Image to Image Translation for Topology Optimization

Master Thesis

June, 2022

Madav Kaushik Sridhar

Maersk McKinney Moller Institute

SDU-Robotics

Image to Image Translation for Topology Optimization

Master Thesis

June, 2022

By

Madav Kaushik Sridhar

Published by: SDU, SDU-Robotics, Campusvej 55, 5230 Odense M Denmark

www.sdu.dk/mmmi

Approval

I hereby declare that the presented work and results in this thesis reflects my own work.

It is assumed that the reader has a basic knowledge in Engineering Mechanics, Artificial Intelligence and Machine Learning.

Madav Kaushik Sridhar - 491322

1st June, 2022

Odense, Denmark

Abstract

Topology optimization has been an essential tool in material design. There are various numerical methods proposed to solve this problem, namely SIMP [1], and MMA [2]. However, these conventional methods provide promising results at the cost of computational time and resources. This work proposes a deep learning approach, where we tackle topology optimization as an image-to-image translation problem. This is achieved by using generative models, such as Conditional Generative Adversarial networks(CGAN)[3]. These models consist of a generator and a discriminator network competing against each other. The generated tries to generate new images with the input domain, and the discriminator tries to distinguish between real (image from the training) and fake (generated image). We also explore other possible architectures for the generator model by implementing (a). UNet, (b). ResNet, and (c). ResUNet architecture. The generated topology images are analyzed with a suitable metric to measure the system's performance. The results show a 98.9 % decrease in computation time compared to the conventional methods. The significant reduction of computational time is an added advantage for the designer. The source code is available on GitHub[1].

[1] https://github.com/masri20/Image-to-Image-Translation-for-Topology-Optimization

Acknowledgements

It is my genuine pleasure to express my deepest gratitude to

[Aljaz Kramberger], [Associate Professor], [Supervisor]
For guidance and support during the thesis project.

[Rasmus Aagaard]
This document template was heavily inspired from Rasmus Aagaard.

I would also like to thank my friends and family for their extended support during the thesis project.

Contents

1 Introduction

1.1 Motivation

The recent developments in design and manufacturing technologies are at an all-time high. With the easy accessibility to designing tools, almost anyone with a little bit of knowledge can develop parts for prototyping or even the final product. This feature is complemented nicely by the advancement in additive manufacturing. Now one can come up with a concept design, 3D-print the part, and test it in their project, bypassing all the tedious fabrication processes (milling, welding, sanding, etc.).

Furthermore, additive manufacturing processes have paved the way for creative artists to design structures that are too difficult or close to impossible to develop with conventional manufacturing methods. Parallelly, there has been a vast development in the Lean Manufacturing solutions, especially with flexible production systems. The ability to have less downtime to change the machines and tools for production of one product to another (Routing Flexibility) is a massive reduction in effort, cost, and time to manufacture new products.

A significant part of this downtime is due to design failure. In this case, we must look at downtime as not just the time at which there is no production output. One must also account for the time consumed for the design and fabrication tools. Therefore, by helping with the design and fabrication, we would reduce the downtime considerably.

SDU Robotics have proposed several simulation and optimization-based solutions for lean manufacturing, especially regarding gripping and grasping applications [4] [5] [6] [7]. On the other hand, the growth in artificial intelligence has been exponential. Mainly, deep neural networks have paved the way to advanced optimization solutions. As a result, we have deep network models solving problems in various fields of science and engineering for classification and regression problems. Besides these tasks, a deep learning network could also be used to create new data. The Generative Adversarial Networks (GAN) are a special branch of deep learning models that can generate new unseen data. It consists of two networks, a generator, and a discriminator, which compete against each other. The generator tries to create data close to reality, and the discriminator tries to

figure out if the data is real or the fake (generated). There are two major types of GAN, conditional and unconditional. The only difference is that unconditional GAN is considered an unsupervised learning method, and conditional GAN is considered supervised.

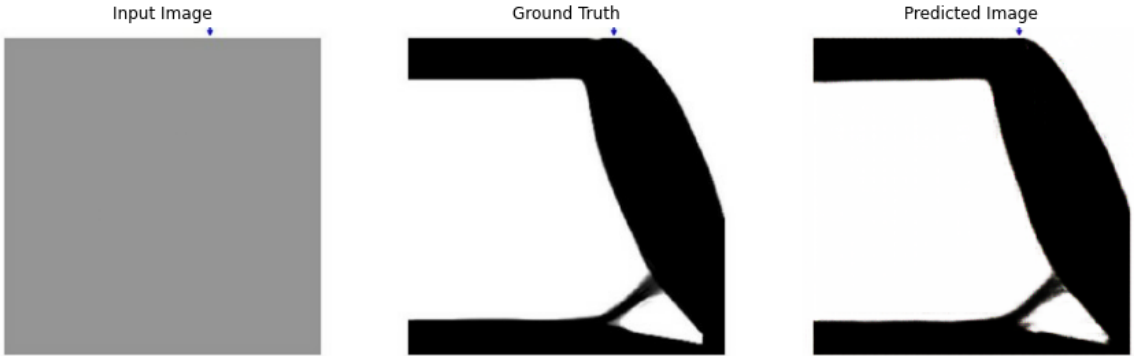


Figure 1.1: Input Image is the initial problem, ground truth is the optimized output using conventional iteration-based solver and the predicted image is the image generated by the conditional GAN

1.2 Problem Statement

With that being the base of this work, the proposed solution tackles topology optimization with a deep learning network as a generative task. Using conditional GAN, we can optimize material distribution by using image-to-image translation. We provide an image pair that consists of the initial topology optimization problem and the solution.

In the figure.1.1 the image on the left is the input image, which is the initial structure that needs to be optimized based on the boundary and load conditions. The ground truth or the optimized solution obtained using a conventional iteration-based topology optimizer is in the middle. The image to the right is the optimized structure generated by the conditional GAN. Essentially, the network learns the mapping from the initial problem (input image in fig.1.1) to the optimized solution (ground truth in fig.1.1) and then tries to generate an optimized topology when a new problem is given. The central part of the study was to test and compare the idea of image-to-image translation based on three network architectures for the generator network, (a) UNet architecture, (b) ResNet architecture, and (c) ResUNet architecture. In section 2 we will look at the past work done in both the fields of topology optimization and neural network. Later in section 3, we will introduce the methods used in this work, and chapter 4 demonstrates the implementation of the three architectures. Chapter 5 shows the results generated by the three networks, and a suitable metric is used to compare the performance. Finally, in chapter 6 we conclude with future work.

2 Background Study

2.1 Topology Optimization

The idea of topology optimization was first introduced in [1], where they explain about the homogenization based approach. But the computation of optimal micro-structures with their orientation is close to impossible, especially for non-compliance problem. Moreover, it lacks the practicality because it can't be build since there is no fixed or defined length-scale. This led to the next breakthrough in this field. By introducing the Power Law in [8] and later in [9], where the design domain is discretized by assuming each element has constant material properties. The topology optimization was also solved as a dual approach for large scale compliance problem [10].

Other methods include evolution based solutions like [11] and [12], which basically removes materials at places with lower stress and add material to places with higher stress based on a number of heuristic and intuition. Compared to the previous method these are easy to implement and understand, especially with compliance minimization problem. Authors in [13] and [14] have introduced an genetic algorithm approach by acquiring the possible connections among a pre-defined point and then determine the cross-sectional area of each connection element.

These are all gradient based search algorithms. We also have some interesting past works with a stochastic approach. [15] uses Simulated Annealing (SA) to increase the material use subjected to the increase in stress. [16] along with [17] used an Ant Colony Optimization (ACO) and finite element analysis (FEA) to compute topology optimization for both 2D and 3D structures. [18] uses yet another bio-inspired algorithm to compute the topology optimization. As mentioned in [19], stochastic has 4 major advantages over the gradient based search algorithm, better optima, ability to handle discrete designs, gradient free and efficient to parallelize.

Although, both stochastic and gradient based methods require high computational power for larger structures. Hence, there are several techniques proposed to help reduce the computation cost and time. In [20] they propose the idea of reducing the number of design variable so that we can cut down on computation time. And in [21], where by using K-

Means algorithm, the authors claim that, we can reduce the computation time by reducing the dimensions of the design variables. Even 3D optimization was possible using parallel computing with domain decomposition as discussed in [22].

This amount of research helps material optimization in several fields, like aerospace [23], [24] and [25], thermal properties[26], and bio-mechanics [27] [28] just to name a few.

2.2 Neural Nets and Deep Learning

The field of artificial neural networks is growing exponentially. Irrespective of the filed, neural networks and deep learning have contributed to some extraordinary solutions like, [29], [30] and [31] in physics and chemistry respectively. Even day to day problems like this, [32]. Most of the deep learning models are based on CNN which was introduced in LeNet-5 [33] to recognize handwritten zip-codes. The next breakthrough was the UNet introduced in [34], which is a deep neural network with CNN Encoder-Decoder architecture. The key characteristics of UNet is the skip-connections which enable the network to pass important features from the encoder block to the decoder block for better regeneration. Followed by that, ResNet was introduced in [35], where the CNN architecture was complemented by residual blocks which was proposed to eliminate exploding or vanishing gradient during back-propagation.

These networks were essential in tasks such as classification, segmentation and regression. But they weren't able to perform well for a generative task. Generative Adversarial Networks (GAN) was introduced in [36] which consist of two networks, a generator and a discriminator, where they try to compete against each-other. After training, we can discard the discriminator and only the generator is used to generate new data. GANs' have been used in so many applications since then, like [37], where they try to recover photo-realistic texture using SR-GAN with two loss functions, adversarial loss and content loss.

The problem was that the generator produces new data from random noise vector and this was almost like unsupervised learning. It was hard to use for problems with labels based classification or regression. Conditional Generative Adversarial Networks (CGAN) [3] were introduce to support the short-comings of GAN. CGAN also constitutes of a generator network and a discriminator network but the difference is that, we input the conditioned data to both generator and the discriminator. This gained much popularity as it was extensively used in translation problems such as text to image [38] and image to im-

age [39](Pix2Pix). Image to image translation was much appreciated by the community as they came up with several applications , like [40] where they reconstruct images of fingerprints to analyze the danger of representation attack. In [41] they experiment with edge to image translation using Pix2Pix [39].

2.3 Topology Optimization with Neural Nets

One of the fist notable work in topology optimization with deep learning and neural network was [42] where a UNet [34] architecture was used in coordination with SIMP method to predict topology optimization of conductive heat transfer. A Similar approach was take in [43], where the SIMP method [44] was used for few iterations and the intermediary image was passed through the proposed network for further optimization. These method cut down the computation time compared to conventional methods mentioned above.

As research in CNN provided more methods, [45] used a super Resolution CNN (SRCNN) which consist of 4 processes, refinement, path extraction and representation, non-linear mapping and image reconstruction in order to produce high quality images. Another deep learning method was introduced [46] where for each iteration the generative simulated annealing (GSA) uses a deep neural network (DNN) to evaluate the objective function. There have been some study done with topology optimization for 3D profile. In [47], the authors used a 3D Encoder-Decoder CNN to enhance 3D optimization.

There have been some interesting use of generative models as well. By using a Conditional Wasserstein Generative Adversarial Network (CWGAN) [48] with word embedding for the volume fraction label, [49] was able to replicate conventional topology optimization. On the other hand, [50] uses a variational auto-encoder (VAC) for sub-optimal structure and use a super resolution - GAN (SR-GAN) to improve the quality. ReCoNN [51] was a network architecture with a WGAN and auto-encoder, where they generated new samples from the latent space of the auto-encoder. A recent approach was to use CGAN for Topology optimization in civil and architecture domain [52]. Their TopoGAN was proposed to deal with active and passive elements in the design.

3 Methods

3.1 Topology Optimization

In a general sense, topology is related to objects under continuous deformation. It is the same understanding in mechanical and architectural design as well. We study an object under constant force and boundary conditions, and try to optimize the material distribution without compromising the structural integrity. In simple terms, we try to reduce the object's overall volume such that it would not deform or break when a certain amount of force is applied to it. However, we consider the location of the applied force and specific boundary conditions (fixed contact or moving contact) while computing the optimized structure.

Topology optimization has many applications, from aerospace to automobile [53] and designs in civil engineering, bio-mechanics and material science [25] [54] [27], where even a volume reduction of a couple of grams has a higher impact to the performance of the system. This creates a bridge between creative and innovative designs, which are both aesthetically pleasing and structurally strong structures, especially in architectural designs, as mentioned in [55].

3.1.1 Minimum Compliance

Problem Statement

We define a body occupying a domain Ω in $\mathbb{R}^2$. This body is subjected to a force f and boundary traction t. The objective function is defined as the problem of finding the optimal value of stiffness tensor $E_{ijkl}(x)$ which varies over the domain. By using Energy Bi-linear Form (the internal work energy at equilibrium u and displacement v) along with linearized strain and load linear form, the minimum compliance becomes,

$$\begin{aligned} &\min_{u \in U, E} l(u) \\ &\text{s.t. } a_E(u, v) = l(v), \qquad \text{for all } v \in U, \\ &\quad E \in \mathsf{E}_{\mathsf{ad}}. \end{aligned} \tag{3.1}$$

$where,$

$$a(u,v) = \int_{\Omega} E_{ijkl}(x)\varepsilon_{ij}(u)\varepsilon_{kl}(v)\mathrm{d}\Omega \quad \Rightarrow \text{Energy Bi-Linear Form} \tag{3.2}$$

with linearized strain,

$$\varepsilon_{ij}(u) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{3.3}$$

and load linear form,

$$l(u) = \int_{\Omega} fu \ \mathrm{d}\Omega + \int_{\Gamma_T} tu \ \mathrm{d}s \tag{3.4}$$

In equation ($3.1^{[56]}$), a_E denotes that the system is in a bi-linear form which depends on the design variable. u and v are equilibrium and displacement respectively where U denotes the space of kinematically admissible displacement fields. E_{ad} is a collection of all stiffness tensors that attain the material property of a given isotropic material from an unknown set of materials (Ω^{mat}), as mentioned in [56].

The fig.(3.1) shows the iterative approach to reach optimal design, which is the most common method to compute the optimized structure [43] [57][56].

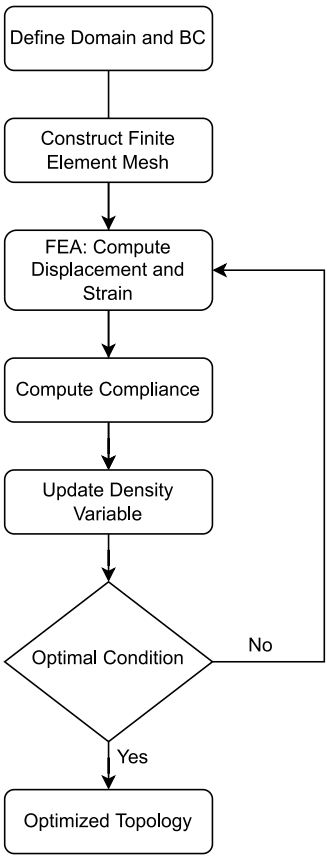


Figure 3.1: Minimum Compliance

When computing, one can reach the optimal design based on two conditions. The most common way is to mention the number of iterations. This is entirely user-dependent and can be determined by trial and error. The other condition is when no considerable amount of material is removed compared to the previous step of the iteration.

3.1.2 Construct Finite Element Mesh

The typical approach to solve equation(3.1) is to discretize. The outcome will be,

$$\begin{aligned} &\min_{\mathbf{u}, E_e} \mathbf{f}^T \mathbf{u} \\ &\text{s.t. } : \mathbf{K}(E_e)\,\mathbf{u} = \mathbf{f} \\ &\quad E_e \in E_{\text{ad}} \end{aligned} \tag{3.5}$$

In equation ($3.5^{[56]}$), u is the displacement vector and f is the load vector. As shown in equation ($3.6^{[56]}$), K being the stiffness matrix which depends on elemental stiffness E_e for $e = 1, \cdots, N$ and K_e is the element stiffness matrix (on a global level).

$$\mathbf{K} = \sum_{e=1}^{N} \mathbf{K}_e(E_e) \tag{3.6}$$

Displacement and strain using FEA

Once the design domain has been discretized, the focus will be on computing displacement and strain at each node. This is done in order to determine which of these nodes should contain material and which of these nodes must be void (no material). To give some visual sense, we would translate to images which would have black and white representation to show material present and void respectively, This causes a binary problem, where the solution would be to provide a continuum between the two and thereby try to nudge the solution in either direction. One of the common methods is by using Solid Isotropic Material Penalization (SIMP).

$$E_{ijkl} = 1_{\Omega^{mat}} E^0_{ijkl}, \quad 1_{\Omega^{mat}} = \begin{cases} 1 & \text{if} \quad x \in \Omega mat \\ 0 & \text{if} \quad x \in \Omega / \Omega mat \end{cases} \tag{3.7}$$

$$\int_{\Omega} 1_{\Omega^{mat}} d\Omega = Vol(\Omega^{mat}) \leq V \tag{3.8}$$

This is achieved by limiting the spatial extension of domain Ω, we are trying to find an optimal subset Ωmat of material nodes (material redistribution). Equation(3.8) is limit on the amount of material available V this is called the volume fraction.

Solid Isotropic Material Penalization (SIMP)

This penalization method was introduced [1] to ensure the solution converges. Element density $\rho(x)$ is allowed to be any value between 0 and 1, i.e., $0 \leq \rho(x) \leq 1$. We would want it to converge to either 0 or 1 and be penalized if it has an intermediary density value. With that in mind, we write it mathematically as.

$$E_i jkl(x_i) = E^0_{ijkl} \quad \rho(x_i)^p, \qquad p > 1, \quad (0 \leq x_i \leq 1) \tag{3.9}$$

Where, in equation(3.9), $E_i(x_i)$ is element stiffness tensor, $\rho(x_i)$ is element density, and p is the penalization term. With $\rho(x_i)$ between 0 and 1 and a p value greater than one, it will add more weight to the term. From a stiffness standpoint, this makes it uneconomical for a vertex to have an intermediary density value between 0 and 1.

How to choose p-value?

The estimation of the p-value based on initial volume is important for the solution to converge faster. We would like the solution to be in black and white compliance design, i.e., ones and zeros, because intermediary density for material does not make practical sense. So, we shall assume that we will be designing for a component made of a particular material with a fixed density across the body; we want to have either 100% material present with the same density or 0% material present. One must choose a p-value that will force the intermediary values to converge. Below are the equations (3.10) and (3.11), for estimating the p-value for 2-D and 3-D use cases respectively, where v is Poisson ratio which is considered to be 0.3, according to [58].

$$p \geq max \left[\frac{2}{(1-v)}, \frac{4}{(1+v)} \right] \quad \text{(in 2-D)} \tag{3.10}$$

$$p \geq max \left[15\frac{(1-v)}{(7-5v)}, \frac{3(1-v)}{2(1+v)} \right] \quad \text{(in 3-D)} \tag{3.11}$$

Convergence

In order to converge the form of minimum compliance, equation(3.1) has to be in a continuum setting as shown in(3.12) where all the variables remind the same but we introduce a lower bound on the elemental density ρ_{min} to prevent any singularities, as mentioned in [56]

$$\begin{aligned} &\min_{u \in U, \rho} l(u) \\ &\text{s.t. } a_E(u, v) = l(v), \quad \text{for all } v \in U \\ &\quad E_{ijkl}(x) = \rho(x)^p E^0_{ijkl} \\ &\quad \int_\Omega \rho(x) \mathsf{d}\Omega \leq V; \quad 0 < \rho_{\mathsf{min}} \leq \rho \leq 1 \end{aligned} \tag{3.12}$$

Update - Material add/remove

The constraints in equation (3.12) gets multiplied with Lagrange Multipliers, Λ, $\lambda^-(x)$ and $\lambda^+(x)$ for the sizing of the design variable ρ, as mentioned in [56], we get,

$$\begin{aligned} \mathcal{L} = l(u) - \left\{ a_E(u, \bar{u}) - l(\bar{u}) \right\} + \Lambda \left(\int_\Omega \rho(x) d\Omega - V \right) + \\ \int_\Omega \lambda^+(x)(\rho(x) - 1) d\Omega + \int_\Omega \lambda^-(x)(\rho_{min} - \rho(x)) d\Omega \end{aligned} \tag{3.13}$$

In equation(3.13), $l(u)$ is the load linear form, $\bar{u}$ is the Lagrange multiplier for equilibrium(where $\bar{u} \in U$). $(a_E(u, \bar{u}) - l(\bar{u}))$ is the internal work energy, $\Lambda(\int_\Omega \rho(x) d\Omega - V)$ is the element density and volumetric constraint. The $\lambda^+(x)(\rho(x) - 1) d\Omega$ term tells us if we have material already present. When $\rho(x) = 1$, the term goes away and we don't add more material. The $\int_\Omega \lambda^-(x)(\rho_{min} - \rho(x)) d\Omega$ term tells us if there is very minimum material present. When $\rho_{min} = \rho(x)$ it means that we don't remove more material.

3.1.3 Method of Moving Asymptotes (MMA)

Topology optimization can be solved using different numerical methods. One of the most commonly used method is SIMP, as discussed earlier. But this study has used Method of moving asymptotes (MMA) to generate training data which is discussed in chapter 4. This method was introduced in [2], which falls under a family of convex approximation, most commonly used for structural optimization. MMA can handle all types of constraints and also non-linear programming. It is easy to implement and a common approach for non-linear optimization, as mentioned in [58].It formulates a local model at each iteration, which is similar in other methods as well, but when presented with large number of design variable, it becomes computationally expensive.

Consider the following,

$P : minimize$

$$f_0(x) \qquad (x \in R^n)$$

$$s.t :$$

$$f_i(x) \leq \hat{f}_i, \qquad \text{for} \quad i = 1, \cdots, m$$

$$\underline{x}_j \leq x_j \leq \overline{x}_j, \qquad \text{for} \quad j = 1, \cdots, n$$

where, in the above equations, $x = (x_1, \cdots, x_n)^T$ is the design variable vector, $f_0(x)$ is the objective function(structural weight), $f_i(x) \leq \hat{f}_i$ are the boundary conditions and the lower and upper limits are given by $\underline{x}_j$ and $\overline{x}_j$, as mentioned in [2].

In simper terms, $f_i^{(k)}$ can be computed by linearizing f_i in variables of equation(3.14), depending on the signs of the derivatives of f_i at $x^{(k)}$

$$\frac{1}{(x_j - L_j)} \quad \text{or} \quad \frac{1}{(U_j - x_j)} \tag{3.14}$$

where, k denotes the current iteration, L_j and U_j are referred to as 'moving asymptotes'[2] because they change in-between iterations. The problem is defined similar to SIMP method where we assume constant material properties within each element in the design domain. By using the power law, the modulus of each element elasticity is modeled as a function of relative density. As mentioned in equation(3.9), $\rho(x_i)$ is the element density, E_i is the element elasticity, E_0 is the modulus of elasticity of the base material and p is the penalization power which nudges the equation to a binary solution (black is material present and white if material is removed). The penalization power p is chosen based on the Poisson's ration v from the equation (3.10) and (3.11) for 2D domain and for 3D domain respectively.

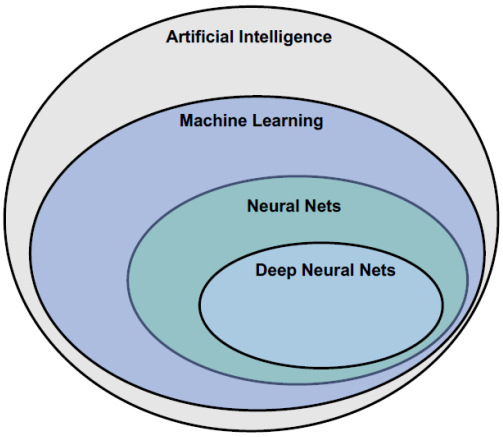
Instead of the convergence method mentioned in section 3.1.2, we use the above mentioned MMA method for convergence. This is achieved by moving the lower L_j and upper U_j asymptotes closer or far from the design variable x_j. Quicker convergence can be achieved by choosing a optimal value for the moving asymptotes, as mentioned in [2]. Since we are working in 2D domain, the model will be discretized in to horizontal elements n_{elx} along the x-axis and vertical elements n_{ely} along the y-axis. Both together

gives us the total number of design variables as show in equation (3.15)

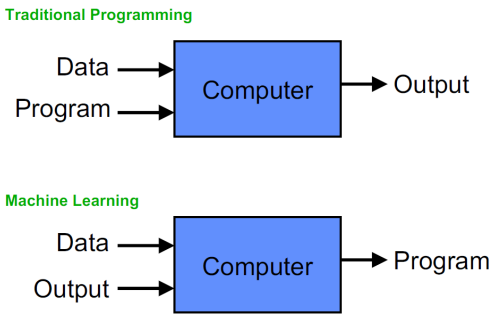
$$n = n_{elx} * n_{ely} \tag{3.15}$$

3.2 Artificial Intelligence (AI)

Artificial Intelligence has been the buzzword in several industries for the past few years. There is a large quantity of research output every year. To understand AI, one should look at this in two contexts, narrow AI and general AI, according to [59]. Narrow AI is the development of systems that can perform and augment natural intelligence classification, perception, reasoning, learning, and/or acting, whereas general AI is complete autonomy. We have had the concept of artificial intelligence since 1950 [61].There are several rea-



(a) Subsets of Artificial Intelligence [60]



(b) Traditional vs Machine Learning[60]

sons behind the delay in advanced development of this field. Some reasons mentioned would be the combined development in big data, computing power and algorithms. These together made AI a fast-developing field.

Machine learning is a subset of artificial intelligence that uses previous experiences to learn the task. It is based on statistical models and algorithms to analyze the patterns in the data. In simple terms, we train the computers to program themselves. Fig. 3.2b shows the difference between traditional programming and machine learning. A machine learning algorithm can perform tasks like: classification, regression, prediction, and clustering.

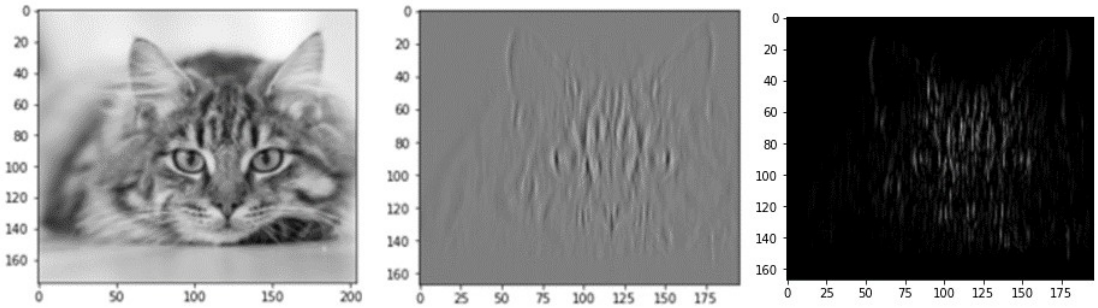


Figure 3.3: Left to right: Input Image, Image after convolution layer and Image after convolution layer with ReLU activation [63]

3.2.1 Convolutional Neural Network (CNN)

Convolutional Neural Networks are a branch of deep learning algorithms mainly used to analyze structured arrays of data like images. They are commonly used in computer vision-based tasks such as image classification. This is because they are good at recognizing patterns in images, such as lines, edges and other subtle features. For example, the first CNN was used to classify handwritten digits [62]. This works with a convolution filter(kernels) which goes through the image, looking for patterns similar to the kernels themselves. By performing a cross-correlation operation, the kernel returns a larger value, if it is identical and returns a smaller value, if it is not identical. This is how the CNN detects horizontal and vertical lines, curves and corners. Generally, the first convolution layer has these basic pattern detectors, and the algorithm typically learns it during training. The later layers extract features from these previous layers to identify the image better. We pass the output through an activation layer to improve the performance of the convolution layers. This helps the convolution network not consolidate and condense the input to single matrix multiplication. In addition, the activation layers add a bit of non-linearity, which helps the higher layers to learn more abstract concepts of the image. The most common activation layers are sigmoid and rectified linear units (ReLU). As we can see from the fig.3.3, the activation layer makes the image more sharper by making the vertical feature lines more prominent. This makes the higher layers recognize more complex patterns.

Convolution and activation layers combined with pooling or downsampling layer, and together it is considered as a Convolution Block. Usually, the last layers are flattening layer followed by a fully connected layer and a soft-max layer.

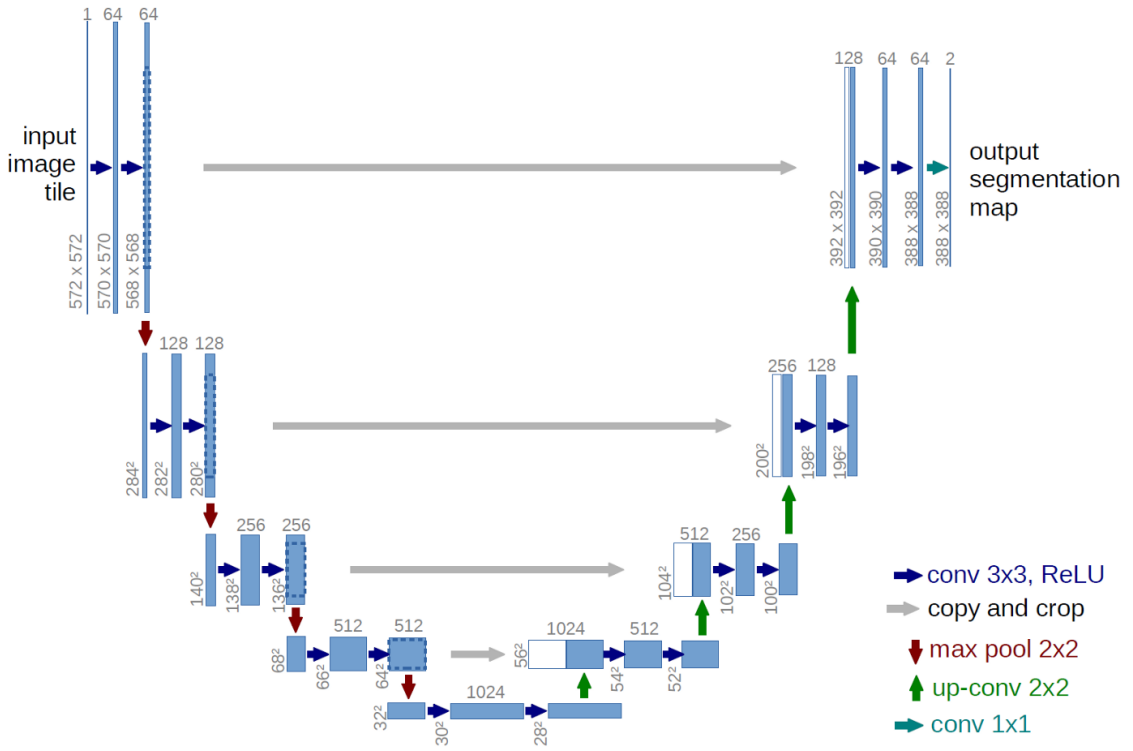


Figure 3.4: U-Net Architecture - Since the network has a U-shaped architecture it was named ”UNet” [34]

UNet

UNet was first introduced by *O.Ronneberger Et al*.[34]. It was specifically design for medical image segmentation. Image segmentation is a process of separating different elements of an image. Since its inception, U-Nets have been used for various applications, even for topology optimization [43] and [27]. In a convolutional neural network (CNN) the network tries to learn the feature mapping of the image and convert the image into a vector which is used for classification. But in the case of U-Net, we would need to reconstruct the image from this feature vector as we want an segmented image as the output. In order to do that, we use the already learnt feature mapping obtained during contraction to expand the feature vector back to an image. This helps the network to preserve the structural integrity of the image. Fig.3.4 shows the general architecture of U-Net.

ResNet

ResNets are another set of networks which are widely used in the field of computer vision. As an improvement to CNN, ResNets were introduced by *K.He Et al*.[35]. While adding more layers to CNN would improve the performance, it comes with a cost. During back propagation, the network sufferer’s from degradation making it hard to train deeper networks. This is not a result of over-fitting rather it is because of vanishing or exploding

gradients. This problem was solved by introducing residual blocks with identity connection. The identity connection sets up and alternative path for the gradients to pass to deeper layer of the network during back propagation. Moreover, the network learns the identity function thereby making sure that the higher layer perform better than the lower layers. Fig.4.7 shows the identity connection.

3.2.2 Generative Adversarial Networks(GAN)

Generative Adversarial Networks are yet another branch of deep learning algorithms where it can learn from the data and generate new unseen data with the same characteristics. For example, if we present pictures of dogs to the network and it would be able to generate new images with the same key features and characteristics of a dog.

Introduced by *I. Goodfellow Et al*.[36], it has an unique architecture where we have two separate neural network, the generator and the discriminator, working against each other. Generator learns to generate fake data from a random noise and the discriminator tries to distinguish between real and fake. Going back to the same example of generating dog images, fig.3.5 shows that the generator is trying to produce images which are realistic and the job of the discriminator is to identify if the image presented is from the training set (real) or a generated image (fake). The adversarial part is when the discriminator identifies the unrealistic data generated by the generator and thereby penalizes the generator.

Generator

The generator network takes random noise as an input to generate a sample data (fake data). The random noise is a vector of D-dimensions from a Gaussian distribution, which used to seed the generative process of the generator. The vector forms a compressed representation of the data distribution because the points in the multidimensional vector space corresponds to the points in the problem domain. The vector space (latent vector) consist of important feature of the domain but they are not directly visible.

Discriminator

Discriminator acts as a binary classification network. It takes a sample from the domain (real/generated) and predicts if the presented image is real/fake. The real sample comes from the training dataset and the generated sample comes from the generator. Once we have trained our model, then we can eliminate the discriminator and only use the generator as it has effectively learnt to extract features from the data presented.

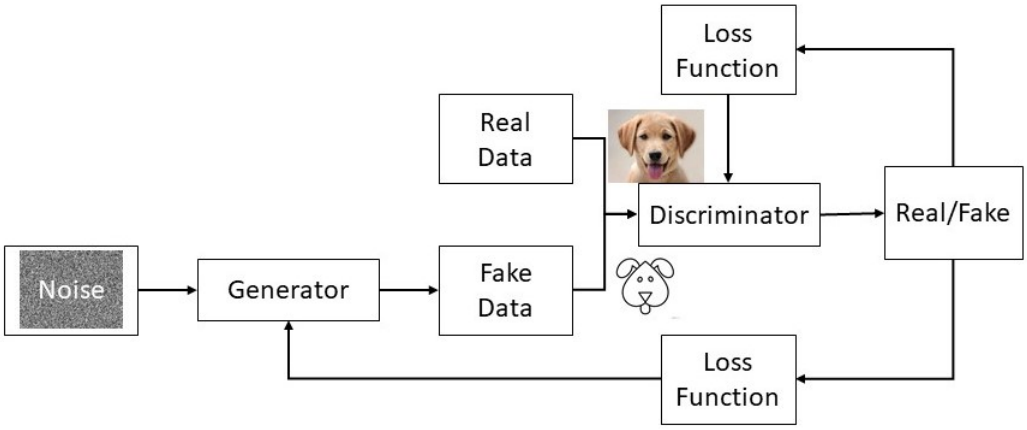


Figure 3.5: Workflow of Generative Adversarial Networks

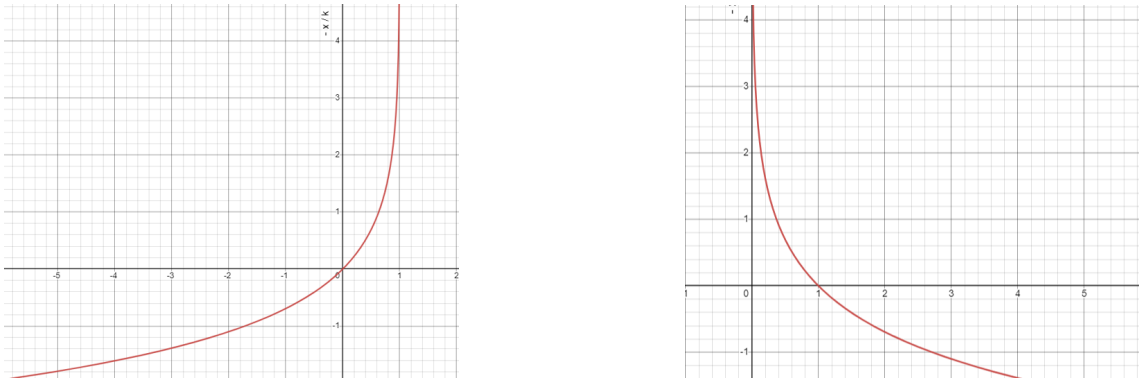
Loss Function

The loss/error is a function used to evaluate how the model is performing. If the error is large, then the model is not doing well. Both the generator and the discriminator use a log-loss function. We assume that we have a label 1 and the network predicts 0.1. The prediction is far off from the actual label. In this case, we want the loss function to return a larger value. On the other-hand, when the prediction is 0.9, which is closer to the actual label, we want the loss function to return a smaller value. So the way to translate is to use a negative logarithmic function on the predicted value, as shown in equation.(3.16) and fig.3.6b. This is how the loss is calculated for the generator network.

$$G_{Error} = -\ln(\text{prediction}) \tag{3.16}$$

$$D_{Error} = -\ln(1 - \text{prediction}) \tag{3.17}$$

On the contrary, we assume we have a label 0 and the network predicts 0.9. The prediction is far off from the actual label. In this case, we want the loss function to return a larger value. And on the other-hand, when the prediction is 0.1, which is closer to the actual label, we want the loss function to return a smaller value. The way to translate this is to use a negative logarithmic function on complement of the predicted values, as shown in equation(3.17). This is how the loss is calculated for the discriminator network. Once we have computed the error based on the prediction done by the network, now we can back-propagate by taking the derivative of the errors' based on all the weights using the chain rule, which tells us how much to update the weights such that we minimize the error. The error is minimized by gradient decent, where we plot the error with respect to



(a) $y = -\ln(1-x)$ (b) $y = -\ln(x)$

Figure 3.6: Assuming x-axis denotes the number of epochs and the y-axis denotes the loss, (a) and (b).shows a sample plot of the discriminator loss and generator loss, respectively.

the parameters and then compute the gradient along the direction of growth. Then we move in the negative direction of gradient in order to find new parameters.

3.2.3 Conditional Generative Adversarial Networks

Conditional Generative Adversarial Networks (CGAN) are an extension of GAN, where the generator is trained to generate new data in the domain of the input data. The input data can be a label/class value. Considering the previous example of generating images dogs, we could assume the label/class to be the breed of the dog. So we train the network with different breeds of the dog and ask the generator to generate images of dog, of a particular breed.

CGAN was introduced by *M.Mirza Et al*.[3]. It has a similar architecture as GAN, but the only difference is that the generator does not generate from the latent noise vector z alone, but it also receives the label of the input data y. The discriminator is tasked with detecting if the data is from training (real) x or generated (fake). In GAN, the parameters for the generator G are changed to minimize $\log(1 - D(G(z)))$, and the parameters for the discriminator are changed to maximize $\log D(x)$, which is similar to a two-player min-max game. The value function $V(G, D)$ is given by equation (3.18)[36]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{3.18}$$

Where in equation (3.18), $D(\boldsymbol{x})$ is the discriminator's estimate of the probability, that the

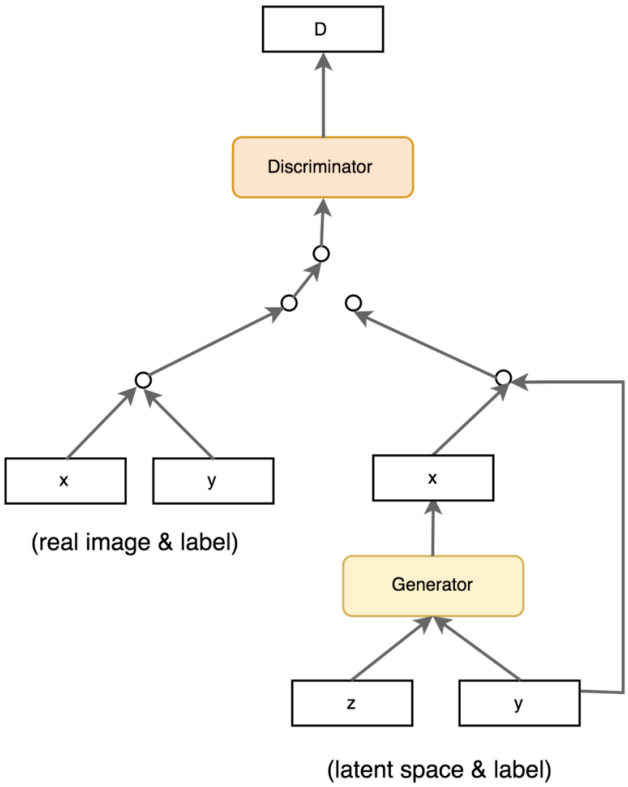


Figure 3.7: Work flow of CGAN[64]

real data instance x is real and $D(G(z))$ is the discriminator's estimate of the probability, that the fake instance $G(z)$ is real. But for CGAN, both the generator and discriminator are conditioned by feeding label y of the data as an additional input layer. This causes the loss function to be changed to accommodated the label of input data y. The value function of CGAN is given by equation (3.19)[3]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x} \mid \boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z} \mid \boldsymbol{y})))] \quad (3.19)$$

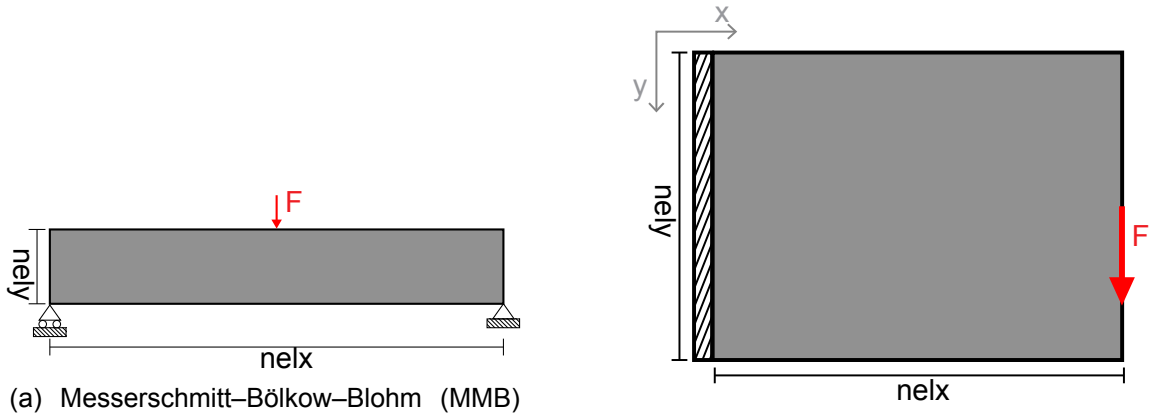
Where in equation (3.19), $\mathbb{E}_{\boldsymbol{x}}$ and $\mathbb{E}_{\boldsymbol{z}}$ are the expected value over all the training data (real) and latent noise vector, respectively. $D(\boldsymbol{x} \mid \boldsymbol{y})$ is the discriminator's estimate of the probability, that the real data instance x with label y is real and $D(G(\boldsymbol{z} \mid \boldsymbol{y}))$ is the discriminator's estimate of the probability, that the fake instance $G(z)$ with label y is real. Fig. 3.7 show the basic work flow of the CGAN architecture. This work uses a CGAN to perform image-to-image translation. This was introduced by *P. Isola Et al*. [39] where the network tried to learn the mapping of an input image to a target image. The implementation of these above mentioned methods are discussed in the following chapter.

4 Implementation

In this chapter we will look at the implementation of the proposed methods. At first we will look at how the dataset was generated. Then we look at the network architecture of the methods.

4.1 Dataset Generation

For any data-driven approach, the essential step is to gather the training dataset. There are different ways to do so. If one is performing unsupervised learning, the data need not have to be conditioned. Data is conditioned when it contains a label attached to the data. On the other hand, one is expected to use conditioned data for supervised learning. For the model to learn critical features and produce good results, it is crucial to have adequately labeled data. It poses to be the most challenging task for specific applications. Usually, we would switch to synthetic data collection methods in these cases. Using advanced mathematics and computer programming, one can synthesize data through simulation. Another interesting method is to use generative AI models to generate synthetic data [65]. In this case, with only a few labeled data, the generative AI model will be able to synthesize a larger dataset. In this study, a python library TOPOPT[66] was used to



(a) Messerschmitt–Bölkow–Blohm (MMB) beam)

(b) Cantilever beam

Figure 4.1: Image (a) and (b) shows the structures used [66].

generate the training data. TOPOPT computes the topology optimization for structural compliance problems, time-harmonic load problems, and von Mises Stress problems. As

discussed in chapter 3, this library uses an iterative-based MMA solver based on the NLopt [67], which is an open-source library for non-linear optimization.

For the case study of the proposed solution, Messerschmitt–Bölkow–Blohm (MMB) beam and a Cantilever beam were considered. Fig.4.1 shows the structures in the design domain. One can note the boundary conditions and where force is applied (red arrow). The structure itself is shaded in gray to indicate that it contains some material.

(a) Unfiltered (Checkerboard Pattern) (b) Filtered - Density-Based

Figure 4.2: Image (a) shows the unfiltered result and image (b) shows the filtered result of topology optimization [68].

In order to compute the optimal topology for these structures, one has to translate the boundary and load conditions from the design domain. For that, we need to set the following parameters.

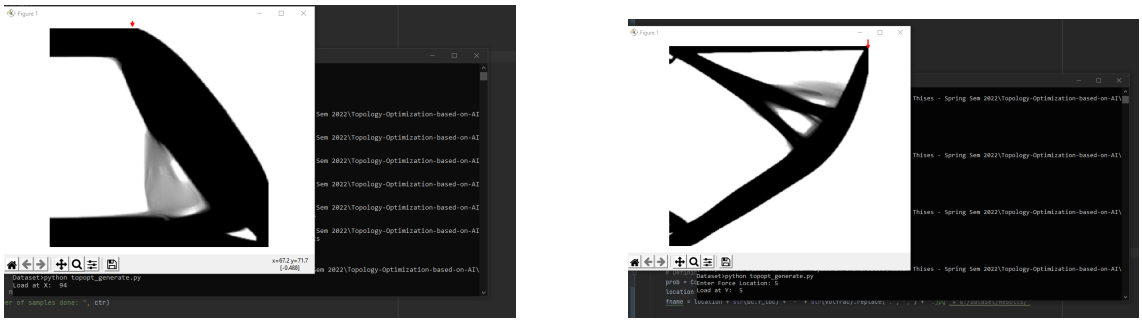
- $nelx$ Number of elements on the x-axis(horizontal)
- $nely$ Number of elements on the y-axis(vertical)
- f Location at which force is applied.
- pen Penalization power (p)
- $volfrac$ Volume Fraction
- $rmin$ Density Filter radius

As mentioned before, $nelx$ and $nely$ together will give us the total number of design variables. The pen is the penalization power p mentioned in 3.9. Volume fraction says the amount of material permitted. Finally, we apply a density-based filter to smooth out the optimized topology. The fig.4.2 shows the difference between the standard output and the filtered output. When these parameters are set, one can use TOPOPT to generate the optimized structure for the mentioned boundary conditions. Since its an iterative-based

Parameter	Value
Structure	MMB and Cantilever
Problem	Structural Compliance
nelx	250
nely	250
volfrac	[0.2, 0.5)
pen	3.0
rmin	1.5
force	Randomized within [250, 250]

Table 4.1: Design Parameters used for the generation of the dataset.

optimizer, the computation time is directly proportional to the total number of design variables. This means that the more the design variable, the more time it takes for the solver to converge to a solution.



(a) MMB Structure

(b) Cantilever Structure

Figure 4.3: Image shows the generation of training data for (a) MMB structures and (b) Cantilever structure.

Since we wanted to generate a huge batch of data for the training of our model, the entire process of generating optimized topology for the given boundary and load condition was automated. Table.4.1 show the diversity of a select few parameters in order to create some variety in the training dataset. During initial trials, different values were tried for n_{elx} and n_{ely}. The problem was that when the number of elements was increased to get a bigger structure, the computation time increased drastically, and the program demanded computational resources.

On the flip side, when the number of elements was decreased, the generated image quality was not suitable for training. So based on these experiments, the number of elements was chosen as shown in the table.4.1. The volume fraction was sampled from a uniform distribution with the limits mentioned in the table.4.1. Although sampling from a normal

distribution was mentioned in [43], uniform distribution ensures an equally likely outcome within the specified range. The penalty p was chosen based on the popular choices in [27] and [43]. The location of force applied was randomly drawn along the x-axis for the MMB structure and along the y-axis for the cantilever structure. The magnitude of the force applied was 1 N. The dataset was generated by randomizing a few of these parameters, as shown in fig.4.3.

Evaluating the network's learning skill on the training set will be biased as the network has already seen this data. We would not be able to notice if the model learns from the dataset or remembers the dataset. So, it is common to split the dataset into training and validation sets. Then, the network learns from the training dataset and uses the validation dataset to validate the performance. The validation set is a collection of held-out samples from the training set. Evaluating the networks on the validation set will be unbiased as we ensure that the data present in the validation set is different from the training set. The same concept was considered while generating the dataset.

(a) MMB Structure - Image Pair

(b) Cantilever Structure - Image Pair

Figure 4.4: Image shows the Image-pair (a) MMB structures and (b) Cantilever structure.

In order for us to teach the mapping from one image to the other, it is recommended [39] that we gather the data as shown in fig.4.4, as image pairs. The image on the left of the pair is assumed as the base material with a red arrow that marks the location with force applied. This will be the input image. The image on the right of the pair is the result we want to achieve, the optimized topology. This will be the expected output or the ground truth during training. The generated images are resized to 512x256 with a fixed aspect ratio to train the network.

During the initial experimentation for the MMB structure, since there was no hard-set rule on the amount of data to collect, [39] was used as a reference. In their experimentation with aerial to map image to image translation, they used 1096 image pairs for training

Structure	Train	Validation
MMB	1472	1000
Cantilever	836	100

Table 4.2: Dataset size for each structure

and 1098 for validation. Once the initial experimentation was conclusive, the Cantilever structure's training and validation data were reduced, as shown in table.4.2.

4.2 Network Architecture

Now that we have gathered all the data, we need to train our network. Let us look at the architecture of the neural network. These network architectures were implemented on Keras [69] with Tensorflow [70] as the back-end.

4.2.1 Data Preprocessing

To train the network, one must preprocess the data, perform data augmentation and prepare the input pipeline.

Loading the image

From the previous section, we can recall the database contains a training and validation set. Each of these sets has several image pairs with a size of 512x256 pixels. This was to ensure the correct input image was paired with the right target image so that there was no mishap during training. So now we have to load this image pair and decode it to a unit8 tensor. The decoded image pair is now split into an input and target image size 256x256 pixels.

Data Augmentation

It is a common practice to augment data before training. This will ensure more diversity in the dataset. Not just that, by performing data augmentation, the network becomes more versatile. The recommended [39] data augmentation techniques for the training set were resize to 286x286 pixels, random crop to 256x256 pixels, random mirroring, and finally normalize to [-1, 1]. And for the validation set, it was recommended only to perform random mirroring and normalize the data to [-1, 1]. However, the recommended data augmentation techniques would not fit this application, as we might lose essential information, especially the location of the load. However, the image was normalized to [-1, 1] because when unnormalized data is fed to an activation function, it could quickly get stuck in a flat region and not learn the features.

Input Pipeline

Now that our dataset is preprocessed, we need to convert the entire training and validation set into its respective dataset. $tf.data$ is an API that allows us to build the input pipeline so we can perform certain operations. Shuffling the dataset is highly recommended. As a result, the network will avoid learning the training order and prevent bias during training. Defining the batch size is crucial as it is considered one of the hyper-parameters. Batch size represents the number of samples given to the network after which the weights are updated. It is usually recommended to be 32 based on standard practices. However, after initial experimentation, the batch size was set to 1. The advantage of having a mini-batch is that the overall training would consume less memory as the network uses fewer samples to train.

Furthermore, as mentioned in [39], "when we have a batch size equal to one and perform batch normalization, it is known as instance normalization." This proves to be an effective approach for image generation tasks [71]. Although other implementations of this image-to-image translation have removed the batch norm in the bottleneck layers, the activation in these layers is zeroed by the batch norm operation, virtually making the innermost layer skipped.

4.2.2 Generator Architecture

Section 3.2.2 gave a brief introduction to a generator network. Here we shall look at the various generator architecture implemented.

UNet based Generator

The architecture used for the generator is a modified UNet [34] architecture. The UNet consists of an input layer, an encoder layer, a decoder layer, the bottleneck layer, and a final output layer, as shown in fig.4.5.

Generally, a GAN or a cGAN generator will generate an image from a random point in the latent space. Instead, the generator takes the image as a tensor of size $(1, 256, 256, 3)$ through the input layer, where the first index indicates the batch size. The second and the third index denotes the size of the input image, and the last index is the number of channels in the input image. Next are the encoder layers are nothing but a downsampler. It consists of a 2D convolution layer (Conv2D) followed by batch normalization and a leaky ReLU. The encoder reduces the spatial dimensions by doubling the number of feature channels for every downsampling step. This will reduce the input image to a latent vector.

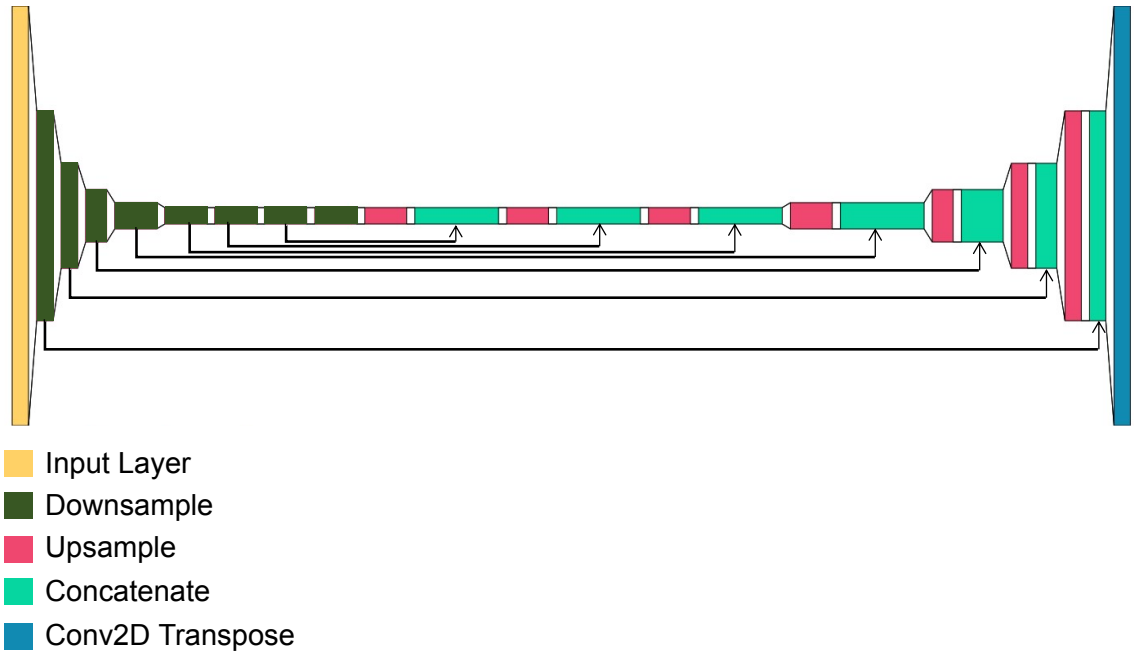


Figure 4.5: UNet generator architecture with skip connections

Now the decoder layers behave as an upsampler. It consists of a 2D transposed convolution layer (Conv2DTranspose) followed by batch normalization, dropout with a rate of 50%(applied to the first three blocks), and ReLU. The decoder tried to regenerate the image from the latent vector of the encoder by expanding the feature channels. The encoder shares higher spatial information with the respective decoder layer of the same dimension. This is called ”Skip Connection”.

ResNet based Generator

The other architecture used for the generator is a modified ResNet architecture from [35] [72]. The ResNet consists of an input layer, an encoder layer, several residual layers, a decoder layer, and a final output layer, as shown in fig.4.6.

Just as in the above-mentioned UNet architecture 4.2.2, the input layer reminds unchanged, which takes an image tensor of size (1, 256, 256, 3). Then it passes through a 2D reflection padding layer. During initial implementation, this layer was skipped, and the resulting output image after training had several artifacts. Based on the recommendation in [72], the reflection padding layer was added, which helped in removing the artifacts. Next, it passes through a 7x7 Conv2D, followed by batch normalization and a ReLU activation.

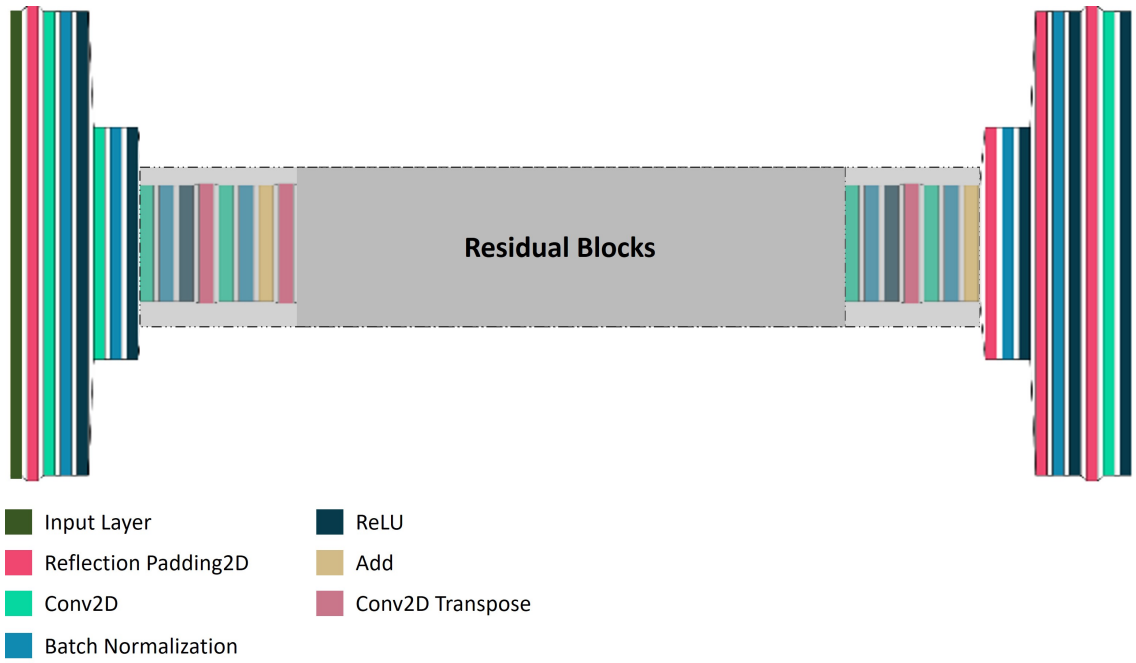


Figure 4.6: ResNet generator architecture

We then downsample two steps to lower spatial dimensions with the encoder block consisting of Conv2D followed by batch normalization and a ReLU activation function. This reduces the spatial dimensions by doubling the number of feature channels for every downsampling step. The successive layers are a series of residual blocks consisting of two sets of, Conv2D layers followed by batch normalization and ReLU activation function. The key feature of a residual block is identity connection. As shown in the fig.4.7, this acts as a shortcut connection to the later stages in a residual block. The identity connection ensures that the higher layers of the network do not perform any worse than the lower layers by learning the identity function.

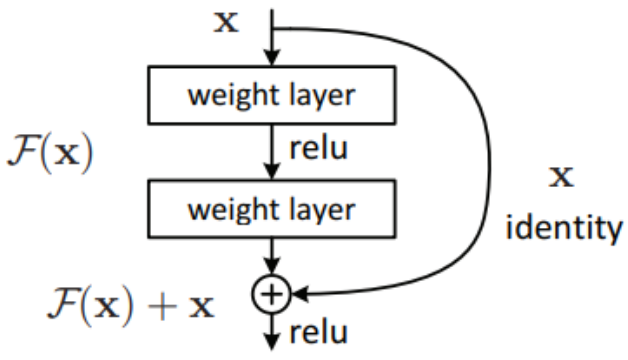


Figure 4.7: Identity connection where the weight layers block consists of Conv2D, batch normalization and ReLU activation function

The next layer is the upsample or the decoder block consisting of a Conv2DTranspose followed by batch normalization and a ReLU activation function. In order to match the dimensions of the input image tensor, we upsample by two steps. Finally, we have a Conv2D to scale up the number of channels to match the input image tensor.

ResUNet based Generator

The last architecture used for the generator is a modified ResUNet architecture from [73]. The ResUNet consists of an input layer, an encoder layer, a bridge, a decoder layer, and a final output layer, as shown in fig.4.8.

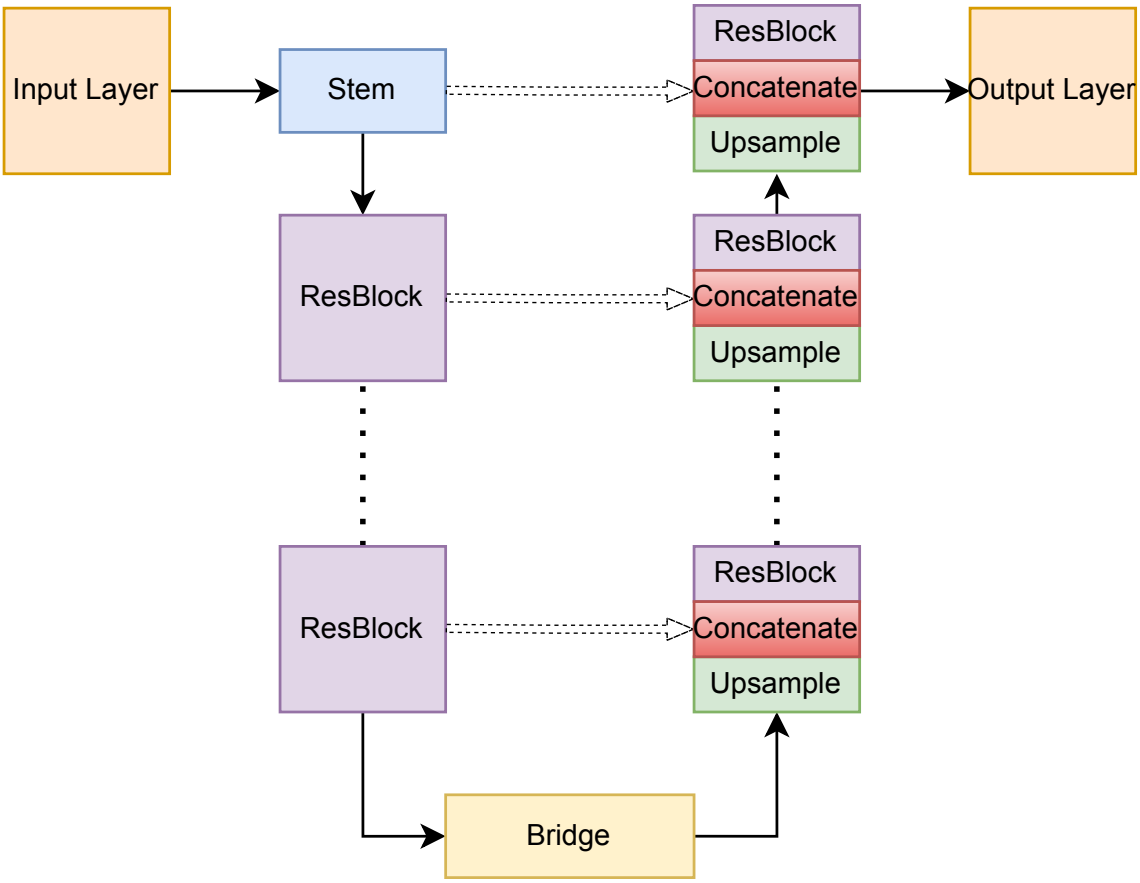


Figure 4.8: ReUNet generator architecture with skip connections

The ResUNet consists of some basic building blocks used in the entire architecture. The first building block is the residual block, consisting of 2 sets of batch normalization, ReLU activation function, and Conv2D layer, as shown in fig.4.10. Similar to the previous method, the residual block has the identity connection, which connects the input to the residual block to the output of the residual block. The next building block is the convolution block, consisting of batch normalization, ReLU activation function, and Conv2D layer.

Like the other two methods, the generator takes the image as a tensor of size $(1, 256, 256, 3)$. Then it passes through a block called a stem. The stem block consists of a Conv2D layer followed by a batch normalization ReLU activation layer and a final Conv2D layer, as shown in fig.4.9. This also acts as a residual block as it has an identity connection that maps the input of the stem directly to the output via Conv2D and a batch normalization layer.

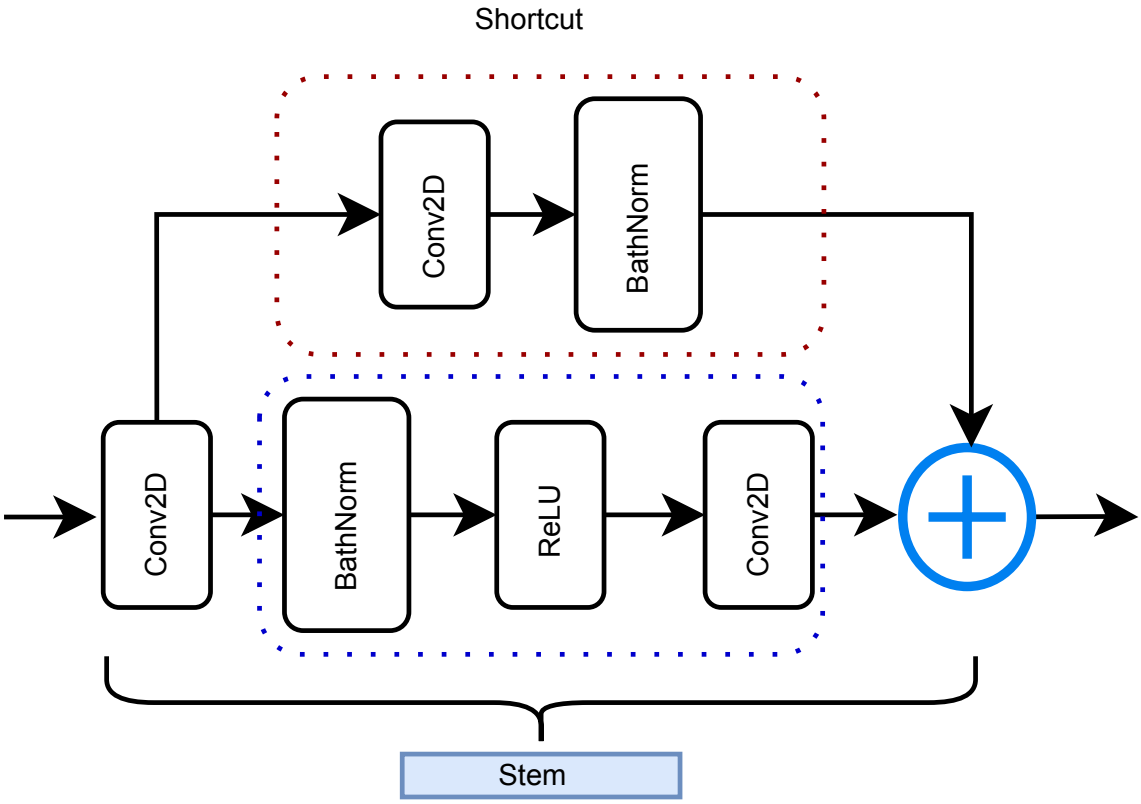


Figure 4.9: ResUNet Building block - Stem

Moving on down the network, we again have an encoder that downsamples the input image tensor into a compact representation by reducing the spatial dimensions. The encoder layer consists of seven residual blocks. Next, we have the bridge in the middle between the encoder and the decoder. The bridge consists of three convolution blocks. Finally, we have the decoder, which recovers the compact representation from the encoder. The decoder layer consists of seven upsampling and residual layers. Each layer receives a skip connection, higher spatial information with the respective encoder layer of the same dimension. The output layer is a Conv2D which produces the reconstructed

image in the same dimension as the input image tensor.

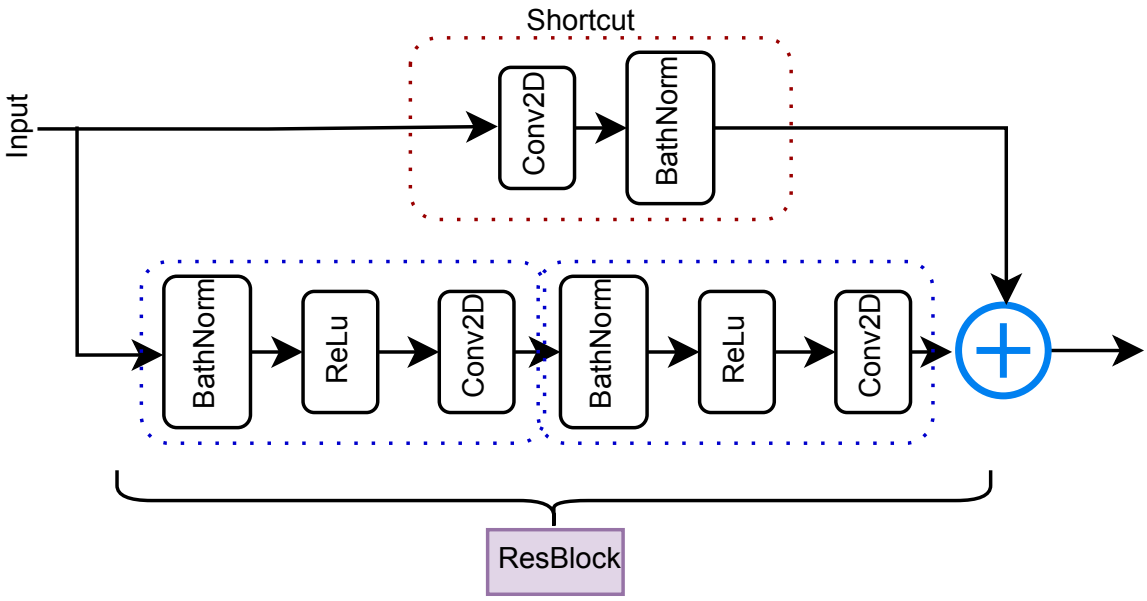


Figure 4.10: ResUNet Building block - Residual block

4.2.3 Discriminator Architecture

The discriminator is a convolutional PatchGAN classifier architecture as defined in [39]. It classifies if each image patch is real or fake (generated). The fig.4.11 shows that the architecture consists of two input layers, where the discriminator network takes the input image and the target image of size $(1, 256, 256, 3)$. From section 4.1, we know that the input image is the structure for which we need to predict the optimized topology. Depending on the training step, the target image can be the generated image from the generator network or an image of the optimized topology from the dataset. The next layer concatenates the two images and downsamples for three steps. The downsample or the encoder block consists of a Conv2D layer followed by batch normalization and Leaky ReLU with $\alpha = 0.3$.

Next, we pass through a 2D Zero padding layer to preserve the dimension of the channel feature and use a Conv2D layer followed by batch normalization and Leaky ReLU with $\alpha = 0.3$ to reduce the spatial dimension to $(1, 33, 33, 512)$. Finally, we add a Conv2D layer to reduce to a 30x30 image patch. Each 30x30 image patch of the output classifies a 70x70 patch in the input image. Averaging all the patches will give a binary output indicating whether the image is real or fake (generated).

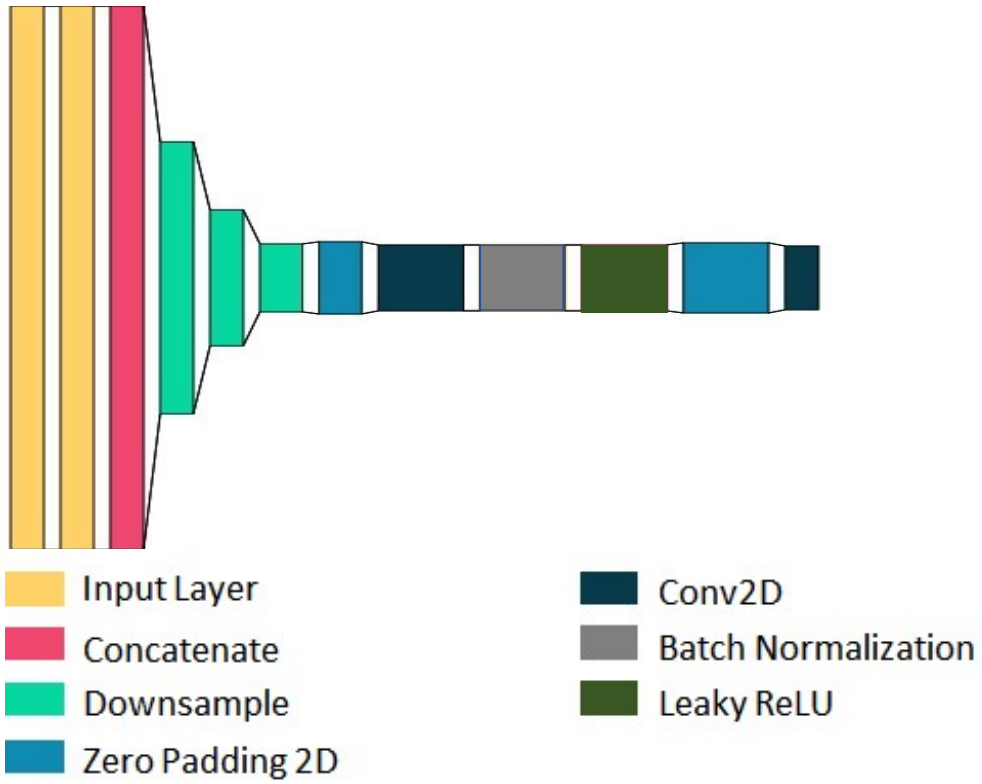


Figure 4.11: Discriminator Architecture

4.2.4 Loss Functions

Here, we have two loss functions, one for the generator and the discriminator. The generator loss is a sigmoid cross-entropy loss between the generated image and an array of ones. For the generated image to be more structurally similar to the target image, we introduce L1 or L2 loss. L1 loss, an MAE (Mean Absolute Error), is used instead of L2 as it produces less blurred output. The combination of a GAN loss and a traditional distance loss helps the generator to be closer to the ground truth, and the GAN loss helps the generator to fool the discriminator, as mentioned in [39]. Since we take two inputs for the discriminator and estimate if the given input is real or fake (generated), we need a $real_loss$ and a $fake_loss$. The real loss takes the real image and an array of ones and performs a sigmoid cross-entropy loss (since these are real images). The fake loss takes the fake (generated) image and an array of zeros and performs a sigmoid cross-entropy loss. Then the total discriminator loss is computed by simply adding real and fake loss. The overall generator loss looks like the equation 4.2.

$$\mathcal{L}_{cGAAN}(G, D) = \mathcal{E}_{x,y}[logD(x, y)] + \mathcal{E}_x[log(1 - D(x, G(x))] \tag{4.1}$$

$$G^* = arg \min_G \max_D \quad \mathcal{L}_{cGAAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{4.2}$$

Where, in equation.4.2, λ acts as a bias to the $\mathcal{L}_{L1}(G)$, L1 loss. And in equation.4.1$D(x, y)$ is the discriminator's prediction when a real image is passed. $D(x, G(x, z))$ is the discriminator's prediction when a fake image is given, and $G(x)$ is the generator's output. $\mathcal{E}_x$ is the expected value over generated data and $\mathcal{E}_{x,y}$ is the expected value over the real data. The loss plots for all the three generator models and the discriminator are shown and discussed in section 5.1.

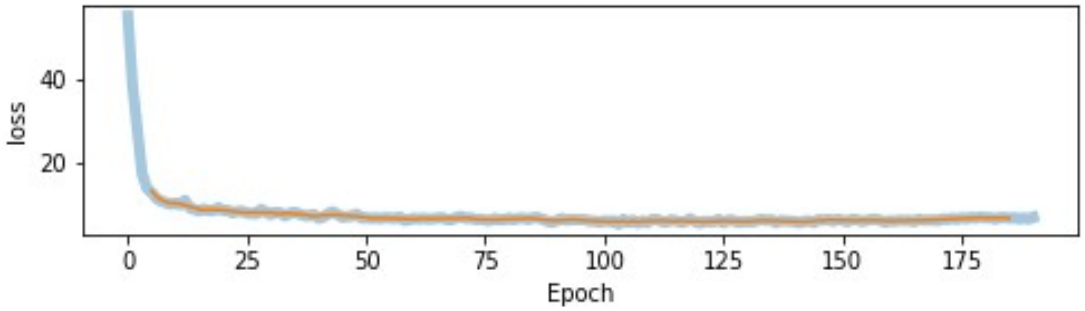
4.2.5 Training

The training step was to alternate between the discriminator and the generator. For one gradient descent on the discriminator, we train one step on the generator. Also, as mentioned in [39], and [36], the training step tries to maximize the $logD(x, G(x, z))$ instead of trying to minimize $log(1 - D(x, G(x, z))$. An Adam optimizer was used with $2\text{e}{-4}$ as the learning rate and 0.5 for the momentum parameter. All the models were trained for 200 epochs, and for every 10 epoch, the model was evaluated against the validation dataset. The resulting images were plotted, and these are discussed in section 5.

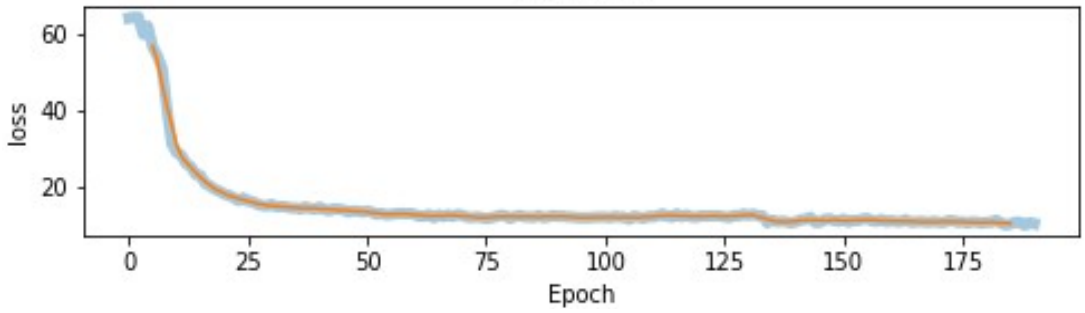
5 Results

In this chapter, we look at the results and compare the performance of the different methods implemented. The images were generated after 200 epochs of training and validated for every ten epoch. To understand the performance, we will compare the generator models against the conventional method with a quantitative and qualitative metric.

(a).UNet Generator



(b). ResNet Generator



(c). ResUNet Generator

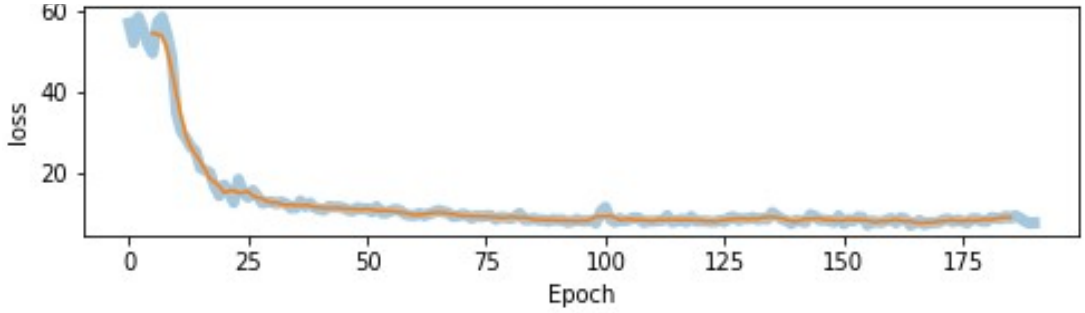


Figure 5.1: The training (thick line) and validation (thin line) loss of (a). UNet, (b). ResNet and (c). ResUNet generators

5.1 Loss Function

A loss function is a function that tracks the performance of the model during training and validation. This is achieved by logging the generator and the discriminator loss, as shown in section 4.2.4. Collecting the losses for every epoch during training is called the training loss. In an epoch, the loss function is calculated across every data item, and it provides the quantitative loss measure at that step. However, plotting the curve across iterations only shows the loss on a subset of the entire dataset. Hence, we plot the validation loss collected using the validation set and the training loss.

The fig.5.1 shows the training (thick line) and the validation (thin line). 5.1(a). has a steeper curve when compared with the other generators. The reason for this is due to the high learning rate. Whereas, the ResNet generator 5.1(b). has a smoother curve due to the momentum term in the Adam Optimizer. We can notice an unstable start to the ResUNet generator 5.1(c). This could be due to the momentum parameter in the Adam optimizer. The momentum smooths the learning rate applied to the model. We can also see the effect of this around 100 epochs. The loss plots were similar for the cantilever structure as well. After training, we will only use the generator for generating the optimized topology. Hence, we don't look at the discriminator loss for the model's performance.

5.2 Speed

Computation speed is one of the critical drawbacks in conventional iterative-based topology optimization. Although the speed could be improved with more resources and computing power, it becomes necessary for the user to have better-performing workstations to optimize simple profiles.

During the training data, the TopOpt Solver [66] took an average of 30 seconds for the MMB structure with 250 x 250 elements along the x and y-direction. It took an average of 37 seconds for the Cantilever structure with the 250 x 250 elements along the x and y-direction. This was performed on a workstation with the following configuration. *Windows 10(64-bit) machine with 16 GB RAM, AMD Ryzen™5 4000 Series Processor and NVIDIA GeForce GTX 1650 Ti graphics processing unit.*

In the case of an AI, one has to look at the computation time and the training time. Training time is the average time taken per epoch. Upon training, the average time the AI takes to predict an output based on the input is related to the conventional computation time.

Network Architecture	MMB Structure	Cantilever Structure
UNet	0.21 sec	0.21 sec
ResNet-50	0.40 sec	0.41 sec
ResUNet	0.39 sec	0.35 sec

Table 5.1: Time take to generate an optimized topology

On average, the UNet-Generator took 25.8 seconds per epoch, the ResNet50-Generator took 45 seconds per epoch, and the ResUNet-Generator took 107 seconds per epoch during training for the MMB problem. For the cantilever problem, the UNet-Generator took 27 seconds per epoch, the ResNet50-Generator took 42 seconds per epoch, and the ResUNet-Generator took 102 seconds per epoch, on average. All generator models took less than a second during testing. The table 5.1 shows the time taken for the generator to generate an optimized topology. It displays a **98.9%** decrease in computation time for both the structures when compared with conventional topology optimizer.

However, as mentioned in [52], although the computation time is very low for AI-based topology optimization, it does not account for the time and resources to train the model. Furthermore, as per their suggestion, it would make more sense not just to use the model to predict one structure and instead to provide assistance for real-time design changes.

5.3 Accuracy

The accuracy of a model is usually evaluated to measure its performance. This tells us how close or far the predicted output is from the ground truth. Whether it is a classification or regression problem, we use different metrics to analyze the performance. The most commonly used methods for classification problems are classification accuracy, confusion matrix, and log-loss, which are used for multi-class classification [74]. Regarding regression problems, Mean Absolute Error and Squared Mean Error are commonly used metrics to measure the performance, as mentioned in [75]

Since the GANs can be viewed as a classification problem [76] or a regression problem [77], one can use a variety of metrics based on the application, as mentioned in [78]. In this work, a Visual Metric and Fréchet Inception Distance (FID) were used to evaluate the performance and compare the methods used.

5.3.1 Visual Metric

This metric is used as the primary analysis. As it is merely a visual inspection, it is more straightforward and intuitive for the user. Fig.5.2 and fig.5.3 shows the results obtained with different models for the generator. More generated examples are available in the GitHub repository [1]

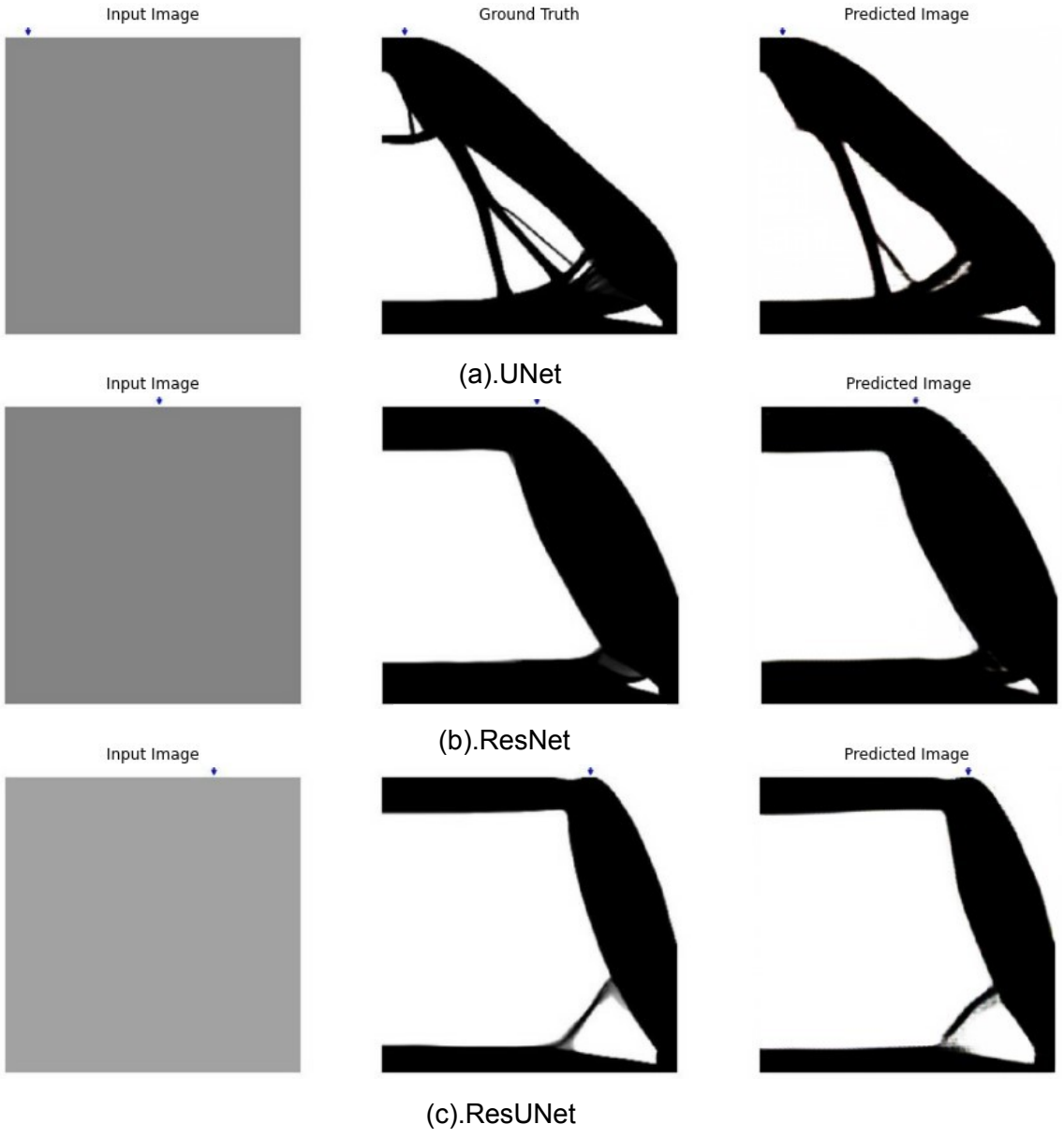


Figure 5.2: (Left to right): input image, output from TopOpt Solver and output from the generator network (a).UNet, (b).ResNet, (c).ResUNet, for MMB structure.

[1] https://github.com/masri20/Image-to-Image-Translation-for-Topology-Optimization

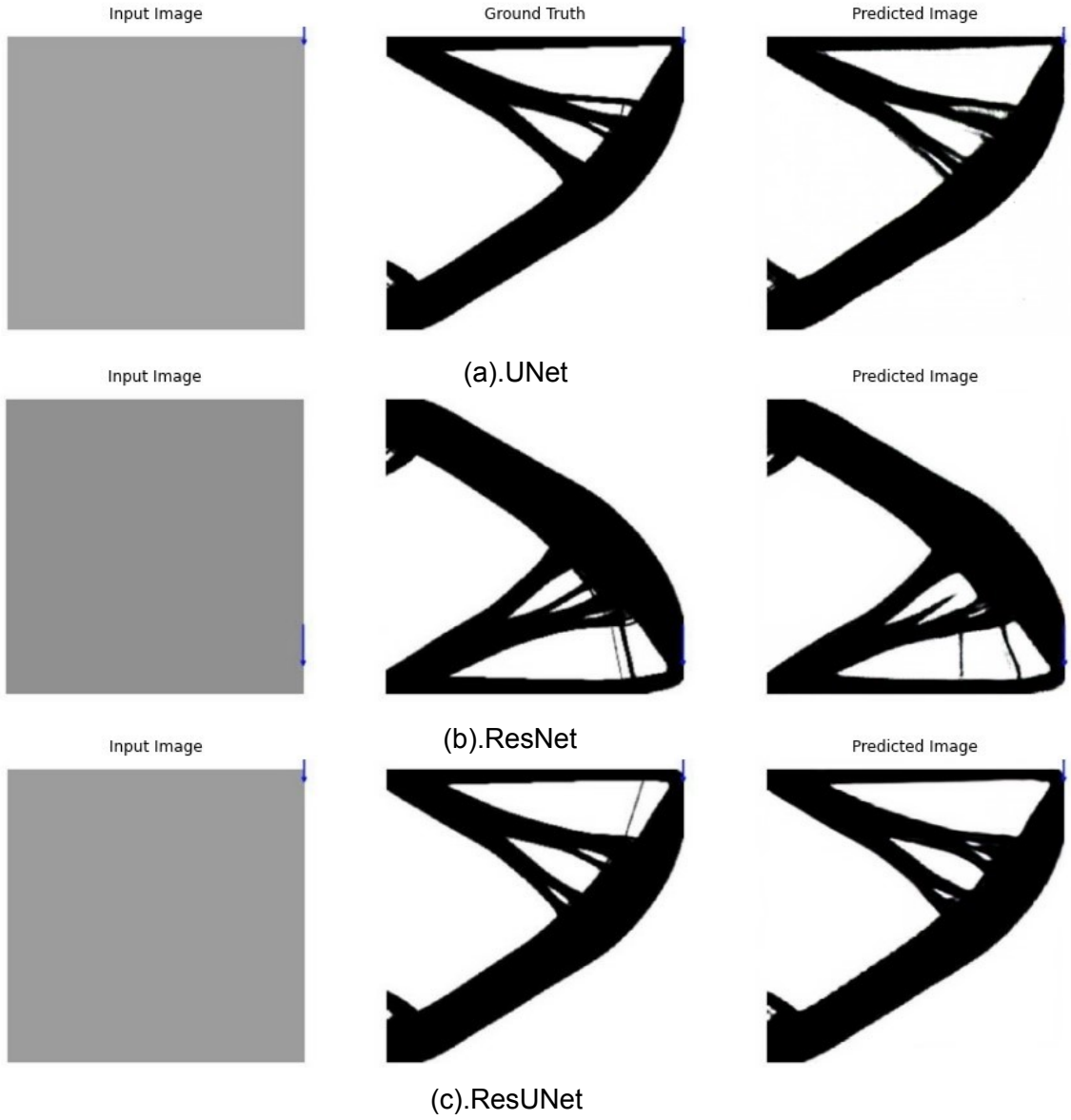


Figure 5.3: (Left to right): input image, output from TopOpt Solver and output from the generator network (a).UNet, (b).ResNet, (c).ResUNet, for cantilever structure.

As explained in section 4.1, one can interpret these fig.5.2 and fig.5.3 in the following way. The first image on to the left is the input image indicating the location at which the force is applied. The image in the middle is the ground truth generated by the TopOpt Solver and the image to the left is the predicted topology optimization by the generator networks.

We can observe that all the generated images are sharp (without any blur). This is because of the L1 loss function in the GAN loss equation.(4.2). L2 loss is preferred in most cases, but minimizing L2 loss is equivalent to maximizing the log-likelihood of a Gaussian distribution for a generative model. In other words, taking the previous example of gen-

erating images of dogs. During training, each image of a dog is different from the other (different breed, color, etc.). This means each image has its corresponding peek. Due to this, when we use L2 loss, we will use a single-peak Gaussian to fit these many kinds of dogs, which is equivalent to averaging dogs of different looks, which will create blurry or splotchy artifacts in the image, as mentioned in [79] and [39].

When we look at the generated images individually, one can see the ResNet generator in fig.5.2.(b) maintains the volume fraction similar to the ground truth. It also tries to complete the profile at places where the TopOpt Solver (ground truth) could not compute the update equation.(3.13). The results produced by ResUNet also maintained the volume fraction and had less unwanted branching, as shown in fig.5.3.(c).

Although the overall generated images are closed to the ground truth in a visual aspect, this metric is entirely based on how one perceives the image. Therefore, using other methods makes it possible to compare the quality of the generated images on a detailed level, where a human eye can miss the specific details.

5.3.2 Fréchet Inception Distance - FID

Fréchet Inception Distance (FID), introduced in [80], is a metric that measures the generated images' quality, similar to the Inception score. For the inception score, the authors in [81] use an Inception Model [82] to predict the class probabilities of the generated images. This will give us conditional label distribution (with low entropy) and a marginal distribution (with high entropy) from the generated image. We want a lower entropy (lower average uncertainty) for the conditional label distribution as this shows the generated images are classified as one class. On the other hand, we would want the marginal distribution to have a higher entropy as this shows the probability distribution of all the generated images.

In this case, we look at the consistency and the quality of generated images over the conditional label distribution and the diversity of the generated images over the marginal distribution. By combining the two distributions, we get equation 5.1, which gives us the Inception Score. Where **KL** denots the Kullback-Leibler divergence [83], or KL divergence (relative entropy between conditional label and marginal distribution).

$$exp(E_{\boldsymbol{x}}\mathbf{KL}(p(y|\boldsymbol{x})||p(y))) \tag{5.1}$$

$where,$

$p(y|\boldsymbol{x})$ $\rightarrow$ Conditional label Distribution

$p(y) = \int p(y|\boldsymbol{x} = G(z))\, dz$ $\rightarrow$ Marginal Distribution

Fréchet Inception Distance is an extension of the Inception score. Here we try to determine if the real data distribution, p_r(.) and generated data distribution, p_g(.) are equal by using the following equation 5.2

$$p_r(.) = p_g(.) \iff \int p_r(x) f(x)\, dx = \int p_g(x) f(x)\, dx \tag{5.2}$$

$where,$

$f(x) \rightarrow$ Basis function which spans over the space in which $p_r(.)$ and $p_g(.)$ live

The basis function $f(x)$ can be a polynomial, which gives us the first and second moments of our distribution. Since we can obtain all the distribution properties by knowing the mean and the covariance, we consider the distribution a Gaussian distribution. We calculate these statistics for the entire set of real and generated images and compute the distance between these distributions using Fréchet Distance [84] or Wasserstein-2 Distance [85], as mentioned in [80]. The below equation 5.3 shows the Fréchet Inception Distance [86].

$$d^2((\boldsymbol{m_g}, \boldsymbol{C_g}), (\boldsymbol{m_r}, \boldsymbol{C_r})) = ||\boldsymbol{m_g} - \boldsymbol{m_r}||_2^2 + Tr(\boldsymbol{C_g} + \boldsymbol{C_r} - 2(\boldsymbol{C_g C_r})^{½}) \tag{5.3}$$

$where,$

$d^2 \rightarrow$ Distance which is considered as the FID Score

$Tr \rightarrow$ Trace, i.e., sum of elements along the diagonal of the square matrix

$(\boldsymbol{m_r}, \boldsymbol{m_g}) \rightarrow$ Feature-wise mean of the real and generated image respectively

$(\boldsymbol{C_g}, \boldsymbol{C_g}) \rightarrow$ Covariance matrix for real and generated feature vectors

$||\boldsymbol{m_g} - \boldsymbol{m_r}||_2^2 \rightarrow$ Sum of squared difference between two mean vectors

In practicality, we use the pre-trained Inception model [82] to obtain the FID score. The model is modified such that a global spatial pooling layer is taken as the output layer instead of the conventional soft-max output layer (the activations of the last pooling layer). This layer has 2,048 activations, containing the image's coding vector or feature vector x. The real and generated images are passed through this modified Inception Model, and outputs feature vectors for real and generated images. Finally, the feature vectors of the real image are taken as the reference to compare the feature vectors from the generated images.

Network Architecture	MMB Structure	Cantilever Structure
UNet	28.9	26.4
ResNet-50	9.16	18.99
ResUNet	13.17	14.15

Table 5.2: FID scores for the generator models implemented for MMB and Cantilever structures.

An image pair was created using [66] and passed to the three different models, UNet, ResNet and ResUNet to give us the predictions. Finally, we pass the real image (ground truth) and the respective generated image to the modified Inception model to compute the FID scores. The table 5.2 shows the FID scores for the three different models. The general understanding is that the lower the FID score, the better. This means that the features of both images are almost alike.

Based on the table 5.2, it is evident that the ResNet-50 generator produces images closer to the ground truth. This evidence is compelling because a ResNet generator containing nine residual blocks was used during the initial trials, resulting in an FID score of 28.16. Further experiments revealed similar results with the ResUNet generator. In initial trials, ResUnet with four residual blocks resulted in an FID score of 44.04. Although it is trivial that the more residual layer means better back-propagation of the gradients, both the architectures were subsequently changed to produce better results, as shown in the table 5.2.

However, all of those mentioned above also depend on the quality of the reference image (ground truth). Therefore, these scores could indicate how the model performs at a detail level. One would recommend a combination of both metrics to evaluate the model's performance.

6 Conclusion

We have looked at a different approach to solve 2D topology optimization for an MMB structure and a Cantilever structure. The conventional methods such as Solid Isotropic Material Penalization (SIMP), where we discretize the design domain into a grid of finite elements called isotropic solid microstructures to compute the density distribution of material. Topology optimization is also solved by many numerical methods, such as Sequential Quadratic Programming (SQP), Method of Moving Asymptotes (MMA), and Sequential Linear Programming (SLP). We also looked at the problems we would encounter when we use them. These numerical methods have been implemented as python libraries to generate optimized topology. However, all of these methods come at the cost of computational time and resources.

Moreover, the computational time increases drastically with an increase in the number of elements in the design domain. The python library used in this work, TopOpt solver [66] uses the Method of Moving Asymptotes (MMA) to optimize the topology. Method of Moving Asymptotes converges to a solution faster than other numerical methods. Nevertheless, it could only handle a maximum of 90000 elements before the system runs out of computational resources, and in section, 5.2 we saw the computational time for both the structures. Therefore, we consider the conventional topology optimization problem as an image-to-image translation problem.

In this work, we look at training a deep learning method, namely conditional generative adversarial networks (CGAN), as a viable solution for topology optimization. Converting the problem from the design domain to the image domain, one can perform image-to-image translation to obtain an optimized topology. We provide an image of the initial boundary and load conditions, and it gets translated to an optimized topology. This is achieved by a generator network that learns the mapping between the input and target images and tries to generate new images in the input domain. A discriminator network tries to identify if the image is real (from the training) or fake (generated).

This data-driven approach was explored with three different network architectures. The first method was a modified UNet architecture, a preferred architecture for generative

tasks, which generated satisfactory results. However, a detailed evaluation showed that even though the generated image was visually close to the ground truth, the FID scores were inconclusive. As a result, efforts were made to make the generator network deeper by adding more layers, but it did not positively impact the generated image's quality. Hence, a different network model, namely ResNet, was implemented to solve this. The ResNet architecture was executed as ResNet-9 with nine residual blocks as the bridge. The ResNet-9 architecture was recommended as an alternative generator architecture in CycleGAN [72], but the results were inconclusive, as evidently shown by the FID score in the table. 5.2. Therefore, the residual layers were increased to ResNet-50, and consequently, we can see an improvement in the quality of the image generated.

Finally, to combine the advantage of using skip connections in UNet architecture and residual blocks with identity connections in the ResNet architecture, a modified ResUNet was implemented. Initial execution with four residual layers was changed to 7 residual blocks in both the encoder and the decoder to improve the quality of the image generated.

Although, a critical remark would be that all the models suffer in the quality of the generated output with smaller volume fractions. Since we do not explicitly state the volume fraction during training, the model becomes inconsistent with lower volume fractions. An easy fix would be to add an input layer, where we encode the volume fraction as a label. However, this could need further examination to find the exact root cause.

Figure 6.1: Image on the right is the generator network generated output and on the right is its respective 3D STL[87]

On the flip side, compared with the conventional methods, one can observe a considerable decrease in computation time and resources. One can take this as an initial tool to generate optimized topology for hobby prototyping. Even though the generated output is in the 2D domain, one can use online applications to convert the generated profile into a mesh, as shown in figure 6.1.

Thus the proposed solution is a comparison study to determine a better solution for optimizing topology. The essential advantage of this system is that it is versatile. It can be used for minimum compliance, heat distribution, and other topology optimization problems provided one has enough training data.

6.1 Future Work

The presented work can be extended in 2 significant ways. First, we can look at the possibility of trying to solve other structures such as L-Bracket and I-beams. The python library used in this work [66] can also generate training data for L-Bracket and I- Beams. Although, during initial trials, the time to optimize for 250x250 elements was extremely high. This made it impossible to collect enough data for training. Another improvement would be trying thermal topology optimization. But, again, we would need to collect the training data, which could be harder using conventional solvers with higher computation time. One can also look into topology optimization in a 3D domain. Again, we need a traditional 3D topology optimizer to generate the training data, such as [88]

The following extension could be trying to encode the volume fraction as an input parameter. This will fix the occasional instability in the quality of the generated images. Another extension could be testing a different generative model. For example, BicycleGAN[89] has shown some impressive results when it comes to one-to-many generative tasks. This means we can get other options for the optimized topology from one input image, and the variation could be in terms of volume fraction.

Although the possibility of experimenting with different methods has no limits, the methods mentioned above are of particular personal interest.

Bibliography

[1] Martin Philip Bendsøe and Noboru Kikuchi. “Generating optimal topologies in structural design using a homogenization method”. In: *Computer Methods in Applied Mechanics and Engineering* 71.2 (1988), pp. 197–224. ISSN: 0045-7825. DOI: https://doi.org/10.1016/0045-7825(88)90086-2. URL: https://www.sciencedirect.com/science/article/pii/0045782588900862.

[2] Krister Svanberg. “The method of moving asymptotes—a new method for structural optimization”. In: *International journal for numerical methods in engineering* 24.2 (1987), pp. 359–373.

[3] Mehdi Mirza and Simon Osindero. “Conditional Generative Adversarial Nets”. In: *CoRR* abs/1411.1784 (2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784.

[4] Aljaľ Kramberger et al. “Automatic Fingertip Exchange System for Robotic Grasping in Flexible Production Processes”. In: *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. 2019, pp. 1664–1669. DOI: 10.1109/COASE.2019.8842911.

[5] Lukas Christoffer Malte Wiuf Schwartz et al. “Designing Fingers in Simulation based on Imprints.” In: *SIMULTECH*. 2017, pp. 304–313.

[6] Frederik Hagelskjær et al. “Combined Optimization of Gripper Finger Design and Pose Estimation Processes for Advanced Industrial Assembly”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 2022–2029.

[7] Adam Wolniakowski et al. “Task and context sensitive gripper design learning using dynamic grasp simulation”. In: *Journal of Intelligent & Robotic Systems* 87.1 (2017), pp. 15–42.

[8] Martin Bendsøe. “Bendsoe, M.P.: Optimal Shape Design as a Material Distribution Problem. Structural Optimization 1, 193-202”. In: *Structural Optimization* 1 (Jan. 1989), pp. 193–202. DOI: 10.1007/BF01650949.

[9] Martin Bendsøe and Ole Sigmund. “Material interpolation schemes in topology optimization”. In: *Archive of Applied Mechanics* 69 (Nov. 1999), pp. 635–654. DOI: 10.1007/s004190050248.

[10] M Marcel Beckers. “Topology optimization using a dual method with discrete variables”. In: *Structural optimization* 17 (1999), pp. 14–24.

[11] A. Baumgartner, L. Harzheim, and C. Mattheck. “SKO (soft kill option): the biological way to find an optimum structure topology”. In: *International Journal of Fatigue* 14.6 (1992), pp. 387–393. ISSN: 0142-1123. DOI: https://doi.org/10.1016/0142-1123(92)90226-3. URL: https://www.sciencedirect.com/science/article/pii/0142112392902263.

[12] YM Xie and GP Steven. “Evolutionary structural optimization for dynamic problems”. In: *Computers & Structures* 58.6 (1996), pp. 1067–1073.

[13] Chongbin Zhao et al. “A generalized evolutionary method for numerical topology optimization of structures under static loading conditions”. In: *Structural optimization* 15.3 (1998), pp. 251–260.

[14] David Greiner and Prabhat Hajela. “Truss topology optimization for mass and reliability considerations - Co-evolutionary multiobjective formulations”. In: *Structural and Multidisciplinary Optimization* 45 (Apr. 2012), pp. 589–613. DOI: 10.1007/s00158-011-0709-9.

[15] Patrick Y Shim and Souran Manoochehri. “Generating optimal configurations in structural design using simulated annealing”. In: *International journal for numerical methods in engineering* 40.6 (1997), pp. 1053–1069.

[16] Guan-Chun Luh and Chun-Yi Lin. “Structural topology optimization using ant colony optimization algorithm”. In: *Applied Soft Computing* 9.4 (2009), pp. 1343–1353.

[17] Ali Kaveh et al. “Structural topology optimization using ant colony methodology”. In: *Engineering Structures* 30.9 (2008), pp. 2559–2565.

[18] Guan-Chun Luh, Chun-Yi Lin, and Yu-Shu Lin. “A binary particle swarm optimization for continuum structural topology optimization”. In: *Applied Soft Computing* 11.2 (2011), pp. 2833–2844.

[19] Ole Sigmund. “A 99 line topology optimization code written in Matlab”. In: *Structural and multidisciplinary optimization* 21.2 (2001), pp. 120–127.

[20] Sun-Yong Kim, Il Yong Kim, and Chris Mechefske. “A new efficient convergence criterion for reducing computational expense in topology optimization: Reducible design variable method”. In: *International Journal for Numerical Methods in Engineering* 90 (May 2012), pp. 752–783. DOI: 10.1002/nme.3343.

[21] Kai Liu et al. “Towards Nonlinear Multimaterial Topology Optimization Using Unsupervised Machine Learning and Metamodel-Based Optimization”. In: Aug. 2015, V02BT03A004. DOI: 10.1115/DETC2015-46534.

[22] Thomas Borrvall and Joakim Petersson. “Large-scale topology optimization in 3D using parallel computing”. In: *Computer methods in applied mechanics and engineering* 190.46-47 (2001), pp. 6201–6229.

[23] Ting Wei Chin and Graeme J. Kennedy. “Large-Scale Compliance-Minimization and Buckling Topology Optimization of the Undeformed Common Research Model Wing”. In: 2016.

[24] Jie Liu et al. “Topological Design of a Lightweight Sandwich Aircraft Spoiler”. In: *Materials* 12 (Oct. 2019), p. 3225. DOI: 10.3390/ma12193225.

[25] Ji-Hong Zhu and Tong Gao. *Topology optimization in engineering structure design*. Elsevier, 2016.

[26] Joe Alexandersen, Ole Sigmund, and Niels Aage. “Large scale three-dimensional topology optimisation of heat sinks cooled by natural convection”. In: *International Journal of Heat and Mass Transfer* 100 (Sept. 2016), pp. 876–891. DOI: 10.1016/j.ijheatmasstransfer.2016.05.013.

[27] Xianmin Zhang and Benliang Zhu. “Topology Optimization of Distributed Compliant Mechanisms”. In: *Topology Optimization of Compliant Mechanisms*. Singapore: Springer Singapore, 2018, pp. 81–119. ISBN: 978-981-13-0432-3. DOI: 10.1007/978-981-13-0432-3_3. URL: https://doi.org/10.1007/978-981-13-0432-3_3.

[28] Alok Sutradhar et al. “Designing patient-specific 3D printed craniofacial implants using a novel topology optimization method”. In: *Medical & biological engineering & computing* 54.7 (2016), pp. 1123–1135.

[29] Kyle Mills, Michael Spanner, and Isaac Tamblyn. “Deep learning and the Schrödinger equation”. In: *Physical Review A* 96.4 (Oct. 2017). DOI: 10.1103/physreva.96.042113. URL: https://doi.org/10.1103%2Fphysreva.96.042113.

[30] Felix Brockherde et al. “Bypassing the Kohn-Sham equations with machine learning”. In: *Nature communications* 8.1 (2017), pp. 1–10.

[31] Jyh-Shyong Chang, Shih-Chieh Lu, and Yu-Lun Chiu. “Dynamic modeling of batch polymerization reactors via the hybrid neural-network rate-function approach”. In: *Chemical Engineering Journal* 130.1 (2007), pp. 19–28.

[32] HR Kirby and GB Parker. “The development of traffic and transport applications of artificial intelligence: an overview”. In: *Artificial Intelligence Application to Traffic Engineering* (1994).

[33] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[35] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[36] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[37] Christian Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 105–114.

[38] Han Zhang et al. *StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks*. 2016. DOI: 10.48550/ARXIV.1612.03242. URL: https://arxiv.org/abs/1612.03242.

[39] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *CoRR* abs/1611.07004 (2016). arXiv: 1611.07004. URL: http://arxiv.org/abs/1611.07004.

[40] Ji-Hwan Moon, Jin-Ho Park, and Gye-Young Kim. “Restore Fingerprints Using Pix2Pix”. In: *Advances in Computer Science and Ubiquitous Computing*. Ed. by James J. Park et al. Singapore: Springer Singapore, 2021, pp. 489–494. ISBN: 978-981-15-9343-7.

[41] Badour AlBahar and Jia-Bin Huang. *Guided Image-to-Image Translation with Bi-Directional Feature Transformation*. 2019. DOI: 10.48550/ARXIV.1910.11328. URL: https://arxiv.org/abs/1910.11328.

[42] Qiyin Lin et al. “Investigation into the topology optimization for conductive heat transfer based on deep learning approach”. In: *International Communications in Heat and Mass Transfer* 97 (2018), pp. 103–109. ISSN: 0735-1933. DOI: https://doi.org/10.1016/j.icheatmasstransfer.2018.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0735193318301593.

[43] Ivan Sosnovik and Ivan V. Oseledets. “Neural networks for topology optimization”. In: *CoRR* abs/1709.09578 (2017). arXiv: 1709.09578. URL: http://arxiv.org/abs/1709.09578.

[44] William Hunter et al. *ToPy - Topology optimization with Python*. https://github.com/williamhunter/topy. 2017.

[45] Liang Xue et al. *An efficient and high-resolution topology optimization method based on convolutional neural networks*. 2020. DOI: 10.48550/ARXIV.2001.04350. URL: https://arxiv.org/abs/2001.04350.

[46] Changyu Deng, Can Qin, and Wei Lu. "Deep-Learning-Enabled Simulated Annealing for Topology Optimization". In: *ArXiv* abs/2002.01927 (2020).

[47] Saurabh Banga et al. "3d topology optimization using convolutional neural networks". In: *arXiv preprint arXiv:1808.07440* (2018).

[48] Ying Yu et al. "CWGAN: Conditional Wasserstein Generative Adversarial Nets for Fault Data Generation". In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2019, pp. 2713–2718. DOI: 10.1109/ROBIO49542.2019.8961501.

[49] Sharad Rawat and M.-H. Herman Shen. "A Novel Topology Optimization Approach using Conditional Deep Learning". In: *CoRR* abs/1901.04859 (2019). arXiv: 1901.04859. URL: http://arxiv.org/abs/1901.04859.

[50] Yonggyun Yu, Taeil Hur, and Jaeho Jung. "Deep learning for topology optimization design". In: *arXiv preprint arXiv:1801.05463* (2018).

[51] Yu Li et al. "Reconstruction of simulation-based physical field by reconstruction neural network method". In: *arXiv preprint arXiv:1805.00528* (2018).

[52] Mathias Bernhard et al. "Topology Optimization with Generative Adversarial Networks". In: ().

[53] Aykut Kentli. "Topology Optimization Applications on Engineering Structures". In: *Truss and Frames*. Ed. by Aykut Kentli. Rijeka: IntechOpen, 2019. Chap. 4. DOI: 10.5772/intechopen.90474. URL: https://doi.org/10.5772/intechopen.90474.

[54] Mariusz Bujny et al. "Evolutionary crashworthiness topology optimization of thin-walled structures". In: *ASMO UK, Munich, Germany* (2016).

[55] Lauren L. Beghini et al. "Connecting architecture and engineering through structural topology optimization". In: *Engineering Structures* 59 (2014), pp. 716–726. ISSN: 0141-0296. DOI: https://doi.org/10.1016/j.engstruct.2013.10.032. URL: https://www.sciencedirect.com/science/article/pii/S0141029613005014.

[56] Martin P. Bendsøe and Ole Sigmund. "Topology optimization by distribution of isotropic material". In: *Topology Optimization: Theory, Methods, and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–69. ISBN: 978-3-662-05086-6. DOI: 10.1007/978-3-662-05086-6_1. URL: https://doi.org/10.1007/978-3-662-05086-6_1.

[57] William Hunter et al. *ToPy - Topology optimization with Python*. https://github.com/williamhunter/topy. 2017.

[58] M Fanni, MN Shabara, and MG Alkalla. “A Comparison between Different Topology Optimization Methods.(Dept M (Production))”. In: *MEJ. Mansoura Engineering Journal* 38.4 (2020), pp. 13–24.

[59] *Mathematics of Big Data and Machine Learning: Artificial Intelligence and Machine Learning | Mathematics of Big Data and Machine Learning | Supplemental Resources | MIT OpenCourseWare*. URL: https://ocw.mit.edu/courses/res-ll-005-mathematics-of-big-data-and-machine-learning-january-iap-2020/resources/mitres_ll_005iap20_supplemental_ses01/.

[60] P. Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Penguin Books Limited, 2015. ISBN: 9780241004555. URL: https://books.google.dk/books?id=pjRkCQAAQBAJ.

[61] Alan M Turing. “Computing machinery and intelligence”. In: *Parsing the turing test*. Springer, 2009, pp. 23–65.

[62] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[63] *Convolutional Neural Network Definition | DeepAI*. [Online; accessed 2022-05-21]. URL: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network.

[64] *CGAN*. https://medium.com/analytics-vidhya/conditional-generative-adversarial-networks-f8f1ce025c5d.

[65] Zekuan Yu et al. “Retinal image synthesis from multiple-landmarks input with generative adversarial networks”. In: *Biomedical engineering online* 18.1 (2019), pp. 1–15.

[66] Zachary Ferguson. *topopt*. https://github.com/zfergus/topopt. 2020.

[67] *stevengj/nlopt: library for nonlinear optimization, wrapping many algorithms for global and local, constrained or unconstrained, optimization*. [Online; accessed 2022-05-25]. URL: https://github.com/stevengj/nlopt.

[68] Meenakshsundaram. *Topology Optimization*. via Wikimedia Commons. URL: https://creativecommons.org/licenses/by-sa/3.0.

[69] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[70] Please look at their website. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[71] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. “Instance normalization: The missing ingredient for fast stylization”. In: *arXiv preprint arXiv:1607.08022* (2016).

[72] Jun-Yan Zhu et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.

[73] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. “Road extraction by deep residual u-net”. In: *IEEE Geoscience and Remote Sensing Letters* 15.5 (2018), pp. 749–753.

[74] Aditya Mishra. *Metrics to Evaluate your Machine Learning Algorithm | by Aditya Mishra | Towards Data Science*. misc. [Online; accessed 2022-05-04]. URL: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

[75] *Regression Metrics for Machine Learning*. [Online; accessed 2022-05-04]. URL: https://machinelearningmastery.com/regression-metrics-for-machine-learning/#:%7B%5Ctextasciitilde%7D:text=There%20are%20three%20error%20metrics,Mean%20Absolute%20Error%20(MAE).

[76] Sayak Paul. *Conditional GAN*. misc. [Online; accessed 2022-05-04]. URL: https://keras.io/examples/generative/conditional_gan/.

[77] Richard Marriott, Sami Romdhani, and Liming Chen. “Taking control of intra-class variation in conditional gans under weak supervision”. In: (2020), pp. 257–264.

[78] Ali Borji. “Pros and Cons of GAN Evaluation Measures”. In: *CoRR* abs/1802.03446 (2018). arXiv: 1802.03446. URL: http://arxiv.org/abs/1802.03446.

[79] Hang Zhao et al. “Loss functions for image restoration with neural networks”. In: *IEEE Transactions on computational imaging* 3.1 (2016), pp. 47–57.

[80] Martin Heusel et al. “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf.

[81] Tim Salimans et al. “Improved Techniques for Training GANs”. In: *CoRR* abs/1606.03498 (2016). arXiv: 1606.03498. URL: http://arxiv.org/abs/1606.03498.

[82] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[83] I. Csiszar. “I-Divergence Geometry of Probability Distributions and Minimization Problems”. In: *The Annals of Probability* 3.1 (1975), pp. 146–158. DOI: 10.1214/aop/1176996454. URL: https://doi.org/10.1214/aop/1176996454.

[84] Mauritius Fréchet. “On the distance of two probability laws”. In: *Weekly Reports of the Sessions of the Academy of Sciences* 244.6 (1957), pp. 689–692.

[85] Leonid Nisonovich Vaserstein. “Markov processes over denumerable products of spaces, describing large systems of automata”. In: *Problemy Peredachi Informatsii* 5.3 (1969), pp. 64–72.

[86] DC Dowson and BV666017 Landau. “The Fréchet distance between multivariate normal distributions”. In: *Journal of multivariate analysis* 12.3 (1982), pp. 450–455.

[87] *Imgage t0 STL*. https://imagetostl.com/.

[88] Kai Liu and Andres Tovar. “An efficient 3D topology optimization code written in Matlab”. In: *Structural and Multidisciplinary Optimization* 50 (Dec. 2014). DOI: 10.1007/s00158-014-1107-x.

[89] Jun-Yan Zhu et al. “Toward multimodal image-to-image translation”. In: *Advances in neural information processing systems* 30 (2017).

A Appendix - Guide to use the code

The proposed methods have been implemented using Keras with Tensorflow as back-end. In order to use the code, one has to follow the below mentioned steps.

Cloning the repository

Clone the repository: https://github.com/masri20/Image-to-Image-Translation-for-Topology-Optimization

Navigation

The repository consist of different networks used in this work, namely UNet, ResNet50 and ResUNet. These are under the folder *./Models*

Set path to respective database

Each program has the option to train and test both MMB and cantilever structures. After cloning please change the file-path for the dataset. If you prefer to use your own dataset, remember to change the file-path. The images have to be 512x256 along with a train-validation split.

Results

All the programs will generate training and validation plot. It also generates images after every 10 epoch. The previous results and plots are in *./Results*.

Evaluation

The code for FID evaluation for the in the root file-path.

University of
Southern Denmark

Campusvej 55
5230 Odense M

www.sdu.dk/mmmi