Noname manuscript No.
(will be inserted by the editor)

TOuNN: Topology Optimization using Neural Networks

Aaditya Chandrasekhar · Krishnan Suresh

the date of receipt and acceptance should be inserted later

Abstract Neural networks, and more broadly, machine learning techniques, have been recently exploited to *accelerate* topology optimization through data-driven training and image processing. In this paper, we demonstrate that one can *directly execute* topology optimization (TO) using neural networks (NN). The primary concept is to use the NN's activation functions to represent the popular Solid Isotropic Material with Penalization (SIMP) density field. *In other words, the density function is parameterized by the weights and bias associated with the NN, and spanned by NN's activation functions; the density representation is thus independent of the finite element mesh.* Then, by relying on the NN's built-in backpropogation, and a conventional finite element solver, the density field is optimized.

Methods to impose design and manufacturing constraints within the proposed framework are described and illustrated. A byproduct of representing the density field via activation functions is that it leads to a crisp and differentiable boundary. The proposed framework is simple to implement, and is illustrated through 2D and 3D examples. Some of the unresolved challenges with the proposed framework are also summarized.

1 Introduction

Topology optimization (TO) is now a well established field encompassing numerous methods including Solid Isotropic Material with Penalization (SIMP) based optimization [1], [2], [3], [4], level set methods [5], evolutionary methods [6] and topological sensitivity methods [7] [8], [9], [10]. The variety of TO methods has not only added richness to the field, it offers design engineers several TO options to choose from.

The objective of this paper is to explore yet another TO method that differs from the above established methods in its construction. The proposed method relies on the popular Solid Isotropic Material with Penalization (SIMP) mathematical formulation, but uses a *neural network's activation functions* [11] to represent the SIMP density. *In other words, the density function is parameterized by the weights and bias associated with the NN, and spanned by NN's activation functions; the density representation is thus independent of the finite element mesh.* While prior work (see Section 2) have used neural networks (NN) to *accelerate* topology optimization, the objective here is to *directly execute* topology optimization using NN. The proposed framework is discussed in Section 3, followed by a description of the algorithm in

Aaditya Chandrasekhar
Department of Mechanical Engineering
University of Wisconsin-Madison
E-mail: achandrasek3@wisc.edu

Krishnan Suresh
Department of Mechanical Engineering
University of Wisconsin-Madison
E-mail: ksuresh@wisc.edu

Section 4. In Section 5, several numerical experiments are carried out to establish the validity, robustness and other characteristics of the method. Open research challenges, opportunities and conclusions are summarized in Section 6.

2 Literature Review

Given the objective of this paper, the literature review is limited here to recent work on exploiting NN for TO. The primary strategy thus far is to use NN, and more broadly machine learning (ML) techniques, to *accelerate* TO through data-driven training, and image processing. For example, an encoder-decoder convolutional neural network (CNN) was used in [12] to accelerate TO, based on the premise that a large data set spanning multiple loads and boundary conditions can help establish a mapping from problem specification to an optimized topology. The authors of [13] also employed a CNN, but they established a mapping from intermediate TO results to the final optimized structure, thereby once again accelerating TO. A data-driven approach for predicting optimized topologies under variable loading cases was proposed in [14], where the binary images of optimized topologies are used as training data; then, a feed-forward neural net was employed to predict the optimized topology under different loading condition. In [15], the authors proposed ML models based on support vector regression and K-nearest-neighbors to generate optimal topologies in a moving morphable component framework. Recently, a data-driven conditional generative adversarial network (GAN) was used [16] to generate optimized topologies based on input vector specification. The authors of [17] used a CNN trained with a data set containing optimized topologies in conjunction with a GAN to generate optimal topologies, while the authors of [18] used a ML framework to recognize and substitute evolving features during the optimization process thereby improving the convergence rate. An encoder-decoder framework was proposed in [19] to learn optimized designs at various loading conditions, as a surrogate to gradient based optimization. A recent effort that seeks to capture optimized topologies using NN is reported in [20]; they used CNN's to obtain an image prior of the optimized designs, and the prior was then post-processed to obtain optimized topologies.

In this paper, instead of using NN (or broadly ML techniques) as a training/acceleration tool, we *directly execute* topology optimization using NN. The primary concept is to use NN's activation functions to represent the SIMP based density field. Then, by relying on the NN's backpropogation and a conventional finite element solver, the density field is optimized. A byproduct of representing the density field via activation functions is that it leads to a crisp and differentiable boundary, with implicit filtering. These and other characteristics of the method are demonstrated later through numerical experiments.

3 Proposed Method

3.1 Overview

We will assume that a design domain with loads and restraints have been prescribed (for example, see Figure 1). The objective here is to find, within this design domain, a topology of minimal compliance and desired volume. We will also assume that the domain has been discretized into finite elements for structural analysis.

The method discussed in this paper builds upon the popular SIMP formulation for topology optimization (TO) [1]. In particular, the TO problem is converted into a continuous optimization problem using an auxiliary density field ρ defined over the domain. Assuming the volume constraint is active, as is typically assumed in such compli-

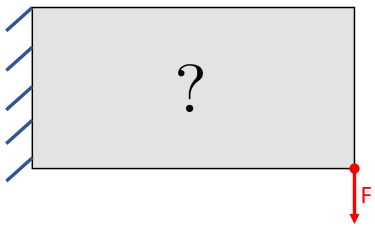

Fig. 1: A classic topology optimization problem.

ance minimization problems [3], the topology optimization problem can be posed as [1], [2], [3]:

$$\underset{\rho}{\text{minimize}} \quad \boldsymbol{u}^{\mathsf{T}}\boldsymbol{K}(\boldsymbol{\rho})\boldsymbol{u} \tag{1a}$$

$$\text{subject to} \quad \boldsymbol{K}(\boldsymbol{\rho})\boldsymbol{u} = \boldsymbol{f} \tag{1b}$$

$$\sum_e \rho_e v_e = V^* \tag{1c}$$

where $\boldsymbol{u}$ is the displacement field, $\boldsymbol{K}$ is the finite element stiffness matrix, $\boldsymbol{f}$ is the applied force, ρ_e is the density associated with element e, v_e is the volume of the element, and V^* is the prescribed volume.

While the density field is typically represented using the finite element mesh, in this paper, it will be constructed independently via activation functions associated with a neural network (NN). In other words, given any point in the domain, the NN will output a density value; see Figure 2. Using this transformation, we convert the constrained optimization problem in Equation 1 into an unconstrained penalty problem by constructing a loss function (as explained later in section 3.4). The loss function is minimized by employing standard machine-learning (ML) techniques. The methodology and various components of the framework are described in the remainder of this section. We note that in ML, optimization is often referred to as *training*, sensitivity analysis as *back-propagation*, and iteration as *epoch*.

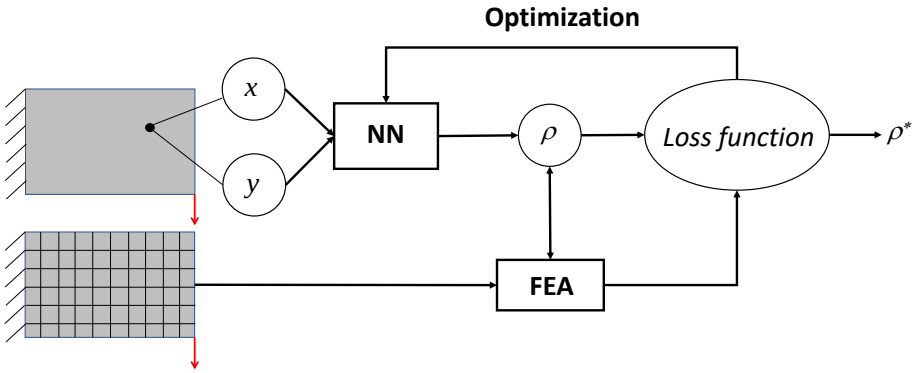

Fig. 2: Overview of the proposed TOuNN framework in 2D.

3.2 Neural Network

While there are various types of neural networks (NN), we employ here a simple fully-connected feed-forward NN [21]. The input to the network is either 2-dimensional (x, y) for 2D problems, or 3-dimensional (x, y, z) for 3D problems; see Figure 3. *The output of the NN is the density value ρ at that point. In other words, the NN is simply an*

evaluator that return the density value for any point within the domain; the value returned will depend on the weights, bias and activation functions, as described below.

The NN itself consists of a series of hidden layers associated with activation functions such as leaky rectified linear unit (LeakyReLU) [22], [23] coupled with batch normalization [24] (this is illustrated in Figure 3 for 2D problems; see description below). The final layer of the NN is a *classifier* layer with a softMax activation function that ensures that the density lies between 0 and 1.

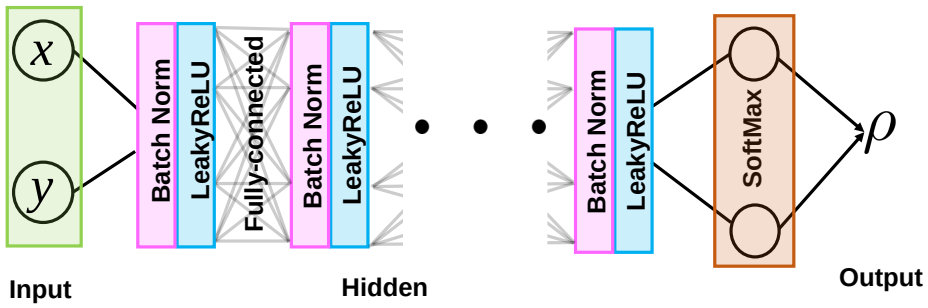

Fig. 3: The architecture of the proposed neural net for 2D problems.

As illustrated in Figure 3, the NN typically consists of several hidden layers (depth); each layer may consist of several activation functions (height). By varying these two, one can increase the representational capacity of the NN. As an example, Figure 4 illustrates a NN with a single hidden layer of height 2. Observe that each connection within the NN is associated with a weight, and each node is associated with an activation function and a bias. The output of any node is computed as follows. In Figure 4, the value of $z_1^{[1]}$ is first computed as $z_1^{[1]} = w_{11}^{[1]}x + w_{21}^{[1]}y + b_1^{[1]}$, where $w_{ij}^{[k]}$ is the weight associated to the j^{th} neuron in layer k from the i^{th} neuron in the previous layer, and $b_j^{[k]}$ is the bias associated with the j^{th} neuron in the k^{th} layer. Then, the output $a_1^{[1]}$ of the node is computed as $a_1^{[1]} = \sigma(z_1^{[1]})$ where σ is the chosen activation function. For example, the ReLU activation function is defined as: $\sigma(z) \equiv max(0, z)$. In this paper, we will rely on a variation of the ReLU, namely, LeakyReLU [23] that is differentiable. This and other differentiable activation functions are supported by various NN software libraries such as pyTorch [25].

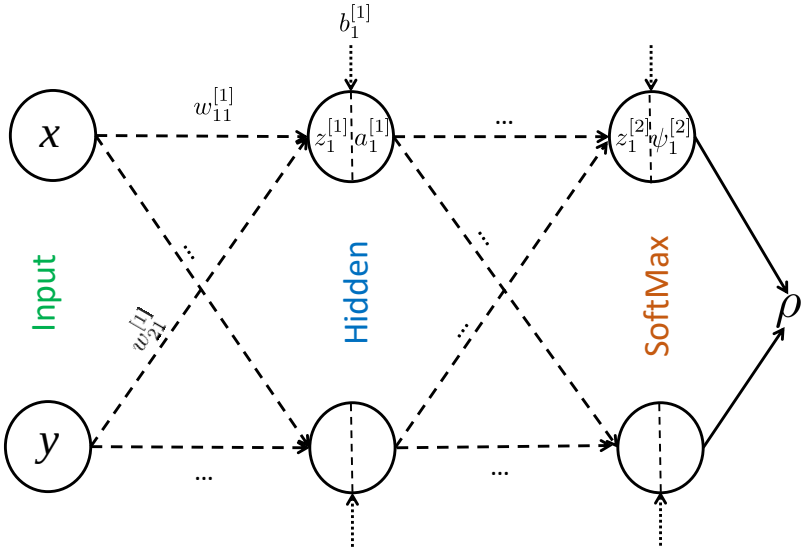

Fig. 4: Illustration of a simple network with one hidden layer of height 2.

The final layer, as mentioned earlier, is a softMax function that scales the outputs from the hidden layers to values between 0 and 1. For example, in Figure 4, we have $\psi_1^{[2]} = \frac{e^{z_1^{[2]}}}{e^{z_1^{[2]}} + e^{z_2^{[2]}}}$. The softMax function has many outputs as inputs; however, we will use only the first output, and interpret it as the density field ($\rho = \psi_1^{[2]}$); the remaining outputs are disregarded, but can be used, for example, in a multi-material setting.

As should be clear from the above description, once the activation function is chosen, the output $\rho(x, y)$ is defined globally, and determined solely by the weights and bias. We will denote the entire set of weights and bias by $\boldsymbol{w}$. Thus, the optimization problem using the NN may be posed as:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \boldsymbol{u}^{\mathsf{T}}\boldsymbol{K}(\boldsymbol{w})\boldsymbol{u} \tag{2a}$$

$$\text{subject to} \quad \boldsymbol{K}(\boldsymbol{w})\boldsymbol{u} = \boldsymbol{f} \tag{2b}$$

$$\sum_e \rho_e(\boldsymbol{w}) v_e = V^* \tag{2c}$$

The element density value $\rho_e(\boldsymbol{w})$ in the above equation is the density function evaluated at the center of the element.

3.3 Finite Element Analysis

For 2D finite element analysis, we use a regular 4 node quad element, and fast Cholesky factorization based on the CVXOPT library [26]. For 3D, we use a regular 8 node hexahedral element, and an assembly free deflated finite element solver [27] [28]. Note that the FE solver is outside of the NN (see Figure 2), and is treated as a black-box by the NN. During each iteration, the density at the center of each element is computed by the NN, and is provided to the FE solver. The FE solver compute the stiffness matrix for each element based on the density evaluated at the center of the element. *Note that, since the density function can be evaluated at multiple points within each element, it is possible to use advanced integration schemes to compute a more accurate estimate of element stiffness matrices; this is however not pursued in this paper.* The assembled global stiffness matrix is then used to compute the displacement vector $\boldsymbol{u}$, and the un-scaled compliance of each element:

$$J_e = \{u_e\}^T [K]_0 \{u_e\} \tag{3}$$

This will be used in sensitivity calculations as explained later on. The total compliance is given by:

$$J = \sum_e \rho_e^p J_e \tag{4}$$

where p is the usual SIMP penalty parameter.

3.4 Loss Function

In this section, we describe how the optimization problem is solved using standard NN capabilities. While the optimization is usually carried out using optimality criteria [29] or MMA [30], here we will rely on neural networks (NN). NNs are designed to minimize an unconstrained loss function using built-in procedures such as Adam optimization [31]. We therefore convert the constrained minimization problem in Equation 1 into an unconstrained

minimization problem by relying on the penalty formulation [32] to define the loss function as:

$$L(\boldsymbol{w}) = \frac{\boldsymbol{u}^\mathsf{T}\boldsymbol{K}\boldsymbol{u}}{J^0} + \alpha\left(\frac{\sum_e \rho_e v_e}{V^*} - 1\right)^2 \tag{5}$$

where α is a penalty parameter, and J^0 is the initial compliance of the system, used here for scaling. As described in [32], the solution of the constrained problem is obtained by minimizing the loss function as follows. Starting from a small positive value for the penalty parameter α, a gradient driven step is taken to minimize the loss function. Then, the penalty parameter is increased and the process is repeated. Observe that, in the limit $\alpha \to \infty$, when the loss function is minimized, the equality constraint is satisfied and the objective is thereby minimized [32]. In practice, a maximum value of 100 is usually assigned for α [32]; the complete update scheme is described later in Algorithm 1. Other methods such as the augmented Lagrangian [32] may also be used to convert the equality-constrained problem into an equivalent unconstrained problem. While this paper largely focuses on an equality constrained topology optimization problems using NN, more recently, researchers have also solved generic optimization problems with inequality-constraints using NN [33], [34]. Adapting these techniques for TO is a topic of future research.

3.5 Sensitivity Analysis

We now turn our attention to sensitivity analysis, a critical part of any optimization framework including NN. NNs rely on backpropagation [35] to analytically compute [36], [37], [38], [25] the sensitivity of loss functions with respect to the weights and bias. This is possible since the activation functions are analytically defined, and the output can be expressed as a composition of such functions.

Thus, in theory, once the network is defined, no additional work is needed to compute sensitivities; it can be computed automatically (and analytically) via backpropogation! However, in the current scenario, the FE implementation is outside of the NN (see Figure 2). Therefore, we need to compute some of the sensitivity terms explicitly.

Note that the sensitivity of the loss function with respect to a particular design variable w_i is given by:

$$\frac{\partial L}{\partial w_i} = \sum_e \frac{\partial L}{\partial \rho_e}\frac{\partial \rho_e}{\partial w_i} \tag{6}$$

The second term $\dfrac{\partial \rho_e}{\partial w_i}$ can be computed analytically by the NN through backpropogation since the density dependence on the weights is entirely part of the NN. On the other hand, the first term involves both the NN and the FE black-box, and must therefore be explicitly provided. Note that:

$$\frac{\partial L}{\partial \rho_e} = \frac{1}{J^0}\frac{\partial}{\partial \rho_e}(\boldsymbol{u}^\mathsf{T}\boldsymbol{K}\boldsymbol{u}) + \frac{2\alpha v_e}{V^*}\left(\frac{\sum_k \rho_k v_k}{V^*} - 1\right) \tag{7}$$

Further, recall that [39], [3]:

$$\frac{\partial}{\partial \rho_e}(\boldsymbol{u}^\mathsf{T}\boldsymbol{K}\boldsymbol{u}) = -p\rho_e^{p-1}J_e \tag{8}$$

where J_e is the element-wise un-scaled compliance defined earlier. Thus one can now compute the desired sensitivity as follows:

$$\frac{\partial L}{\partial w_i} = \frac{1}{J^0}\left[-p\sum_e \rho_e^{p-1}J_e\frac{\partial \rho_e}{\partial w_i}\right] + \left[\frac{2\alpha}{V^*}\left(\frac{\sum_k \rho_k v_k}{V^*} - 1\right)\sum_e \frac{\partial \rho_e}{\partial w_i}v_e\right] \tag{9}$$

Due to the compositional nature of the NN, the gradient is typically stabilized using gradient-clipping [40].

4 Algorithm

In this section, the proposed framework is summarized through an explicit algorithm. We will assume that the NN has been constructed with a desired number of layers, nodes per layer and activation functions. Here, we use pyTorch [25] to implement the NN. The weights and bias of the network are initialized using Glorot normal initialization [41].

The first step in the algorithm is to sample the domain at the center of each element; this is followed by the initialization of the penalty parameter α and the SIMP parameter p. In the main iteration, the element densities are computed using the NN using the current values of $\boldsymbol{w}$. These densities are then used by the FE solver to solve the structural problem, and to compute the un-scaled element compliances J_e defined in Equation 3. Further, in the first iteration, a reference compliance J^0 is also computed for scaling purposes. Then, the loss function is computed using Equation 5 and the sensitivities are computed using Equation 9. The weights $\boldsymbol{w}$ are then updated using the built-in optimizer (here Adam optimizer). This is followed by an update of the penalty parameter α. Finally, we use the continuation scheme where the parameter p is incremented to avoid local minima [42], [43]. The process is then repeated until termination. In typical mesh-based density optimization, the algorithm terminates if the maximum change in element density is less than a prescribed value. Here, since the density is a globally function, the algorithm is set to terminate if the percentage of grey elements (elements with densities between 0.05 and .95) $\epsilon_g = N_{grey}/N_{total}$ is less than a prescribed value. Through experiments, we observed that this criteria is robust and consistent with the formulation.

Algorithm 1 TOuNN

procedure TopOpt(NN, Ω^h, V^*) ▷ NN, discretized domain, desired vol
  $\boldsymbol{x} = \{x_e, y_e\}_{e\in\Omega^h}$ or $\{x_e, y_e, z_e\}_{e\in\Omega^h}$ ▷ center of elements in FE mesh; optimization input
  epoch = 0; $\alpha = \alpha_0$; $p = p_0$ ▷ Penalty factor initialization
  $J_0 \leftarrow FEA(\boldsymbol{\rho} = v_f^*, \Omega^h)$ ▷ FEA with uniform gray
  repeat ▷ Optimization (Training)
    $\boldsymbol{\rho} = NN(\boldsymbol{x})$ ▷ Call NN to compute $\boldsymbol{\rho}$ at $\boldsymbol{p}$; (Forward propagation)
    $J_e \leftarrow FEA(\boldsymbol{\rho}, \Omega^h)$ ▷ Solve FEA
    $\mathrm{L} = \frac{\sum_e \rho_e^p J_e}{J_0} + \alpha\left(\frac{\sum_e \rho_e v_e}{V^*} - 1\right)^2$ ▷ Loss function
    Compute ∇L ▷ Sensitivity(Backward Propagation)
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w}(\nabla L)$ ▷ Adam optimizer step
    $\alpha \leftarrow \min(\alpha_{max}, \alpha + \Delta\alpha)$ ▷ Increment α
    $p \leftarrow \min(p_{max}, p + \Delta p)$ ▷ Continuation
    epoch $\leftarrow$ epoch + 1
  until $\epsilon_g < \epsilon_g^*$ ▷ Check for convergence

Further, once the algorithm terminates, the density function can be sampled at a finer resolution to extract crisp boundaries, as illustrated below through numerical experiments. Finally, one can easily and accurately compute the gradient of the density field to compute, for example, boundary normals.

5 Numerical Experiments

In this section, we conduct several numerical experiments to illustrate the TOuNN framework and algorithm. The 2D implementation is in Python, and uses the pyTorch library [25] for the neural network. The 3D implementa-

tion is in C++, and uses the C++ implementation of pyTorch. The default parameters in the implementation are as follows.

- The material properties were $E = 1$, $E_{min} = 1e^{-6}$ and $\nu = 0.3$.
- A mesh size of 60×30 was used for all 2D experiments, unless otherwise stated.
- The NN is composed of 5 layers with 20 nodes (neurons) per layer, for both 2D and 3D, unless otherwise specified. The number of design variables, i,e, the size of $\boldsymbol{w}$, is 1782, that corresponds approximately to the total number of 2D elements in a $60x30$ mesh; this is equal the number of design variables in typical SIMP based formulations.
- The LeakyReLU was chosen as the activation function for all nodes.
- The learning rate for the Adam optimizer was set at the recommended value of 0.01 [44].
- A threshold value for gradient clipping was also set at the recommended value of 0.1 [40].
- The α parameter is updated as follows: $\alpha_0 = 0.1$, $\alpha_{max} = 100$ and $\Delta\alpha = 0.05$.
- The p parameter is updated as follows: $p_0 = 2.0$, $p_{max} = 4$ and $\Delta p = 0.01$.
- The termination criteria were as follows: $\epsilon_g^* = 0.035$; a maximum of 499 iterations is also imposed.
- After termination, the density in each element was sampled on a 15×15 grid to extract the topology.

All experiments were conducted on a Intel i7 - 6700 CPU @ 2.6 Ghz with 16 GB of RAM. Through the experiments, we investigate the following.

1. *Validation*: The first task to validate the TOuNN framework by comparing the computed topologies for standard 2D benchmark problems, against those obtained via established methods [26]. Typical convergence plots for the loss function, objective and constraint are also included.
2. *Computational Cost*: Typical computational costs are tabulated and compared.
3. *NN Dependency*: Next, we vary the NN size (depth and width), and study its impact on the computed 2D topologies.
4. *Mesh Dependency*: Similarly, we vary the FE mesh size in 2D and study its impact on the topology.
5. *High Resolution Boundary*: Boundary extraction via post-process sampling of the density field is illustrated through examples.
6. *Three-dimensional*: The proposed methodology is demonstrated using a 3D example.
7. *Design and Manufacturing Constraints*: Finally, we explore extension of the presented framework to include manufacturing constraints that render the obtained designs manufacturable.

5.1 Validation

We begin by comparing the topologies obtained from the proposed framework, against those obtained using the popular 88-line implementation of SIMP-based optimization [26]. With the default parameters listed above, and with a filtering radius of 2.0 [26], the results are summarized in Figure 5, where v_f^* is the desired volume fraction. We observe that although the topologies are sometimes different (due to the infinitely many solutions), the compliances are marginally lower (better) for the TOuNN framework. The number of finite element operations is also summarized for each example. The sharpness of the boundary for the TOuNN framework stems from the analytic representation of the density field.

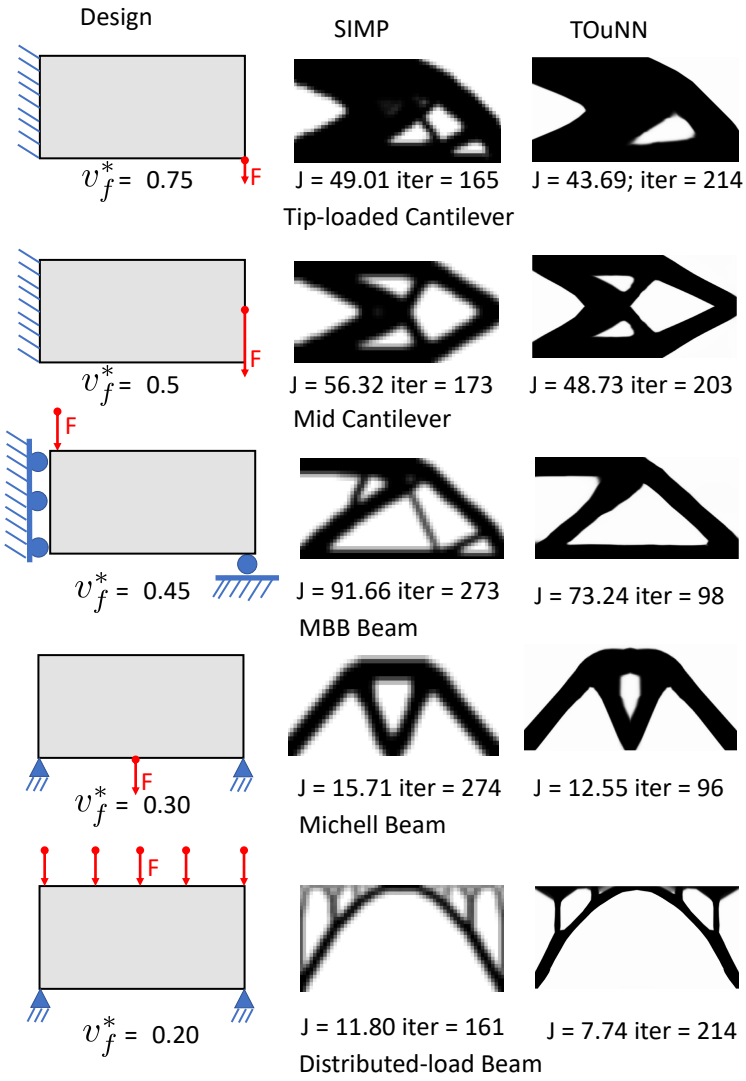

Fig. 5: Validation of TOuNN.

Typical convergence plots for the mid-loaded cantilever beam is illustrated in Figure 6, and the convergence plot for the Michell beam is illustrated in Figure 7. In the two figures, the relative number of grey elements denotes the ratio of elements with densities between 0.05 and .95, to the total number of elements, i.e., $\epsilon_g = N_{grey}/N_{total}$. As described at the beginning of this section, optimization terminates when $\epsilon_g < 0.035$.

Finally, Table 1 summarizes the number of FE iterations for all 5 examples, for a wide range of volume fractions. We could not observe any pattern. However, for the distributed-load problem, the algorithm failed to converge for two of the volume fractions. This is discussed further later in the paper.

Volume Fraction →	0.90	0.85	0.70	0.50	0.35	0.25	0.15
Tip-loaded cantilever	103	181	256	128	129	115	129
Mid-loaded cantilever	81	170	136	150	141	161	177
MBB beam	176	248	177	113	98	106	121
Michell beam	122	96	72	54	112	75	105
Distributed load bridge	350	176	51	499*	499*	218	214

Table 1: The number of iterations for various examples; (*) denotes lack of convergence.

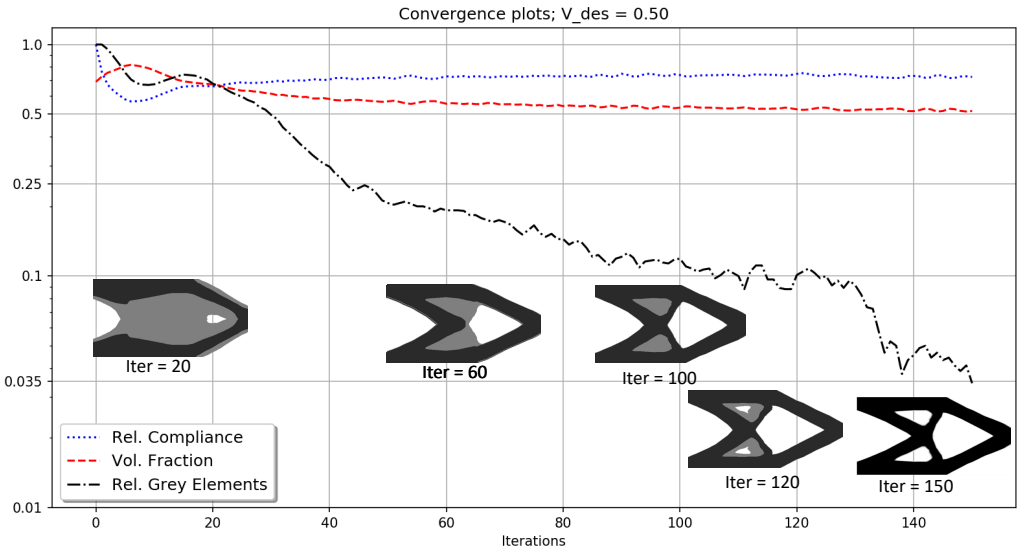

Fig. 6: Convergence plots for the problem in Figure 5 mid-cantilever.

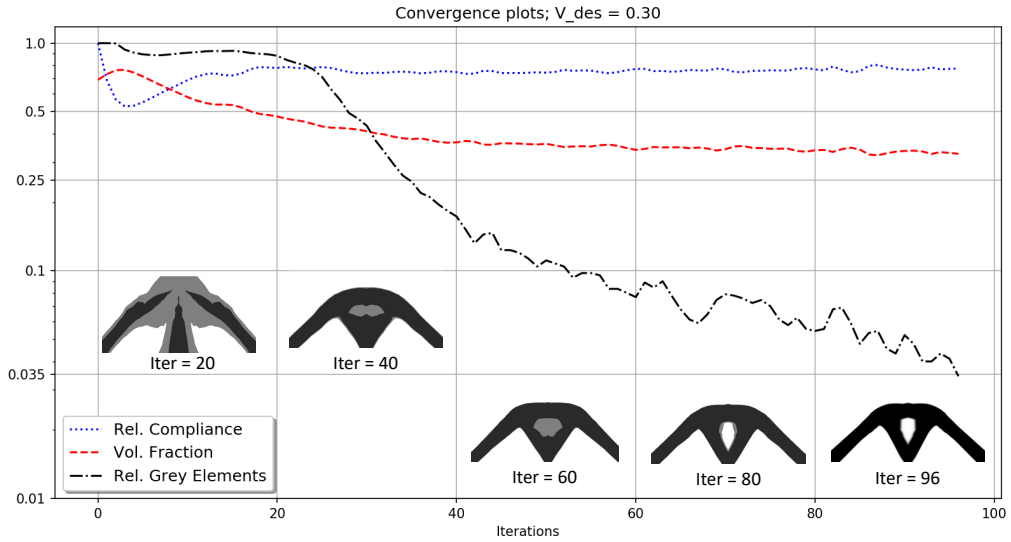

Fig. 7: Convergence plots for the problem in Figure 5 Michell beam.

5.2 Computational Cost

Next, we briefly summarize the computational costs in TOuNN. Recall that the framework (see Figure 2) can be separated into the following components:

(i) **Forward** : Computing density values at the sampled points, a.k.a, forward propagation.

(ii) **FEA** : Finite element analysis to compute displacements, compliance, etc.

(iii) **Wt.Update** : Computing the loss function, gradient clipping and updating weights through back-propagation (includes sensitivity analysis).

(iv) **Other** : Computing the convergence criteria and other book-keeping tasks.

Table 2 summarizes the time taken for each example, and for each of the components. For the default mesh size of 60x30, FEA consumes approximately 50% of the total computation. For larger mesh sizes, the percentage increases

as one would expect. We note that the cost of each TOuNN iteration is roughly twice the cost of each iteration in the 88-line code [26].

Example (Vol.Frac)	Forward	FEA	Wt. update	Other	Total Time (Iter)	Total Time [26](Iter)
Tip Cantilever (0.75)	0.79	4.49	2.01	1.25	8.54 (214)	4.32 (165)
Mid Cantilever (0.5)	0.75	4.2	1.98	1.40	8.34 (203)	4.51 (173)
MBB (0.45)	0.42	3.13	1.25	1.44	6.23 (98)	10.93 (273)
Michell (0.3)	0.42	2.97	1.21	1.61	6.21 (96)	10.68 (274)
Distributed beam (0.2)	0.68	4.55	1.86	1.76	8.85 (214)	4.31 (161)

Table 2: Time taken in seconds for each component of the framework, for each example.

5.3 NN Size Dependency

The functional representation of a neural network (here the density function) is global, highly non-linear, and evades simple characterization [45], [46], [47]. It is therefore difficult to predict *a priori* the minimum size of the neural network (depth and height) required to capture a particular topology, except in trivial scenarios. One such scenario is illustrated in Figure 8a where the expected topology is a tensile bar. Due to the simplicity of the topology a neural network with a single hidden layer with a single node (size: 1 x 1) is sufficient to capture the topology of volume fraction 0.4 as illustrated in Figure 8b. The number of design variables, i.e., the size of $\boldsymbol{w}$, associated with this 1 x 1 network is 7.

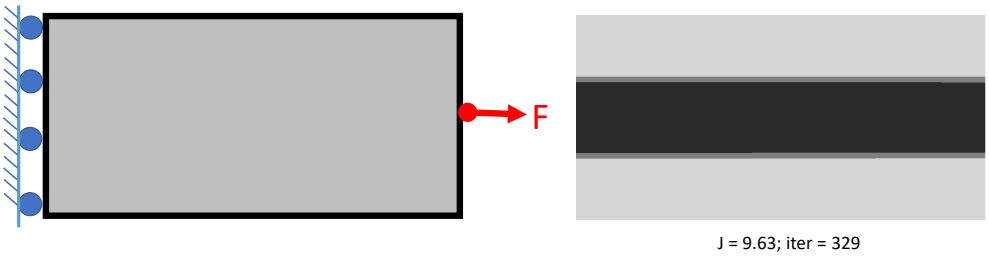

Fig. 8: The tensile load problem; a neural net of size 1 x 1 is sufficient to capture the topology.

As expected, for other non-trivial scenarios, a larger NN is required. To illustrate, we compute the topology for the tip-cantilever problem posed in Figure 5, with varying neural net size (depth and height), keeping everything else at default values, as listed at the beginning of this section. The results are summarized in Figure 9, where a NN: 2 x 8 (114) implies that the neural network has 2 layers, with each layer having 8 nodes, and the total number of design variables is 114. The algorithm did not converge when the NN size was too small (2 x 8). We observe that the complexity of the resulting topology remains unchanged with increasing NN-size, while the computational cost (as captured by seconds/iteration) is weakly dependent on the NN size. It is a reasonable to expect that the minimum feature size will depend on the size of the neural network; further investigation is needed to validate this hypothesis.

5.4 Effect of Mesh Size

We next briefly study the effect of the mesh size on the computed topology; all other parameters were kept constant at default values. Since the density field is independent of the mesh, the computed topology can also be

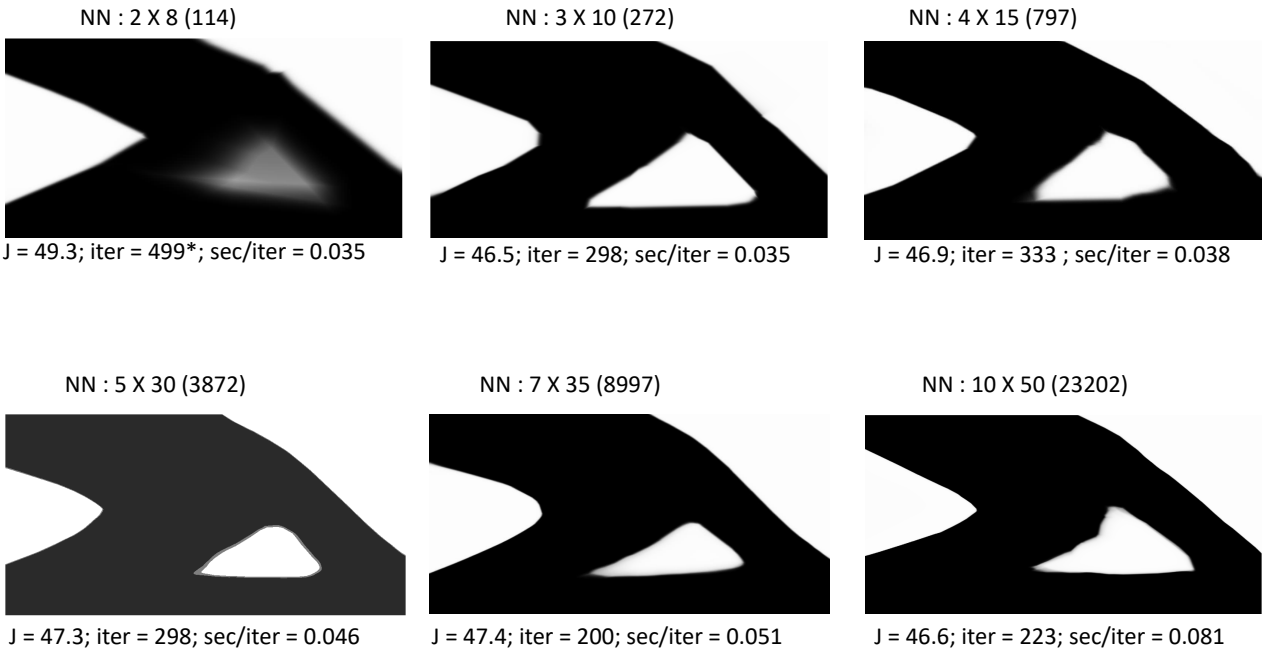

Fig. 9: Optimized topologies for the tip-loaded cantilever beam, for varying neural net size: depth x height, with number of design variables in parenthesis.

expected to be independent of the mesh (provided the disretization is fine enough to capture the underlying physics). This can be observed in the results illustrated in Figure 10.

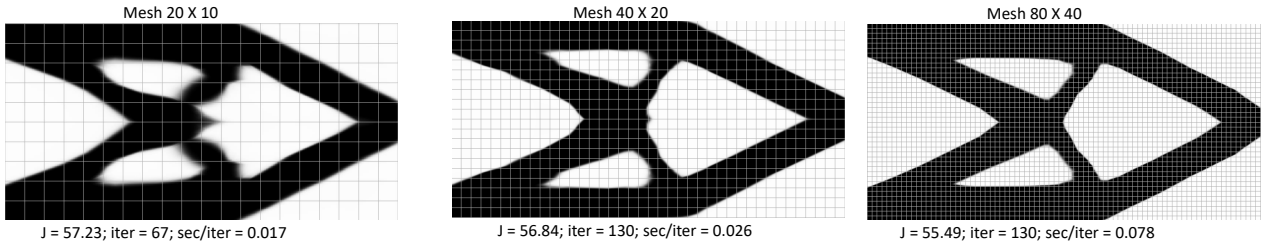

Fig. 10: Impact of mesh size on the computed topology.

5.5 High Resolution Boundary

One of the key challenges in TO is boundary resolution. This is often achieved by increasing the size of the mesh (which increases the computational cost), or by employing multi-resolution schemes [48] [49]. In the proposed framework, one can extract a high resolution boundary at no additional cost as follows. First the optimization is carried out; then the optimized (i.e. trained) weights, i.e., w^*, are used to sample the domain at a fine resolution as illustrated in Figure 11.

5.6 Extension to 3D

We now consider the implementation of the TOuNN framework to 3D. To achieve this, the fundamental changes are: (1) we add an input neuron corresponding to the z coordinate, and (2) we use a 3D FEA solver; this is illustrated in Figure 12. All other aspects of the NN remain unchanged.

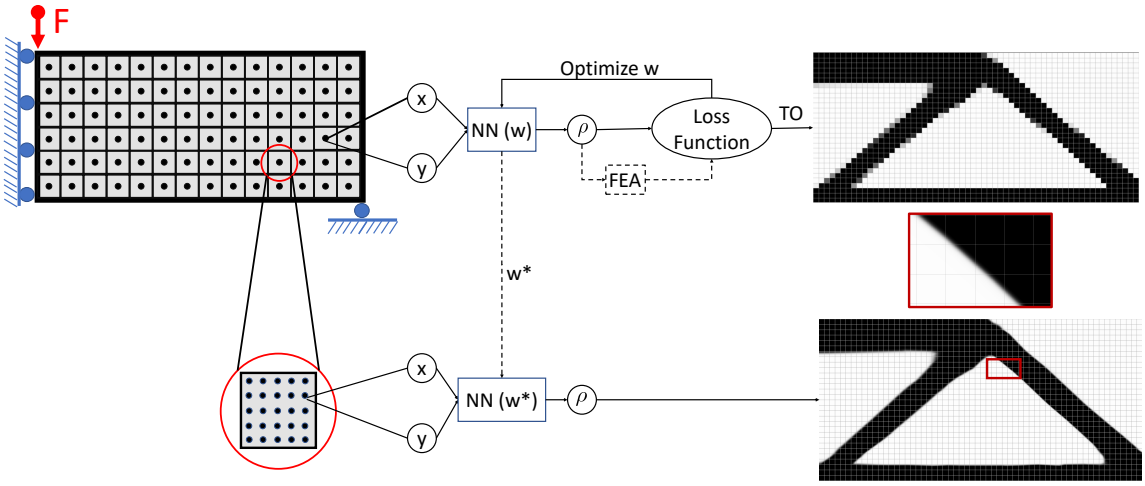

Fig. 11: High resolution boundary is extracted by first optimizing, and then using the optimized weights to sample the density at a high resolution.

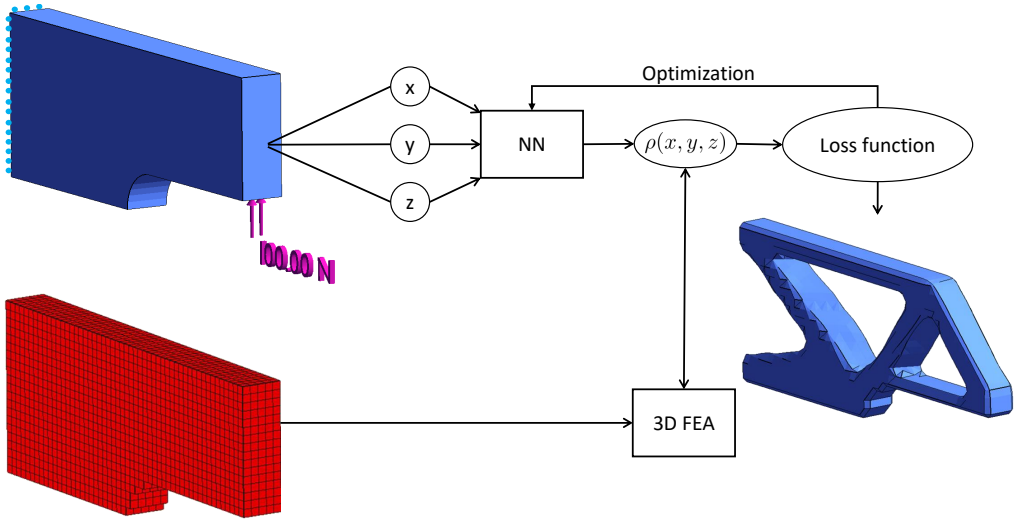

Fig. 12: Implementation of TOuNN in 3D.

To validate the 3D implementation, we consider the edge-cantilever beam problem [50] posed in Figure 13a. The domain is discretized by a 15 x 30 x 9 hexahedral mesh. The topology obtained and the compliance using the code provided by [50] is illustrated in Figure 13b. In our formulation we used an identical mesh; all other parameters including the NN size were set at default values. The computed topology and compliance is illustrated in Figure 13c. For this example, the 3D TOuNN framework takes fewer iterations and leads to a design with comparable compliance. Further, the number of finite elements is 4050, while the size of NN, i.e., the number of design variables, is 1800.

5.7 Design and Manufacturing Constraints

Typically, design and manufacturing constraints are imposed in TO through projection operators [51] that act on the density field ρ. Similarly, in the current formulation, we introduce input and output projection operators as illustrated in Figure 14. As explained below, certain types of constraints can be handled as input projections, and others as output projections.

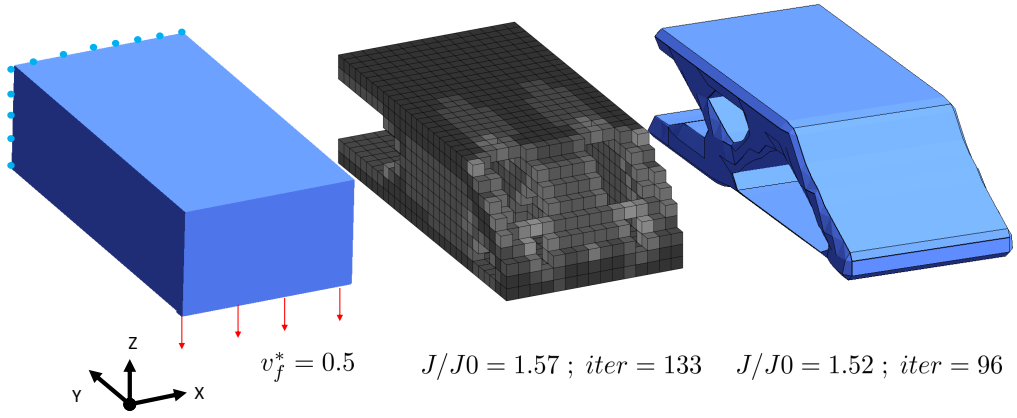

Fig. 13: (a) 3D edge cantilever [50]. (b) Computed topology using [50]. (c) Computed topology using 3D TOuNN.

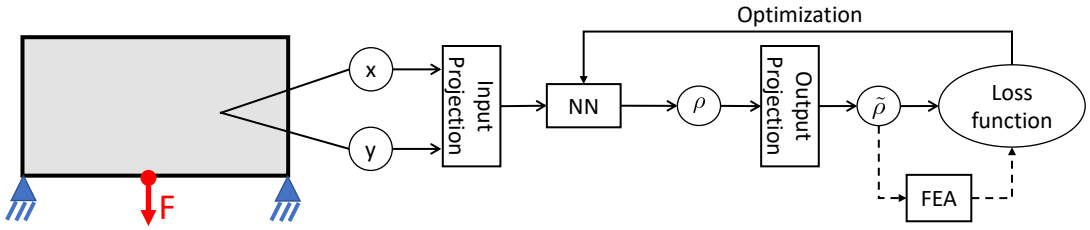

Fig. 14: Input and output projections can be used to impose design/manufacturing constraints.

5.7.1 Symmetry

First we consider classic symmetry constraints within this framework. To enforce symmetry about, say, the x axis passing through (x_0, y_0), we transform the y coordinate as follows:

$$y \leftarrow y_0 + |y - y_0| \tag{10}$$

In other words, we treat symmetry as an input projection. No other change is needed to the framework. As an example, Figure 15 illustrates imposing symmetry about the y axis for a tip-loaded cantilever beam. For comparison, see Figure 5 where the symmetry was not imposed. Imposing symmetry results in a 13% increase in compliance.

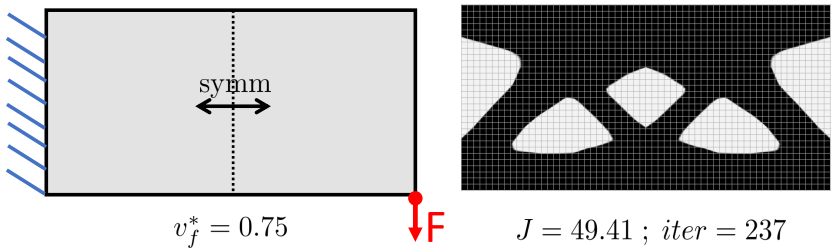

Fig. 15: Symmetry about the y axis for a tip-cantilever beam

5.7.2 Non-Design Region

We now consider the *non-design constraint* where material *should not* be removed from certain a region Ω_N. Essentially, the density value should be forced to a value of 1 in this region. However, to facilitate backward propagation, i.e., sensitivity analysis, we treat this as an output projection via a functional approximation of the max operator:

$$\rho \leftarrow 0.5(\rho + |1 - \rho|) \quad \{x, y\} \in \Omega_N \tag{11}$$

As an example, Figure 16a illustrates the Michell beam problem with a non-design constraint, where material should not be removed from an annular region as shown. The computed topology is illustrated in Figure 16b. Imposing this constraint results in a 14% increase in compliance, compared to the design without this constraint in Figure 5.

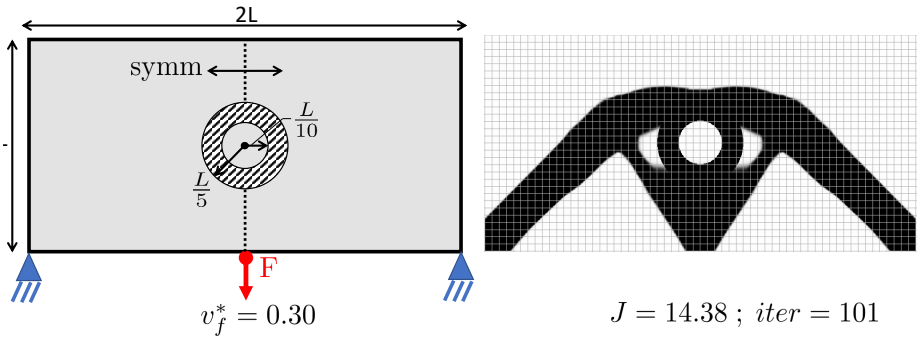

Fig. 16: (a) Michell beam problem with a non-design constraint. (b) Optimized topology that satisfies the constraint.

5.7.3 Extrusion

The final constraint explored here is extrusion in 3D, where the cross section of the topology must not change along the extruded direction. Extrusion constraint can be easily implemented here as an input projection. Specifically, suppose the extruded direction is x, then the x input is simply discarded in the 3D NN architecture; everything else remains the same (including the use of 3D FEA):

$$(y, z) \leftarrow (x, y, z) \tag{12}$$

As an example, Figure 17a illustrates a simple design problem in 3D; the desired volume fraction is 0.5. Figure 17b illustrates the optimized topology without extrusion constraint, while Figure 17c illustrates the topology with an extrusion constraint along x. The extrusion constraint leads to a 6% increase in compliance, but takes fewer iterations.

6 Opportunities and Challenges

There are several extensions and opportunities that we foresee. A possible extension is multi-material topology optimization; this would entail extending the NN *output* to multiple dimensions. A potentially rich opportunity is

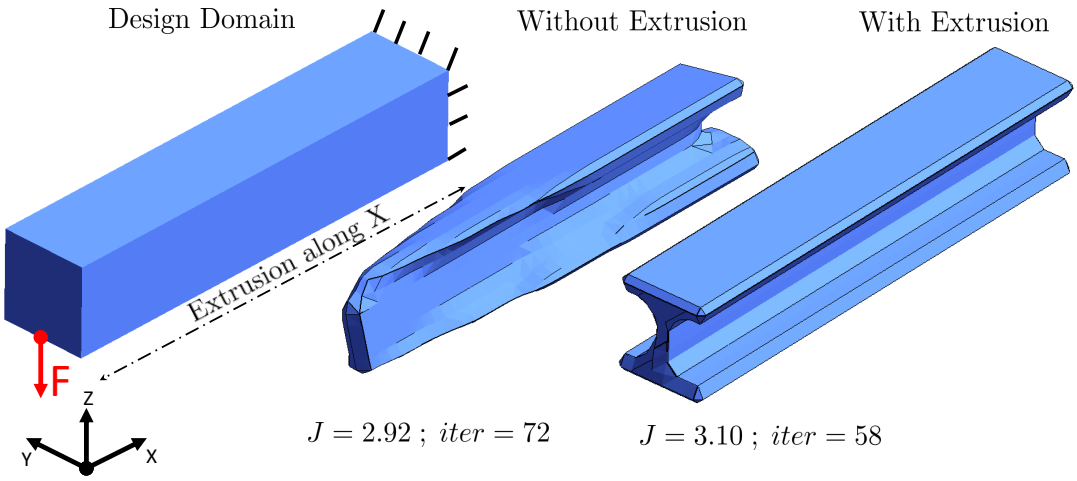

Fig. 17: (a) A 3D beam problem. (b) Optimized topology without extrusion constraint. (c) Optimized topology with extrusion constraint along x.

in the infinitely differentiable density function. This translate into accurate computation of boundary normal, that could be useful in several applications.

We also observed some challenges. For example, the lack of detailed features in the computed topologies (see Figure 5) could be of concern in certain applications. This limitation directly stems from the use of global activation functions in TOuNN. It may be possible to overcome this by exploring other activation functions.

A second (and related) challenge is the handling of distributed loads. For example, consider the distributed load problem in Figure 18; the topologies computed via 88-line [26] and TOuNN are also illustrated. While the 88-line code [26] converged to a possible solution (with gray regions), TOuNN failed to converge due to the imposed termination criteria of requiring close to zero grey elements. Alternate termination criteria suitable for our framework are being explored. One simple solution, we observed, was to increase the penalization factor; alternately, one can penalize presence of gray elements through the loss function.

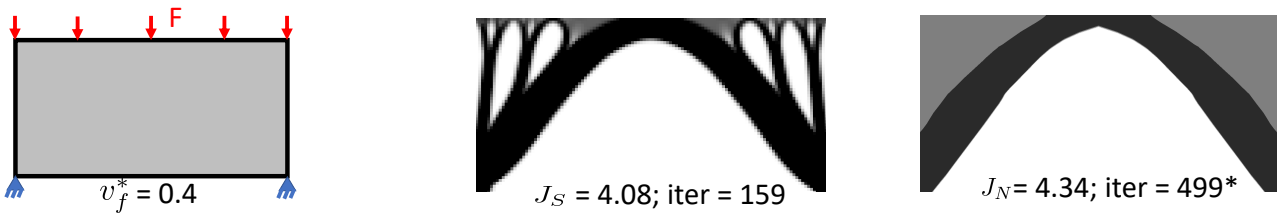

Fig. 18: Distributed loads leads to gray regions, and may therefore fail to converge.

In this paper, a simple volume-constrained compliance minimization was explored; extensions to handle other objectives with multiple constraints are currently being explored; see [33], [34].

An open and interesting question is the geometric interpretation of the NN design variables. However, even for the simple tensile bar problem in Figure 8a where a NN of 1 x 1 was used, it was difficult to geometrically interpret the weights and bias values. This can be attributed to the use of non-linear softMax function at the output. Alternate output functions need to be explored.

Checkerboard patterns commonly occur in SIMP based topology optimization [52], [53], and restriction methods such as filtering, perimeter control, etc. are often used in this context [43]. However, in the current formulation, no checkerboard patterns were observed in any of the experiments. However, it is possible that checkerboard might begin to appear if the NN size is increased significantly; further investigation is needed.

In conclusion, in this paper, a direct topology optimization framework using a conventional neural network (NN) was proposed. The salient features of the framework are: (1) the direct use of NN's activation functions, (2) exploiting built-in backpropogation for sensitivity analysis, (3) implicit filtering, (4) sharpness of the boundary, and (5) scope for relearning. The framework was validated and characterized through several benchmark problems in 2D and 3D. However, several challenges remain as highlighted above.

7 Replication of Results

The Python code used in generating the examples in this paper is available at www.ersl.wisc.edu/software/TOuNN.zip.

Acknowledgments

The authors would like to thank the support of National Science Foundation through grant CMMI 1561899. Prof. Suresh is a consulting Chief Scientific Officer of SciArt, Corp.

Compliance with ethical standards

The authors declare that they have no conflict of interest.

References

1. M. P. Bendsoe and O. Sigmund. *Topology optimization: theory, methods, and applications*. Springer Berlin Heidelberg, 2 edition, 2003.
2. M. P. Bendsoe et al. An Analytical Model to Predict Optimal Material Properties in the Context of Optimal Structural Design. *Journal of Applied Mechanics*, 61(4):930, dec 2008.
3. O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127, 2001.
4. A. Chandrasekhar et al. Build optimization of fiber-reinforced additively manufactured components. *Structural and Multidisciplinary Optimization*, 61(1):77–90, jan 2020.
5. M. Y. Wang et al. A level set method for structural topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 192(1-2):227–246, jan 2003.
6. Y. M. Xie and G. P. Steven. A simple evolutionary procedure for structural optimization. *Computers & structures*, 49(5):885–896, 1993.
7. K. Suresh. Efficient generation of large-scale pareto-optimal topologies. *Structural and Multidisciplinary Optimization*, 47(1):49–61, jan 2013.
8. S. Deng and K. Suresh. Multi-constrained topology optimization via the topological sensitivity. *Structural and Multidisciplinary Optimization*, 51(5):987–1001, may 2015.
9. A. M. Mirzendehdel et al. Strength-based topology optimization for anisotropic parts. *Additive Manufacturing*, 19:104–113, jan 2018.
10. A. M. Mirzendehdel and K. Suresh. A pareto-optimal approach to multimaterial topology optimization. *Journal of Mechanical Design*, 137(10):101701, 2015.
11. K. Gurney. *An introduction to neural networks*. CRC press, 1997.
12. S. Banga et al. 3D Topology Optimization using Convolutional Neural Networks. *arXiv preprint arXiv:1808.07440*, aug 2018.
13. I. Sosnovik and I. Oseledets. Neural networks for topology optimization. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 34(4):215–223, aug 2019.
14. E. Ulu et al. A data-driven investigation and estimation of optimal topologies under variable loading configurations. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 4(2):61–72, 2016.

15. X. Lei et al. Machine learning-driven real-time topology optimization under moving morphable component-based framework. *Journal of Applied Mechanics*, 86(1), 2019.
16. Z. Nie et al. TopologyGAN: Topology Optimization Using Generative Adversarial Networks Based on Physical Fields Over the Initial Domain. *arXiv preprint arXiv:2003.04685*, mar 2020.
17. Y. Yu et al. Deep learning for determining a near-optimal topological design without any iteration. *Structural and Multidisciplinary Optimization*, 59(3):787–799, mar 2019.
18. Q. Lin et al. Investigation into the topology optimization for conductive heat transfer based on deep learning approach. *International Communications in Heat and Mass Transfer*, 97:103–109, oct 2018.
19. Y. Zhang et al. A deep convolutional neural network for topology optimization with strong generalization ability. *arXiv preprint arXiv:1901.07761*, 2019.
20. S. Hoyer et al. Neural reparameterization improves structural optimization. *arXiv preprint arXiv:1909.04240*, 2019.
21. C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
22. I. Goodfellow et al. *Deep Learning*. MIT Press, 2016.
23. L. Lu et al. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.
24. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on Machine Learning, ICML 2015*, volume 1, pp. 448–456. International Machine Learning Society (IMLS), feb 2015.
25. A. Paszke et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.
26. E. Andreassen et al. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, jan 2011.
27. P. Yadav and K. Suresh. Large Scale Finite Element Analysis via Assembly-Free Deflated Conjugate Gradient. *Journal of Computing and Information Science in Engineering*, 14(4):041008, Oct 2014.
28. A. M. Mirzendehdel and K. Suresh. A Deflated Assembly Free Approach to Large-Scale Implicit Structural Dynamics. *Journal of Computational and Nonlinear Dynamics*, 10(6):061015, Nov 2015.
29. M. P. Bendsøe and O. Sigmund. *Optimization of structural topology, shape, and material*, volume 414. Springer, 1995.
30. K. Svanberg. The method of moving asymptotes—a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2):359–373, 1987.
31. D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, dec 2015.
32. J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
33. H. Kervadec et al. Constrained deep networks: Lagrangian optimization via log-barrier extensions. *arXiv preprint arXiv:1904.04205*, 2019.
34. P. Márquez-Neila et al. Imposing hard constraints on deep networks: Promises and limitations. *arXiv preprint arXiv:1706.02025*, 2017.
35. D. E. Rumelhart et al. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
36. A. G. Baydin et al. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.
37. S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
38. M. Abadi et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*, mar 2016.
39. M. P. Bendsoe and O. Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.
40. R. Pascanu et al. Understanding the exploding gradient problem. *Computing Research Repository*, abs/1211.5063, 2012.
41. X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.
42. S. Rojas-Labanda and M. Stolpe. Automatic penalty continuation in structural topology optimization. *Structural and Multidisciplinary Optimization*, 52(6):1205–1221, dec 2015.
43. O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural Optimization*, 16(1):68–75, 1998.

44. L. N. Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.
45. R. Arora et al. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.
46. G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
47. E. D. Sontag. Vc dimension of neural networks. *NATO ASI Series F Computer and Systems Sciences*, 168:69–96, 1998.
48. T. H. Nguyen et al. A computational paradigm for multiresolution topology optimization (MTOP). *Structural and Multidisciplinary Optimization*, 41(4):525–539, apr 2010.
49. J. P. Groen et al. Higher-order multi-resolution topology optimization using the finite cell method. *International Journal for Numerical Methods in Engineering*, 110(10):903–920, jun 2017.
50. K. Liu and A. Tovar. An efficient 3D topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, 50(6):1175–1196, dec 2014.
51. S. L. Vatanabe et al. Topology optimization with manufacturing constraints: A unified projection-based approach. *Advances in Engineering Software*, 100:97–112, oct 2016.
52. A. Díaz and O. Sigmund. Checkerboard patterns in layout optimization. *Structural Optimization*, 10(1):40–45, aug 1995.
53. O. Sigmund and K. Maute. Topology optimization approaches: A comparative review, dec 2013.