



POLITECNICO
MILANO 1863

DIPARTIMENTO DI
INGEGNERIA CIVILE E AMBIENTALE

DYNUSTOP: Dynamically Updating Topology Optimization Prediction using Artificial Intelligence

TESI DI LAUREA MAGISTRALE IN
CIVIL ENGINEERING - INGEGNERIA CIVILE

Author: **Gabriel Garayalde**

Student ID: 942916

Advisors: Prof. Alberto Corigliano & Prof. Matteo Bruggi & Ing. Matteo Torzoni

Academic Year: 2022-23

Abstract

Topology Optimization (TO) is a powerful tool in computational design that optimizes the distribution of material within a specified domain to create structures that adhere to prescribed design constraints. The resulting topology provides an efficient use of material, leading to cost savings and lightweight structures, which is valuable for engineers. However, computing time remains a bottleneck, limiting its application in real engineering contexts. To address this challenge, researchers are exploring the use of artificial intelligence to accelerate the process. This thesis proposes a multi-stage machine learning model that aims to predict an optimal topology in 2D or 3D, in a single shot without an initial form-finding process. The proposed method utilizes a combination of machine learning models to solve distinct parts of the TO problem, resulting in a near-instantaneous optimization. The thesis explores three problem cases, and the results are presented in an interactive computer application that visualizes and updates the predicted topology in real-time in response to user changes of the loading parameters. The proposed method can generate mean average error accuracy of less than 0.035 for the density field for all three test cases, with a predictive speed that is less than 0.5% that of a traditional TO algorithm.

Keywords: Topology optimization, Machine learning, Autoencoder, data-driven prediction

Abstract in lingua italiana

L'ottimizzazione topologica è un potente strumento di progettazione automatica, che cerca la distribuzione ottimale di materiale all'interno di un dato dominio, in modo da minimizzare una assegnata funzione obiettivo e rispettando i vincoli di progettazione prescritti. La topologia risultante consente un efficiente uso del materiale, permettendo strutture più leggere dal costo inferiore. Tuttavia, il tempo di calcolo necessario per questi strumenti è un collo di bottiglia, che ne limita l'applicazione in contesti ingegneristici reali. Per affrontare questa sfida, la comunità scientifica sta esplorando l'utilizzo di algoritmi di intelligenza artificiale. Questa tesi propone un modello di machine learning multi-stage che mira a prevedere una topologia ottimale in 2D o 3D, in modo non iterativo e senza un processo iniziale di ricerca della forma. Il metodo proposto utilizza una combinazione di modelli di machine learning per risolvere parti distinte del problema di ottimizzazione topologica, consentendo un'ottimizzazione quasi istantanea. La tesi esplora tre problemi diversi, e i risultati sono presentati tramite un'applicazione informatica interattiva che visualizza e aggiorna la topologia prevista in tempo reale, in risposta alle modifiche dei parametri di carico da parte dell'utente. Il metodo produce risultati caratterizzati da un errore medio inferiore al 0.035 per il campo di densità in tutti i casi di test considerati, con una velocità predittiva inferiore allo 0,5% di un algoritmo di ottimizzazione topologica tradizionale.

Parole chiave: Ottimizzazione topologica, Machine learning, Autoencoder, predizione data-driven

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
1 Introduction	1
2 Literature Review	3
2.1 Topology Optimization	3
2.1.1 History of SIMP method	4
2.1.2 Modern research of SIMP method	8
2.1.3 Density-based codes for specialized problems	10
2.2 Artificial Intelligence and Machine Learning	11
2.2.1 Origins of AI and ML	11
2.2.2 Deep Supervised learning for computer vision	13
2.3 Topology Optimization and Machine Learning	17
2.3.1 Accelerative Methods	18
2.3.2 Non-iterative, fully data-driven methods	19
2.3.3 Non-iterative, Physics Aided Methods	21
2.3.4 Multistage and Non-iterative methods	22
2.3.5 Objective Function and Sensitivity Filter for Non-iterative Methods	23
2.3.6 Dimensionality Reduction-based approaches	24
3 Topology Optimization Formulation	27
3.1 Topology optimization problem formulation in 2D	27
3.1.1 Modified SIMP approach	28
3.1.2 Optimality Criteria Method	29
3.1.3 Filtering	29

3.2	Topology optimization: extension to 3D	30
4	Matlab Implementation and Dataset formulation	33
4.1	2D	33
4.1.1	MBB	33
4.2	Extension to 3D	40
4.2.1	3D Cantilever	42
4.2.2	3D Bridge	47
5	Machine learning formulation	51
5.1	Proposed DL architecture	52
5.1.1	Step 1: Training the AE	53
5.1.2	Step 2: Training the MLP	62
5.1.3	Step 3: Test the combined deep learning pipeline	64
6	Results	65
6.1	2D	66
6.1.1	2D MBB	66
6.2	3D	72
6.2.1	3D cantilever	72
6.2.2	3D bridge	79
7	Conclusions and future developments	87
	Bibliography	93
	A Appendix	103
	List of Figures	111
	List of Tables	115
	Acknowledgements	117

1 | Introduction

Topology Optimization (TO) is a robust tool for computational design, capable of creating intricate and high-performance structures that adhere to a set of prescribed design constraints for loads and boundary conditions. TO methods optimize the distribution of material within a specified domain by redistributing it to areas where it is most effective according to a chosen optimality criterion.

TO is widely used in aerospace, mechanical, and civil engineering to conceive designs that satisfy specific performance criteria, such as minimizing compliance or maximizing thermal conductivity [1] [2]. The resulting topology provides an efficient use of material within the design domain, leading to cost savings and lightweight structures - all valuable features for engineers. The field of TO has gained momentum for decades, with improvements in hardware cost, computational algorithms, and additive manufacturing techniques all contributing to its boom. Nevertheless, computing time remains a bottleneck, particularly for larger domains and more complex problems, limiting its application in real engineering contexts. To address this challenge, researchers are exploring the application of artificial intelligence (AI) to engineer novel solutions capable of reducing computational times and accelerating the TO process.

This thesis explores the combination of AI and TO to propose a multi-stage machine learning (ML) model, which aims to predict an optimal topology in 2D or 3D, in a single shot without an initial form-finding process. This proposed method would be near instantaneous, and offer clear benefits over the traditional and computationally heavy TO algorithms. The proposed multi-stage pipeline utilizes a combination of ML models to solve distinct parts of the TO problem. First, a multi-layer perceptron (MLP) model that takes as inputs the loading parameters for the specific problem and outputs the corresponding latent space representation. Second, the 'decoder' branch of an autoencoder (AE) that takes as input this previously generated latent space representation and predicts the optimized solution. The underlying hypothesis of this thesis, is that all the topologies in a diverse dataset can be compressed into small latent space representations without significant information loss. These topologies share enough structure and common features

that can be encoded in a reduced vector representation.

In total three different problem cases will be investigated, a 2D Messerschmitt-Bölkow-Blohm (MBB) beam, a 3D cantilever, and a 3D bridge case. Firstly, a training dataset will be created for each of these problems using the popular 'top88' and 'top3D125' Matlab codes [3] [4]. For each of the three cases, three different outputs to the optimization problem are produced, the optimized density field (or the optimized topology), the Von Mises stress field (VM) and the tension or compression (TorC) regions in the structure.

In summary, the proposed method offers clear advantages over the traditional TO methods by allowing the user to generate near optimal solutions almost instantaneously. This can be especially important in the conceptual design phase of a project when it is important to investigate many iterations of the design problem quickly and accurately. The possibility of predicting optimal topologies in a short time, allows the designer to gain an understanding of how the problem responds to variations in the input loading parameters, and make a more informed engineering decision.

The manuscript is organized as follows. In the following chapter, the most significant historical developments in the TO field are reviewed, as well as the current state of the art of AI and the combined field of AI and TO. In chapter 3, a theoretical background to the TO formulation is provided. A complete explanation of the content of the dataset adopted to train the multistage machine learning model is provided in chapter 4. The proposed machine learning strategy is described in chapter 5. The obtained results are presented in chapter 6, along with an interactive computer application that visualizes and updates the predicted topology in real time, in response to user changes of the loading parameters. Finally conclusions and discussion of the results are presented in 7.

2 | Literature Review

This chapter is devoted to review the most important historical advances in the TO field, as well as the current state of the art techniques. The chapter is organized as follows: the first section focuses on the literature review of TO, whilst the second section gives a theoretical background to the TO formulation.

2.1. Topology Optimization

TO is a powerful computational design tool, capable of generating complex and high-performance structural, optimized to comply with a set of given design constraints in terms of loads and boundary conditions. In particular, TO techniques optimize the material distribution within the specified domain, by redistributing it to areas where it is more effective, in terms of a chosen optimality criterion. During this process, the material distribution can take any feasible shape within the assigned domain; in this sense, TO differs from size optimization, which instead deals with configurations established a priori.

TO has become popular in aerospace, mechanical and civil engineering, where engineers can use TO methods to conceptualize designs that fulfil certain performance criteria, such as minimizing the compliance or maximising thermal conductivity [1] [2]. The resulting topology provides many advantages for a design engineer, as the effective use of material within the design domain yields a lightweight structure that saves on material cost. However, many traditional manufacturing processes have difficulty in producing these designs, due to the the high complexity of the resulting geometries. For this reason, the recent advances in additive manufacturing and three-dimensional (3D) printing - capable of handling more complicated 3D geometries - have led to the creation of interesting and practical structural shapes, thus setting a new standard of what could be the future of manufacturing.

The earliest published work on optimization of structures was that by Australian Michell [5], who in 1904 derived the optimality criteria for the layout of minimum compliance trusses. In 1977, Prager and Rozvany [6] published their paper detailing the analytical

optimization of the layout of trusses and grillages, termed 'optimal layout theory'. In 1981, Cheng and Olhoff [7] considered the optimization of ribs systems in solid plates, further extending the research for the optimization of discrete elements and thus setting the basis for the optimization of continua.

The seminal paper by Bendsoe and Kikuchi (1988) [8] introduced the concept of finite-element based TO with the 'homogenization' approach, and was further extended in 1992 by Bendsoe and Diaz [9] for multiple loading conditions, as well as by Suzuki and Kikuchi (1991) [10] and Allaire et al. (1997) [11]. Almost contemporaneously, the landmark paper by Bendsoe in 1989 [12] proposed another numerical method for topological optimization: the Solid, Isotropic Microstructure with Penalty (SIMP) method, which has been further developed by Rozvany et al. (1992) [13] and by Zhou and Rozvany (1991) [14]; over time, the SIMP method has become a more popular optimization method than homogenization methods, due to its simplicity and effectiveness.

Both homogenization methods and SIMP today represent the most common TO approaches, and are collectively considered as density based methods. Other numerical methods, such as level-set methods (LSMs), offer different advantages, yet have not gained as widespread popularity as density based methods. LSMs can handle complex geometries, including holes and irregular shapes, by utilising a continuous representation of the boundary (iso-contours of a level-set function). The level-set function can also be differentiated analytically with respect to the design variables, which allows for the efficient calculation of the sensitivities. These are two advantageous features that LSMs offer over traditional density-based TO methods that can lead to more accurate and efficient numerical models.

The following sections review density based methods, with particular attention devoted to the SIMP method, as this is the method adopted in this thesis for dataset population purposes.

2.1.1. History of SIMP method

First proposed in the late 1980s, the SIMP method is today generally recognized as the most popular TO method, with many uses in research and commercial Finite Element Analysis (FEA) software. The earliest mentions of FE-based topology optimization stem from the work of Rossow and Taylor (1973) [15] who considered the optimal design of a variable thickness sheet. The numerical formulation reduces to the unpenalized relation described by $\rho = s$ in Fig. 2.1. Here, ρ denotes the specific cost or density of an element and s refers to the normalised stiffness value. Empty (white) and solid (black) elements

will have values $\rho = 0$ and $\rho = 1$ respectively. However, the resulting solutions using the unpenalized relation will feature mostly intermediate or grey elements $0 \leq \rho \leq 1$. In TO, the objective is generally to achieve a topology with near binary (black-white) solutions, such that the topology has a clear and distinct form. The landmark paper by Bendsoe in 1989 [12] introduced a SIMP-like method with penalization, under the names of *direct approach* or *artificial density* method. This is illustrated in Fig. 2.1 by the curved dash-dot relation $\rho = s^{\frac{1}{p}}$. This non-linear relation penalizes and suppresses intermediate values of stiffnesses and steers the solution to a more black and white design.

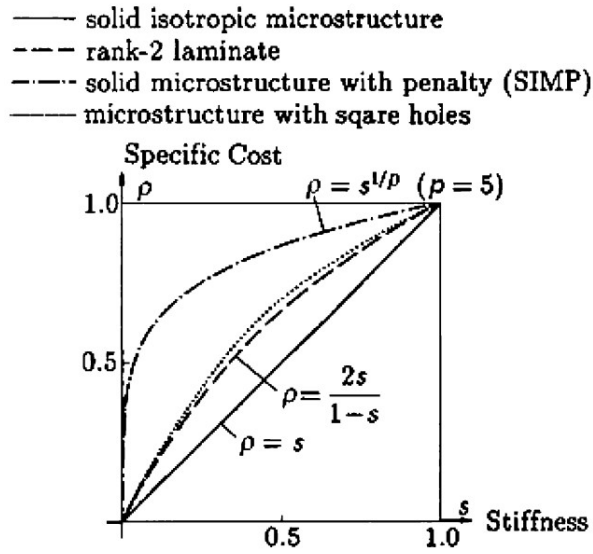


Figure 2.1: Penalization relationship in the stiffness-density graph.

However, despite the simplicity of the SIMP approach, the homogenization method was generally preferred in the research community during this time because it provided more accurate and physically meaningful results, particularly for problems involving periodic structures or composite materials.

In 1991 however, Rozvany and Zhou [16] proposed a physical justification for the power-law relation used in the SIMP method, based on the allowance for fictitious manufacturing costs for intermediate thicknesses of material. The specific cost ρ in Fig. 2.1 can thus be interpreted as the specific cost per unit area of the plate, resulting from the sum of the material and fabrication costs (see Fig. 2.2). Material costs are simply linearly proportional to the thickness of the material. Fabrication costs can be instead interpreted as the cost associated with "machining" down an element of initial thickness t_0 , and is linearly proportional to the amount of material "machined down". We observe however, that for an element of zero thickness the fabrication cost is zero, as there is no cost

associated with cutting out empty elements. The superposition of these two graphs yields a bi-linear relationship that can be approximated by the power law relation seen in Fig. 2.1.

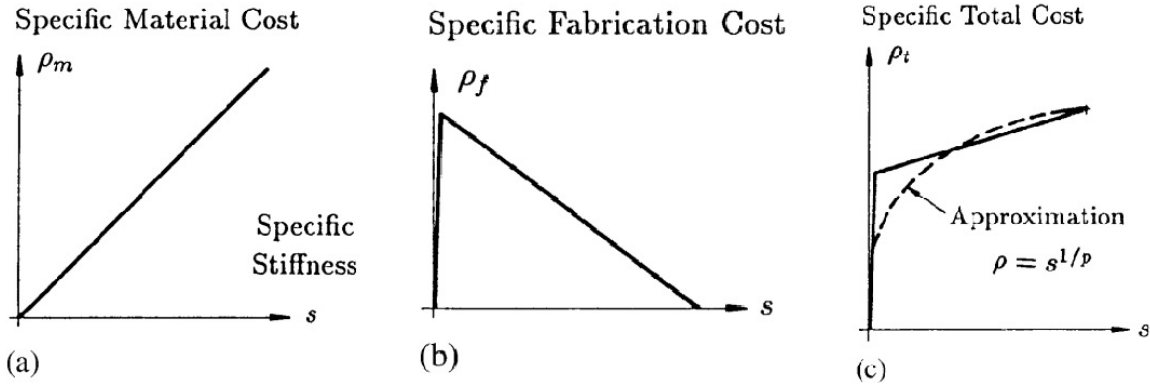


Figure 2.2: Interpretation of the penalization parameter on the basis of fabrication costs.

Rozvany and Zhou [16] further presented a SIMP solution for the MBB beam, which consists of a rectangular beam simply supported on both ends by pinned supports which prevent horizontal movement. Their result, for one half of the MBB beam is shown in Fig. 2.3(a), exhibiting a close approximation to the exact analytical solution presented by Lewinski et al. (1994) [17], shown in Fig. 2.3(b). The latter, along with Zhou and Rozvany [14], presented the Continuum Optimality Criteria (COC) in the context of layout optimization, extending the theory initially developed by W. Prager in the late seventies. COC are iterative optimization techniques, yielding a necessary condition (which is also sufficient in convex problems) for cost minimization. In 1992, Zhou and Rozvany [18] reformulated the COC methods in terms of discretized matrix methods, enabling to increase the efficiency and effectiveness of the SIMP framework. This new method was termed discretized continuum-type optimality criteria (DCOC), and was shown capable of handling a variety of design constraints, such as stress constraints, multiple load conditions, natural frequency constraints and temperature strains.

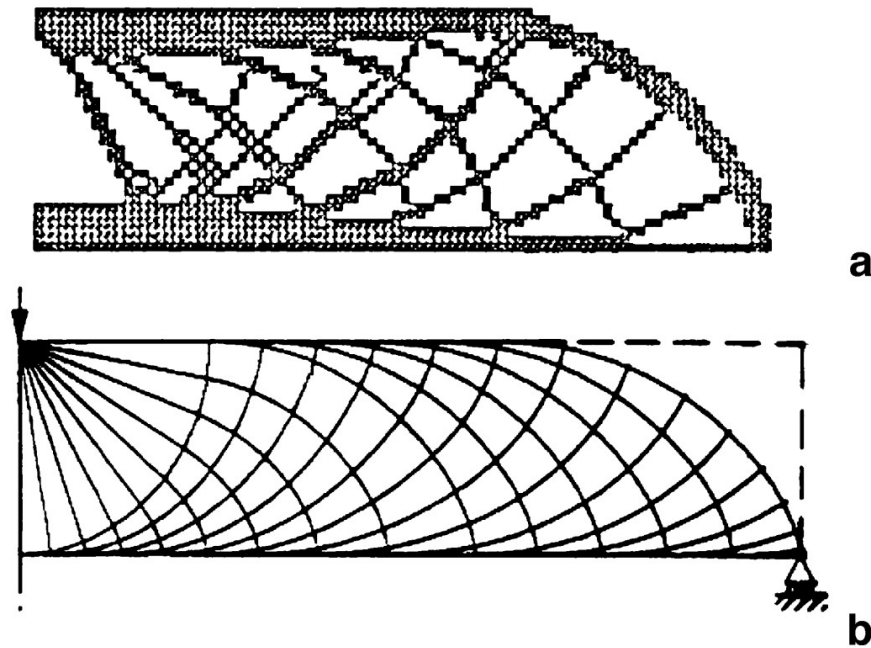


Figure 2.3: a. A SIMP Solution the MBB beam problem presented by Rozvany and Zhou (1991) b. the exact analytical truss solution presented by Lewinski et al. (1994).

The paper by Bendsoe and Sigmund (1999) [1] on material interpolation schemes led to a wider acceptance of SIMP in the research community. This work presented physically feasible microstructures that lent a physical justification to the power-law relation. Other research activities, such as the 99-line MATLAB code for SIMP formulation [19] and the web-based topology optimization program [20] have further increased the popularity of the SIMP method. SIMPs popularity was especially cemented in the wider community after the successful textbook *Topology Optimization: Theory, Methods, and Applications* [21] written by Bendsoe and Sigmund in 2003.

Rozvany and Zhou (1994) [22] first described what would later become known as the 'continuation method' [23], where a global optimum is found first for the unpenalized problem $\rho = 1$, yielding a solution with intermediate grey densities; in successive iterations, the penalization parameters is then slowly increased to yield more black-and-white densities, and resulting in a final solution that is not far from the global optimum. Another feature that is common to modern SIMP methodologies is the filtering method, a highly efficient but partially heuristic solution suggested by Sigmund in 1994 [24] in response to the checkerboarding problem, an issue first noticed in relation to homogenization methods [25] in 1993. The 2007 paper by Sigmund provides a more detailed examination of such filtering methods [26].

2.1.2. Modern research of SIMP method

The boom in modern educational research papers pertaining to density-based methods is in large part due to the original paper published by Sigmund (2001) [19], which published the educational 99-line topology optimization code 'top99'. Written in MATLAB, the code can handle 2D compliance minimization problems, with extensions for multiple load cases, passive elements and different boundary conditions. In 2010, Andreassen et al. [3] improved the code, creating 'top88', a shortened 88-line optimization code in which considerable speed-ups were achieved by pre-allocating arrays and vectorizing loops. Optional black-and-white projection filtering methods are also discussed and provided.

In 2012, Talischi et al. [27] published a general purpose mesh generator for polygonal elements written in MATLAB which provides the input required for finite element and optimization codes that use linear convex polygons. Polygonal discretizations have the advantage of not being susceptible to numerical instabilities such as checkerboarding, which otherwise affect lower order triangular and quadrilateral meshes. This polygonal discretization method was then utilized in Talischi et. al [28], which provided the MATLAB code 'Polytop', a structural topology optimization tool that includes a general finite element routine based on isoparametric polygonal elements (see Fig. 2.4). The advantage of the latter is that, the finite element and sensitivity analysis routines are decoupled from the optimization formulation and can thus be extended and modified independently.

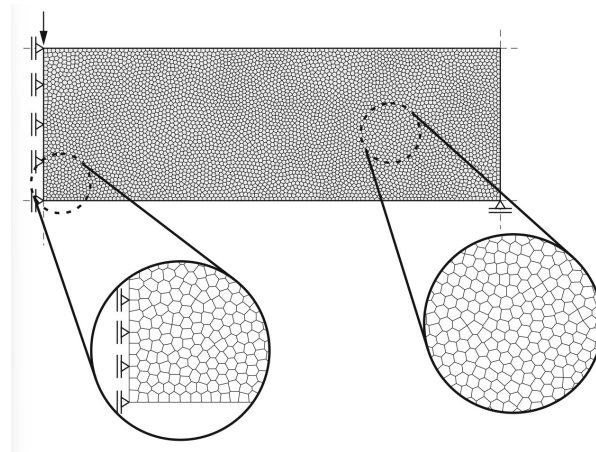


Figure 2.4: MBB beam discretized into polygonal elements using 'Polytop'.

The 2D '88-line' and '99-line' codes were used as the basis by many authors to extend the minimum compliance problem to 3D. In 2014, Liu and Tovar [29] published their work on 3D minimum compliance TO problems, with the efficient 169-line MATLAB code 'top3d', built upon the 'top88' code. The paper also includes instructions to define multiple load

cases, active and passive elements, heat conduction problems, as well as the capacity to implement general non-linear programming strategies such as 'sequential quadratic programming' and 'method of moving asymptotes' (MMA) [30]. Likewise, Lagaros et al. (2018) [31] solved the 3D minimum compliance problem, and released their code implemented in C language, also allowing to model the obtained results in SAP2000. Zeng and Ma (2020) [32] presented an efficient gradient projection-based method for structural topological optimization problems characterized by a nonlinear objective function. Their code is similar in structure to 'Polytop' [28], and includes benchmark problems such as the MBB beam and the 3D cantilever. Chi et al. (2020) [33] proposed a 3D topology optimization framework employing the virtual element method (VEM) [34], using polyhedral discretization methods capable of handling arbitrary shapes.

For increasingly larger domains, the computational cost of numerical optimization routines can be very high, and thus the efficiency of the code is of high significance. Amir et al. (2014) [35] reduced the computational cost associated with the nested analysis problem, by exploiting the specific characteristics of a multigrid preconditioned conjugate gradients solver. The applicability of the proposed procedure is demonstrated on several 2D and 3D examples involving up to hundreds of thousands of degrees of freedom.

The research in parallel computing has also enabled the significant improvement in computational times for TO problems. Niels et al. (2014) [36] published an easy-to-use parallel computing framework which solves the 3D minimum compliance problem on structured grids, using standard FEM and filtering techniques; the resulting fully parallelized framework is capable of handling more than a 100 million design variables. The latter has been further extended by Zhang et al. (2021) [37] in 'TopADD', a 2D and 3D integrated TO parallel-computing framework developed to deal with arbitrary design domains. It can seamlessly switch between 2D and 3D, and besides compliance minimisation, also allows for compliant mechanisms and the heat conduction problem.

In 2020, Ferrari and Sigmund [4] published the MATLAB codes 'top99neo' and 'top3D125' for 2D and 3D topology optimization, respectively. Both codes showed considerable improvements in computational efficiency, with 'top99neo' entailing a speed-up of about 2.55 to 5.5 times with respect to its 'top88' predecessor [3], and 'top3D125' showing a speed-up of up to 1.9 times compared to the code of Amir et al (2014) [35]. In both cases, these improvements are due to more efficient procedures for the assembly and implementation of filters in the design update phase.

2.1.3. Density-based codes for specialized problems

Many researchers have extended the standard density based codes to more complex and specialised design problems, such as multiscale, multiple materials, buckling criteria and stress constraints, amongst many others.

Multiscale problems refer to structures where a macroscale structure consists of a periodic repetition of a local microstructure (see Fig. 2.5). Bone or bamboo are examples of strong, yet lightweight multi-scale materials found in nature. With the rapid development of additive manufacturing in recent years, there has been a growth in interest for optimizing multiscale structures, to produce new materials with superior performance characteristics. In 2015, Xia and Breitkopf [38], built on 'top88' [3] a code that adopts an energy-based homogenization approach, generating 2D microstructures with optimal material properties such as maximum shear and bulk modulus.

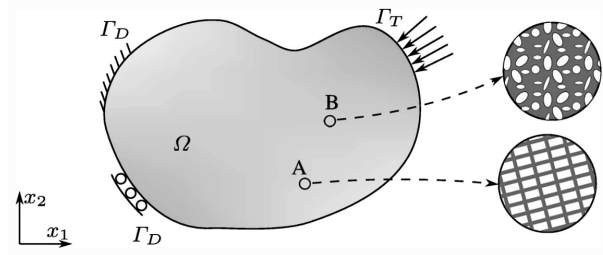


Figure 2.5: Periodic repetition in the local microstructure of multiscale materials.

In the case of problems with multiple materials, Tavakoli and Mohseni (2014) [39] presented a new algorithm for the solution of multi-material TO problems. The proposed method subdivides the multi-phase TO problem into a series of binary sub-problems, which are solved sequentially using a binary phase TO solver. The algorithm is useful for solving multi-material minimum structural and thermal compliance TO problems, based on the classical optimality criteria method. Sanders et al. (2018) [40] published 'PolyTop', a MATLAB code built on top of 'PolyMat' for compliance minimization on unstructured polygonal FE meshes that can solve for many materials and volume constraints.

Ferrari et al. (2021) [41] developed a 250-line MATLAB code for topology optimization with linearized buckling criteria. This code utilizes the efficiency improvements introduced in 'top99neo' [4] to speed up the buckling analysis, and included stiffness, volume, and buckling load factors either as the objective function or as constraints.

Regarding TO with stress constraints, Giraldo-Londoño and Paulino (2021b) [42] developed 'PolyStress', a MATLAB code built upon 'PolyTop' [40] with local stress constraints

handled by means of the Lagrangian method. The 'PolyStress' code can account for both linear and nonlinear material properties and is provided with a library of benchmark problems. A 3D extension was proposed by Deng et al. (2021) [43], proposing a 146-line MATLAB code for topology optimization with stress minimization. This work adopts the adjoint method for the 3D sensitivity analysis of the p-norm global stress measure, and MMA [30] as nonlinear optimization solver.

2.2. Artificial Intelligence and Machine Learning

Over the recent years, Artificial Intelligence (AI) has become an undeniable and unstoppable force that has both captivated and alarmed the scientific community. It has become deeply embedded into the fabric of our every day lives, from our phones, cars or even email spam filters. Recent examples such as ChatGPT, a large language AI model, or Dall-E-2, an AI model that can create images from text, generate results that are incomprehensibly accurate, and yet again raise the bar for what AI algorithms are capable of. This section seeks to provide context around AI and machine learning (ML), from its origins to current state of the art research that is pushing the boundaries of what is possible. The focus will be mainly on ML algorithms for computer vision and image recognition, as these form the basis of this thesis.

2.2.1. Origins of AI and ML

Artificial intelligence, contrary to popular belief, is not a new and emerging field, but rather a field of study born in the 1950's, when data scientists posed the question of whether computers could be made to "automate intellectual tasks normally performed by humans" [44]. AI is a general field, encompassing both ML and deep learning (DL), but also encompassing areas which do not strictly involve 'learning', such as early chess programs with hard-coded explicit rules. The relationship between these fields is shown in Fig. 2.6.

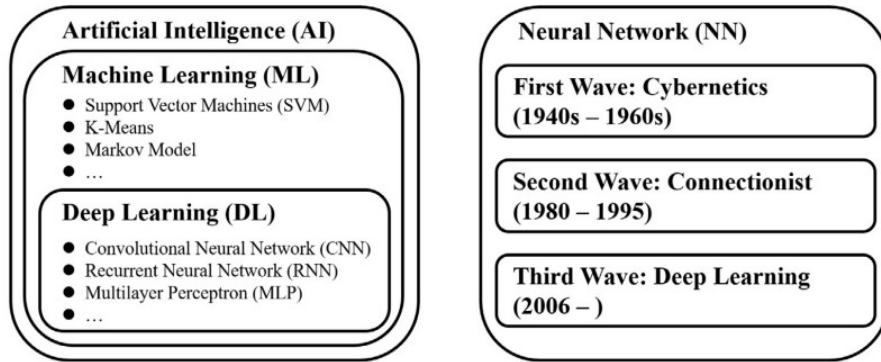


Figure 2.6: Relationship between Artificial Intelligence, Machine learning and Deep learning. [44]

In 1990's, with the advent of increasing computing capabilities at lower cost, and the advances in software engineering, the field of ML began to experience a flourish of research. Whereas traditional programming requires the user to program explicit hard-coded rules, yielding the computation of an output answers for a given data set, ML completely reverts this programming paradigm. The user inputs the data (such as photos of animals) and the answers (the tag labeling the names of the animals in the photos first, and then the ML model outputs the rules describing the underlying functional link (see Fig. 2.7). To this aim, the ML model is trained on input and output pairs to learn a statistical structure correlating the two. The user is not tasked with learning or guessing the rules between the data a priori, but rather of setting up the algorithm that allows to automatically learn these rules, often in the form of tunable parameters.

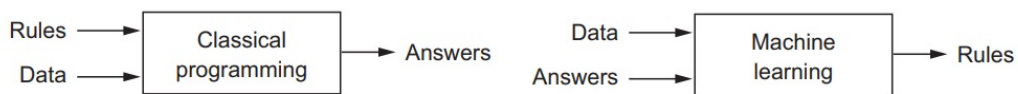


Figure 2.7: Difference between classical programming and machine learning [44].

An artificial neural network (ANN) is a popular type of ML algorithm. The term 'neural' is a reference to the neurons and neural pathways in our brains. The most fundamental building block of an ANN is a perceptron, shown in Fig. 2.8. A perceptron takes a vector of input values, multiplies each input by a corresponding weight value, and sums the weighted inputs. The resulting summation is then usually ruled by an activation function; this is often called neuron in the ML community.

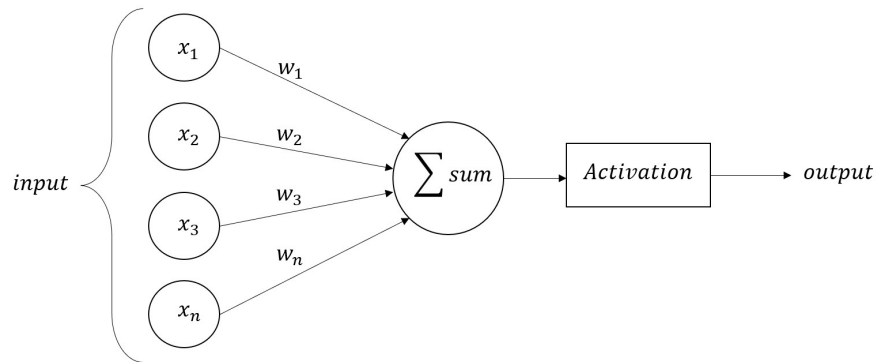


Figure 2.8: A simplified diagram of a perceptron

Perceptrons are often arranged in layers to form a neural network (see Fig. 2.9), which is called multi-layer perceptron model. Typically, when there are two or more hidden layers, a neural network is said to become 'deep'. These deep learning models are capable of learning increasingly complex representations of the input data. Deep learning is a sub-field of machine learning, involving the learning of neural network models characterized by a deep architecture.

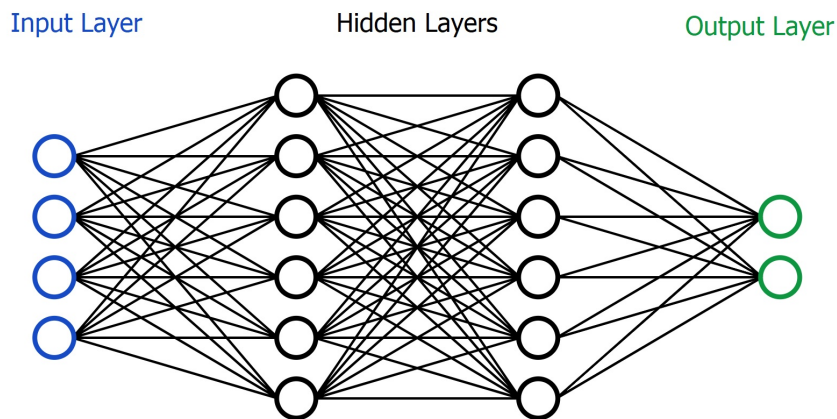


Figure 2.9: A generic representation of a neural network.

2.2.2. Deep Supervised learning for computer vision

Machine learning has grown into a boundless field of immeasurable scope, with new sub-fields and categories being added every day. However, the majority of cases can be roughly divided into supervised and unsupervised learning paradigms. Supervised learning occurs when the model learns a mapping between the input and the known target data, given a dataset of examples. This is often the case of classification or regression problems; other

common applications that typically fall under this category are character and speech recognition. On the other hand, unsupervised learning learns from unlabeled data that has not been classified or categorized, with the aim of discovering hidden patterns or structure in the data

One of the earliest examples of pattern recognition research was published in 1959 by Selfridge [45], who introduced a simplified multilayer model (termed 'Pandemonium') that tested many hypotheses on the input data. Each hypotheses received feedback based on the correctness of their response, which allows the model to adjust the weights and improve the general performance over time. This presented one of the earliest examples of an error-driven learning mechanism and contains many parallels to the behaviour of modern neural networks and machine learning techniques. How to properly train these multilayer networks however was not fully understood until the 1970's and 1980's, when back-propagation techniques and stochastic gradient descent became popular [46].

One particular deep feedforward network proved much easier to train, yet yielded higher prediction accuracy for image detection and classification, was the convolutional neural network (CNN) [47] [48]. CNNs are a type of neural network that contains convolutional layers, which apply a set of filters to the input image. Each filter is convolved with the image to produce a feature map, which captures different patterns or features present in the image. CNNs have proved significantly more effective in image recognition tasks compared with densely connected layers largely due to the fundamental differences in their layer architecture. Dense layers are trained to recognise *global* patterns in the input space, whereas convolutional layers learn *local* patterns and features. The network is capable of distinguishing local patterns in the input space such as edges and textures, which in turn combine to form higher-level features. We can think of images in terms of compositional hierarchies; where high-level features are composed of lower-level features (see Fig. 2.10).

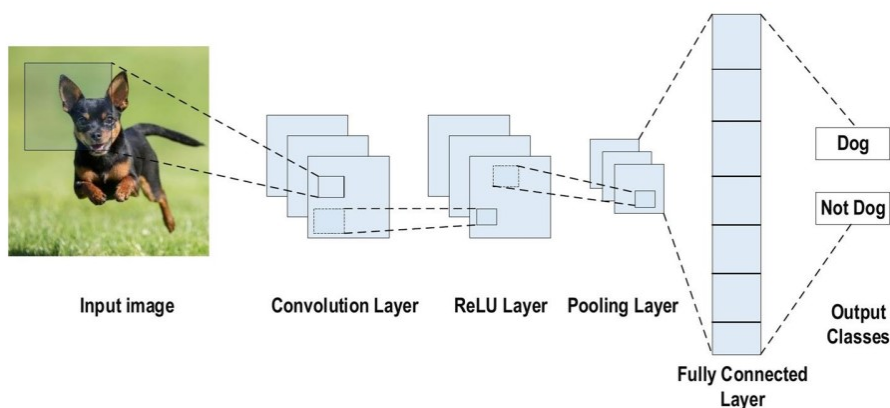


Figure 2.10: Example of CNN architecture for image classification.

LeCun et al. (1990) [49] published the first paper on CNNs trained with backpropagation for classifying low-resolution images of handwritten digits. Backpropagation is a technique for training neural networks whereby the gradient of the loss function with respect to the weights is calculated, and then this gradient information is used to update the weights in the direction that minimizes the loss. This procedure is repeated multiple times until the networks performance converges to a satisfactory level. Another paper published in 1998 by LeCun et al. [50], further showed how neural networks, and in particular CNNs, are able to handle the recognition of complex interdependent outputs, such as sequences of characters in a bank check. By the late 1990's this system was responsible for reading over 10% of all bank cheques in the United States.

Despite significant progresses in the fields of image recognition, such as face detection [47], embryo cell detection and image segmentation [51] (image segmentation here refers to the process of training algorithms to divide and identify different regions or objects in an image), the success of CNNs in computer vision was still largely limited in the early 2000's due to, amongst other factors, insufficient hardware capabilities and still evolving algorithmic models. In 2010 however, Leon Bottou [52], made an important breakthrough by demonstrating the amazing performance of stochastic gradient descent as an optimization algorithm. Another significant turning point occurred in 2012, with the success of the ImageNet competition, which tasked teams to classify 1.2 million images into 1000 different classes. The winning entry produced outstanding results, with errors almost halved with respect to previous state of the art model [53]. This was largely due to the use of GPUs for parallelizing the training process, the advent of the ReLU activation function, and a newly-developed regularization method called 'dropout'. In particular, the effectiveness of exploiting the non-linear ReLU activation function $f(z) = \max(z, 0)$ in the hidden layers for the supervised training of deep neural networks was shown earlier in 2011 [54]. On the other hand, dropout regularization was another important innovation introduced in 2014 by Srivastava et al. [55] to prevent overfitting of the training data (when the model simply memorizes the training data, instead learning a possible model behind them). Dropout works by randomly dropping out (i.e., setting to zero) a fraction of the neurons in a layer during training. By doing this, dropout prevents the neurons from co-adapting too much to each other, and forces the network to learn more robust features that generalize well to new data.

Many innovative and improved CNN architectures were developed since the first ImageNet competitions. CNNs have been made capable to match human performance or even out-perform humans in many particular image recognition tasks [48]. For instance, a notable contribution is the ResNet model proposed by He et al. in 2015 [56], a significantly deeper

network, in which the layers are reformulated as residual blocks. Residual blocks exploit an identity mapping of the input, to provide an extra path toward the terminal part of the network by skipping some layers, which is then added to the outputs of the stacked layers through an element-wise summation. The idea behind this strategy is that for the trainable layers, it is easier to fit the result of the desired mapping, instead of the desired underlying mapping directly. Accordingly, multiple residual blocks are usually exploited within the same architecture to perform an iterative refinement of the output features, with each block slightly improving the representation. This strategy has proven effective in allowing for enhanced performance, without adding extra parameters or computational complexity, when further improvements were no longer achievable through deeper architectures. The ResNet model achieved a 3.57 % error rate in the ImageNet classification task, exceeding human performance, which stands at approximately 5% error. Image classification tasks refer to the process of training an algorithm to recognize and categorize images based on their content. The aim is to create a model that can automatically assign images to specific classes or categories, for instance identifying whether an image contains a dog or a cat. In 2017, Hu et al. [57] further reduced this error rate in the ImageNet classification task down to 2.25 % with the development of squeeze-and-excitation networks.

Given the CNN's outstanding performance in image classification and segmentation, they have been recently exploited in many successful applications in diverse fields. In 2015, Rosenberg et al. proposed the U-Net [58], a specific type of CNN-based encoder-decoder network, which concatenates representations from the the contracting path to the expanding path (see Fig. 2.11). This was initially proposed in the context of biomedical image segmentation, but more recently has also been generalized to the field of topology optimization with great success. In the medical industry CNNs have been also used for the classification of patients with Alzheimer's disease (2020) [59], and the 3D segmentation of the human brain (2020) [60]. In materials science, deep learning sees rapidly emerging applications spanning atomistic simulation, materials imaging and spectral analysis (2022) [61].

Other pioneering machine learning techniques include batch normalization, which was introduced in 2015 by Ioffe and Szegedy [62]. Batch normalization works by normalizing the input data at each layer of the neural network, so that the data has a mean of zero and a standard deviation of one. This helps to ensure that the input data to each layer is more consistent, which can help to speed up training and improve the accuracy of the model [63]. Weight decay has become a standard tool for improving the generalization performance of neural networks [64] and the accuracy of the model [53]. This regularization technique

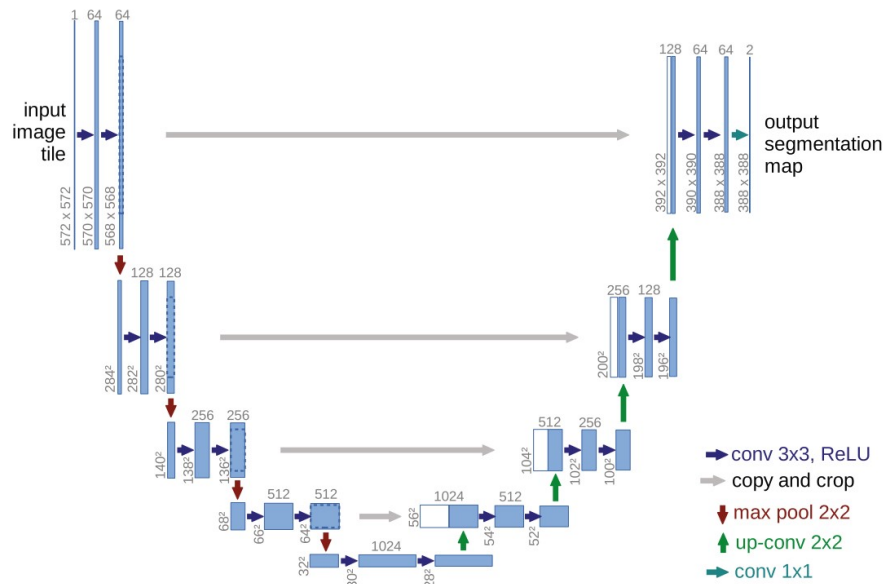


Figure 2.11: Example of a U-Net architecture.

affects the learning by scaling the weights of the model before they are updated in the gradient direction. Such a shrinking effect implies that the model behavior will not change much for similar inputs, thus avoiding learning local noise in the data.

2.3. Topology Optimization and Machine Learning

Topology optimization as a field has been gaining momentum for many decades; with the lowering cost of hardware, improved computational algorithms and the advent of additive manufacturing techniques all contributing to this boom. However, despite these improvements, the bottleneck for many problems is still the computing time, which grows exponentially for larger domains and complex problems, thus hampering the application to real engineering design contexts. For this reason, many are looking towards the field of AI and ML to engineer new and novel solutions capable of reducing the computational times and speed up the topology optimization process. Publications in this new research area fall into a few generalized categories depending on how ML is exploited within the TO algorithms. The main branches are the following: *accelerative*, which utilises the TO algorithm for a part of the optimization process; *non-iterative* (both fully data-driven and physics-aided), which conversely aims to predict the optimal topology in one-shot without an initial form finding process; and *multistage*, which utilises a combination of ML models to solve distinct parts of the TO problem.

2.3.1. Accelerative Methods

In 2017, Sosnoviks and Oseledets [65] proposed one of the first deep learning approaches to 2D topology optimization by considering it as an image segmentation task using a deep CNN (see Fig. 2.12). The final black-and-white topology is predicted by using as input to the neural network two grayscale images: the density distribution and the gradient of the densities, both attained after only a handful of iterations of the SIMP solver. This method is coined as an accelerative method; the SIMP algorithm is utilised for a small number of iterations, whilst the more time consuming *refinement* phase of the solver is replaced with a deep learning model, greatly accelerating the task.

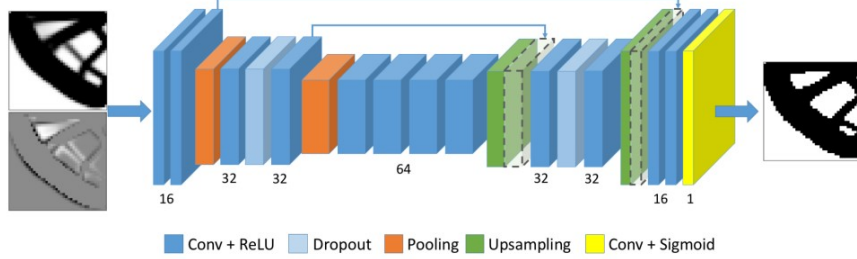


Figure 2.12: ML Architecture used in Sosnoviks and Oseledets work in 2017 [65].

Lin et al. (2018) [66] followed a similar approach, to greatly reduce the second refinement stage of the SIMP solver. The work focused on 2D heat conduction problems, using a CNN based AE (see Fig. 2.13). Kolliaris et al (2020.) [67] developed 'DLTOP', a novel accelerative method in which 'deep belief networks' (DBN) are used for discovering connections between the density values of each finite element of the domain along the first iterations of SIMP approach with its final outcome. First, the intermediate optimization result from the SIMP method serves as input to the DBN, which predicts an optimal elements density, then the DBN-predicted topology is again refined through the SIMP method. Banga et al. (2018) [68] extended these methods to 3D, by providing three types of input into a 3D CNN: 3D density distribution of voxels after a number of intermediate iterations, gradient of voxel densities, and forces and boundary conditions along the x, y, and z directions. This yielded a 40% reduction in computational time, yet with a binary and root mean square deviation accuracy of 96.2% and 79.7%, respectively. Senhora et al. (2022) [69] proposed an accelerative deep learning model consisting of a CNN with residual connections, characterized by a strong generalization capability of solving a wide variety of problems with different geometries, boundary conditions, mesh sizes, volume fractions and filter radius. The model uses a two-resolution approach, with inputs to the deep learning model organized in a coarse and a fine mesh. The proposed model

shows speed-ups of up to 30 times compared to traditional topology optimization, and is demonstrated for both 2D and 3D compliance minimization problems.

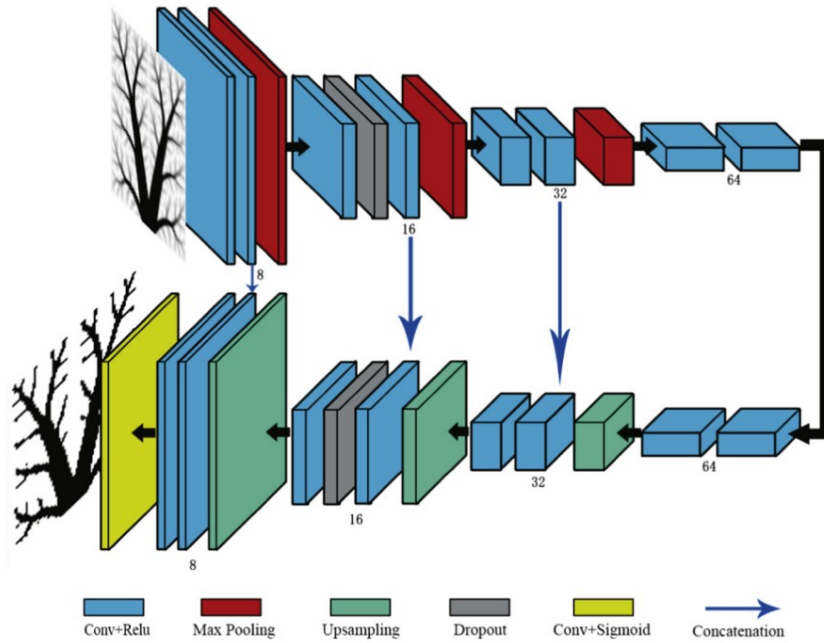


Figure 2.13: ML Architecture used in Lin et al. [66].

2.3.2. Non-iterative, fully data-driven methods

Non-iterative optimization methods, as opposed to accelerative methods, predict an optimal shape that satisfies the design conditions, without an iterative form finding process.

Abueidda et al. (2020) [70] proposed a CNN ResUnet that directly derives an optimized 2D geometry for assigned loading, boundary conditions, and volume fraction in the case of (i) linear elasticity with small deformation (without nonlinear constraints), (ii) nonlinear hyper-elasticity (neo-Hookean material) with geometric nonlinearity, and (iii) linear elasticity with stress constraint, a nonlinear constraint. The ResUnet takes the advantages of the U-Net architecture [58] and combines it with the effectiveness of residual learning. Kollmann et al. (2020) [71] also utilised the ResUnet for the non-iterative prediction of optimal 2D meta-materials. The volume fraction, filter radius, and design objective (e.g. maximum bulk modulus, maximum shear modulus, or minimum Poisson's ratio) were used as inputs to the network. Zheng et al. (2021) [72] presented another method for predicting 3D topologies, yielding good generalization ability for variable design domains and different loading and boundary conditions. The boundary conditions, load conditions, volume fraction and domain size are directly encoded onto an input tensor, and

a U-Net encoder decoder architecture then predicts the corresponding optimal topology (see Fig. 2.14).

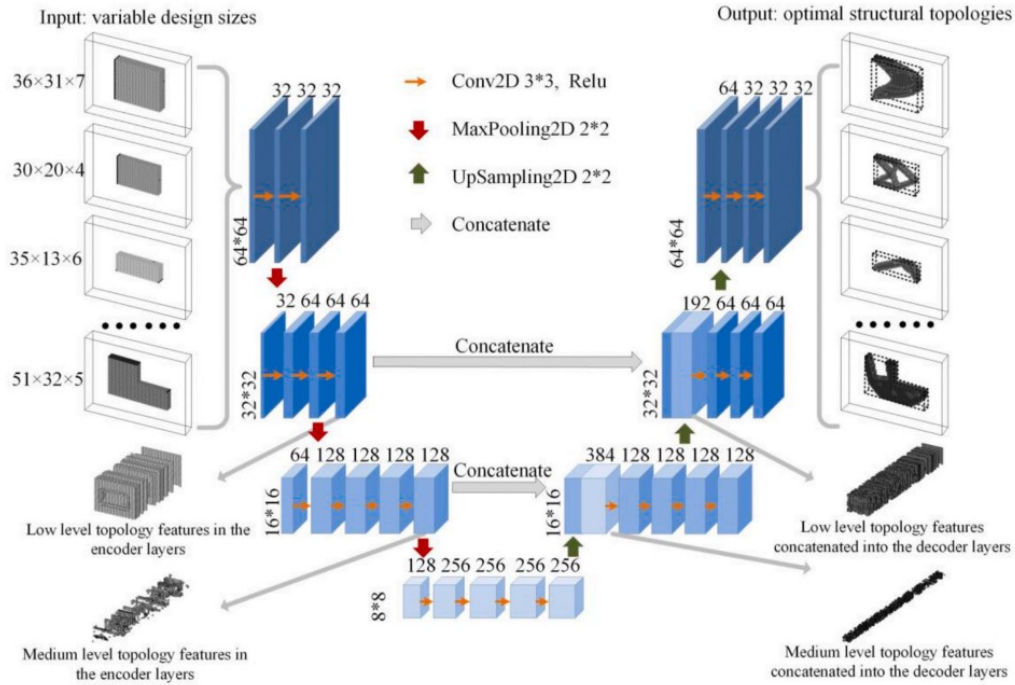


Figure 2.14: U-Net Architecture used in Zheng et al. (2021) [72].

Zhang et al. (2021) [73] introduced a non-iterative topology optimization neural reparameterization framework, whereby the objective function in the TO formulation is directly used as the loss function to train the network. The update of the pseudo-density design variables in the conventional TO method is therefore transformed into the update of the neural networks parameters, termed 'reparameterization'. Compared with existing methods, this has proved effective to overcome the phenomenon of 'structural disconnections'. Rawat and Shen released a number of papers using generative adversarial networks (GANs) to predict optimal topologies. In 2018 [74], they presented a paper in conditional Wasserstein GANs (CWGANs) predict 2D topologies, and also map the generated shape to the corresponding optimization condition behind it, such as volume fraction, penalty and radius of the smoothing filter. In 2019 [75], extended this strategy to 3D problems, by predicting 3D optimal topologies using a similar architecture of CWGANs. Herath and Haputhanthri (2021) [76] also utilised a combined CNN and conditional GANs (cGAN) framework to generate the optimal design, plot the Von Mises stress contours on the optimal design and predict the locations characterized by the maximum Von Mises stress values.

2.3.3. Non-iterative, Physics Aided Methods

A subset of non-iterative methods is that of non-iterative physics-aided methods, that in contrast to the former, are also informed by the physics of the problem (either incorporated as an additional term in the loss function to be minimized or in a principled formulation of the input and target data tensors). A handful of these works require a first run of the FEA algorithm to generate an initial field of interest, for instance in terms of displacement or strain, however they are still considered non-iterative for the sake of simplicity. Zhang et al. (2019) [77] utilised a U-net, whose input tensor contains information regarding the final displacement, strain and volume fraction of the SIMP formulation (see Fig. 2.15). The training set involved 80,000 images of cantilevers, yielding excellent performance on unseen boundary conditions. Xiang et al. (2022) [78] proposed a similar approach in 3D with 10 input tensors - consisting of displacements, longitudinal strains, shear strains and volume fraction - into the neural network. The proposed method shows good generalization capabilities, even in the case of stress minimization problems.

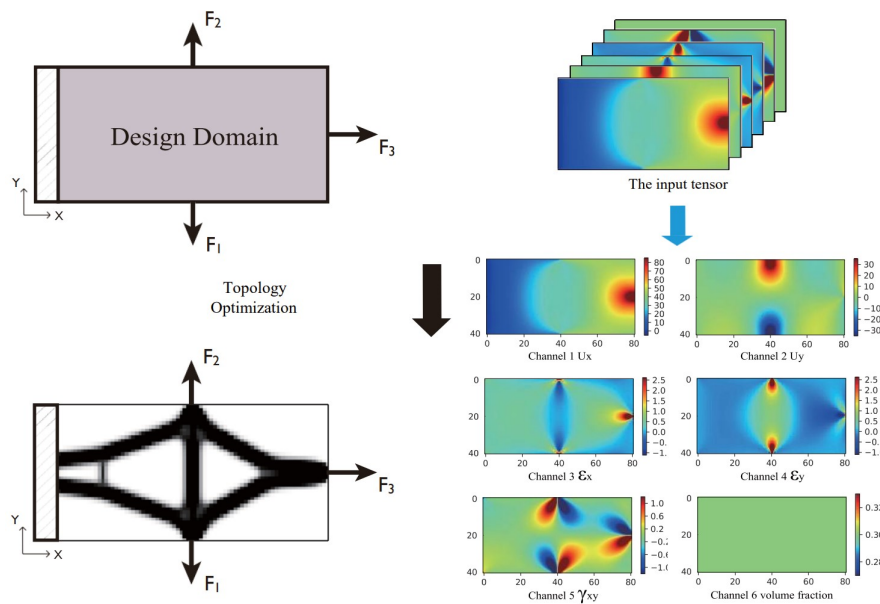


Figure 2.15: Input tensor fields used in the network proposed by Zhang et al. (2019) [77].

Yan et al. (2022) [79] proposed a framework that takes the principal stress field from the first SIMP iteration as input tensor for a CNN based neural network. The authors reported results of an acceptable level of accuracy for 2D topologies, even exploiting training datasets of limited size. Nie et al. (2021) [80] proposed 'TopologyGAN', a framework that takes the strain energy density and Von Mises stress values obtained from the FEA as the inputs for a GAN, along with the volume fraction, boundary conditions and

load. In particular, the generator leverages on a hybrid architecture called U-SE-ResNet, which is a U-Net with a SE-ResNet module in a downsampling-upsampling fashion.

Cang et al. (2018) [81] proposed a novel non-iterative method termed 'theory-driven' learning (which differs from purely data-driven supervised learning), whereby an adaptive sampling procedure is adopted rather than a batch-mode training method. The deviation of the predicted solutions from the optimal solution is quantified and used to select the new data points to sample in the learning process. Xiang et al. (2022) [82] introduced a non-iterative U-Net based encoder decoder framework that computes the optimized topology and the corresponding Von Mises stress for a stress-constrained problem. The p-norm stress is adopted to measure the global stress level, and the MMA is used to generate a training dataset for varying loading conditions and volume fractions. This strategy has been also extended to 3D problems with negligible added computational cost. Jeong et al. (2023) [83] proposed a physics-informed neural network-based topology optimization scheme, where an energy-based deep learning model replaces the FEA to numerically determine the displacement field. In this model, the governing physics equations are embedded in the loss function of the neural network. The proposed approach has been also extended to multiple loading conditions, passive domains and 3D linear elastic compliance minimization

2.3.4. Multistage and Non-iterative methods

Other authors have approached the combination of topology optimization and machine learning by means of a multistage non-iterative method, whereby two or more neural networks are synergistically exploited to solve distinct parts of the TO problem; this often results in a more complex architecture that can lead to a better accuracy. For instance, Yu et al. (2018) [84] proposed a CNN-based encoder-decoder network, followed by a cGAN serving as the second stage refinement (see Fig. 2.13). The training datasets consisted in optimal topologies, respectively featuring low and high resolution. The CNN-based encoder-decoder first predicts the low resolution image by taking as input the constraints information and the load components along the x and y directions. The resulting image is then upsampled to high resolution using the cGAN. Li et al. (2019) [85] applied a similar architecture for conductive heat transfer problems, with heat sink position, heat source position, and mass fraction coded into the input tensor. Rade et al. 2020 [86] developed two approaches, density sequence and coupled density compliance sequence models, both respectful of the underlying physics, as well as of the topological constraints and of the result obtained with the SIMP topological optimization algorithm. In this case, the training datasets were generated both for 2D and 3D problems, providing notable

advantages compared to the baseline density-based approach in both cases.

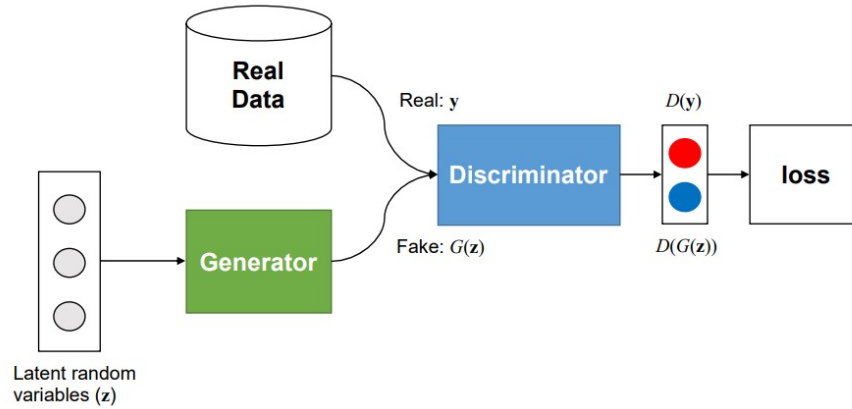


Figure 2.16: Generator and discriminator GAN network taken from Yu et al. (2018) [84]

Behzadi and Ilies (2021) [87] introduced a non-iterative deep transfer learning method based on CNNs. The proposed method uses low resolution datasets to train a source network and a much smaller high resolution dataset to fine-tune a target network. This multistage combination of networks is capable of leveraging the learned knowledge from the source network and transfer it to the target network so that the latter requires a much smaller number of training cases than an equivalent deep CNN to make predictions with the same level of accuracy. Additionally, it is capable of handling high-resolution design domains with good generalization capability.

2.3.5. Objective Function and Sensitivity Filter for Non-iterative Methods

A significant portion of the computing time of TO algorithms is often devoted to evaluating the objective function and the FEA-based sensitivity analysis. To this aim, deep learning models can be applied in order to accelerate or replace these computationally expensive operations. For instance, Lee et al. (2020) [88] used a CNN to predict the compliance information by using low resolution density images as an input to the network. The evaluation of optimality may thus be performed without having to run any FEA code, resulting in a significant speed up. Kim et al. (2021) [89] exploited machine learning algorithms to predict the continuous anisotropic effective material properties for simultaneous design of the overall topology configuration and local fiber material layout in functionally graded composite structures. The methodology is applied to both 2D and 3D test cases for the minimization of structural compliance and compared with benchmark cases that use the typical asymptotic homogenization design method. Takahashi et

al. (2019) [90] created a predictive model for the sensitivity using a CNN on cantilever examples. Chi et al. (2021) [91] proposes a similar method for mapping the design variables to their corresponding sensitivities. The latter also includes a novel online training concept using data from earlier iterations of the TO procedure, and also in this case the method is tested on 2D and 3D benchmarks.

2.3.6. Dimensionality Reduction-based approaches

Following a different perspective, other researchers have chosen to create more efficient predictive models and reduce the associated computational burden by reducing the dimensionality of the design space. For instance, Guo et al. (2018) [92] presented a novel approach that uses an augmented variational autoencoder (VAE) leveraging on style transfer [93] to encode 2D topologies into a lower-dimensional latent space, which is then decoded back into 2D topologies (see Fig. 2.17). Moving the optimization process to the latent space was shown to be successful with regards to: generating new designs, improving solution quality, and adapting training data from related problems to construct a design representation enabling a proper solution to the considered problem. Ulu et al. (2015) [94] devised a projection-based mapping of optimal 2D topologies onto a lower dimensional space using principal component analysis (PCA). Thus, optimal topologies for unseen loading configurations can be predicted, by exploiting a ML model previously fitted on the input loading configurations and the corresponding PCA interpolation coefficients.

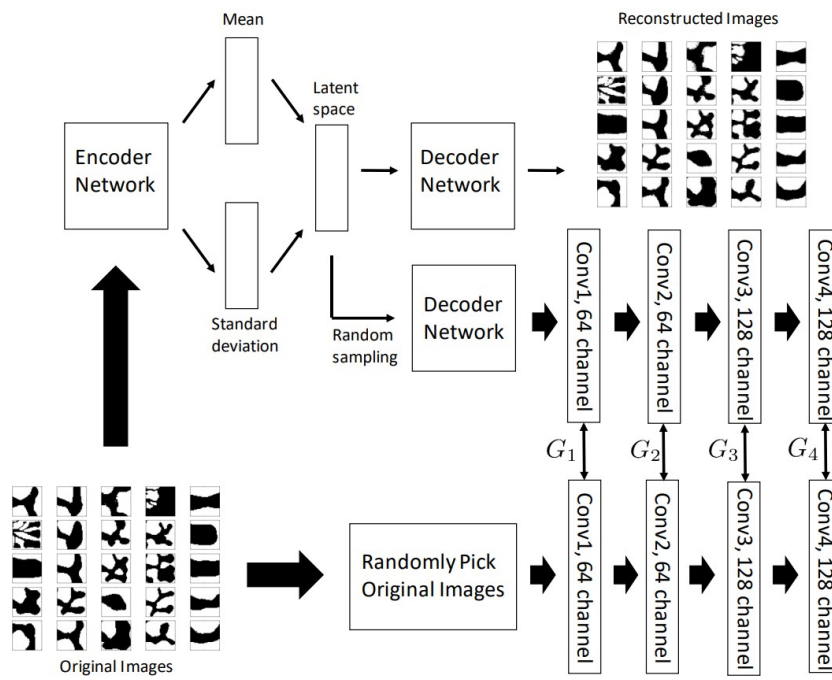


Figure 2.17: VAE with style transfer in Guo et al. (2018) [92].

3 | Topology Optimization Formulation

This chapter presents the general formulation for the topology optimization problem, and the solution algorithm based on the SIMP method. This will be adopted for dataset generation purposes, as detailed in the next chapter. In particular, we adopt the 88-line MATLAB code 'top88.m' developed by Andreassen et al. (2010) [3]¹ for 2D problems, whilst the extension to 3D is based on the 125-line code proposed in the paper by Ferrari and Sigmund (2020) [4]. The following sections review the theoretical notions underlying these two codes.

3.1. Topology optimization problem formulation in 2D

The focus of the following formulation is restricted to minimum compliance problems with a statically applied load. The MATLAB code is divided into the main program, the optimality criteria optimizer, mesh independency filter and the finite element code. Extensions to the problem such as multiple load conditions, and diverse boundary conditions are considered in the next sections.

Fig 3.1 illustrates the basic input and output of the topology optimization algorithm. The first step is to specify the domain, boundary conditions and loading conditions of the problem. Subsequently parameters such as the volume fraction, filter radius and sensitivity filtering can be adjusted to produce an output as shown. The aim of the optimization problem is to find the optimal material distribution within a given domain that satisfies the minimum compliance problem for a fixed constraint on the amount of material, expressed as a fraction of the total design volume.

¹the 88-line code was derived from the 99-line code first presented in 2001 by Sigmund [19]

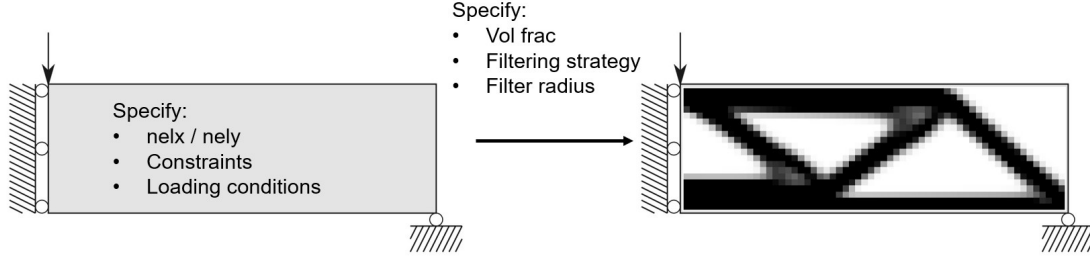


Figure 3.1: Example of the input and output of the topology optimization problem for the 2D MBB

3.1.1. Modified SIMP approach

By adopting the SIMP approach, the user is first required to input the number of rectangular finite element along the x and y directions, in which the design domain should be discretized. Each finite element e is characterized by a density value x_e that affect the corresponding value of the Young's Modulus E_e , as follows:

$$E_e(x_e) = E_{min} + x_e^p(E_0 - E_{min}), \quad x_e \in [0, 1], \quad (3.1)$$

which corresponds to the modified SIMP approach, featuring a number of advantages over its classical counterpart. Herein: E_{min} is the minimum value of stiffness to avoid singularities in the stiffness matrix, E_0 is the max value of stiffness, and the exponent p is a penalization factor to ensure black-and-white solutions.

The mathematical formulation of the TO problem is given in the following set of equations:

$$\begin{aligned} \min_{\mathbf{x}} : \quad & \mathbf{c}(\mathbf{x}) = \mathbf{U}^T \mathbf{K} \mathbf{U} = \sum_{e=1}^N E_e(x_e) \mathbf{u}_e^T \mathbf{k}_0 \mathbf{u}_e, \\ \text{subject to :} \quad & V(\mathbf{x})/V_0 = f, \\ & \mathbf{K} \mathbf{U} = \mathbf{F}, \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{1}. \end{aligned} \quad (3.2)$$

The first line describes the objective function, the so-called structural compliance. It is defined as the work of external loads at equilibrium, providing a measure of the overall stiffness. The minimization problem is subject to a set of constraints on the optimization variables, respectively in terms of prescribed volume fraction, global force-displacement relationship, and admissibility of the element densities. Herein: adopting a regular mesh

of finite elements, \mathbf{x} is the vector of design variables (one per element); c is the compliance to be minimized; \mathbf{U} and \mathbf{F} are the global displacement and force vectors, respectively; \mathbf{K} is the global stiffness matrix, assuming a linear elastic isotropic material; \mathbf{u}_e is the element displacement vector; and \mathbf{k}_0 is the element stiffness matrix for an element of unit Young's Modulus; N refers to the number of elements within the discretized domain; f refers to the prescribed volume fraction; $V(\mathbf{x})$ and V_0 refer to the volume of the placed material and to the target design volume, respectively. The assumption of small displacement holds.

3.1.2. Optimality Criteria Method

To update the element density values over each iteration of the optimization problem, we adopt the following standard optimality criteria method with heuristic updating strategy:

$$x_e^{new} = \begin{cases} \max(0, x_e - m) & \text{if } x_e B_e^\eta \leq \max(0, x_e - m) \\ \min(1, x_e + m) & \text{if } x_e B_e^\eta \geq \min(1, x_e + m) \\ x_e B_e^\eta & \text{otherwise} \end{cases}, \quad (3.3)$$

where m is a suitable threshold on the iteration step, η ($=1/2$) is a numerical damping coefficient, B_e is derived from the optimality condition, as below:

$$\mathbf{B}_e = \frac{-\frac{\partial c}{\partial x_e}}{\lambda \frac{\partial V}{\partial x_e}}, \quad (3.4)$$

with λ being a suitable Lagrangian multiplier value, selected using a bisection algorithm whilst still satisfying the volume constraint.

Eq.(3.4) involves the sensitivities of the objective function c and the material volume V with respect to the element densities x_e , computed under the assumption that each element has unit volume, as follows:

$$\begin{aligned} \frac{\partial c}{\partial x_e} &= -p x_e^{p-1} (E_0 - E_{min}) \mathbf{u}_e^T \mathbf{k}_0 \mathbf{u}_e, \\ \frac{\partial V}{\partial x_e} &= 1. \end{aligned} \quad (3.5)$$

3.1.3. Filtering

Another feature related to the adoption of the SIMP method is the filtering strategy; that is, the application of sensitivity filter which involves a weighted average over different

elements. This is a partially heuristic, yet efficient solution to the arising of undesired checkerboard patterns (unphysical minima) and mesh dependence issues, first suggested by Sigmund in 1994 [24]. Indeed, it ensures the existence of solutions to the optimization problem by placing restrictions on the design. For the generation of the datasets in both 2D and 3D cases, the original sensitivity $\frac{\partial c}{\partial x_e}$ of the objective function obtained in Eq. 3.5 is modified as follows to obtain the filtered sensitivities $\widehat{\frac{\partial c}{\partial x_e}}$:

$$\widehat{\frac{\partial c}{\partial x_e}} = \frac{1}{\max(\gamma, x_e) \sum_{i \in N_e} H_{ei}} \sum_{i \in N_e} H_{ei} x_{ei} \frac{\partial c}{\partial x_e}. \quad (3.6)$$

Herein: $\gamma = 10^{-3}$ is a small positive term introduced to avoid division by zero. This differs from the classical SIMP approach proposed by Sigmund [19], in which the density variables x_e cannot become zero, thus not requiring to define the γ term; H_{ei} is the weight factor for the i -th set of elements, for which the center-to-center distance to the e -th element is smaller than the filter radius r_{min} ; H_{ei} is defined as:

$$H_{ei} = \max(0, r_{min} - \Delta(e, i)). \quad (3.7)$$

For a more detailed analysis of filtering methods, the reader can refer to the 2007 paper by Sigmund [26]. It must be remarked that the adoption of a filtering scheme implies the arising of grey boundaries in the achieved optimal layouts. To get a crisp transition between full material and void, projection schemes may be conveniently implemented, as discussed in Guest et. al. (2004) [95].

3.2. Topology optimization: extension to 3D

As mentioned earlier, the generation of training datasets for 2D problems is carried out by means of the popular 'top88' optimization code [3]. The extension to 3D problems is instead based on the '125-line' code, known as 'top3D125', and released in 2020 by Ferrari and Sigmund [4]. At the time of writing, this has proved the most efficient MATLAB implementation to date, with speed-ups in the computing time of about 1.9 times compared to the code of Amir et al. (2014) [35]. Both of these codes employ significantly more efficient filter assembly and implementation procedures compared to the 'top88' code, as well as shortcuts in the design update step. One of the most useful improvements is the 'fsparse' subroutine developed by Engblom and Lukarski (2016) [96], which improves over the basic sparse assembly used in the original 'top88' code through a better sorting of

operations.

One of the other main differences employed in the 3D formulation, as opposed to the 2D formulation, is the filtering of the densities (in the 2D formulation the filtering is applied only to the sensitivities). The density filter transforms the original densities x_e as follows:

$$\tilde{x}_e = \frac{1}{\sum_{i \in N_e} H_{ei}} \sum_{i \in N_e} H_{ei} x_i, \quad (3.8)$$

where the original densities x_e are known as the design variables, and the filtered densities \tilde{x}_e are referred to as the physical densities. The filtering of the densities necessitates a minor reformulation to the sensitivity equations. The sensitivities of the objection function c and the material volume V with respect to the physical densities \tilde{x}_e are still given by Eq. 3.5, provided the variable x_e is replaced with \tilde{x}_e . The sensitivities with respect to the design variables x_j are obtained through use of the chain rule:

$$\frac{\partial \psi}{\partial x_j} = \sum_{e \in N_j} \frac{\partial \psi}{\partial \tilde{x}_e} \frac{\partial \tilde{x}_e}{\partial x_j} = \sum_{e \in N_j} \frac{1}{\sum_{i \in N_e} H_{ei}} H_{je} \frac{\partial \psi}{\partial \tilde{x}_e}, \quad (3.9)$$

where the function ψ represents either the objective function c or the volume V . The only other minor modifications from the 2D formulation is the definition of the elemental stiffness matrix K_e^s for the 8-node hexahedron, and the addition of the extra 'space-dimension' in the necessary lines to account account for 3D. For full details about the problem formulation for the 'top3D125' refer to the article published by Ferrari and Sigmund 2020 [4].

4 | Matlab Implementation and Dataset formulation

For both the 2D and 3D cases, it is necessary to create a robust and large enough dataset to properly train the machine learning model. We define *cases* here to mean unique sets of boundary conditions, which can be implemented by means of a suitable parametrization of the MATLAB code. In particular, we consider case studies involving an MBB beam as 2D example and a cantilever and a bridge as extension to 3D. To properly sample the parametric input space (which accounts for the loading conditions, and also the boundary conditions in the bridge case), a Latin hypercube sampling (LHS) strategy was employed. LHS is a statistical method for generating a near-random sample of parameter values from a multidimensional distribution and allows a uniform sampling of the full sample space.

4.1. 2D

For the 2D case, the original '88-line' MATLAB code has been modified to allow for the generation of a dataset for our machine learning model.

The diagram in Fig 4.1 illustrates the connectivity of the finite elements for the 2D case. The design domain is rectangular and discretized into square elements, with a width `nelx` in the x-direction and a height `nely` in the y-direction of 4 and 3 respectively. The total number of elements `nel` is 12 with 4 nodes per element. There are two degrees of freedom (DOFs) per node, shown in red. The numbering of both the elements and nodes is performed top to bottom column-wise and left to right, with the DOFs $2n - 1$ and $2n$ corresponding to the horizontal and vertical displacement of the node n respectively. The sign convention for loads and displacements is taken to be positive up and positive right.

4.1.1. MBB

The MBB case was discretized in a domain with size `nelx` = 120 and `nely` = 40. The volume fraction was prescribed as `volfrac` = 0.5, the penalization power `penal` = 3 and

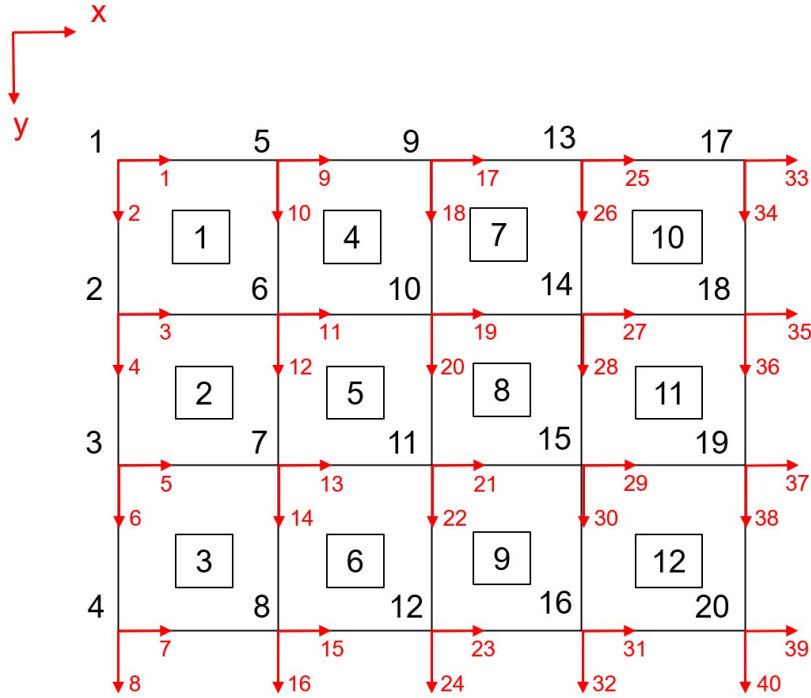


Figure 4.1: Numbering and connectivity for the 2D finite element formulation.

the filter radius $r_{\min} = 1.5$.

The code is split into a main file *'Main_MBB.m'* which iteratively calls the function *'top88_MBB'* implementing the topology optimization procedure for each unique set of loading parameters. Each topology optimization loop produces three outputs; a grey scale optimal topology, a VM Stress map, and a TorC field. These are stored locally in the variables `topopt`, `VM` and `TorC` respectively, and then ultimately exported to a .csv file format once the optimization loop was completed for every load condition. A basic flowchart of the data generation procedure is shown in Fig. 4.2. The two loading parameters that are allowed to vary are the `angle = θ` , which controls the angle of the force vector, measured anti-clockwise from the positive x-axis, and the `nodeID` which controls the node number onto which the force vector is applied. As a constraint, the force vector may be applied only on the nodes on the free surface boundary, shown in Fig. 4.3.

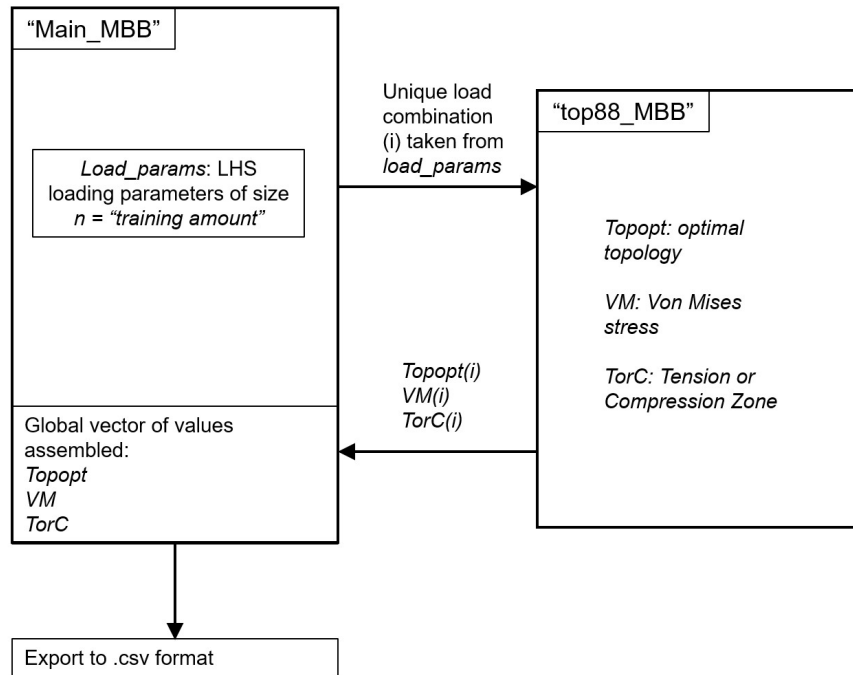


Figure 4.2: Flowchart of main program '*Main_MBB.m*' calling upon function '*top88_MBB*' to produce topologies for each load condition.

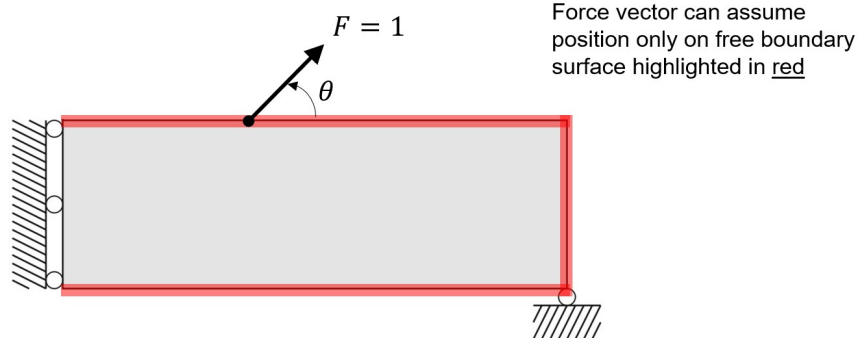


Figure 4.3: Allowable force vector positions in the MBB case.

Loading Parameters

The output of the topology optimization algorithm is not sensitive to the magnitude of the loading vector, and thus a unit force vector $F = 1$ is utilized. The finite element analysis step is performed in the linear elastic regime and it is therefore possible to exploit the superposition of effects; the force vector can be decomposed into its x- and y-direction components F_x and F_y , shown in Fig.4.4. This is necessary since our finite element code

requires the acting loads to be specified in the same direction of the DOFs.

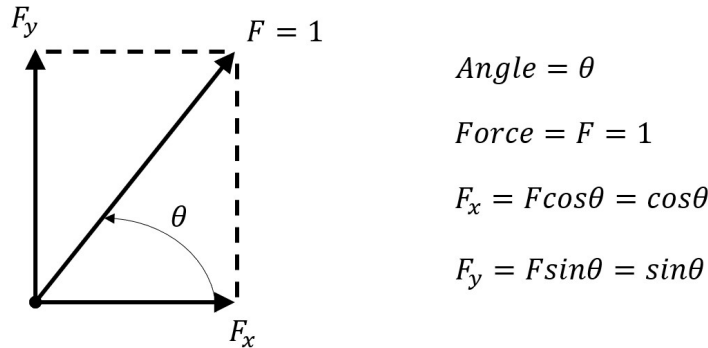


Figure 4.4: Decomposition of unit force vector in x- and y- components for use in the MATLAB program.

Latin Hypercube Sampling (LHS)

LHS is a method that can be used to sample random numbers in which the samples are evenly selected over a sample space. It is a popular method of generating what are known as *controlled random samples* since it can produce a highly diverse and representative data set with a comparatively small amount of samples. For this method to be truly effective, the variables must be independent. To illustrate in 2 dimensions refer to Fig. 4.4. The sample space of each variable is subdivided into n evenly spaced regions. A random sample is chosen by selecting a combination of two subregions across the two dimensions. For the current case of the 2D MBB beam, the two independent variables are the `angle` and `node_ID`. The quantity of n random samples generated is equal to the samples we desire for our dataset. For this case we have created a dataset of $n = 2500$ topologies. A `for` loop introduces each unique pair `i` of `angle` and `node_ID` at a time into our optimization algorithm to produce data containing the corresponding optimized topology, Von Mises (VM) stress and tension or compression (TorC) zones. This data is then stored along with the corresponding loading parameters behind them, in view of the future training of our ML model, as detailed in the next sections.

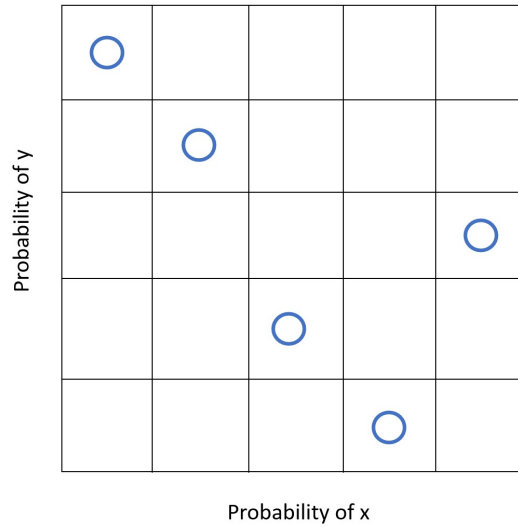


Figure 4.5: Illustration of LHS sampling method in 2 dimensions

Von Mises Stress field and Tension or Compression Field

At each iteration of the optimization algorithm, the code performs a finite element analysis to update the objective function. In this step, information about the local displacements \mathbf{u} is obtained, and then used to calculate the global compliance c . The FEA is the most computationally demanding step, however, once this is performed and the local displacements are obtained, it is computationally efficient to calculate additional useful physical data about the problem, such as the principal stresses or normal and shear stresses.

To calculate the VM stresses, we introduce the [3x8] internal compatibility matrix B for a square element, and the plane stress constitutive matrix D , given by equations 4.1 and 4.2, respectively:

$$B = \begin{bmatrix} -1/2 & 0 & 1/2 & 0 & 1/2 & 0 & -1/2 & 0 \\ 0 & -1/2 & 0 & -1/2 & 0 & 1/2 & 0 & 1/2 \\ -1/2 & -1/2 & -1/2 & 1/2 & 1/2 & 1/2 & 1/2 & -1/2 \end{bmatrix}, \quad (4.1)$$

$$D = \frac{E}{1-v^2} \begin{bmatrix} 1 & v & 0 \\ v & 1 & 0 \\ 0 & 0 & \frac{1-v}{2} \end{bmatrix}. \quad (4.2)$$

The stress vector for the element σ_e is then computed as:

$$\sigma_e = \begin{bmatrix} \sigma_{xx} & \sigma_{yy} & \tau_{xy} \end{bmatrix}^\top = DBu_e. \quad (4.3)$$

The normal and shear stresses are then passed through the VM function to yield the VM stress for each element, computed as follows:

$$\sigma_{VM,e} = \sqrt{(\sigma_{xx}^2 + \sigma_{yy}^2 - \sigma_{xx}\sigma_{yy} + 3\tau_{xy}^2)}. \quad (4.4)$$

regardless of whether the element is part of the topology or void space. In practice, a VM stress field is obtained at every iteration of the FEA for the whole domain. This 'full field' of stresses, further referred to as the 'initial VM stress field' is then saved at the end of the optimization procedure, and stored as part of the training dataset. However, this 'full field' of stress is computed assuming the elastic modulus of the full material everywhere, see Eq. 4.2. Clearly, this field has no physical meaning since any converged optimal solution is characterized by two main region only: solid, with full elastic modulus, and void, with nil elastic modulus. Visualising the VM stresses on the topology is achieved by simply performing an element-wise multiplication between the 'full field' of stress with the obtained optimal topology. The void spaces in the optimal topology, characterized by a zero density of material, do not contribute when multiplied with the VM stresses, and those elements with a density field greater than 0 will retain a VM stress value scaled proportionally to their density value. Such an interpolation allows reconstructing the real stress field, further referred to as the 'final VM stress field', in both void and solid regions. No critical bias is introduced in the intermediate density regions encountered at the boundary of the optimal layout. This discussion holds for 3D, as well.

Fig. 4.6 illustrates this concept. The initial VM stress field (left) is passed through the optimal topology 'mask' to yield a plot of the topology including the VM stresses information.

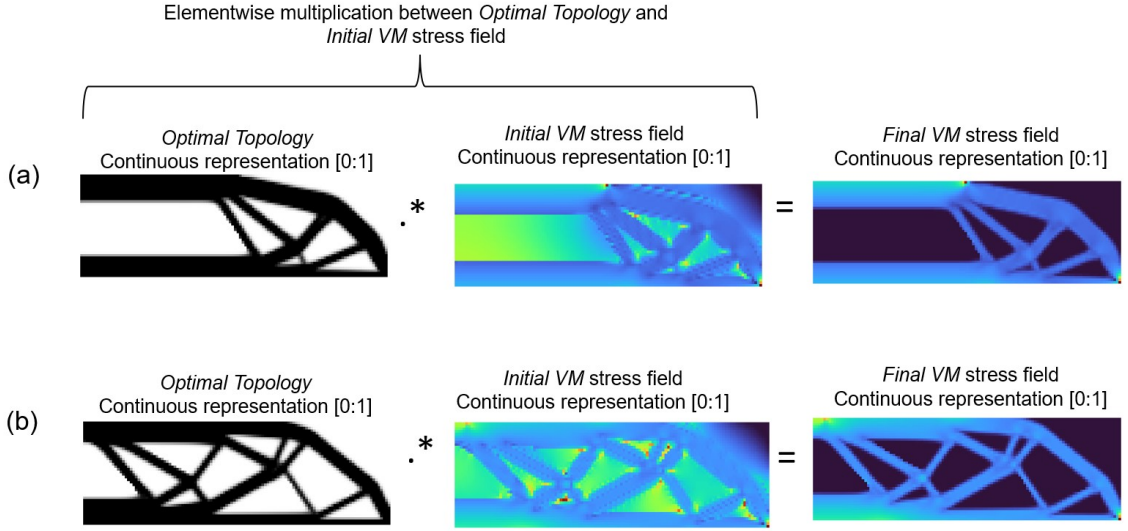


Figure 4.6: Element-wise multiplication between Von Mises stress field with the corresponding optimal topology.

The TorC zones utilise a similar concept of passing a stress field through the optimal topology 'masking' layer. However, in this case, the considered stress field is characterized by values normalized to range from -1 to 1, with negative and positive values representing elements mainly in tension and compression, respectively. This kind of representation is assumed to incorporate important information about the dominant principal stress acting on each element during the optimization procedure.

To determine the dominant principal stresses on each element, the stress vector is first rearranged into a $[2 \times 2]$ matrix as:

$$\sigma_{2 \times 2} = \begin{bmatrix} \sigma_{xx} & \tau_{xy} \\ \tau_{xy} & \sigma_{yy} \end{bmatrix}, \quad (4.5)$$

which is then passed through the *eig* MATLAB function to compute its eigenvalues, one positive and one negative, corresponding to the principal stresses:

$$\sigma_p = \text{eig}(\sigma_{2 \times 2}) = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}. \quad (4.6)$$

For each element, the dominant principal stress σ_{TorC} is finally selected to be the eigenvalue that is greatest in magnitude.

This field, when element-wise multiplied with the optimal topology, 'projects' the tension and compression information onto the optimal topology. If negative values can be rounded

to -1, and positive values rounded to +1, a more distinct visual representation of the TorC regions is shown. This is illustrated in Fig. 4.7 with tension shown in red, compression in blue, and void space in white for two test cases (a) and (b).

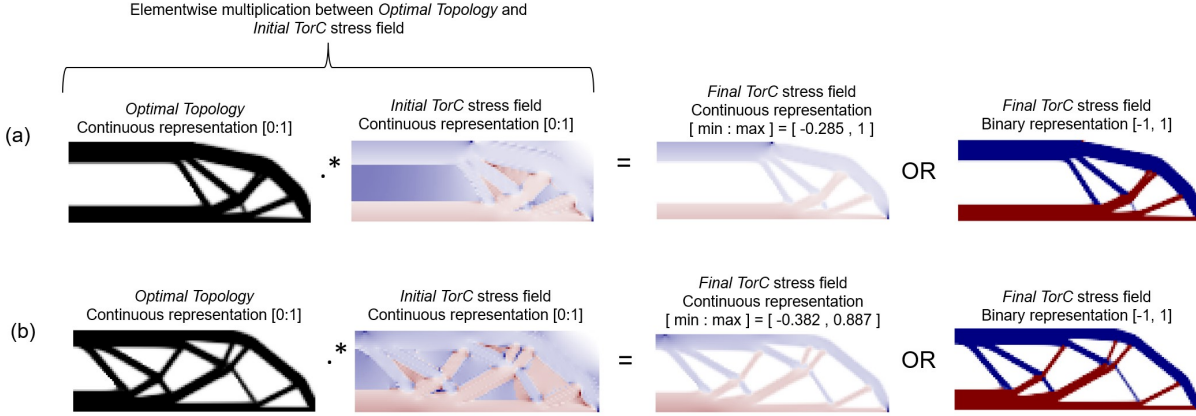


Figure 4.7: Element-wise multiplication between σ_{TorC} stress field and the corresponding optimal topology.

4.2. Extension to 3D

For the 3D case, the 'top3D125' [4] MATLAB code has been modified to allow for the generation of a dataset for our machine learning model. The two considered 3D case studies involves a cantilever beam and a bridge.

The design domain is a prism discretized into cubic elements, with a total length n_{elx} in the x-direction, height n_{ely} in the y-direction, and width n_{elz} in the z-direction. There are three degrees of freedom (DOFs) already defined per node. The numbering of both the element number and node numbers is performed first top to bottom column-wise (down the y-axis) and then left to right along the x-axis), while moving along the z-axis. Fig. 4.8(a) shows the node (black) and DOF numbers (red) associated with the starting plane $z = 0$, whilst Fig.4.8(b) shows the numbering for elements between $z = 0$ and $z = 1$. For node n , the DOFs $3n - 2$, $3n - 1$ and $3n$ correspond to the x-, y- and z-displacements, respectively. The sign convention for the loads and displacements matches the positive sign conventions of the axes.

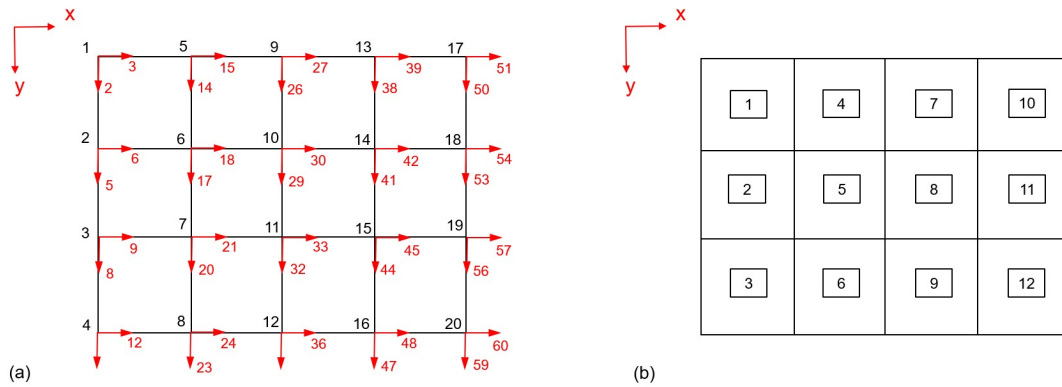


Figure 4.8: (a) Nodes and DOFs numbering for $z=0$ (b) elements numbering for the elements between $z=0$ and $z=1$.

The three dimensional isometric view of the domain is shown in Fig.4.9 (a) with the first element (as per the numbering order) highlighted in blue. A detailed view of this first element is reported in Fig.4.9 (b) showing the node numbers (black) and associated DOFs (red).

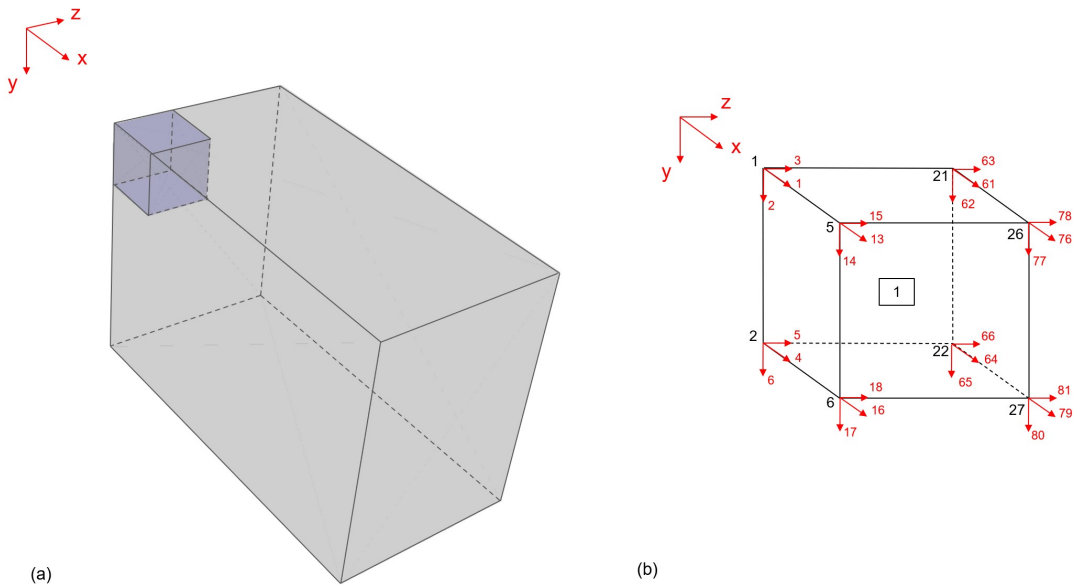


Figure 4.9: (a) Isometric view of the design domain with element 1 highlighted in blue (b) detailed view of element 1 showing node numbers and DOFs.

4.2.1. 3D Cantilever

The 3D Cantilever case was discretized in a domain with size $\text{nelx} = 24$ and $\text{nely} = 12$ and $\text{nelz} = 12$. The volume fraction was prescribed as $\text{volfrac} = 0.12$, the penalization power $\text{penal} = 3$ and the filter radius $\text{r_min} = \sqrt{3}$.

Like the 2D case, the code was split into a main file *'Main_3DCantilever.m'* which called upon, internally, the function *'top3D125_3DCantilever'* which ran the topology optimization procedure for each unique set of loading parameters. The same procedure is followed for the 3D Bridge case. Each topology optimization loop produces the same three outputs as the 2D case; a grey scale density field for the optimal topology, an initial VM stress field, and an initial TorC stress field which was stored for every load combination. The 3D cases follow the same basic flowchart of data generation as shown in Fig. 4.2. A total of 2500 topologies were generated for the 3D cantilever training set.

Loading Parameters

For the 3D case, additional parameters are necessary to define the unique position and direction of the force vector in a 3D space. Like the 2D case, the topology is invariant to the magnitude of the vector and thus a unit force vector is taken for the calculations. The position of the force vector origin is defined by three parameters; the x-, y- and z-coordinate in space. Two additional parameters define the the direction of the unit vector; the inclination and azimuth angle (see Fig. 4.10). These are independent angles that can trace a unique point on the surface of a unit sphere, and are given by θ and φ for the inclination angle and azimuth angle, respectively. Typically, a third parameter r , which controls the radial distance is required, however for a unit vector this is kept equal to one.

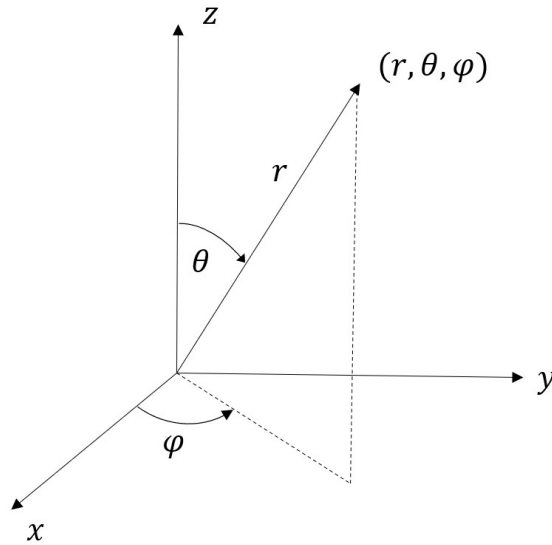


Figure 4.10: Spherical coordinate system: inclination θ , azimuth φ , and radius r .

Together, these 5 independent parameters (x, y, z, θ, φ) are necessary to define the position and direction of the force vector. Within each optimization run, the θ, φ values are decomposed into the F_x, F_y, F_z components, as follows:

$$\begin{aligned}
 F_x &= F \cdot \sin\theta \cos\varphi = \sin\theta \cos\varphi, \\
 F_y &= F \cdot \sin\theta \sin\varphi = \sin\theta \sin\varphi, \\
 F_z &= F \cdot \cos\theta = \cos\theta,
 \end{aligned}
 \tag{4.7}$$

which are then exploited in the finite element procedure of the MATLAB code.

Von Mises Stress field and Tension or Compression Field

Like the 2D cases, the 'top3D125' code can also generate the VM stresses and TorC zones, by simply reformulating the problem into three dimensions.

To calculate the VM stresses, we introduce the [6x24] internal compatibility matrix B for

a cubic element, and the constitutive matrix D given by:

$$D = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}. \quad (4.8)$$

The product of these two and the vector of local displacements u_e yields the stress vector σ_e for the e -th element:

$$\sigma_e = \begin{bmatrix} \sigma_{xx} & \sigma_{yy} & \sigma_{zz} & \tau_{xy} & \tau_{xz} & \tau_{yz} \end{bmatrix} = DBu_e. \quad (4.9)$$

A procedure analogous to that adopted for the 2D case is used to map the VM stress field onto the optimal topology also in the 3D case. First, a field of VM stresses is generated for the full domain, regardless of the computed optimal topology. This is computed at the element level as:

$$\sigma_{VM,e} = \sqrt{\frac{1}{2}((\sigma_{xx} - \sigma_{yy})^2 + (\sigma_{xx} - \sigma_{zz})^2 + (\sigma_{yy} - \sigma_{zz})^2 + 6 \cdot (\tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2))}. \quad (4.10)$$

The resulting stress field is then normalized to take values between 0 and 1 for each unique loading combination, whereby 0 represents a state of no stress, and the 1 represents the greatest magnitude of VM stress. The optimal topology is finally adopted as a 'masking' layer through which the VM stress field is passed. This is achieved by an element wise multiplication; void spaces remain voids in the output, and regions with non-zero densities assume a scaled value of the VM stress. This is illustrated in Fig.4.11 for the 3D cantilever.

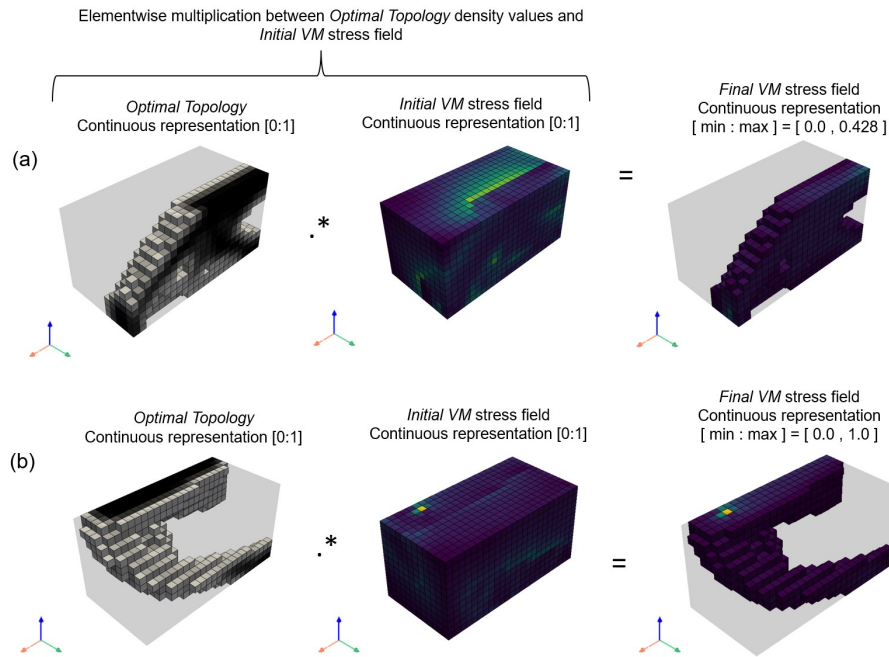


Figure 4.11: Projecting the Von Mises stresses onto the optimal topology using the masking procedure for two different cases (a) and (b).

As shown in Fig.4.11 (a), it can happen that the element containing the greatest magnitude of VM stress (with a value of 1) in the initial VM stress field is not necessarily projected onto the optimal topology, and is in fact element-wise multiplied with a void space, rendering it null. As such, the element-wise multiplication yields a final VM stress field with values that do not necessarily span between 0 and 1. In Fig. 4.11 (b) however, the element with a VM stress of 1 is also an element with a physical density of 1 for the optimal topology, for which the element-wise multiplication yields a topology containing a VM stress of 1.

The TorC regions for the 3D case follow the same procedure and format as with the 2D case; the values for each element are defined based on the dominant principal stress. First, the [6x1] stress vector σ_e is rearranged as a [3x3] stress matrix:

$$\sigma_{3x3} = \begin{bmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{bmatrix}, \quad (4.11)$$

which is then passed through the *eig* function in MATLAB to yield the following [3x3]

matrix of principal stresses:

$$\sigma_p = eig(\sigma_{3x3}) = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}. \quad (4.12)$$

The element dominant principal stress is finally selected as the principal stresses of greatest magnitude.

A TorC field σ_{TorC} is thus generated for the entire domain, which is eventually normalized to lie between -1 and 1; again, negative values represent a dominant tensile principal stress and positive values represent compressive principal stresses. This field is passed through the optimal topology 'mask' by means of an element wise multiplication, as shown in Fig.4.12. Similarly to the VM stresses, the element containing the maximum compressive or tensile principal stress value does not necessarily coincide with an element containing a physical density in the optimal topology regime; thus the element-wise product does not necessarily yield a map of values containing either a -1 or a 1. Nevertheless, this representation still encodes the information about TorC zones.

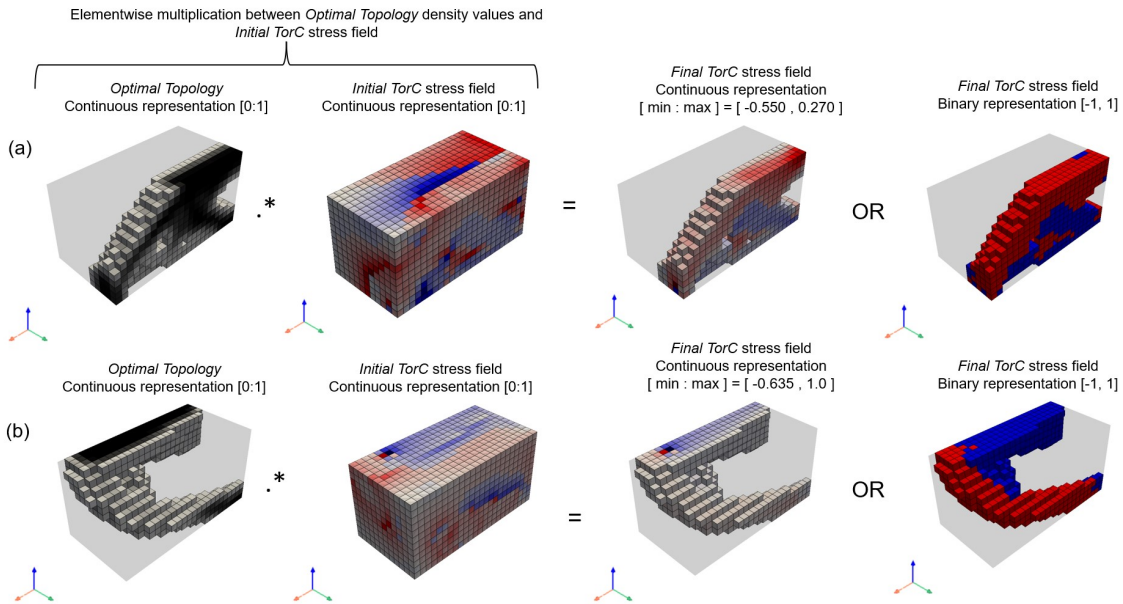


Figure 4.12: Projecting the σ_{TorC} stresses onto the optimal topology using masking procedure for two different cases (a) and (b).

4.2.2. 3D Bridge

The 3D Bridge case was discretized in a domain with size $\text{nelx} = 60$ and $\text{nely} = 20$ and $\text{nelz} = 4$. The volume fraction was prescribed as $\text{volfrac} = 0.12$, the penalization power $\text{penal} = 3$ and the filter radius $r_{\text{min}} = \sqrt{3}$. The deck is constrained as full material (density = 1) and has a unit thickness. The void region located below the deck has a width of 30 units. The deck region, the void region, and the region in which the topology is free to develop are highlighted in Fig. 4.13 in red, orange and grey, respectively.

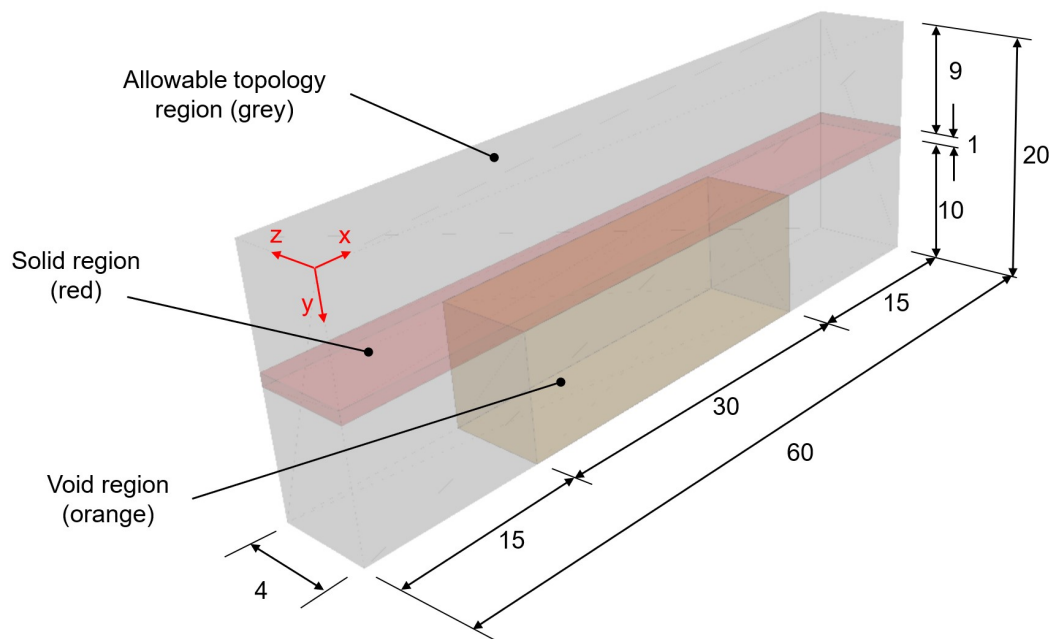


Figure 4.13: 3D (half-)bridge domain: dimensions and solid, void, and allowable regions.

This domain represents one half of the bridge, with the $x - y$ symmetry plane placed at $z = 0$; the full bridge configuration can be easily recovered as shown in Fig. 4.14. In this case, exploiting symmetry yields a considerable computational advantage when generating the training dataset. A total of 2500 topologies were generated for the 3D bridge training set.

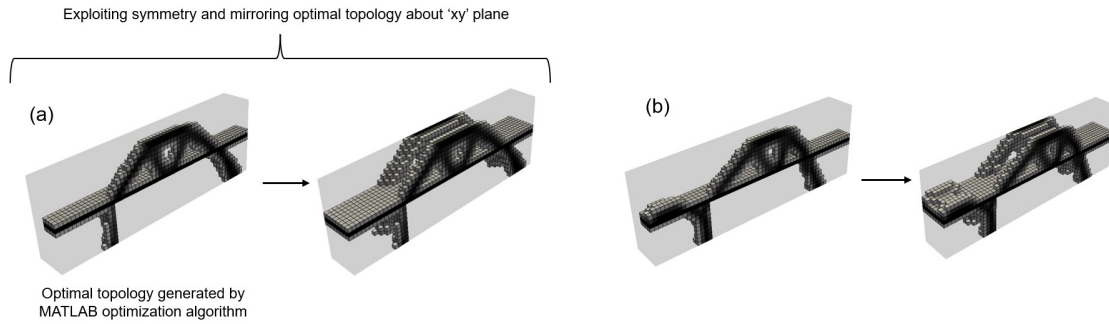


Figure 4.14: Mirroring the half-bridge domain to create the full bridge for different test cases (a) and (b).

Loading Parameters

For the 3D bridge case, the parametric input space accounts for 4 parameters adopted to produce unique topologies: two control the x-axis position of the acting loads and the other two control the x-axis position of the supports (see Fig. 4.14).

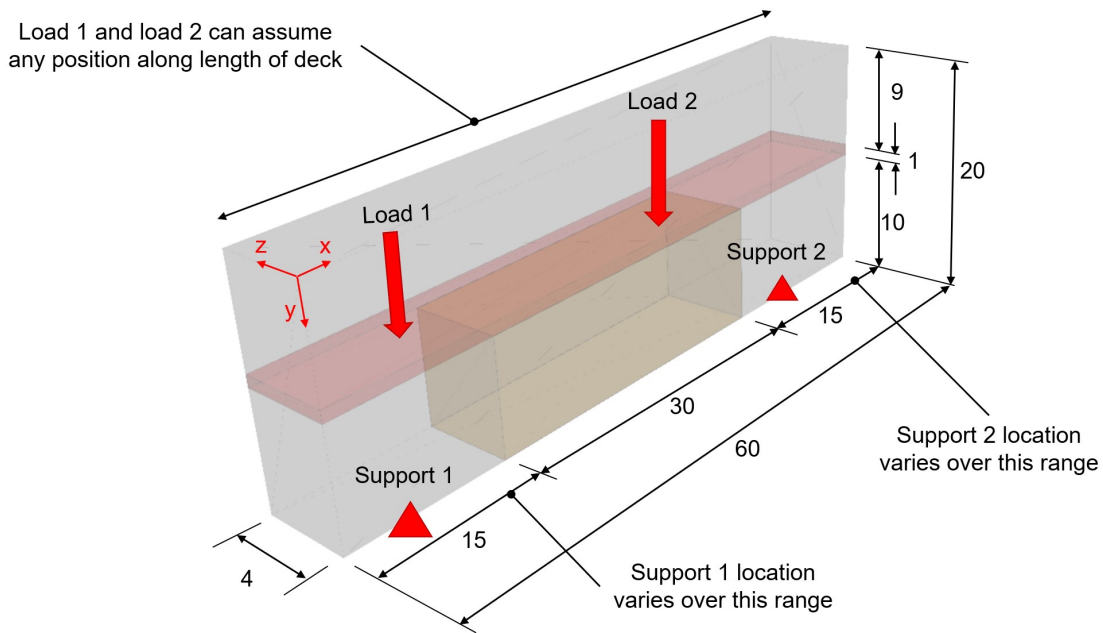


Figure 4.15: 3D Bridge case: schematic representation of the 4 parameters adopted to create unique topologies.

The acting loads are pointing vertically downward and have a unit magnitude applied on the deck of the bridge. These can be applied on any node along the length (x-axis) of bridge, and their positions are not in any way dependent on each other. The two supports

can instead be placed anywhere in the two regions to the left and right of the void region, and are also not dependent on each other.

VM Stress field and TorC Field

The 3D bridge stress formulation follows an identical procedure to that of the 3D cantilever. Fig. 4.16 illustrates the 'masking' procedure used with the optimal topology and the initial VM stress field to produced the final VM stress field for two different cases.

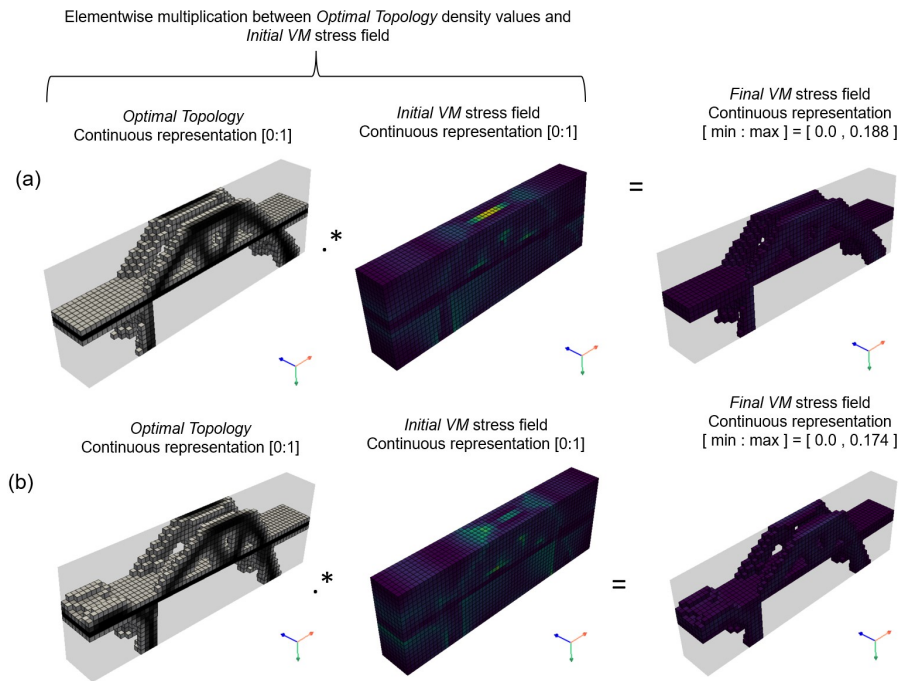


Figure 4.16: Projecting the VM stresses onto the optimal topology using the masking procedure for two different cases (a) and (b).

The initial TorC field assumes normalized values ranging between -1 to 1, where values between -1 and 0 represent a dominant tensile stress and values between 0 and 1 represent a dominant compressive stress for each element. This is illustrated in Fig. 4.17.

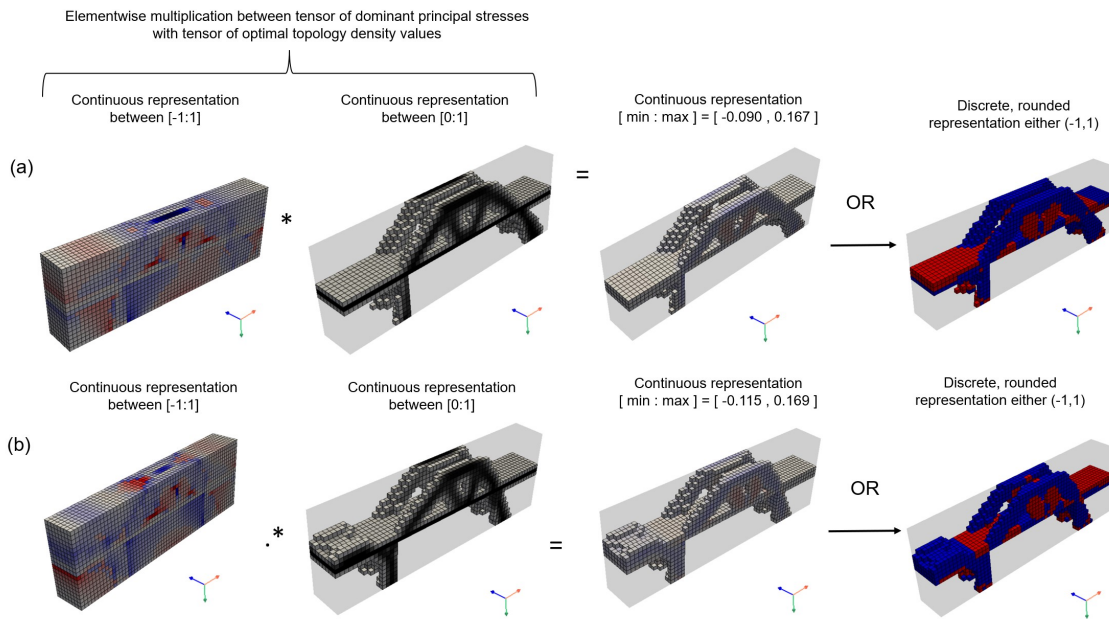
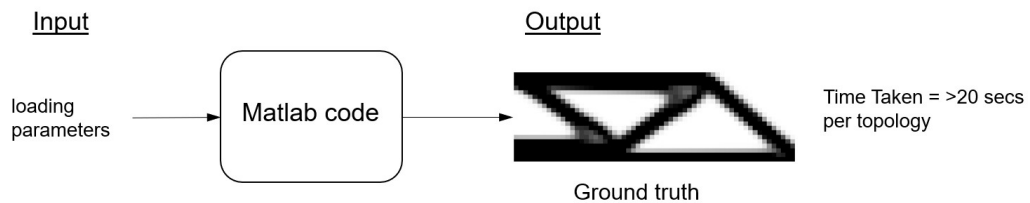


Figure 4.17: Projecting the $\sigma_{T_{orC}}$ stresses onto the optimal topology using the masking procedure for two different cases (a) and (b).

5 | Machine learning formulation

The traditional method of topology optimization, as outlined in chapter 3, takes unique loading parameters as inputs and outputs the topology that best minimizes compliance and satisfies the given volume and boundary constraints. The time taken to generate a topology using this algorithm is dependent on the size of the domain, and the wait times for larger domains can become impractically long. The proposed DL model, on the other hand, aims to take the loading parameters as inputs, and output a topology of acceptable accuracy in a fraction of the time.

(a) Traditional Matlab Optimization Algorithm



(a) Proposed deep learning predictive model

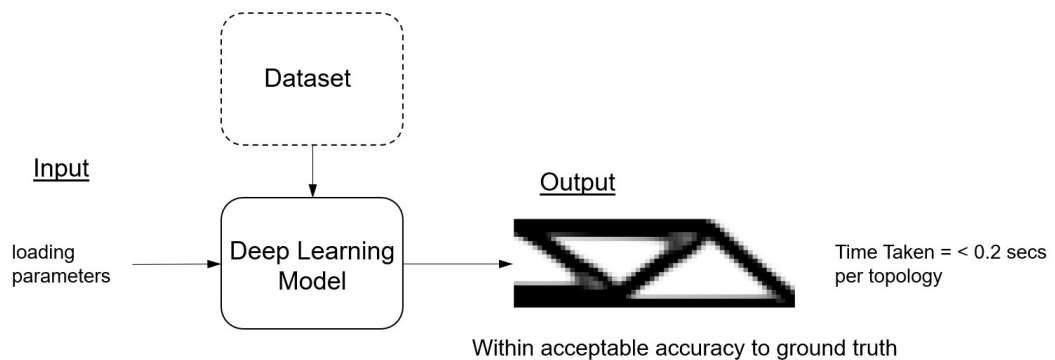


Figure 5.1: Topology optimization approaches: (a) traditional methodology; (b) proposed deep learning model pipeline.

Fig. 5.1 shows a comparison between the two methodologies. In Fig. 5.1 (b), the *dataset* refers to the large set of topologies and loading parameters computed, as outlined in chapter 4 for the specific case study (2D MBB, 3D cantilever, 3D bridge). These are

saved and exploited during the training phase of the DL model in order to learn a possible underlying mapping between the two. Although such a training phase will inevitably require a significant amount of time, once trained, the DL model will be able to predict a topology in a very short amount of time with a satisfactory level of accuracy. This chapter details the adopted DL model, in terms of its architecture and the theoretical notions behind of its conceptualization.

5.1. Proposed DL architecture

The proposed DL architecture can be classified as part of the subgroup of ML + TO research termed *non-iterative* methods. Non-iterative optimization methods, as outlined in Chapter 2, predict an optimal shape that satisfies the design conditions, *without* an iterative form finding process. These directly predict the topology from the input loading parameters almost in real-time. The proposed pipeline makes use of two important types of deep learning architectures; a CNN based AE, and a MLP model.

The proposed DL framework is schematized in Fig. 5.2. The first two steps of the process consist of training the two DL models, whilst the final testing step combines the trained models to predict the optimal topology for a given set of loading parameters. In particular, the three steps involve: first, an AE is trained to map optimal topologies into itself; second, a MLP model is trained to map the input loading parameters onto the latent space representation of the corresponding optimal topology, as provided by the encoding branch of the trained AE; finally, the MLP model trained in step 2 and the decoding branch of the AE trained in step 1 are synergistically exploited to create a DL pipeline capable of predicting, almost in real-time, the correct topology for given a set of loading parameters. These steps will be explained in more detail in the following sections.

It should be noted that the proposed deep learning pipeline has been coded in Python, a popular scientific programming language. As such, training datasets and loading parameters have been converted into NumPy arrays. NumPy is a library built specifically for use in Python, which allows for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays. The DL models are instead implemented using TensorFlow, a free and open-source software library for ML and AI. All the computations are performed using Google Colab, a cloud-based platform that provides access to powerful CPUs, GPUs and pre-installed software libraries. The virtual machine provided by Google Colab used 2 CPUs with x86-64 processors and 11 GB of RAM. Additionally, TensorFlow version 2.12.0 was run using Python version 3.9.16.

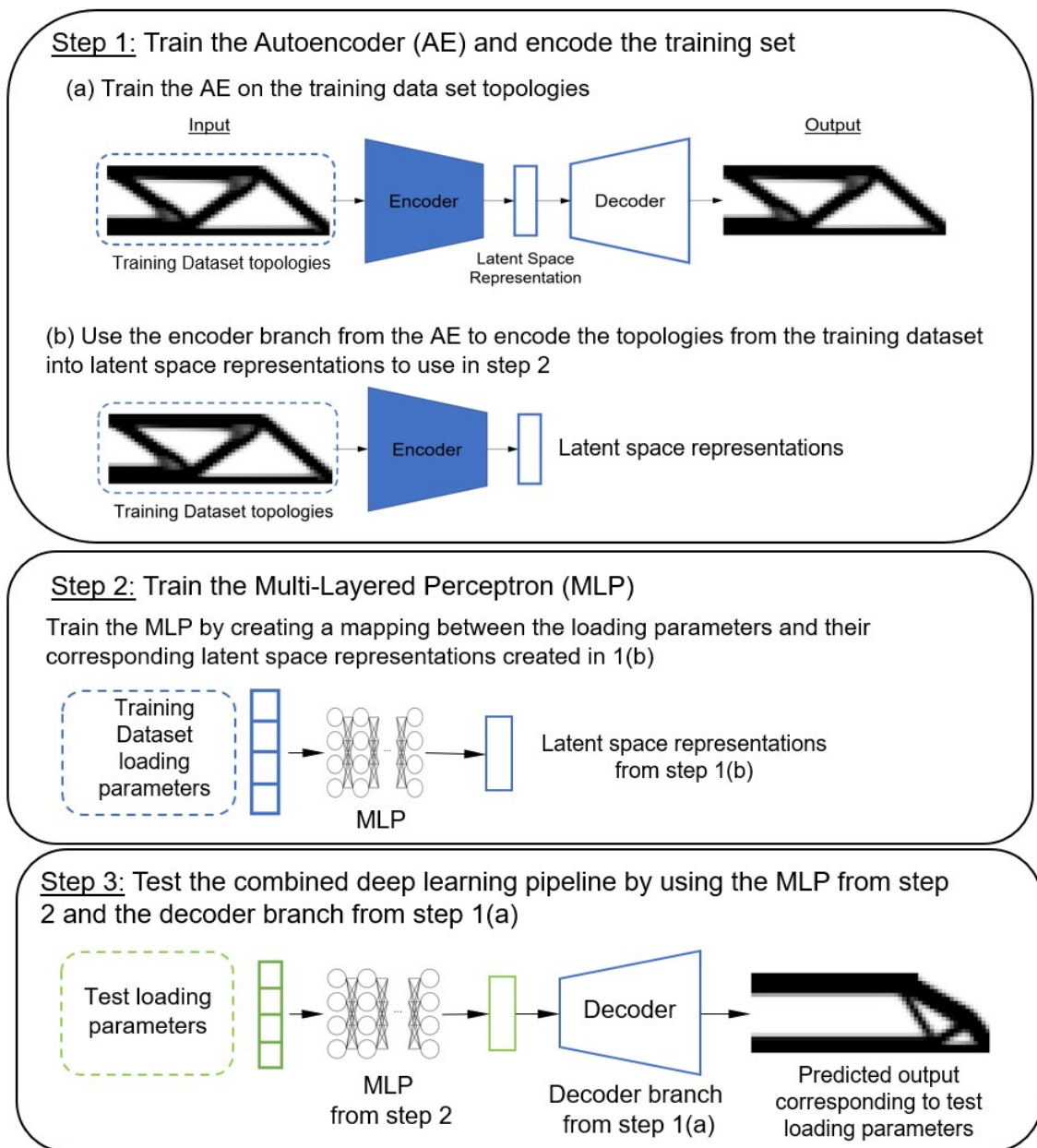


Figure 5.2: Scheme of the proposed deep learning pipeline.

5.1.1. Step 1: Training the AE

AEs are an unsupervised learning technique in which neural networks are leveraged for the task of representation learning. An input (typically an image), is passed through the AE with the aim of reconstructing this input as the output. Such an identity mapping is typically learned by imposing a bottleneck in the middle of AE architecture, such that the original input is compressed into a lower dimensional space. This forces the AE to learn efficient representations of the input data. This technique belongs to the non-linear

dimensionality reduction strategies and is one of the most useful functions of AEs. In other words, AEs learn to encode complex inputs into a latent space representation of a much lower dimension, whilst minimising the loss of information content.

Step 1 of the DL pipeline consists of training the AE on the training dataset of topologies generated as detailed in Chapter 4. Although the formulation varies slightly between 2D and 3D cases, the basic concept is the same. The training dataset consists of topologies that have been previously converted to NumPy tensors, with values for the pixel densities normalised between 0 and 1. These topologies are passed through the encoding branch ('encoder') of the AE and compressed into a lower dimensional latent space representation. Once encoded, these latent space representations are then passed through the decoding branch ('decoder') with the aim of reconstructing the original input topology. A basic diagram illustrating this architecture is shown in Fig. 5.3.

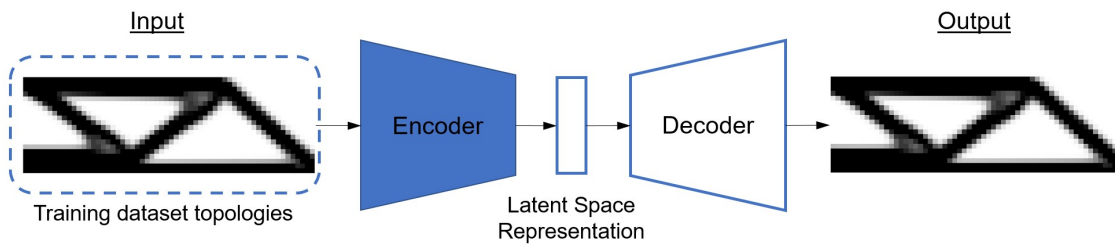


Figure 5.3: Basic AE architecture consisting of encoding and decoding branches

The AE model is trained by minimizing the reconstruction error, which measures the differences between our original input and the predicted output reconstruction. The bottleneck that compresses the input data thus forms a critical element of the network design; without the presence of an information bottleneck, the network could easily learn to simply memorize a 1-for-1 mapping of the input values to the output values. Instead the AE must learn efficient representations of the *features* of the input. All images can be thought of as being made up of smaller local patterns, such as edges, textures and so on. In a dataset that is composed of many images - in our case many images of topologies - there are many local features that are common between them, such as the edge of a truss diagonal, void space, or fully dense space. These shared attributes between the input features mean there is structure in our data, and where there is structure, there is the potential to reduce the dimensionality and encode it into a lower dimensional latent space representation.

The main reason AEs are so effective at detecting structured patterns and features from an image or a dataset is due to the use of *convolutional layers*. Convolutional layers apply small filters over the input image to detect and compute the presence of certain features,

and eventually output a feature map, which shows the activation of such filters for a given input (see Fig. 5.4). In the case of convolutional layers, the parameters that are optimized during the training process turn out to be such convolutional filters.

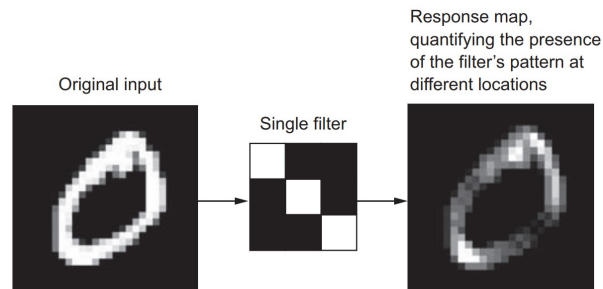


Figure 5.4: An example of a response map produced by applying a 3x3 filter over an input image (adapted from [97]).

Two key parameters define the convolutional operation:

- Size of the filters (or kernels) calculated and applied over the inputs - These are typically 3×3 or 5×5 pixels in size for the 2D case;
- Depth of the output feature map (or number of channels) - These are referred to as the number of filters exploited during the convolution. These can typically range from 32 to over 256.

The second aspect that makes convolutional AEs so effective is the availability of the MaxPooling and convolutional transpose layers. The MaxPooling operation works to downsample the convolutional feature maps, reducing its size after every MaxPooling layer. This is achieved by sliding a window over the feature maps with a particular stride length, and taking the max value for each window. This produces an output that is reduced in size by varying the size of the window or the stride length. In the case of a 2×2 window with a stride length of 2, the output is downsampled by a factor of 2. The MaxPooling operation has two main benefits:

- Successive convolutional layers look at increasingly larger windows (in terms of the fraction of the original input they cover) by downsampling the feature map. This inherently creates spatial-filter hierarchies, where larger and larger features are detected and processed as the sum of smaller features, see Fig. 5.5;
- Reduces the number of parameters to be tuned, by halving the size of the feature map.

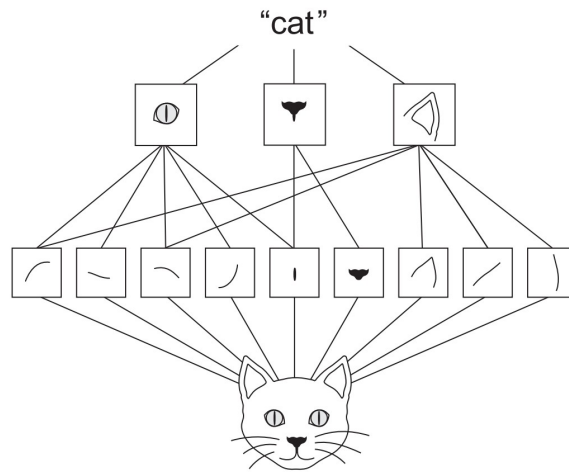


Figure 5.5: The spatial hierarchy of features that forms the image of a cat. Smaller textures and patterns combine to form larger recognisable details such as an ear, or a nose, which combine into the higher level concept of "cat".[97]

The Convolutional transpose layer (or *Conv2DTranspose*) is a layer that, generally speaking, combines a modified convolutional operation and upsampling layers into one. In particular, instead of sliding the kernel over the input and performing element-wise multiplication and summation, a transposed convolutional layer slides the input over the kernel and performs element-wise multiplication and summation. This results in an output that is larger than the input, and the size of the output can be controlled by the stride and padding parameters of the layer. This is used in the 'decoder' branch of the AE to increase the dimensionality of the latent space representation into the reconstructed image.

2D AE

A detailed diagram of the adopted AE architecture is shown in Fig. 5.6 below. The 'encoder' architecture consists of convolutional layers followed by MaxPooling and batch normalization layers. At the end of the encoding branch the $15 \times 5 \times 32$ convolutional response map is reshaped into a 2400×1 vector. This vector is subsequently passed through a dense layer which yields a size 40 vector, which is the latent space representation of the original input. The 'decoder' branch reverses this process in order to reconstruct the input. The 40-dimensional latent space vector is passed through a dense layer to become a 2400×1 vector and then subsequently reshaped to yield a $15 \times 5 \times 32$ layer. These are then passed through a series of Convolutional Transpose, batch normalization and convolutional layers.

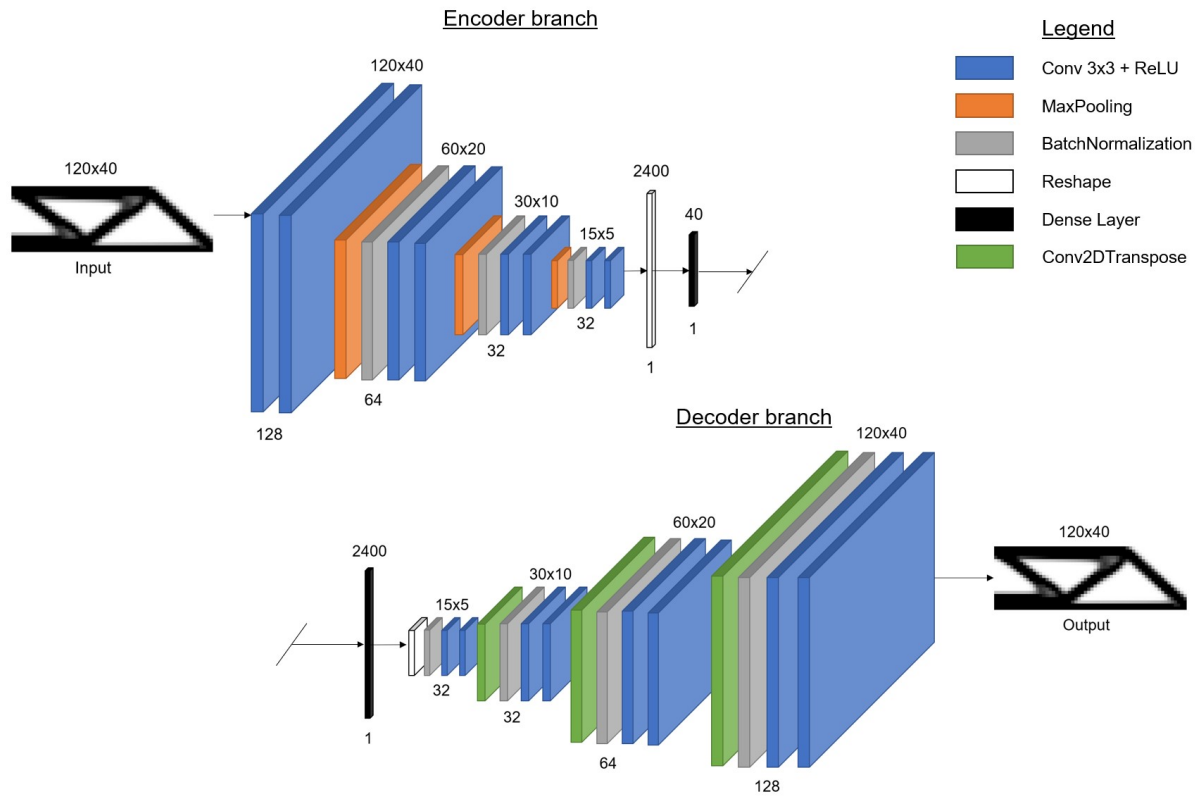


Figure 5.6: Adopted architecture of the AE for the 2D MBB.

In this architecture the kernel sizes are chosen as 3x3, and each convolutional layer features a ReLU activation function. The two middle layers and the last layer feature a sigmoid activation function, which maps its input to an output between 0 and 1. This is particularly relevant to our problem since the reconstructed output topology contains density values between 0 and 1. Additionally the sigmoid function is smooth and continuously differentiable, which makes it suited to the gradient-based optimization algorithms used in deep learning. The two activation functions are shown in Fig. 5.7 below.

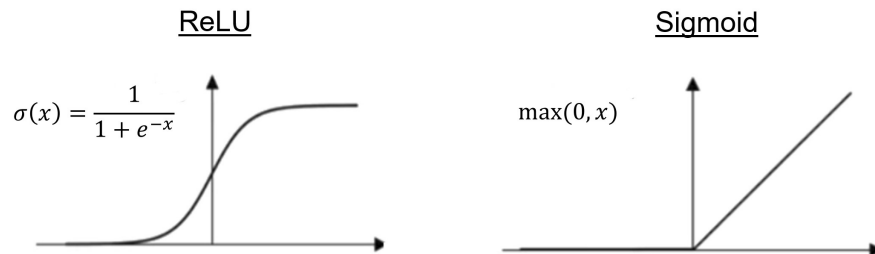


Figure 5.7: Graphs of the ReLU and sigmoid activation functions.

It should be noted that if the AE is equipped with linear activation functions only, without

exploiting non-linear activation functions as done here, the AE would collapse to the same dimensionality reduction function provided by Principal Component Analysis (PCA).

The next key element of our network architecture is the loss function. A loss function (or objective function) is a function that measures the difference between the predicted and actual values of a model. The calculated quantity is a measure of the 'success' of the model, and the goal is to minimize the loss function by adjusting the model parameters. In other words, the model is trained to find the parameter values that produce the lowest possible loss. It is an optimization metric, and is a measure of how well the model fits the training data. For the optimal topologies and the VM stresses the loss function selected is the binary cross entropy loss function, given in Eq. 5.1:

$$H_p(q) = - \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)). \quad (5.1)$$

Herein: N is the number of data points (or in the case of a topology, the number of elements), y_i is the label for element i (1 for fully dense element and 0 for void element), and $p(y_i)$ is the predicted probability of element i being 1. In an AE, the aim is to reconstruct the input data as accurately as possible. Since the target density values can take values between 0 and 1, a threshold is set, such that values above the threshold are set to 1 and values below are set to 0. These rounded values are used for the y_i labels when performing the computation of the binary cross entropy loss function.

The way in which the gradient of the loss function will be used to update parameters is specified by the 'optimizer'. Simply put, the loss function produces a loss score, and the optimizer determines the direction in which to update the weights to further minimize the loss score. The optimizer chosen for the optimal topologies and VM stresses is 'ADAM', which stands for Adaptive Moment Estimation. ADAM is a popular stochastic gradient descent optimization algorithm which uses momentum of the gradients and an adaptive learning rate to speed up convergence.

The last important element of the AE architecture is the accuracy metric. The accuracy metric evaluates the ability of the model to make correct predictions and is a good measure of the overall performance of the model. For the optimal topologies and the VM stresses, the accuracy metric chosen to monitor the training process is the mean absolute error (MAE) (see Eq. 6.1). The MAE is a common metric in regression analysis which measures the average magnitude of errors between predicted and actual values. The MAE is calculated by taking the absolute value of the difference between the predicted \hat{y} and actual y values and then taking the average of these absolute differences.

$$MAE = \frac{1}{n} \sum |y - \hat{y}|. \quad (5.2)$$

For the TorC zone dataset which consists of images with continuous values between -1 and 1 (where negative values represent tensile dominant principal stresses and positive values represent compressive dominant principal stresses), the adopted architecture presents a few modifications. In particular, the middle dense layers and last layer feature a *linear* activation function (shown in Fig. 5.8) in place of a sigmoid one. This is because the model must be capable of predicting values in the range between -1 and 1, for which the sigmoid activation is not suited for.

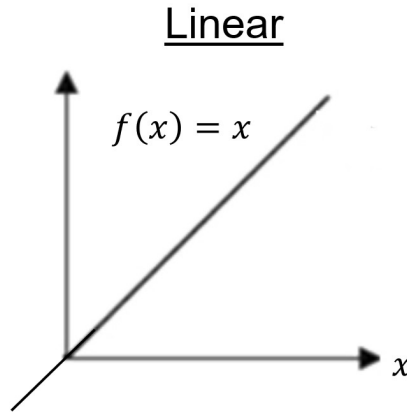


Figure 5.8: Graph of the linear activation function.

The next modification is using a mean squared error (MSE) (see Eq. 5.3) as the loss function to be minimized for training instead of binary cross entropy. The binary cross entropy loss function is suited to a binary classification task between 0 and 1, which makes it ill-suited for the TorC zones which consists of continuous values between -1 and 1. The MSE loss function, however is well suited to this task as it always gives a positive quantity in evaluating difference between the predicted and actual values of a model, regardless of whether the predicted or actual values are positive or negative.

$$MSE = \frac{1}{n} \sum (y - \hat{y})^2. \quad (5.3)$$

A summary of the 2D MBB AE architecture is given in the appendix.

3D AE

A detailed diagram of the AE for the 3D cantilever case is shown in Fig. 5.9 below. The architecture largely resembles that of the 2D MBB case, except the Convolution, MaxPooling and convolutional transpose layers are applied in 3D rather than 2D. The dimensions of the tensors at each layer must be in whole numbers, and thus the downsampling and upsampling operations in the encoder and decoder branches introduce some inconsistencies in the output shapes due to rounding. Due to this, a cropping layer is introduced to trim the borders of the tensor and yield a tensor with the exact same dimensions as the input tensor.

Similarly to the 2D case, a different AE model is employed for the TorC zone dataset, compared to the optimal topologies and VM stresses datasets. This is because the TorC zone dataset contains continuous values between -1 and 1, whereas the optimal topologies and VM stress datasets contain values between 0 and 1. The AE model for the optimal topologies and VM stress datasets utilizes a sigmoid activation function for the middle dense layers and the output layer, and the loss function employed is binary cross entropy. The AE model for the TorC dataset utilizes a linear activation function for the middle dense layers and the output layer, and the loss function employed is MSE. The optimizer chosen for all cases was ADAM.

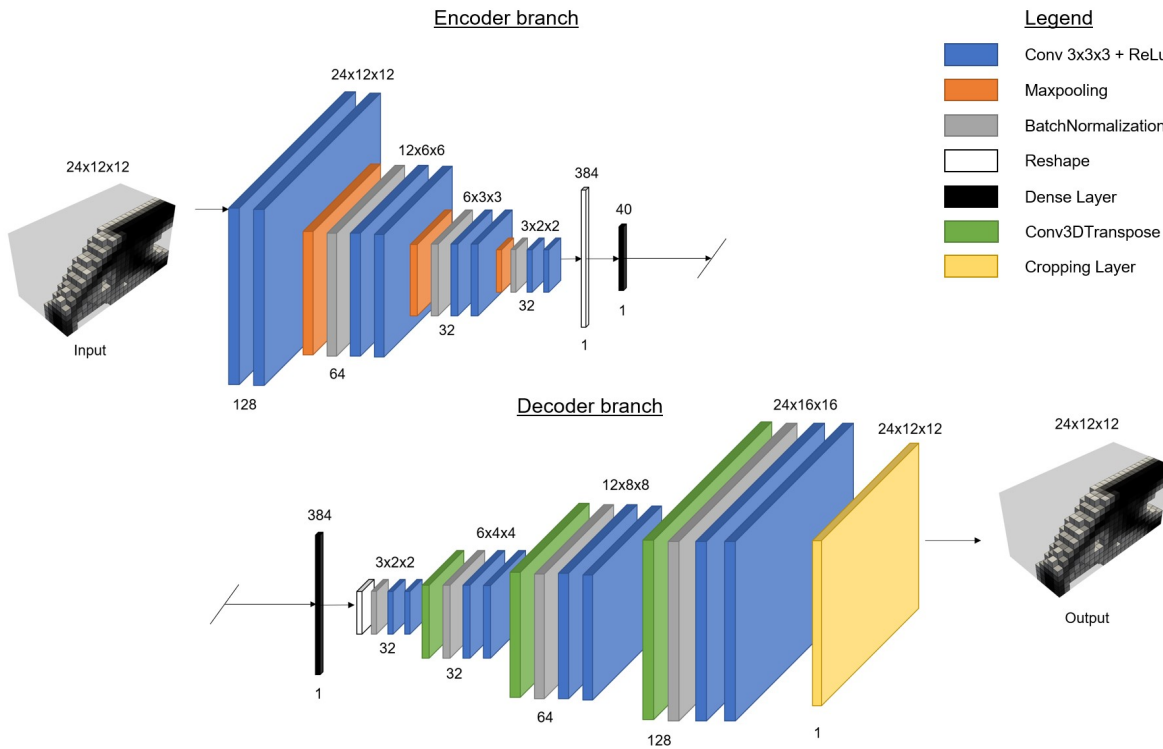


Figure 5.9: AE architecture for the 3D cantilever case.

Likewise the, architecture for the 3D Bridge case is shown in Fig: 5.10. The only differences in the AEs between the 3D cantilever and the 3D bridge is the size of the layers.

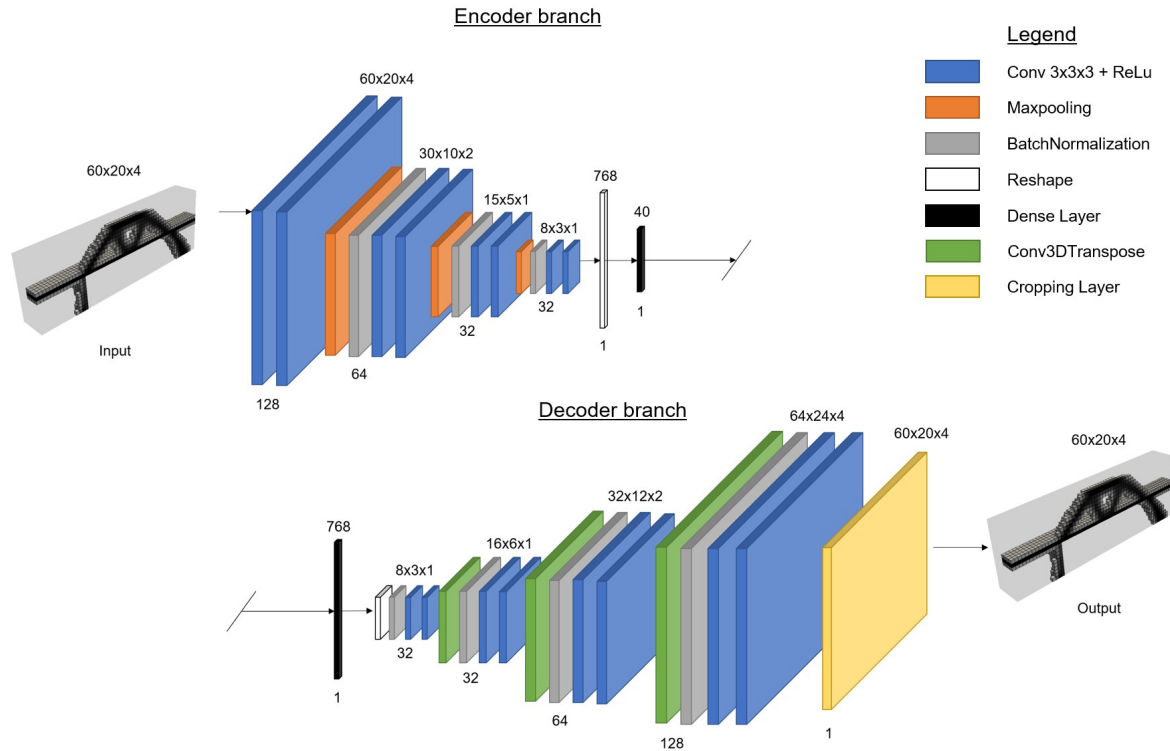


Figure 5.10: AE architecture for the 3D bridge case.

A detailed table of the AE architecture for the 3D cantilever and bridge is given in the appendix.

The final part of this first training step is to encode the training dataset into latent space representations that can be used in the following step. This uses the encoding branch of the autoencoder trained previously. This is step 1 (b) of Fig. 5.2, shown for clarity in Fig. 5.11 below:

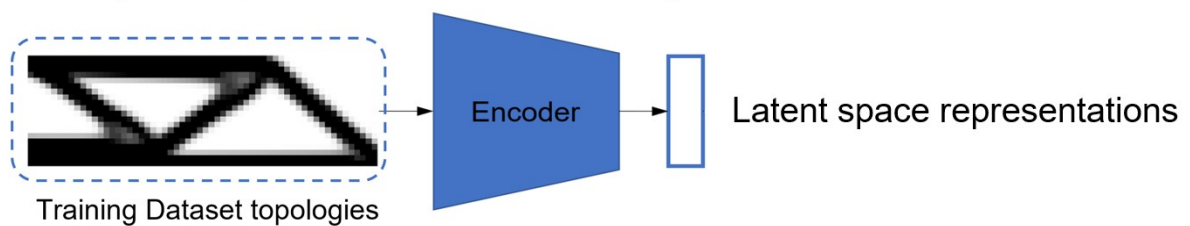


Figure 5.11: Encoding the training dataset using the encoder branch of the trained AE.

5.1.2. Step 2: Training the MLP

The second step of the proposed deep learning pipeline consists of training the MLP model. The MLP is a fully connected class of feedforward artificial neural network, which consists of an input layer, hidden layers and an output layer. The input layer receives the input data, which is a vector of the loading parameters introduced in chapter 4. The hidden layers perform intermediate computations on the input data and learn complex relationships between the input and output features. Finally the hidden layers are connected to the 40-dimensional latent space representations that have been encoded in step 1(b). In this way the model learns to predict the latent space representation that correspond the input loading parameters. A generalised MLP architecture is shown in Fig. 5.12. The following sections will summarize the details of the MLP architectures for the 2D and 3D cases.

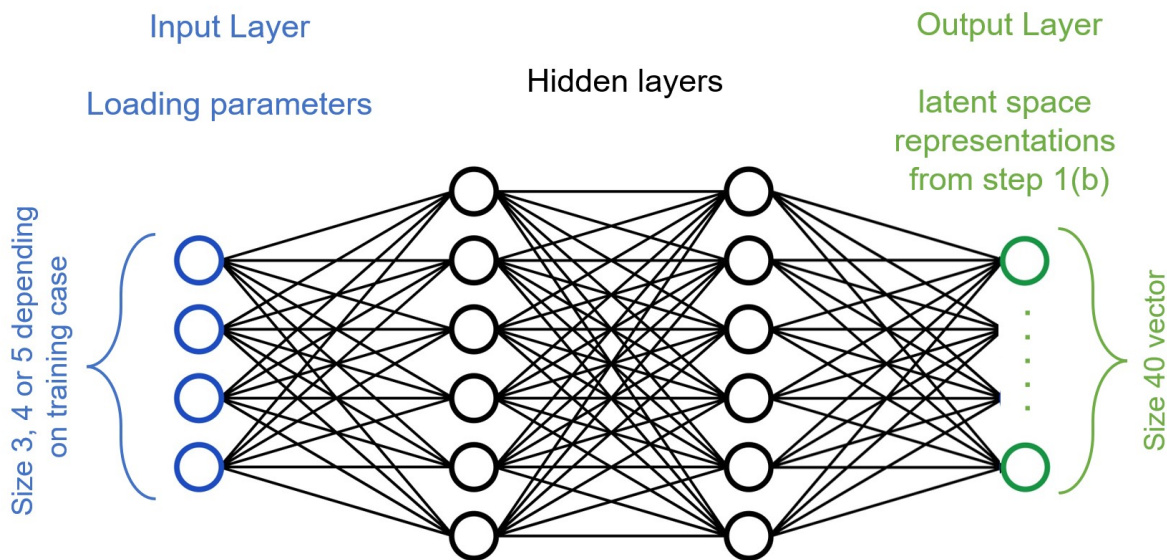


Figure 5.12: A generalised schematic of the MLP architecture.

2D MLP

The MLP model for the 2D MBB consists firstly of an input layer. As stated in chapter 4, the parametrized loading parameters that are being varied to create the dataset are the the x- and y- coordinates of the applied load, and the angle of the load with respect to the x-axis. These 3 loading parameters are normalized between 0 and 1. Normalizing the data between 0 and 1 is a common data pre-processing step in machine learning because it helps improve the training process and the performance of the model. One of the reasons for this is that normalizing puts all the features on the same scale. Without

normalization, features with larger values will have a disproportionately large influence on the model. Normalization is achieved by dividing the loading parameter by its maximum possible value before it is fed in the network; for the x- and y- coordinates they are divided by 120 and 40 respectively, since this is the maximum size of the domain and thus the largest value that it can assume, for the angle, it is divided by 90, since the units of measurement being used is degrees. Only the first quadrant concerning the angles is of interest, since for the MBB example, the topologies produced are 'quadrant invariant', that is, the topologies produced for the angles between 0 and 90 degrees are identical to those produced by the second quadrant between 90 and 180 degrees, and so on and so forth for the third and fourth quadrants.

These normalized loading parameters are introduced into the hidden layers. The hidden layers consist of 2 dense layers, each of which is followed by a batch normalization layer. All of these layers feature 720 neurons. The final layer is the output layer which features 40 neurons. In the training phase the target output layer is the 40-dimensional latent space representation that has been encoded by the encoder branch of the AE in the first step. In this way the MLP learns to create a mapping between the loading parameters and the latent space representations that they produce.

Similarly to the AE, the MLP model used to train on the the TorC dataset, has slight difference to the MLP model used to train on the optimal topologies and VM stress dataset. For the optimal topologies and VM stress dataset, the activation functions used for the hidden dense layers in the MLP is ReLU, whilst the activation function for the output layer is sigmoid. This is to match the activation function used in the output of the 'encoder' branch of the AE, such that the output values fall in the same range. The loss function used is binary cross entropy and the accuracy metric is MAE. For the TorC dataset on the other hand the only differences are in the activation function for the output layer, which is a linear activation function, and the loss function used, which is MSE.

A detailed table of the MLP architecture for the 2D MBB case is given in the appendix.

3D MLP

For the 3D datasets, the MLP model is almost identical to the 2D case. The only difference is the size of the input layer, which changes with respect to the number of input loading parameters used to create the dataset.

For the 3D cantilever case, the size of the input layer is 5, which takes the x-, y- and z- coordinates, and azimuth and inclination angle $(x, y, z, \theta, \varphi)$. Just like the 2D case, the loading parameters are first normalized between 0 and 1, which for the coordinates

involves dividing by the maximum coordinate in the domain; 24, 12, 12 for the x-, y-, and z- coordinates respectively. For the azimuth and inclination angle, these are divided by the maximum angle allowed in the dataset, 360 and 180 degrees respectively.

For the 3D bridge case, the size of the input layer is 4; two of which correspond to the location along the x-axis of the two acting loads on the bridge deck, whilst the remaining two correspond to the x-axis location of the supports to the left and right of the central void region. Again, these loading parameters must be normalized between 0 and 1; the loads are divided by 60, which represents the length of the deck and maximum allowable location of the load, whilst the values taken for each of the supports is divided by 15, as this is the maximum allowable range in which the supports can assume a position.

Besides the loading parameters, the size and architecture of the MLP including activation functions, loss function and accuracy metrics are identical to those used for the 2D case, for the optimal topologies, VM stress and TorC datasets. A detailed summary is included in the appendix.

5.1.3. Step 3: Test the combined deep learning pipeline

The third step consists of testing the combined AE and MLP models that have trained in step 1 and step 2 (see Fig. 5.13). In this step, we utilise a test set of 500 images (and their corresponding loading parameters) to test the accuracy of the proposed multi-stage model.

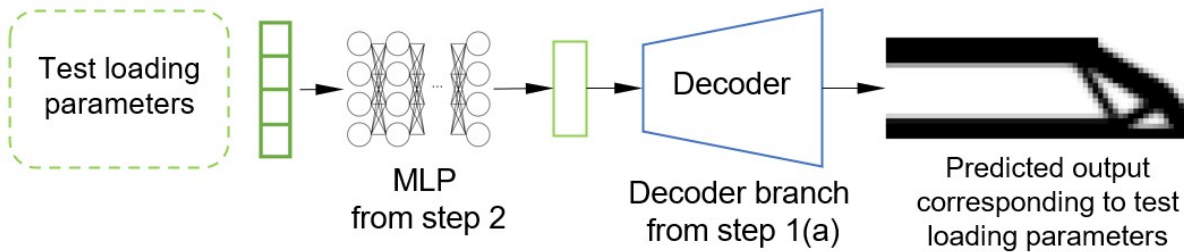


Figure 5.13: Procedure for testing the proposed DL pipeline.

For this testing step, unseen loading parameters are introduced into the MLP trained in step 2. This produces a predicted 40-dimensional latent space representation, which is then introduced as the input to the decoder branch trained in step 1(a). This decoder branch produces a predicted output, that will be either an optimal topology, VM stress distribution, or TorC zone depending on the test case selected. The results of this testing step are shown in chapter 6.

6 | Results

To evaluate the similarity between the predicted structures and the optimal topologies produced using the SIMP algorithm, three accuracy metrics are used: the binary accuracy (BA) [65], the MAE and the root mean squared accuracy (RMS). The BA metric is given by:

$$BA = \frac{\omega_{00} + \omega_{11}}{n_0 + n_1}. \quad (6.1)$$

Herein:, n_l with $l \in \{0, 1\}$, is the total number of pixels of class l , and ω_{tp} , with $t \in \{0, 1\}$ and $p \in \{0, 1\}$, is the total number of pixels of class t predicted to belong to class p . Because the predicted structures contained pixel values within a continuum range (between 0 and 1 for optimal topologies and VM stress, and between -1 and 1 for TorC zones), a threshold function was utilised to binarize the predicted outputs.

$$x_{i,binary} = \begin{cases} 1, & \text{if } x_i > x_{threshold} \\ 0, & \text{if } x_i < x_{threshold} \end{cases} \quad \text{for } i = 1, \dots, n. \quad (6.2)$$

Herein: x_i is the input topology, $x_{i,binary}$ is the binarized output field, and $x_{threshold}$ is the binary threshold value: 0.5 for optimal topologies and VM stress, and 0 for TorC zones. The MAE and RMS accuracy metrics are given as follows:

$$MAE = \frac{1}{n} \sum_{i=1}^n |x_i - \hat{x}_i| \quad \text{for } i = 1, \dots, n. \quad (6.3)$$

$$RMS = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2} \quad \text{for } i = 1, \dots, n. \quad (6.4)$$

Where x_i refers to the density value assumed by element i , \hat{x}_i refers to the predicted density value assumed by element i , and n is the total number of elements. Additionally,

qualitative assessment will also be done in the following section by visually comparing the outputs.

6.1. 2D

6.1.1. 2D MBB

For the 2D MBB case, a testing set consisting of 500 topologies is generated and used for evaluation. These topologies are generated using the SIMP algorithm with the same LHS sampling strategy discussed in chapter 4. The three accuracy metrics; BA, MAE and RMS are utilised to evaluate the accuracy of the optimal topologies, VM stress field, and TorC zones predicted by the trained multi-stage neural network discussed in chapter 5.

The first quantitative evaluation of the benefit of the proposed method is the comparison between the time taken to generate the optimal topology using the SIMP algorithm, and the time taken to predict the optimal topology using the trained neural network. This is shown below in table. 6.1.

Table 6.1: Comparison of average computational run-time between SIMP algorithm and proposed methodology for the 2D MBB case.

	SIMP	Proposed Method: Optimal Topology	Proposed Method: Optimal topology + VM or TorC
Dataset creation	-	8.10 hours	8.10 hours
Training DL model	-	0.44 hours	0.88 hours
Average run-time (in seconds)	12.00 s	0.08 s	0.16 s

The first key distinction between the SIMP algorithm and the proposed method, is that the proposed method necessitates the creation of a training dataset to train the DL model. The 2500 image dataset took 8.1 hours to create (refer to chapter 4 for details on the formulation). The benefit of this method is that once the SIMP algorithm has calculated the optimal topology, additional physical data about the problem, such as the VM stress field, and principal stresses can be calculated at no extra computational cost. As such, the dataset creation time for 'optimal topology + VM and TorC' in table. 6.1 is also 8.1 hours. The second step of the proposed method is the training of the DL model, which necessitates additional computational time. For the optimal topology only, the DL model

can be trained in 0.44 hours. If however, the model is trained for both optimal topology and one of either VM stresses or TorC datasets, an additional 0.44 hours of training time is required, as an additional DL model is required for either the VM or TorC datasets. These two preliminary steps; the dataset creation and the model training can be done 'offline'.

For testing, the proposed method takes a fraction of the time compared with the traditional SIMP method. To predict the optimal topology, the first step of the proposed multistage pipeline takes on average 0.02 seconds. This is the MLP which takes as input the loading parameters and returns as output the latent space representations. The second part of the pipeline is the decoder branch, which takes as input the latent space representations from the first part, and returns as output the predicted topology. This second part takes on average 0.06 seconds. Used together, these two parts take on average 0.08 seconds. The proposed method takes on average 0.67% of the time it takes to run an equivalent SIMP algorithm. To predict the final VM stress, or TorC zones mapped onto the optimal topology it takes on average 0.16 seconds. This is twice as long because the proposed model must be run twice, first to predict the optimal topology (0.08 seconds) and second to predict either the full VM stress field, or TorC zones (0.08 seconds). Together, the element-wise multiplication of these two fields produces the final output, which is either the VM stresses, or TorC zones mapped onto the optimal topology. This takes on average 1.33% of the time it takes to run an equivalent SIMP algorithm.

Fig. 6.1 shows six topologies from the testing dataset chosen to showcase a variety of forms and results. The 'ground truth' topology refers to that produced by the SIMP algorithm, whilst the 'prediction' topology refers to that predicted using the proposed model. The location and direction of the acting force is specified below each test case and is also overlaid on the diagram for clarity. The three accuracy metrics are reported for each specific topology.

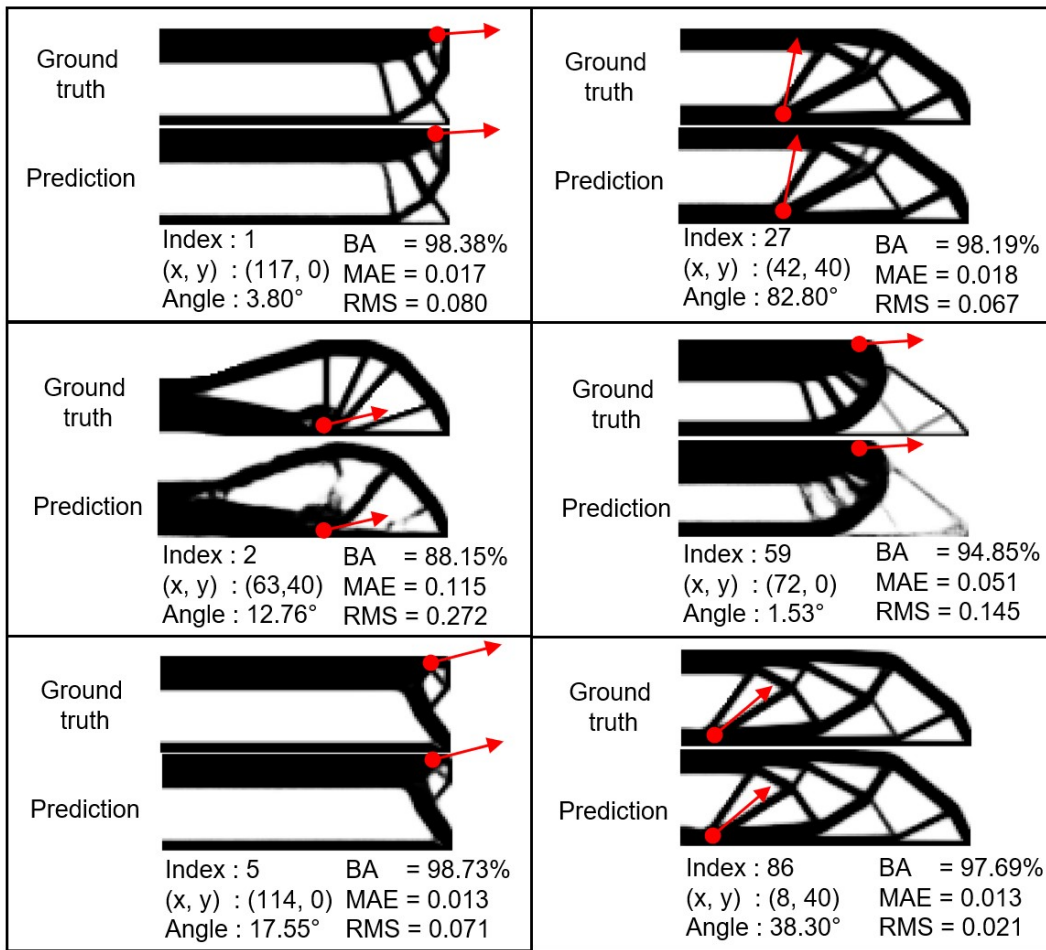


Figure 6.1: Accuracy comparison of the ground truth and prediction for six optimal topologies for the 2D MBB case.

Fig. 6.2 shows the results for the predicted VM stresses for the same six ground truth topologies shown in Fig. 6.1. Each 'case' is shown within its own border, within which the top row represents the ground truth results generated using the SIMP algorithm, and the bottom row represents the predictions generated by the DL model. The first column presents the same optimal topologies as shown in Fig. 6.1. The second column shows the stress field over the whole domain. The three accuracy metrics are shown underneath for each test case. The third column is the final VM result; the projection of the VM stress field onto the optimal topology via an element-wise multiplication between the two matrices.

Fig. 6.3 shows the results for the predicted TorC zones, again for the same six ground truth topologies shown in Fig. 6.1. The format is the same as that in Fig. 6.2, with the addition of a fourth column; in which the dominant principal stress values on the final TorC topology are 'binarized'; that is, made a fully tension or compression diagram for

visual clarity.

The results for the accuracy metrics presented in table. 6.2 are averages taken over the full testing dataset.

Table 6.2: Comparison of average BA, MAE and RMS accuracy metrics for the predicted topologies when compared to the ground truth for the 2D MBB case.

	Optimal Topologies	Initial VM stress field	Initial TorC stress field
BA	96.46%	99.29%	95.26%
MAE	0.035	0.018	0.025
RMS	0.107	0.030	0.048




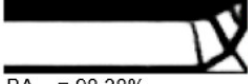
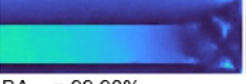








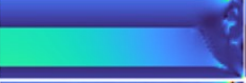

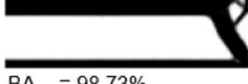



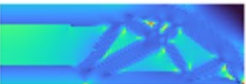


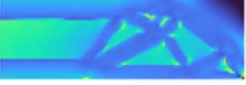



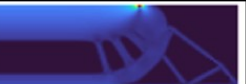




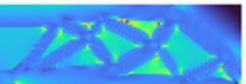


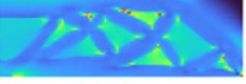

	<i>Optimal topology (topopt)</i>	<i>Initial VM stress field (VM)</i>	<i>Final VM = topopt .* VM</i>
Ground truth			
Prediction			
	BA = 98.38% MAE = 0.017 RMS = 0.080	BA = 99.98% MAE = 0.023 RMS = 0.033	
Ground truth			
Prediction			
	BA = 88.15% MAE = 0.115 RMS = 0.272	BA = 99.94% MAE = 0.016 RMS = 0.030	
Ground truth			
Prediction			
	BA = 98.73% MAE = 0.013 RMS = 0.071	BA = 99.96% MAE = 0.014 RMS = 0.027	
Ground truth			
Prediction			
	BA = 98.19% MAE = 0.018 RMS = 0.067	BA = 99.92% MAE = 0.010 RMS = 0.020	
Ground truth			
Prediction			
	BA = 94.85% MAE = 0.051 RMS = 0.145	BA = 99.94% MAE = 0.018 RMS = 0.034	
Ground truth			
Prediction			
	BA = 97.69% MAE = 0.013 RMS = 0.021	BA = 99.58% MAE = 0.013 RMS = 0.021	

Figure 6.2: Comparison between ground truth and predictions for VM stresses for the 2D MBB case.

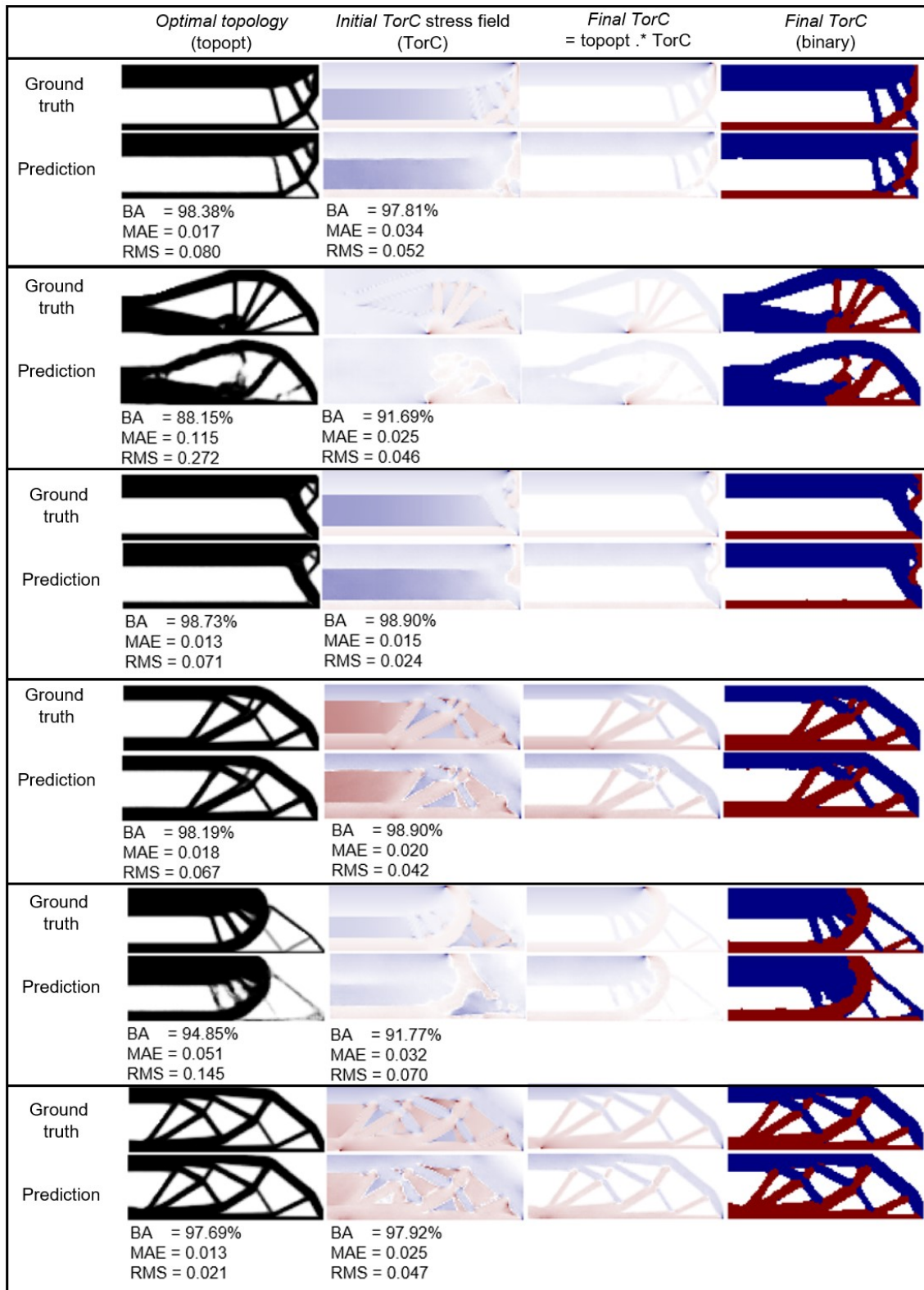


Figure 6.3: Comparison between ground truth and predictions for TorC zones for the 2D MBB case.

6.2. 3D

For the 3D cantilever and 3D bridge cases, testing sets consisting of 500 topologies are generated for each case and used for evaluation. These topologies are generated using the SIMP algorithm with the same LHS sampling strategy discussed in chapter 4. Similarly to the 2D case, the three accuracy metrics; BA, MAE and RMS are utilised to evaluate the accuracy of the optimal topologies, VM stress field, and TorC zones predicted by the trained multi-stage neural network discussed in chapter 5.

6.2.1. 3D cantilever

The first quantitative evaluation of the benefit of the proposed method is the comparison between the time taken to generate the optimal topology using the SIMP algorithm, and the time taken to predict the optimal topology using the trained neural network. This is shown below in table. 6.3.

Table 6.3: Comparison of average computational run-time between SIMP algorithm and proposed methodology for 3D cantilever.

	SIMP	Proposed Method: Optimal Topology	Proposed Method: Optimal topology + VM or TorC
Dataset creation	-	27.72 hours	27.72 hours
Training DL model	-	0.61 hours	1.22 hours
Average run-time (in seconds)	39.91 s	0.13 s	0.26 s

As in the 2D case, the proposed method for the 3D case necessitates a training dataset creation stage, and a DL model training stage. Both of these steps can again be done 'offline'. For the 3D cantilever, the 2500 image dataset took 27.72 hours to create (refer to chapter 4 for details on the formulation). The VM stresses and principal stresses are calculated at no extra computational cost and as such the the dataset creation time for 'optimal topology + VM and TorC' in table. 6.3 is also 27.72 hours. For the optimal topology only, the DL model can be trained in 0.61 hours. If however, the model is trained for both optimal topology and one of either VM stresses or TorC datasets, an additional 0.61 hours of training time is required, as an additional DL model is required for either the VM or TorC datasets.

To predict the optimal topology for the 3D cantilever, the first step of the proposed

multistage pipeline - the MLP which takes as input the loading parameters and returns as output the latent space representations - takes on average 0.02 seconds. This takes the same as the 2D MBB case since the model architecture is almost identical. The second part of the pipeline is the decoder branch, which takes as input the latent space representations from the first part, and returns as output the predicted topology. This second part takes on average 0.11 seconds. This takes marginally longer than the 2D MBB case, since the decoder branch uses 3D convolutional layers which are applied over a slightly larger domain. Used together, these two parts take on average 0.13 seconds. The proposed method takes on average 0.33% of the time it takes to run an equivalent SIMP algorithm. To predict the final VM stress, or TorC zones mapped onto the optimal topology it takes on average 0.26 seconds. This is twice as long because, just as in the 2D case, the proposed model must be run twice, first to predict the optimal topology (0.13 seconds) and second to predict either the full VM stress field, or TorC zones (0.13 seconds). Together, the element-wise multiplication of these two fields produces the final output, which is either the VM stresses, or TorC zones mapped onto the optimal topology. This takes on average 0.66% of the time it takes to run an equivalent SIMP algorithm.

Fig. 6.4 shows five topologies from the testing dataset chosen to showcase a variety of forms and results. The 'ground truth' topology refers to that produced by the SIMP algorithm, whilst the 'prediction' topology refers to that predicted using the proposed model. The location and direction of the acting force is specified below each test case and is also overlaid on the diagram for clarity. The three accuracy metrics are reported for each specific topology.

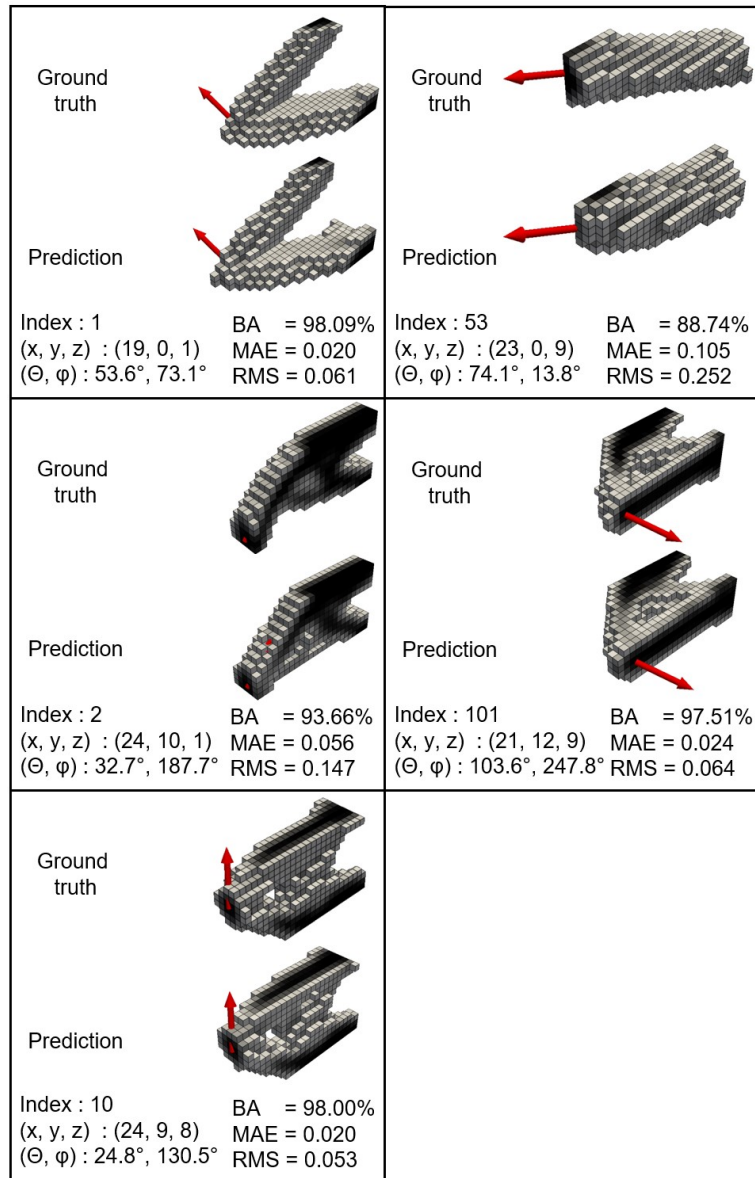


Figure 6.4: Accuracy comparison of the ground truth and prediction for six optimal topologies for the 3D cantilever case.

Fig. 6.5 shows the results for the predicted VM stresses for the same five ground truth topologies shown in Fig. 6.4. Each 'case' is shown within its own border, within which the top row represents the ground truth results generated using the SIMP algorithm, and the bottom row represents the predictions generated by the DL model. The first column presents the same optimal topologies as shown in Fig. 6.4. The second column shows the stress field over the whole domain. The three accuracy metrics are shown underneath for each test case. The third column is the final VM result; the projection of the VM stress field onto the optimal topology via an element-wise multiplication between the two

matrices. The fourth column displays the same topology as shown in column 3 but with a normalization applied, such that the values lie in a continuum between 0 and 1.

Fig. 6.6 shows the results for the predicted TorC zones, again for the same five ground truth topologies shown in Fig. 6.4. The format is the same as that in Fig. 6.5, with the exception of the fourth column; in which the dominant principal stress values on the final TorC topology are 'binarized'; that is, made a fully tension or compression diagram for visual clarity.

The results for the accuracy metrics presented in table. 6.4 are averages taken over the full testing dataset.

Table 6.4: Comparison of average BA, MAE and RMS accuracy metrics for the predicted topologies when compared to the ground truth for the 3D cantilever case.

	Optimal Topologies	Initial VM stress field	Initial TorC stress field
BA	96.59%	97.22%	82.15%
MAE	0.032	0.054	0.090
RMS	0.082	0.078	0.133

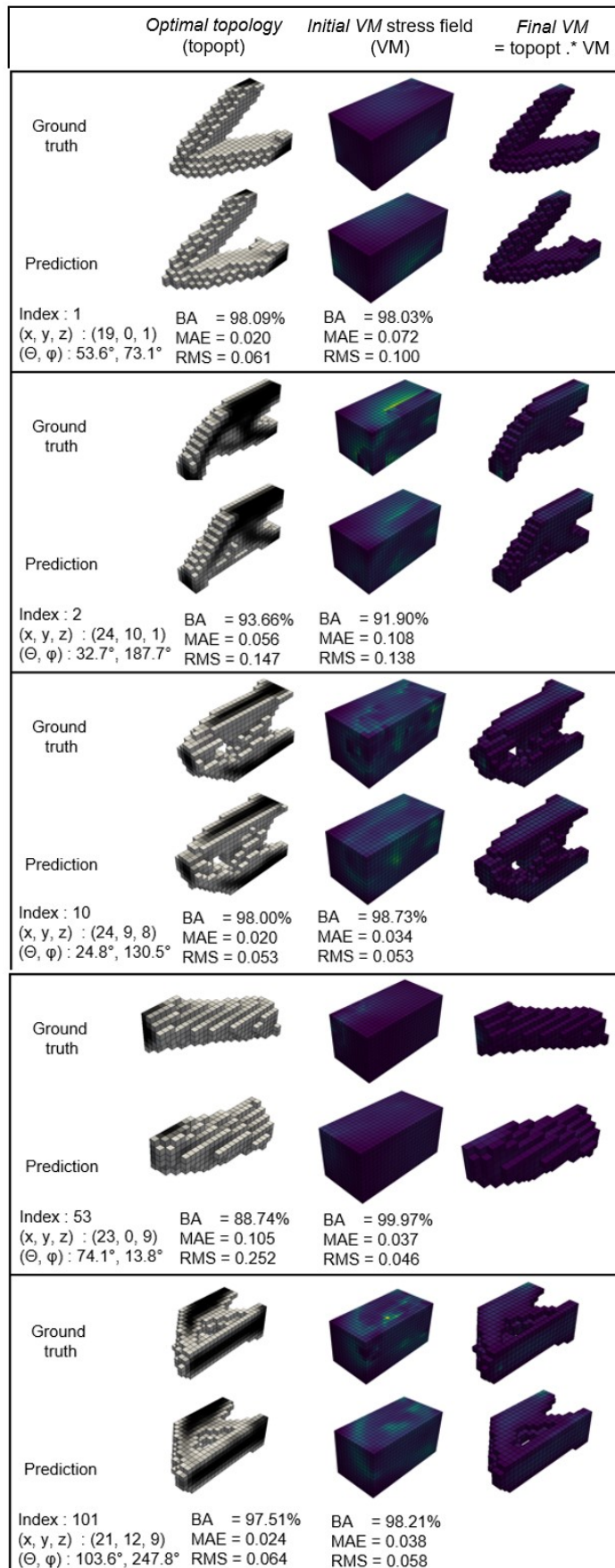


Figure 6.5: Comparison between ground truth and predictions for VM stresses for the 3D cantilever case.

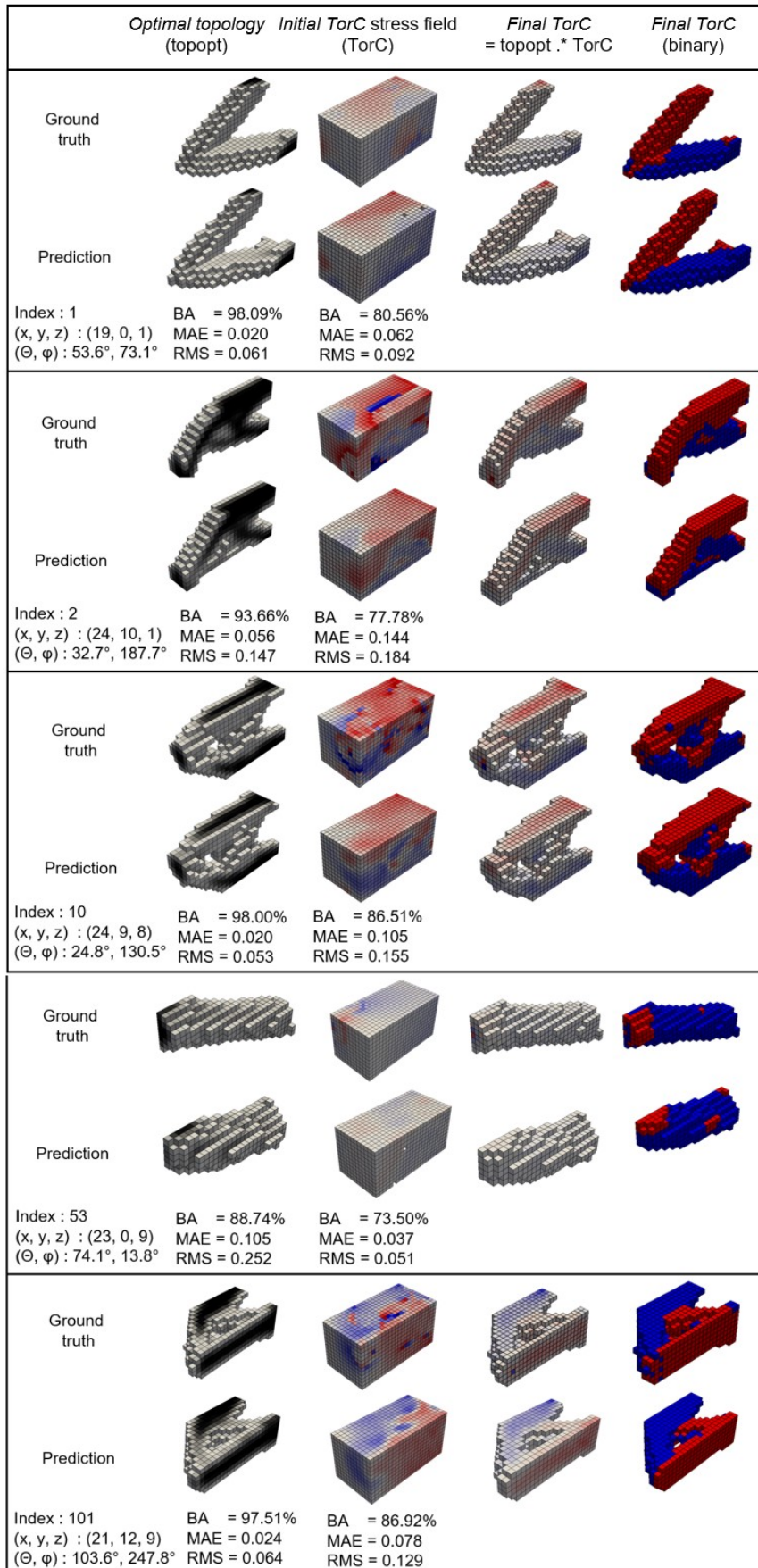


Figure 6.6: Comparison between ground truth and predictions for TorC zones for the 3D cantilever case.

App

Although the traditional SIMP algorithm can provide very optimized solutions for structural problems, the long computation times, especially for increasingly larger domains, has prevented it from developing into a serious and widespread design tool for structural engineers. Design tools that can provide solutions with shorter computation times, and allow an engineer to iterate over many potential solutions to get an understanding of how the problem changes with respect to variations in the loading and boundary conditions can save time and increase efficiency. In the following section an app is presented, whereby the loading parameters for each 3D case (explained in detail in chapter 4 is parametrized and controlled by the user. This app has been created using the Python package *PyVista* [98]. *PyVista* is a Python library for 3D visualization and analysis of scientific datasets such as meshes, point clouds, and volumetric data and provides a user-friendly interface to generate 3D plots and interactive visualizations with advanced rendering capabilities. In this app, the predicted topology changes in response to the chosen loading parameters, and updates almost instantaneously as the loading parameters are updated. The loading parameters are controlled using a sliding bar with a predefined range. The topology is displayed within the window of the user interface as a 3D object. Fig. 6.7 shows three screenshots of the app with randomly chosen loading parameters and predicted topologies.

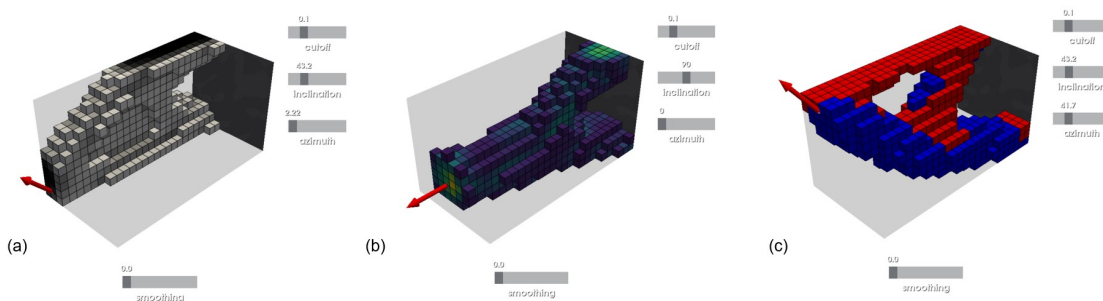


Figure 6.7: PyVista render of three randomly predicted topologies for (a) optimal topology (b) final VM stress and (c) final TorC stress.

The topology is highly interactive, and the user can click on the surface of the 3D cantilever domain (displayed as a light grey box) to change the position of the loading vector. The loading vector updates in real time, and the new loading parameters are passed through the trained DL model to predict and display the new updated topology. The sliding bars on the right control the inclination and azimuth angle. Likewise, these can be changed by the user to update the displayed topology in real time. As seen in Fig. 6.7, the user can toggle between displaying the optimal topology, final VM stress or final TorC stress. Another additional feature is to control the cutoff level for the predicted optimal topology

densities. With the cutoff, any element values that are below a predefined level are not displayed. This can make it easier to visualise the 'core' structural shape. An example topology is shown in Fig. 6.8 with three different cut off points, 0.1, 0.2 and 0.4 for Fig. 6.8 (a), (b) and (c) respectively.

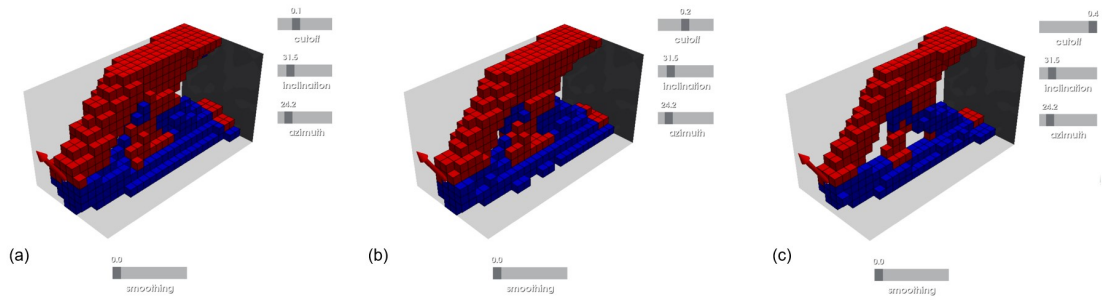


Figure 6.8: PyVista render of a predicted final VM stress for cutoff values (a) 0.1 (b) 0.2 and (c) 0.4.

Finally, PyVista provides the capability to add smoothing to the predicted mesh. This is visualised in Fig. 6.9 for the predicted final TorC stress.

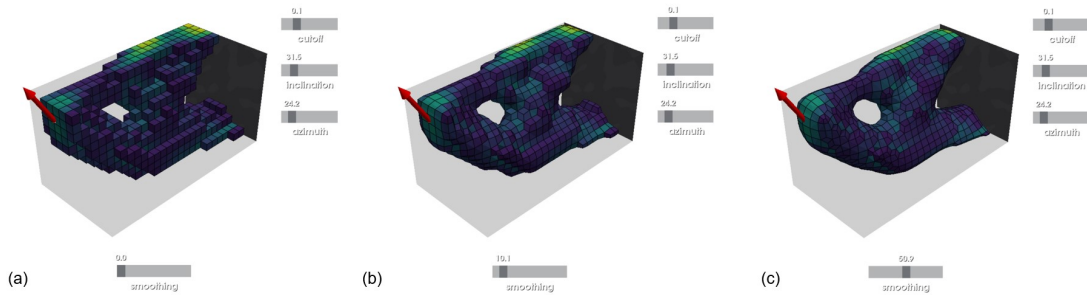


Figure 6.9: PyVista render of a predicted final TorC stress for smoothing values (a) 0.0 (b) 10.1 and (c) 50.9.

6.2.2. 3D bridge

The same quantitative and qualitative evaluations used for the 3D cantilever are used for the 3D bridge. The comparison between the time taken to generate the optimal topology using the SIMP algorithm, and the time taken to predict the optimal topology using the trained neural network is shown below in table. 6.5.

Table 6.5: Comparison of average computational run-time between SIMP algorithm and proposed methodology for 3D bridge.

	SIMP	Proposed Method: Optimal Topology	Proposed Method: Optimal topology + VM or TorC
Dataset creation	-	25.91 hours	25.91 hours
Training DL model	-	0.61 hours	1.22 hours
Average run-time (in seconds)	37.31 s	0.13 s	0.26 s

For the 3D bridge, the 2500 image dataset took 25.91 hours to create (refer to chapter 4 for details on the formulation). The VM stresses and principal stresses, are calculated at no extra computational cost and as such the dataset creation time for 'optimal topology + VM and TorC' in table. 6.5 is also 25.91 hours. For the optimal topology only, the DL model can be trained in 0.61 hours, and if both optimal topology and one of either VM stresses or TorC datasets, an additional 0.61 hours of training time is required, as an additional DL model is required for either the VM or TorC datasets.

To predict the optimal topology for the 3D bridge, takes the DL model on average 0.13 seconds. This consists of the first part (MLP) which takes on average 0.02 seconds, and the second part (decoder branch of the AE), which takes on average 0.11 seconds. The proposed method takes on average 0.35% of of the time it takes to run an equivalent SIMP algorithm. To predict the final VM stress, or TorC zones mapped onto the optimal topology it takes on average 0.26 seconds. This takes on average 0.70% of the time it takes to run an equivalent SIMP algorithm.

Fig. 6.10 shows five topologies from the testing dataset chosen to showcase a variety of forms and results. The 'ground truth' topology refers to that produced by the SIMP algorithm, whilst the 'prediction' topology refers to that predicted using the proposed model. The location and direction of the acting force is specified below each test case and is also overlaid on the diagram for clarity. The three accuracy metrics are reported for each specific topology.

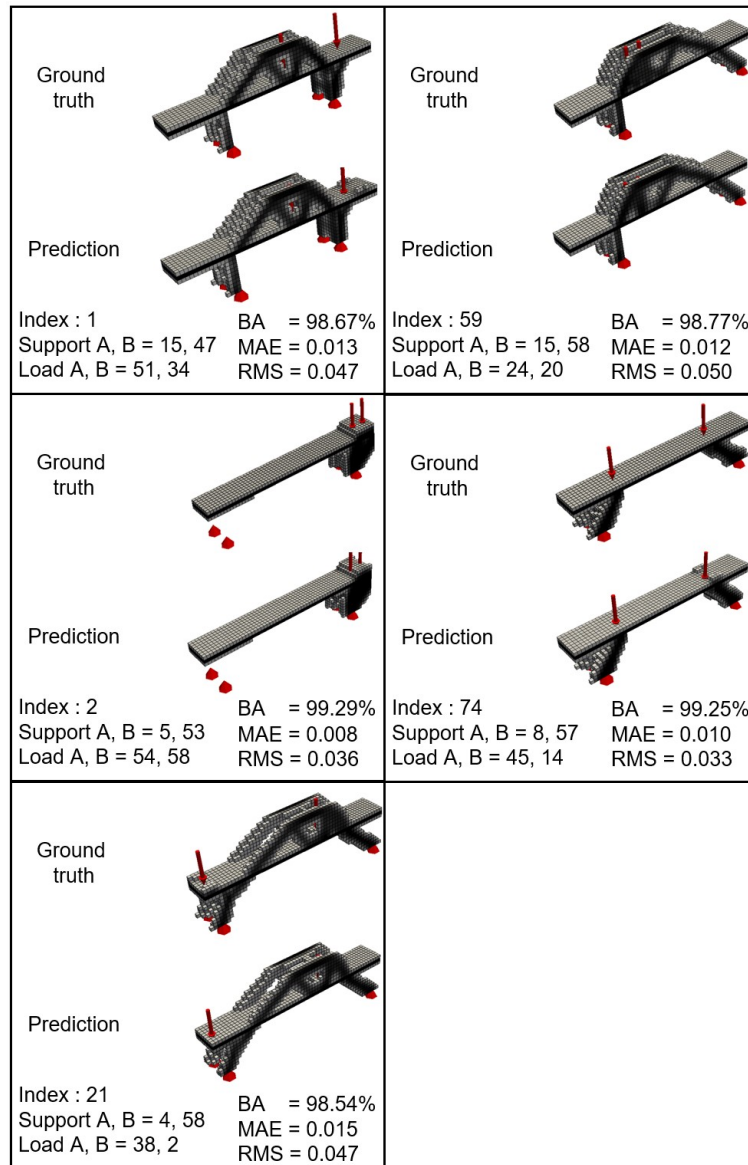


Figure 6.10: Accuracy comparison of the ground truth and prediction for six optimal topologies for the 3D bridge case.

Fig. 6.11 shows the results for the predicted VM stresses for the same five ground truth topologies shown in Fig. 6.10. The format of this figure is similar to that shown in Fig. 6.4 for the 3D cantilever case. Fig. 6.12 shows the results for the predicted TorC zones, again for the same five ground truth topologies shown in Fig. 6.10. The format is the same as that in Fig. 6.12 for the 3D cantilever case.

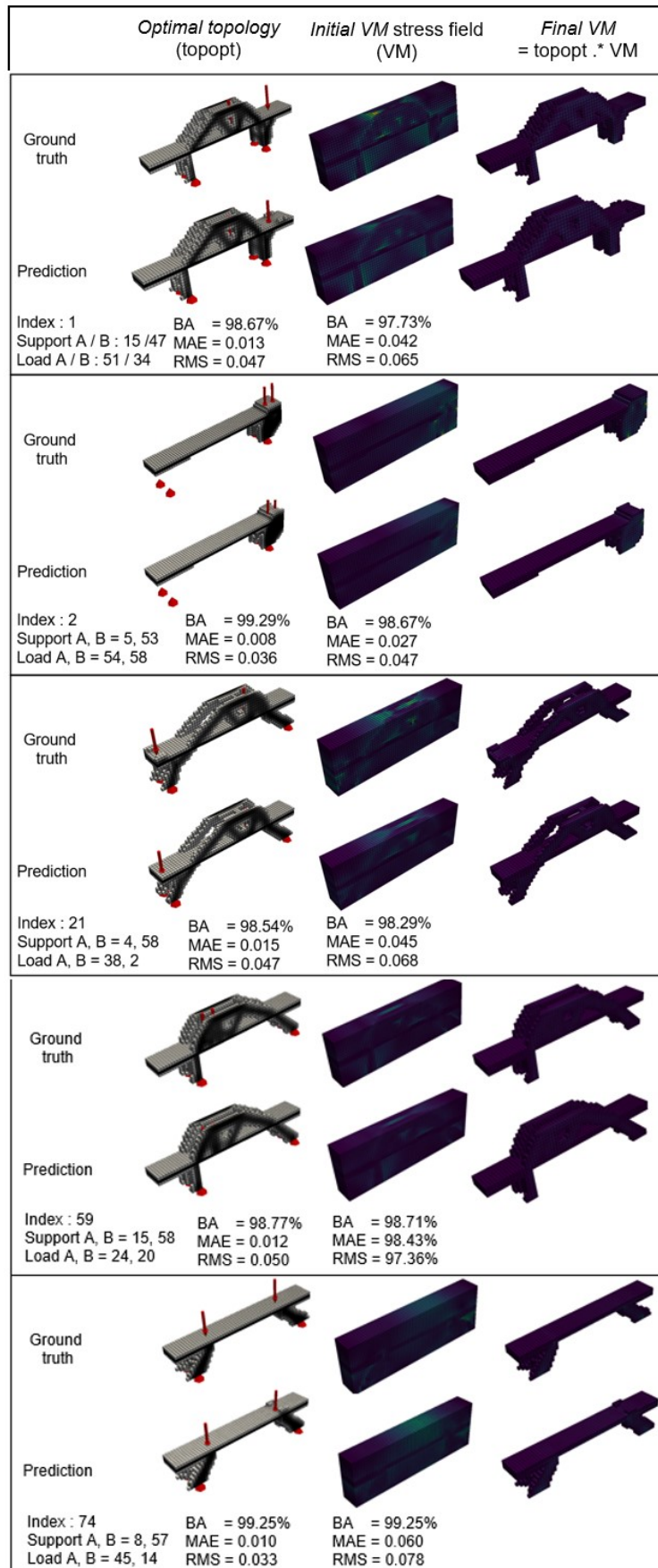


Figure 6.11: Comparison between ground truth and predictions for VM stresses for the 3D bridge case.

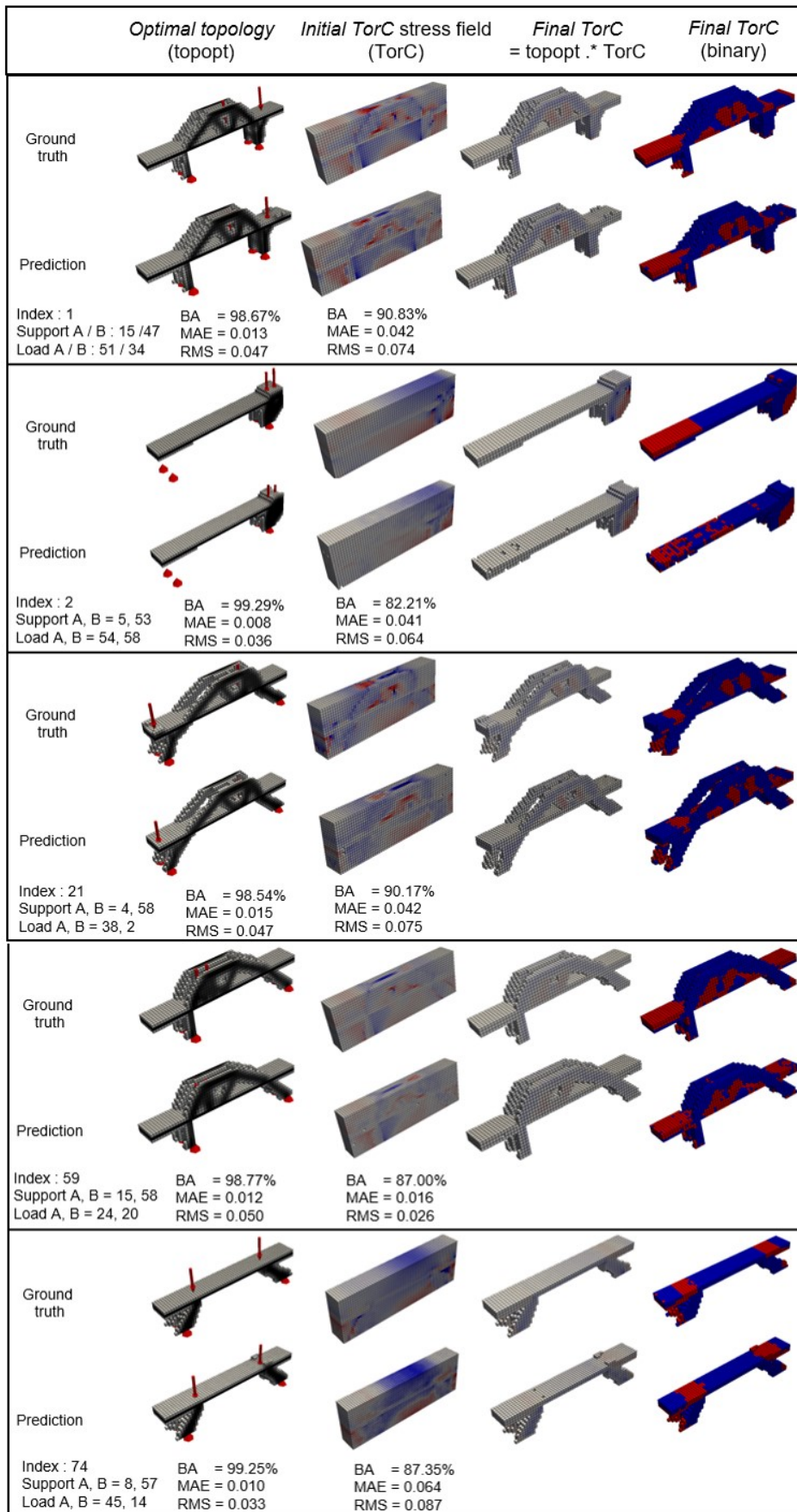


Figure 6.12: Comparison between ground truth and predictions for TorC zones for the 3D bridge case.

The results for the accuracy metrics presented in table. 6.6 are averages taken over the full testing dataset.

Table 6.6: Comparison of average BA, MAE and RMS accuracy metrics for the predicted topologies when compared to the ground truth for the 3D bridge case.

	Optimal Topologies	Initial VM stress field	Initial TorC stress field
BA	98.50%	98.27%	88.36%
MAE	0.015	0.037	0.047
RMS	0.051	0.057	0.078

App

The PyVista app has been created also for the 3D bridge case. The aim is the same; the allow the user to dynamically change the loading parameters to experiment with the different predicted outputs, and visualize and interact with the results in real time. The 3D bridge case however, utilizes different loading parameters, and the app was modified to reflect this. the location of the two supports, and the location of the two vertical forces on the deck are controlled by slider bars, see Fig. 6.13 for three randomly generated 3D bridge topologies.

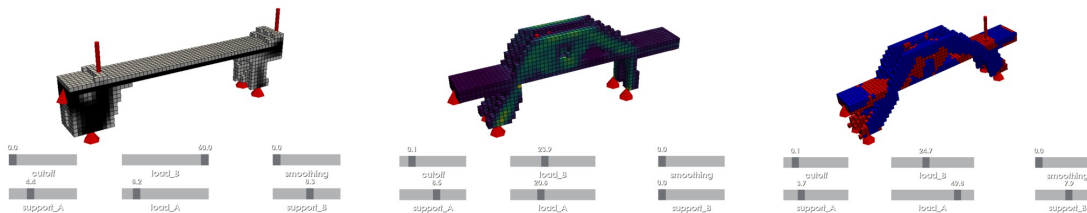


Figure 6.13: PyVista render of three randomly predicted topologies for (a) optimal topology (b) final VM stress and (c) final TorC stress.

As for the 3D cantilever case the following Fig. 6.14 renders the same final predicted TorC topology with with three different cut off points, 0.1, 0.2 and 0.4 for Fig. 6.14 (a), (b) and (c) respectively.

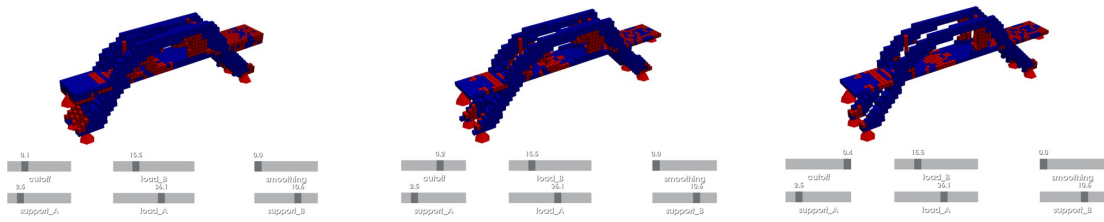


Figure 6.14: PyVista render of a predicted final VM stress for cutoff values (a) 0.1 (b) 0.2 and (c) 0.4.

7 | Conclusions and future developments

This thesis proposes an efficient non-iterative multistage DL model that is capable of predicting a near optimal solution for the topology optimization problem in 2D and 3D. The model consists of two parts; first, a multi-layer perceptron model that takes as inputs the loading parameters for the specific problem case and outputs the corresponding latent space representation. Second, the decoder branch of an autoencoder that takes as input this previously generated latent space representation and predicts either the optimal topology, the VM stress field, or the TorC field. Together, these two parts combine to produce the final DL pipeline that is able to predict accurate solutions almost instantaneously. The final VM or TorC results, are an element-wise multiplication between the predicted optimal topology and either the predicted VM stress field or predicted TorC field respectively. The predicted optimal topology thus acts as a masking layer, and the predicted VM stress field or predicted TorC field will be mapped onto those elements with a positive density value, whilst the elements with a null value will remain void spaces. A total training set of 2,500 images was generated for all three problem cases. The training time varied depending on the size of the domain for each case; for the 2D MBB, 3D cantilever, and 3D bridge, the training time was 8.1 hours, 27.72 hours, and 25,91 hours, respectively. Additionally, time was required to train the DL models, ranging from 0.44 hours to 0.66 hours for the 2D and 3D cases, respectively. The one key feature of the proposed method is that it requires a large up front time investment to create the training dataset and train the DL model. This training was all done 'offline', that is, completed before the model was deployed. Once the training was completed however, the prediction times are near-instantaneous, and achieve near-optimal results when compared to the traditional SIMP topology optimization algorithms.

Discussion. The proposed method offers clear advantages over the traditional SIMP method by allowing the user to generate near optimal solutions almost instantaneously. This permits the designer to investigate many design solutions, and gain an understanding of how the problem responds to variations in the input loading parameters. To this end,

an app was created with the aim of allowing the user to interact with the problem, such that it updates dynamically on screen in response to user changes of the parametrized loading parameters. In the conceptual design phase especially, when it is important to investigate many iterations of the design problem quickly and accurately, this app can provide instant feedback and facilitate the task of finding an effective solution. Screenshots and explanations of this app are provided in chapter 6.

Interpretations of the three accuracy metrics are now discussed. Comparing the total average scores for the predicted *optimal topologies* when compared to the ground truth, the 3D bridge scored the highest (98.50%, 0.015, 0.051), followed by 3D cantilever (96.60%, 0.032, 0.082) and then the 2D MBB (96.46%, 0.035, 0.107), for the three accuracy metrics (binary accuracy, mean absolute error, root mean square). A potential explanation for the accuracy of these results is as follows:

- The 3D bridge dataset yields the highest scores for all three accuracy metrics despite having the highest number of total elements within the domain ($6 * 20 * 4 = 4800$ equal with the 2D MBB). One explanation for this is the recurring structure within the training dataset. The deck (with a total number of elements of $60 * 1 * 4 = 240$) is solid for every topology in the training dataset. The same occurs for the void space under the deck (with a total number of elements of $30 * 10 * 4 = 1200$), which contains empty void elements for every topology in the training dataset. This provides a recurring pattern that the machine learning model is able to learn accurately, and translates this well to unseen but similar cases in the test set. Fig. 6.10 illustrates this argument; the predicted topologies all show a fully dense deck layer, and an empty void space despite the variation in loading and boundary conditions. Although the 3D cantilever had less total elements in the domain ($24 * 12 * 12 = 3456$), it produced lower test set scores for the optimal topology case, and this is in part because all elements within the domain were allowed to vary, whereas the 3D bridge case contained only $4800 - 240 - 1200 = 3360$ elements which are free to vary.
- Another explanation for the success of the 3D bridge case with respect to the 3D cantilever is the nature of the input loading parameters for each of the models. The capacity of the DL model to learn the complex non-linear relationships between input and outputs of the model can be complicated if small variations in the combinations of the inputs lead to large variations in the outputs. For the 3D cantilever, the inclination and azimuth angles of the loading vector varied between 180 and 360 degrees respectively. When considering these values had to be normalized between 0 and 1 to be introduced into the neural network, small changes in the decimal values of these inputs, had large impacts in the outputs. For example, a change of 0.02 in

the input of the azimuth angle translates to a change of 7.2 degrees, which can affect the topology quite dramatically. The 3D cantilever model is thus quite sensitive to changes input parameters and this could be a reason for the lower performance with respect to the 3D cantilever.

- The 2D MBB performed the lowest for the three accuracy metrics for the optimal topology. This result could be counter-intuitive; the 2D MBB topologies are in 2D, rather than 3D, and there are only 3 loading parameters as opposed to 4 or 5, both arguments suggesting a reduction in complexity in the training set, and thus a simpler, easier model for the network to learn. One possible explanation for this is due to the types of topologies produced in 2D compared to 3D. Observing Fig. 6.1, the topologies for the 2D MBB are highly complex; the examples showcase topologies made up of an intricate combination of large elements, and small, thin, multi-directional connection elements. The fifth topology in Fig. 6.1 has very thin and subtle elements in the bottom right corner. Likewise, the second topology has many thin, multi-directional diagonal elements, which the model has difficulty predicting. When compared to Fig. 6.4 and Fig. 6.10, the optimal topologies for the 3D cases are composed of primarily larger, repeatable elements rather than a complex combination of large and thin elements. Machine learning models are able to train better when there are many repeatable patterns present in the dataset. The results suggest that potentially, the high variety in the size and direction of the patterns in the 2D MBB training dataset have meant it has achieved the lowest in the accuracy scores with respect to the 3D examples.

Comparing the total average scores for the predicted *initial VM stress field* when compared to the ground truth, the 2D MBB scored the highest (99.29%, 0.018, 0.030), followed by 3D bridge (98.27%, 96.32%, 94.28%) and then the 3D cantilever (97.22%, 0.054, 0.078), for the three accuracy metrics (binary accuracy, mean absolute error, root mean square). Some comments regarding these results are as follows:

- For this output, the model was tasked with predicting an initial VM stress field over the whole domain. The 2D MBB performed the best out of all the test cases in this task. The second and third columns in Fig. 6.1 show the high similarity between the ground truth and the models predictions for the predicted initial VM stress field, and the final VM stress field. Despite the apparent complexity in the stress fields, the results indicate the model is able to learn structure within the VM stresses for the training data for the 2D MBB. For the 3D cases however, the prediction accuracy for the models decreased, and one explanation can be that the 'large, repeatable' elements that were a feature for the optimal topologies is no longer present for the

VM stress field. An inspection of Fig. 6.4 and Fig. 6.10 shows a full initial VM stress field with abrupt and highly varying values. The higher the complexity in the patterns, the more difficulty in the capacity for the neural network to learn structure and commonality in the data.

- One limitation of the proposed model is that the VM stress is normalized between 0 and 1 for all of the topologies. It is meant to be read as a guide for the *relative difference* in stress for a given load combination. It can give a designer an insight into the most highly stressed regions relative to the whole topology, but should not be considered an *absolute measure* of the stress, and the stresses between predicted topologies should not be compared with each other in terms of absolute magnitude.

Comparing the total average scores for the predicted *initial TorC stress field* when compared to the ground truth, the 2D MBB scored the highest (95.26%, 0.025, 0.048), followed by 3D bridge (88.36%, 0.047, 0.078) and then the 3D cantilever (82.15%, 0.090, 0.133), for the three accuracy metrics (binary accuracy, mean absolute error, root mean square). Some comments regarding these results are as follows:

- The test cases rank in the same order for the TorC as for the VM stresses. However the TorC accuracy is observed to be lower for all the accuracy metrics when compared with the VM stresses amongst all the test cases. One explanation for this can be the difficulty in the DL model in learning negative values for the inputs and outputs, which are present in the TorC dataset, and not present in the VM dataset. For example, both the multi-layer perceptron model and the decoder branch of the autoencoder contained linear activation functions for the TorC dataset, whilst VM model contained sigmoid functions. Linear activation functions are suited for mapping inputs to both negative and positive inputs, whilst sigmoid activation functions are useful for a non-linear mapping of inputs to an output between 0 and 1. This non-linearity can be useful in learning more complex representations in the dataset, and one potential explanation as to why the VM dataset performed better all round when compared to the TorC dataset.
- One accuracy metric that is particularly suitable for the TorC dataset is the binary accuracy metric. This is because, for the initial TorC stress field, it yields the percentage of elements predicted correctly as either compression or tension with respect to the total number of elements in the domain. The binarized final TorC field visually illustrates this metric.

Limitations. The proposed method is capable of predicting almost instantaneous predictions of optimal topologies, VM and TorC stresses at a high degree of accuracy. One

feature however, of trained DL models, is that they are only able to perform well on cases they have seen during training because they rely on patterns and relationships in the data that they have been exposed to. The types of loading parameters that are fed into the model, as well as the types of topologies that are predicted as outputs are designated a priori when creating the dataset and training the model. If for example the user wishes to predict a 2D MBB topology with a different volume fraction, this would not be possible, as the volume fraction was not a variable parameter introduced into the training dataset. This is an inherent feature of the models, they provide large benefits in terms of near instant accurate predictions, but the user must be aware that the models have a predefined mode of use.

Future work. The work contained in this thesis has opened up some exciting directions for future work:

- **Test cases with larger domains:** The test cases proposed within this thesis are of modest size and provide a solid proof of concept model for the proposed method. However, for there to be a more real-world applicability to this technology, test cases with larger domains could be explored.
- **Different training dataset sampling strategy:** The accuracy of the trained model depends highly on the robustness of the dataset. The LHS sampling strategy has proved effective in selecting a representative training dataset, however a future direction could be to explore how the training data set could be selected to achieve higher accuracy results.
- **Stress constraints:** the topology optimization problem could be extended to include stress constraints, that is, design the shape of the topology to limit the VM stresses, as well as minimize compliance.

Bibliography

- [1] M. P. Bendsøe and O. Sigmund. Material interpolation schemes in topology optimization. *Archive of Applied Mechanics*, 69:635–654, 1999.
- [2] A. Bejan. Constructal-theory network of conducting paths for cooling a heat generating volume. *International Journal of Heat and Mass Transfer*, 1997.
- [3] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S. Lazarov, and Ole Sigmund. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, nov 2010.
- [4] Federico Ferrari and Ole Sigmund. A new generation 99 line matlab code for compliance topology optimization and its extension to 3d. 62(4):2211–2228, May 2020.
- [5] A. G. M. Michell. The limits of economy of material in frame structures. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 8:589–597, 1904.
- [6] W. Prager and G. I. N. Rozvany. Optimization of structural geometry. *Dynamical Systems*, pages 265–293, 1977.
- [7] Keng-Tung Cheng and Niels Olhoff. An investigation concerning optimal design of solid elastic plates. *International Journal of Solids and Structures*, 17:305–323, 1981.
- [8] Martin Philip Bendsøe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71(2):197–224, nov 1988.
- [9] A. R. Diaz and M. P. Bendsøe. Shape optimization of structures for multiple loading conditions using a homogenization method. *Structural Optimization*, 4(1):17–22, mar 1992.
- [10] Katsuyuki Suzuki and Noboru Kikuchi. A homogenisation method for shape and topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 93:291–318, 1991.

- [11] Gregoire Allaire, Eric Bonnetier, Gilles Francfort, and Francois Jouve. Shape optimization by the homogenization method. *Numerische Mathematik*, 76:27–68, 1997.
- [12] M. P. Bendsøe. Optimal shape design as a material distribution problem. *Structural Optimization*, 1(4):193–202, dec 1989.
- [13] G. I. N. Rozvany, M. Zhou, and T. Birker. Generalised shape optimization without homogenization. *Structural optimization*, 4:250–252, 1992.
- [14] M. Zhou and G. I. N. Rozvany. The coc algorithm, part ii: Topological, geometrical and generalized shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 89:309–336, 1991.
- [15] M. P. Rossow and J. E. Taylor. A finite element method for the optimal design of variable thickness sheets. *AIAA*, 11(11), 1973.
- [16] G. I. N. Rozvany and M. Zhou. Applications of coc method in layout optimization. *Engineering Optimization in Design Processes*, pages 59–70, 1991.
- [17] T. Lewiński, M. Zhou, and G. I. N. Rozvany. Extended exact solutions for least-weight truss layouts—part i: Cantilever with a horizontal axis of symmetry. *International Journal of Mechanical Sciences*, 36(5):375–398, May 1994.
- [18] M. Zhou and G. I. N. Rozvany. A new discretized optimality criteria method in structural optimization. *AIAA*, 1992.
- [19] O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, apr 2001.
- [20] D. Tcherniak and O. Sigmund. A web-based topology optimization program. *Structural and Multidisciplinary Optimization*, 22:179–187, 2001.
- [21] Martin P. Bendsøe and Ole Sigmund. *Topology Optimization: Theory, Methods, and Applications*. Springer, 2003.
- [22] G. I. N. Rozvany and M. Zhou. Optimality criteria methods for large structural systems. *Advances in Design Optimization*, 1994.
- [23] O. Sigmund and J. Peterson. Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural Optimization*, 16:68–75, 1998.
- [24] Ole Sigmund. *Design of material structures using topology optimization*. PhD thesis, Department of Solid Mechanics, University of Denmark, 1994.

- [25] Martin Philip Bendsøe, Alejandro Díaz, and Noboru Kikuchi. Topology and generalized layout optimization of elastic structures. *NSSE*, pages 159–205, 1993.
- [26] Ole Sigmund. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, 33:401–424, 2007.
- [27] Cameron Talischi, Glaucio H. Paulino, Anderson Pereira, and Ivan F. M. Menezes. Polymesh: a general-purpose mesh generator for polygonal elements written in matlab. *Structural and Multidisciplinary Optimization*, 45:309–328, 2012.
- [28] Cameron Talischi, Glaucio H. Paulino, Anderson Pereira, and Ivan F. M. Menezes. Polytop: a matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Structural and Multidisciplinary Optimization*, 45:329–357, 2012.
- [29] Kai Liu and Andrés Tovar. An efficient 3d topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 50:1175–1196, 2014.
- [30] Krister Svanberg. The method of moving asymptotes—a new method for structural optimization. *International Journal of Numerical Methods in Engineering*, 24:359–373, 1987.
- [31] Nikos D. Lagaros, Nikos Vasileiou, and Georgios Kazakis. A c# code for solving 3d topology optimization problems using sap2000. *Optimization and Engineering*, 20:1–35, 2018.
- [32] Zhi Zeng and Fulei Ma. An efficient gradient projection method for structural topology optimization. *Advances in Engineering Software*, 2020.
- [33] Heng Chi, Anderson Pereira, Ivan F. M. Menezes, and Glaucio H. Paulino. Virtual element method (vem)-based topology optimization: an integrated framework. *Structural and Multidisciplinary Optimization*, 62:1089–1114, 2020.
- [34] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, and A. Russo. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23:199–214, 2013.
- [35] Oded Amir, Niels Aage, and Boyan Stefanov Lazarov. On multigrid-cg for efficient topology optimization. *Structural and Multidisciplinary Optimization*, 49:815–829, 2014.
- [36] Niels Aage, Erik Andreassen, and Boyan S. Lazarov. Topology optimization using

- petsc: An easy-to-use, fully parallel, open source topology optimization framework. *Structural and Multidisciplinary Optimization*, 51, August 2014.
- [37] Zhidong Brian Zhang, Osezua Ibhadode, Ali Bonakdar, and Ehsan Toyserkani. Topadd: a 2d/3d integrated topology optimization parallel-computing framework for arbitrary design domains. *Structural and Multidisciplinary Optimization*, 64, June 2021.
- [38] Liang Xia and Piotr Breitkopf. Design of materials using topology optimization and energy-based homogenization approach in matlab. *Structural and Multidisciplinary Optimization*, 52:1229–1241, 2015.
- [39] R. Tavakoli and Mohammad Mohseni. Alternating active-phase algorithm for multi-material topology optimization problems: A 115-line matlab implementation. *Structural and Multidisciplinary Optimization*, 49, April 2014.
- [40] Emily D. Sanders, Anderson Pereira, Miguel A. Aguiló, and Glaucio H. Paulino. Polymat: an efficient matlab code for multi-material topology optimization. *Structural and Multidisciplinary Optimization*, 58:2727–2759, 2018.
- [41] Federico Ferrari, Ole Sigmund, and James K. Guest. Topology optimization with linearized buckling criteria in 250 lines of matlab. *Structural and Multidisciplinary Optimization*, 63:3045–3066, April 2021.
- [42] Oliver Giraldo-Londoño and Glaucio H. Paulino. Polystress: a matlab implementation for local stress-constrained topology optimization using the augmented lagrangian method. *Structural and Multidisciplinary Optimization*, 63:2065–2097, 2021.
- [43] Hao Deng, Praveen S. Vulimiri, and Albert C. To. An efficient 146-line 3d sensitivity analysis code of stress-based topology optimization written in matlab. *Optimization and Engineering*, 2021.
- [44] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.
- [45] O. G. Selfridge. Pandemonium: A paradigm for learning. *Proc. Symposium on Mechanisation of Thought Processes*, pages 511–529, 1959.
- [46] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323, October 1986.
- [47] Christophe Garcia and Manolis Delakis. Convolutional face finder: a neural architecture for fast and robust face detection. *IEEE transaction on pattern analysis and machine intelligence*, 26(11):1408–1423, November 2004.

- [48] Dan Ciresan, Ueli Meier, Jonathan Masci, and Jurgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 2012.
- [49] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. *Proc. Advances in Neural Information Processing Systems*, pages 396–404, 1990.
- [50] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86:2278–2324, 1998.
- [51] Feng Ning, Damien Delhomme, Yann LeCun, Fabio Piano, Leon Bottou, and Paolo Emilio Barbano. Toward automatic phenotyping of developing embryos from videos. *IEEE Trans Image Process*, 14(9):1360–1371, September 2005.
- [52] Leon Bottou. Large-scale machine learning with stochastic gradient descent. *Proc. of COMPSTAT 2010*, 2010.
- [53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Proc. Advances in Neural Information Processing Systems*, 25:1090–1098, 2012.
- [54] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 15, 2011.
- [55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.
- [57] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. *CVF Conference on Computer Vision and Pattern Recognition*, 2017.
- [58] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Part of the Lecture Notes in Computer Science*, 9351, 2015.
- [59] Junhao Wen, Elina Thibeau-Sutre, Mauricio Diaz-Melo, Jorge Samper-Gonzalez, Alexandre Routier, Simona Bottani, Didier Dormont, Stanley Durrleman, Ninon

- Burgos, and Olivier Colliot. Convolutional neural networks for classification of alzheimer's disease: Overview and reproducible evaluation. *Med Image Anal.*, 2020.
- [60] Pierrick Coupé, Boris Mansencal, Michaël Clément, Rémi Giraud, Baudouin Denis de Senneville, Vinh-Thong Ta, Vincent Lepetit, and José V. Manjon. Assemblynet: A large ensemble of cnns for 3d whole brain mri segmentation. *NeuroImage*, 2020.
- [61] Kamal Choudhary, Brian DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok Choudhary, Ankit Agrawal, Simon J. L. Billinge, Elizabeth Holm, Shyue Ping Ong, and Chris Wolverton. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials*, 8, 2022.
- [62] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning*, 37:448–456, July 2015.
- [63] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.
- [64] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 1991.
- [65] Ivan Sosnovik and Ivan Oseledets. Neural networks for topology optimization. September 2017.
- [66] Qiyin Lin, Jun Hong, Zheng Liu, Baotong Li, and Jihong Wang. Investigation into the topology optimization for conductive heat transfer based on deep learning approach. *International Communications in Heat and Mass Transfer*, 97:103–109, 2018.
- [67] Nikos Ath. Kallioras, Georgios Kazakis, and Nikos D. Lagaros. Accelerated topology optimization by means of deep learning. *Structural and Multidisciplinary Optimization*, 62(3):1185–1212, mar 2020.
- [68] Saurabh Banga, Harsh Gehani, Sanket Bhilare, Sagar Patel, and Levent Kara. 3d topology optimization using convolutional neural networks. *Part of the Lecture Notes in Mechanical Engineering book series*, August 2018.
- [69] Fernando V. Senhora, Heng Chi, Yuyu Zhang, Lucia Mirabella, Tsz Ling Elaine Tang, and Glaucio H. Paulino. Machine learning for topology optimization: Physics-based

- learning through an independent training strategy. *Computer Methods in Applied Mechanics and Engineering*, 398:115116, aug 2022.
- [70] Diab W. Abueidda, Seid Koricv, and Nahil A. Sobh. Topology optimization of 2d structures with nonlinearities using deeplearning. *Computers and Structures*, 2020.
- [71] Hunter T. Kollmann, Diab W. Abueidda, Seid Koric, Erman Guleryuz, and Nahil A. Sobh. Deep learning for topology optimization of 2d metamaterials. *Materials & Design*, 196:109098, nov 2020.
- [72] Shuai Zheng, Zhenzhen He, and Honglei Liu. Generating three-dimensional structural topologies via a u-net convolutional neural network. *Thin-Walled Structures*, 2021.
- [73] Zeyu Zhang, Yu Li, Weien Zhou, Xiaoqian Chen, Wen Yao, and Yong Zhao. TONR: An exploration for a novel way combining neural network with topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 386:114083, dec 2021.
- [74] Sharad Rawat and M. H. Herman Shen. 2018.
- [75] S. Rawat and M. H. Shen. Application of adversarial networks for 3d structural topology optimization. *SAE Technical Paper*, 2019.
- [76] Sumudu Herath and Udith Haputhanthri. Topologically optimal design and failure prediction using conditional generative adversarial networks. *International Journal for Numerical Methods in Engineering*, sep 2021.
- [77] Yiquan Zhang, Bo Peng, Xiaoyi Zhou, Cheng Xiang, and Dalei Wang. A deep convolutional neural network for topology optimization with strong generalization ability. January 2019.
- [78] Cheng Xiang, Dalei Wang, Yue Pan, Airong Chen, Xiaoyi Zhou, and Yiquan Zhang. Accelerated topology optimization design of 3d structures based on deep learning. *Structural and Multidisciplinary Optimization*, 2022.
- [79] Jun Yan, Qi Zhang, Qi Xu, Zhirui Fan, Haijiang Li, Wei Sun, and Guangyuan Wang. Deep learning driven real time topology optimisation based on initial stress learning. *Advanced Engineering Informatics*, 51:101472, jan 2022.
- [80] Zhenguo Nie, Tong Lin, Haoliang Jiang, and Levent Burak Kara. Topologygan: Topology optimization using generative adversarial networks based on physical fields over the initial domain. *Journal of Mechanical Design*, 143:1–13, 2021.
- [81] Ruijin Cang, Hope Yao, and Yi Ren. One-shot generation of near-optimal topology through theory-driven machine learning. July 2018.

- [82] Cheng Xiang, Airon Chen, and Dalei Wang. Real-time stress-based topology optimization via deep learning. *Thin-Walled Structures*, 2022.
- [83] Hyogu Jeong, Jinshuai Bai, C. P. Batuwatta-Gamage, Charith Rathnayaka, Ying Zhou, and YuanTong Gu. A physics-informed neural network-based topology optimization (PINNTO) framework for structural optimization. *Engineering Structures*, 278:115484, mar 2023.
- [84] Yonggyun Yu, Taeil Hur, Jaeho Jung, and In Gwun Jang. Deep learning for determining a near-optimal topological design without any iteration. *Structural and Multidisciplinary Optimization*, 59(3):787–799, oct 2018.
- [85] Baotong Li, Congjia Huang, Xin Li, Shuai Zheng, and Jun Hong. Non-iterative structural topology optimization using deep learning. *Computer-Aided Design*, 2019.
- [86] Jaydeep Rade, Aditya Balu, Ethan Herron, Jay Pathak, Rishikesh Ranade, Soumik Sarkar, and Adarsh Krishnamurthy. Algorithmically-consistent deep learning frameworks for structural topology optimization. *Engineering Applications of Artificial Intelligence*, 106, 2021.
- [87] Mohammad Mahdi Behzadi and Horea T. Ilies. Real-time topology optimization in 3d via deep transfer learning. *Computer-Aided Design*, 135:103014, jun 2021.
- [88] Seunghye Lee, Hyunjoo Kim, Qui X. Lieu, and Jaehong Lee. Cnn-based image recognition for topology optimization. *Knowledge-Based Systems*, 198, June 2020.
- [89] Cheolwoong Kim, Jaewook Lee, and Jeonghoon Yoo. Machine learning-combined topology optimization for functionary graded composite structure design. *Computer Methods in Applied Mechanics and Engineering*, 387:114158, dec 2021.
- [90] Yusuke Takahashi, Yoshiro Suzuki, and Akira Todoroki. Convolutional neural network-based topology optimization (cnn-to) by estimating sensitivity of compliance from material distribution. *Journal of Computing and Information Science in Engineering (ASME)*, 2019.
- [91] Heng Chi, Yuyu Zhang, Tsz Ling Elaine Tang, Lucia Mirabella, Livio Dalloro, Le Song, and Glaucio H. Paulino. Universal machine learning for topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 375:112739, mar 2021.
- [92] Tinghao Guo, Danny J. Lohan, Ruijin Cang, Max Yi Ren, and James T. Allison. An indirect design representation for topology optimization using variational autoencoder and style transfer. jan 2018.

- [93] Akhil Singh, Vaibhav Jaiswal, Gaurav Joshi, Adith Sanjeeve, Shilpa Gite, and Ketan Kotecha. Neural style transfer: A critical review. *IEEE Access*, 9:131583–131613, 2021.
- [94] Erva Ulu, Rusheng Zhang, and Levent Burak Kara. A data-driven investigation and estimation of optimal topologies under variable loading configurations. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 4(2):61–72, aug 2015.
- [95] J. K. Guest, J. H. Prevost, and T. Belytschko. Achieving minimum length scale in topology optimization using nodal design variables and projection functions. *Int. J. Numer. Meth. Engng.*, 2004.
- [96] Stefan Engblom and Dimitar Lukarski. Fast matlab compatible sparse assembly on multicore computers. *Parallel Computing*, 56:1–17, 2016.
- [97] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.
- [98] Bane Sullivan and Alexander Kaszynski. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, May 2019.

A | Appendix

Summary of 2D MBB encoder architecture.

Layer	Output Shape	Activation function	Params
Input Layer	(120, 40, 1)	ReLU	0
Conv2D	(120, 40, 128)	ReLU	1280
Conv2D	(120, 40, 128)	ReLU	147584
MaxPooling2D	(60, 20, 128)	ReLU	0
BatchNormalization	(60, 20, 128)	ReLU	512
Conv2D	(60, 20, 64)	ReLU	73792
Conv2D	(60, 20, 64)	ReLU	36928
MaxPooling2D	(30, 10, 64)	ReLU	0
BatchNormalization	(30, 10, 64)	ReLU	256
Conv2D	(30, 10, 32)	ReLU	18464
Conv2D	(30, 10, 32)	ReLU	9248
MaxPooling2D	(15, 5, 32)	ReLU	0
BatchNormalization	(15, 5, 32)	ReLU	128
Conv2D	(15, 5, 32)	ReLU	9248
Conv2D	(15, 5, 32)	ReLU	9248
Reshape	(2400)	ReLU	0
Dense	(40)	Sigmoid (Linear*)	96040
Total	-	-	402728

Table A.1: 2D MBB encoder architecture.

Summary of 2D MBB decoder architecture.

Layer	Output Shape	activation function	Params
Input Layer	(40)	ReLU	0
Dense	(2400)	ReLU	1280
Reshape	(15, 5, 32)	ReLU	147584
BatchNormalization	(15, 5, 32)	ReLU	512
Conv2D	(15, 5, 32)	ReLU	73792
Conv2D	(15, 5, 32)	ReLU	36928
Conv2DTranspose	(30, 10, 32)	ReLU	0
BatchNormalization	(30, 10, 32)	ReLU	256
Conv2D	(30, 10, 32)	ReLU	18464
Conv2D	(30, 10, 32)	ReLU	9248
Conv2DTranspose	(60, 20, 64)	ReLU	0
BatchNormalization	(60, 20, 64)	ReLU	256
Conv2D	(60, 20, 64)	ReLU	18464
Conv2D	(60, 20, 64)	ReLU	9248
Conv2DTranspose	(120, 40, 128)	ReLU	0
BatchNormalization	(120, 40, 128)	ReLU	256
Conv2D	(120, 40, 128)	ReLU	18464
Conv2D	(120, 40, 128)	ReLU	9248
Conv2D	(120, 40, 1)	Sigmoid (Linear*)	9248
Total	-	-	608193

Table A.2: 2D MBB decoder architecture.

Summary of 2D MBB autoencoder architecture.

Input Layer	Output Shape	Params
Input Layer	(120, 40, 1)	0
Encoder	(40)	402728
Decoder	(120, 40, 1)	608193
Total	-	1010921

Table A.3: 2D MBB autoencoder architecture.

Summary of 3D cantilever encoder architecture

Layer	Output Shape	Activation function	Params
Input Layer	(24, 12, 12, 1)	ReLU	0
Conv3D	(24, 12, 12, 128)	ReLU	3584
Conv3D	(24, 12, 12, 128)	ReLU	442496
MaxPooling3D	(12, 6, 6, 128)	ReLU	0
BatchNormalization	(12, 6, 6, 128)	ReLU	512
Conv3D	(12, 6, 6, 64)	ReLU	221248
Conv3D	(12, 6, 6, 64)	ReLU	110656
MaxPooling3D	(6, 3, 3, 64)	ReLU	0
BatchNormalization	(6, 3, 3, 64)	ReLU	256
Conv3D	(6, 3, 3, 32)	ReLU	55328
Conv3D	(6, 3, 3, 32)	ReLU	27680
MaxPooling3D	(3, 2, 2, 32)	ReLU	0
BatchNormalization	(3, 2, 2, 32)	ReLU	128
Conv3D	(3, 2, 2, 32)	ReLU	27680
Conv3D	(3, 2, 2, 32)	ReLU	27680
Reshape	(384)	ReLU	0
Dense	(40)	Sigmoid	15400
Total	-	-	932648

Table A.4: 3D cantilever encoder architecture.

Summary of 3D cantilever decoder architecture

Layer	Output Shape	Activation function	Params
Input Layer	(40)	ReLU	0
Dense	(384)	ReLU	15744
Reshape	(3, 2, 2, 32)	ReLU	0
BatchNormalization	(3, 2, 2, 32)	ReLU	128
Conv3D	(3, 2, 2, 32)	ReLU	27680
Conv3D	(3, 2, 2, 32)	ReLU	27680
Conv3DTranspose	(6, 4, 4, 32)	ReLU	27680
BatchNormalization	(6, 4, 4, 32)	ReLU	128
Conv3D	(6, 4, 4, 32)	ReLU	27680
Conv3D	(6, 4, 4, 32)	ReLU	27680
Conv3DTranspose	(12, 8, 8, 64)	ReLU	55360
BatchNormalization	(12, 8, 8, 64)	ReLU	256
Conv3D	(12, 8, 8, 64)	ReLU	110656
Conv3D	(12, 8, 8, 64)	ReLU	110656
Conv3DTranspose	(24, 16, 16, 128)	ReLU	221312
BatchNormalization	(24, 16, 16, 128)	ReLU	512
Conv3D	(24, 16, 16, 128)	ReLU	442496
Conv3D	(24, 16, 16, 128)	ReLU	442496
Conv3D	(24, 16, 16, 1)	Sigmoid	3457
Cropping 3D	(24, 12, 12, 1)	-	0
Total	-	-	1541601

Table A.5: 3D cantilever decoder architecture.

Summary of 3D Cantilever autoencoder architecture

Input Layer	Output Shape	Params
Input Layer	(24, 12, 12, 1)	0
Encoder	(40)	932648
Decoder	(24, 12, 12, 1)	1541601
Total	-	2474249

Table A.6: 3D cantilever autoencoder architecture.

Summary of 3D bridge encoder architecture.

Layer	Output Shape	Activation function	Params
Input Layer	(60, 20, 4, 1)	ReLU	0
Conv3D	(60, 20, 4, 128)	ReLU	3584
Conv3D	(60, 20, 4, 128)	ReLU	442496
MaxPooling3D	(30, 10, 2, 128)	ReLU	0
BatchNormalization	(30, 10, 2, 128)	ReLU	512
Conv3D	(30, 10, 2, 64)	ReLU	221248
Conv3D	(30, 10, 2, 64)	ReLU	110656
MaxPooling3D	(15, 5, 1, 64)	ReLU	0
BatchNormalization	(15, 5, 1, 64)	ReLU	256
Conv3D	(15, 5, 1, 32)	ReLU	55328
Conv3D	(15, 5, 1, 32)	ReLU	27680
MaxPooling3D	(8, 3, 1, 32)	ReLU	0
BatchNormalization	(8, 3, 1, 32)	ReLU	128
Conv3D	(8, 3, 1, 32)	ReLU	27680
Conv3D	(8, 3, 1, 32)	ReLU	27680
Reshape	(768)	ReLU	0
Dense	(40)	Sigmoid	30760
Total	-	-	948008

Table A.7: 3D bridge encoder architecture.

Summary of 3D bridge decoder architecture

Layer	Output Shape	Activation function	Params
Input Layer	(40)	ReLU	0
Dense	(768)	ReLU	31488
Reshape	(8, 3, 1, 32)	ReLU	0
BatchNormalization	(8, 3, 1, 32)	ReLU	128
Conv3D	(8, 3, 1, 32)	ReLU	27680
Conv3D	(8, 3, 1, 32)	ReLU	27680
Conv3DTranspose	(16, 6, 2, 32)	ReLU	27680
BatchNormalization	(16, 6, 2, 32)	ReLU	128
Conv3D	(16, 6, 2, 32)	ReLU	27680
Conv3D	(16, 6, 2, 32)	ReLU	27680
Conv3DTranspose	(32, 12, 4, 64)	ReLU	55360
BatchNormalization	(32, 12, 4, 64)	ReLU	256
Conv3D	(32, 12, 4, 64)	ReLU	110656
Conv3D	(32, 12, 4, 64)	ReLU	110656
Conv3DTranspose	(64, 24, 8, 128)	ReLU	221312
BatchNormalization	(64, 24, 8, 128)	ReLU	512
Conv3D	(64, 24, 8, 128)	ReLU	442496
Conv3D	(64, 24, 8, 128)	ReLU	442496
Conv3D	(64, 24, 8, 1)	Sigmoid	3457
Cropping 3D	(60, 20, 8, 1)	-	0
Total	-	-	1557345

Table A.8: 3D bridge decoder architecture.

Summary of 3D bridge autoencoder architecture.

Input Layer	Output Shape	Params
Input Layer	(24, 12, 12, 1)	0
Encoder	(40)	948008
Decoder	(24, 12, 12, 1)	1557345
Total	-	2505355

Table A.9: 3D bridge autoencoder architecture.

List of Figures

2.1	Penalization relationship in the stiffness-density graph.	5
2.2	Interpretation of the penalization parameter on the basis of fabrication costs.	6
2.3	a. A SIMP Solution the MBB beam problem presented by Rozvany and Zhou (1991) b. the exact analytical truss solution presented by Lewinski et al. (1994).	7
2.4	MBB beam discretized into polygonal elements using 'Polytop'.	8
2.5	Periodic repetition in the local microstructure of multiscale materials.	10
2.6	Relationship between Artificial Intelligence, Machine learning and Deep learning. [44]	12
2.7	Difference between classical programming and machine learning [44].	12
2.8	A simplified diagram of a perceptron	13
2.9	A generic representation of a neural network.	13
2.10	Example of CNN architecture for image classification.	14
2.11	Example of a U-Net architecture.	17
2.12	ML Architecture used in Sosnoviks and Oseledets work in 2017 [65].	18
2.13	ML Architecture used in Lin et al. [66].	19
2.14	U-Net Architecture used in Zheng et al. (2021) [72].	20
2.15	Input tensor fields used in the network proposed by Zhang et al. (2019) [77].	21
2.16	Generator and discriminator GAN network taken from Yu et al. (2018) [84]	23
2.17	VAE with style transfer in Guo et al. (2018) [92].	25
3.1	Example of the input and output of the topology optimization problem for the 2D MBB	28
4.1	Numbering and connectivity for the 2D finite element formulation.	34
4.2	Flowchart of main program ' <i>Main_MBB.m</i> ' calling upon function ' <i>top88_MBB</i> ' to produce topologies for each load condition.	35
4.3	Allowable force vector positions in the MBB case.	35
4.4	Decomposition of unit force vector in x- and y- components for use in the MATLAB program.	36

4.5	Illustration of LHS sampling method in 2 dimensions	37
4.6	Element-wise multiplication between Von Mises stress field with the corresponding optimal topology.	39
4.7	Element-wise multiplication between σ_{TorC} stress field and the corresponding optimal topology.	40
4.8	(a) Nodes and DOFs numbering for $z=0$ (b) elements numbering for the elements between $z=0$ and $z=1$	41
4.9	(a) Isometric view of the design domain with element 1 highlighted in blue (b) detailed view of element 1 showing node numbers and DOFs.	41
4.10	Spherical coordinate system: inclination θ , azimuth φ , and radius r	43
4.11	Projecting the Von Mises stresses onto the optimal topology using the masking procedure for two different cases (a) and (b).	45
4.12	Projecting the σ_{TorC} stresses onto the optimal topology using masking procedure for two different cases (a) and (b).	46
4.13	3D (half-)bridge domain: dimensions and solid, void, and allowable regions.	47
4.14	Mirroring the half-bridge domain to create the full bridge for different test cases (a) and (b).	48
4.15	3D Bridge case: schematic representation of the 4 parameters adopted to create unique topologies.	48
4.16	Projecting the VM stresses onto the optimal topology using the masking procedure for two different cases (a) and (b).	49
4.17	Projecting the σ_{TorC} stresses onto the optimal topology using the masking procedure for two different cases (a) and (b).	50
5.1	Topology optimization approaches: (a) traditional methodology; (b) proposed deep learning model pipeline.	51
5.2	Scheme of the proposed deep learning pipeline.	53
5.3	Basic AE architecture consisting of encoding and decoding branches	54
5.4	An example of a response map produced by applying a 3x3 filter over an input image (adapted from [97]).	55
5.5	The spatial hierarchy of features that forms the image of a cat. Smaller textures and patterns combine to form larger recognisable details such as an ear, or a nose, which combine into the higher level concept of "cat".[97]	56
5.6	Adopted architecture of the AE for the 2D MBB.	57
5.7	Graphs of the ReLU and sigmoid activation functions.	57
5.8	Graph of the linear activation function.	59
5.9	AE architecture for the 3D cantilever case.	60

5.10	AE architecture for the 3D bridge case.	61
5.11	Encoding the training dataset using the encoder branch of the trained AE.	61
5.12	A generalised schematic of the MLP architecture.	62
5.13	Procedure for testing the proposed DL pipeline.	64
6.1	Accuracy comparison of the ground truth and prediction for six optimal topologies for the 2D MBB case.	68
6.2	Comparison between ground truth and predictions for VM stresses for the 2D MBB case.	70
6.3	Comparison between ground truth and predictions for TorC zones for the 2D MBB case.	71
6.4	Accuracy comparison of the ground truth and prediction for six optimal topologies for the 3D cantilever case.	74
6.5	Comparison between ground truth and predictions for VM stresses for the 3D cantilever case.	76
6.6	Comparison between ground truth and predictions for TorC zones for the 3D cantilever case.	77
6.7	PyVista render of three randomly predicted topologies for (a) optimal topology (b) final VM stress and (c) final TorC stress.	78
6.8	PyVista render of a predicted final VM stress for cutoff values (a) 0.1 (b) 0.2 and (c) 0.4.	79
6.9	PyVista render of a predicted final TorC stress for smoothing values (a) 0.0 (b) 10.1 and (c) 50.9.	79
6.10	Accuracy comparison of the ground truth and prediction for six optimal topologies for the 3D bridge case.	81
6.11	Comparison between ground truth and predictions for VM stresses for the 3D bridge case.	82
6.12	Comparison between ground truth and predictions for TorC zones for the 3D bridge case.	83
6.13	PyVista render of three randomly predicted topologies for (a) optimal topology (b) final VM stress and (c) final TorC stress.	84
6.14	PyVista render of a predicted final VM stress for cutoff values (a) 0.1 (b) 0.2 and (c) 0.4.	85

List of Tables

- 6.1 Comparison of average computational run-time between SIMP algorithm and proposed methodology for the 2D MBB case. 66
- 6.2 Comparison of average BA, MAE and RMS accuracy metrics for the predicted topologies when compared to the ground truth for the 2D MBB case. 69
- 6.3 Comparison of average computational run-time between SIMP algorithm and proposed methodology for 3D cantilever. 72
- 6.4 Comparison of average BA, MAE and RMS accuracy metrics for the predicted topologies when compared to the ground truth for the 3D cantilever case. 75
- 6.5 Comparison of average computational run-time between SIMP algorithm and proposed methodology for 3D bridge. 80
- 6.6 Comparison of average BA, MAE and RMS accuracy metrics for the predicted topologies when compared to the ground truth for the 3D bridge case. 84

- A.1 2D MBB encoder architecture. 104
- A.2 2D MBB decoder architecture. 105
- A.3 2D MBB autoencoder architecture. 106
- A.4 3D cantilever encoder architecture. 106
- A.5 3D cantilever decoder architecture. 107
- A.6 3D cantilever autoencoder architecture. 108
- A.7 3D bridge encoder architecture. 108
- A.8 3D bridge decoder architecture. 109
- A.9 3D bridge autoencoder architecture. 110

Acknowledgements

This thesis is dedicated to my parents, whose continual support and love have allowed my passions for my studies to flourish. And to Anna, my love, you are a shining light.

