Proceedings of the ASME 2020
International Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
IDETC/CIE2020
August 17-19, 2020, Virtual, Online

DETC2020-22399

GENERATIVE ADVERSARIAL NETWORKS WITH SYNTHETIC TRAINING DATA FOR ENFORCING MANUFACTURING CONSTRAINTS ON TOPOLOGY OPTIMIZATION

Michael Greminger[1]
University of Minnesota Duluth
Duluth, MN

ABSTRACT

Topology optimization is a powerful tool to generate mechanical designs that use minimal mass to achieve their function. However, the designs obtained using topology optimization are often not manufacturable using a given manufacturing process. There exist some modifications to the traditional topology optimization algorithm that are able to impose manufacturing constraints for a limited set of manufacturing methods. These approaches have the drawback that they are often based on heuristics to obtain the manufacturability constraint and thus cannot be applied generally to multiple manufacturing methods. In order to create a general approach to imposing manufacturing constraints on topology optimization, generative adversarial networks (GANs) are used. GANs have the capability to produce samples from a distribution defined by training data. In this work, the GAN is trained by generating synthetic 3D voxel training data that represent the distribution of designs that can be created by a particular manufacturing method. Once trained, the GAN forms a mapping from a latent vector space to the space of manufacturable designs. The topology optimization is then performed on the latent vector space ensuring that the design obtained is manufacturable. The effectiveness of this approach is demonstrated by training a GAN on designs intended to be manufacturable on a 3-axis computer numerically controlled (CNC) milling machine.

Keywords: Topology optimization, design for manufacturability, deep learning, manufacturing constraints, generative adversarial networks

1. INTRODUCTION

Topology optimization is a powerful tool for creating efficient designs that minimize material usage [1]. While topology optimization can generate an optimal design for a particular loading, the designs that result are often not manufacturable using conventional manufacturing methods. There exist modifications to traditional topology optimization algorithms that can impose manufacturing constraints, such as symmetry or pull direction, but these approaches are not generalizable to other manufacturing methods [2, 3]. There is a need for a general approach to imposing manufacturing constraints on topology optimization that can handle a large variety of manufacturing methods. Generative adversarial networks (GANs) are a potential method to achieve this goal due to their ability to approximate a distribution of multidimensional input data used during training [4]. GANs have been shown to be able to generate realistic images of faces and other objects. GANs have also been extended to 3D objects using a voxel representation [5].

Generative approaches to design are not new and this work builds on these previous approaches. A proven approach to generative design is to use a grammar defined parametric model approach with stochastic optimization as described by Shea et al. [6]. This approach generates efficient designs but it is best suited to frame structures or to objects manufactured using flexible manufacturing methods such as 3D printing. Another successful approach described by Matejka et al. [7] builds upon existing topology optimization approaches by varying the parameters of the topology optimization such as voxel size, load distribution, and target volume. These variations result in a large family of thousands of potential designs. The innovation with this approach is how the designs are presented to the user in a way to aid in design selection. Manufacturing constraints can be a way to curate the designs generated by this approach, but the designs are not constrained to be manufacturable by any particular manufacturing method during the generation process. Additionally, GANs have been explored has a means to generate designs. Oh et al. [8] presented working show GANs generating

[1] Contact author: mgreming@d.umn.edu

2D wheel designs for automotive applications. The present work builds on this effort by extending it to 3D and by adding manufacturing constraints to the generated designs.

The GAN structure consists of two neural networks (see Fig. 1), a generator and a discriminator. A generator takes a latent vector and converts it through a series of convolution and

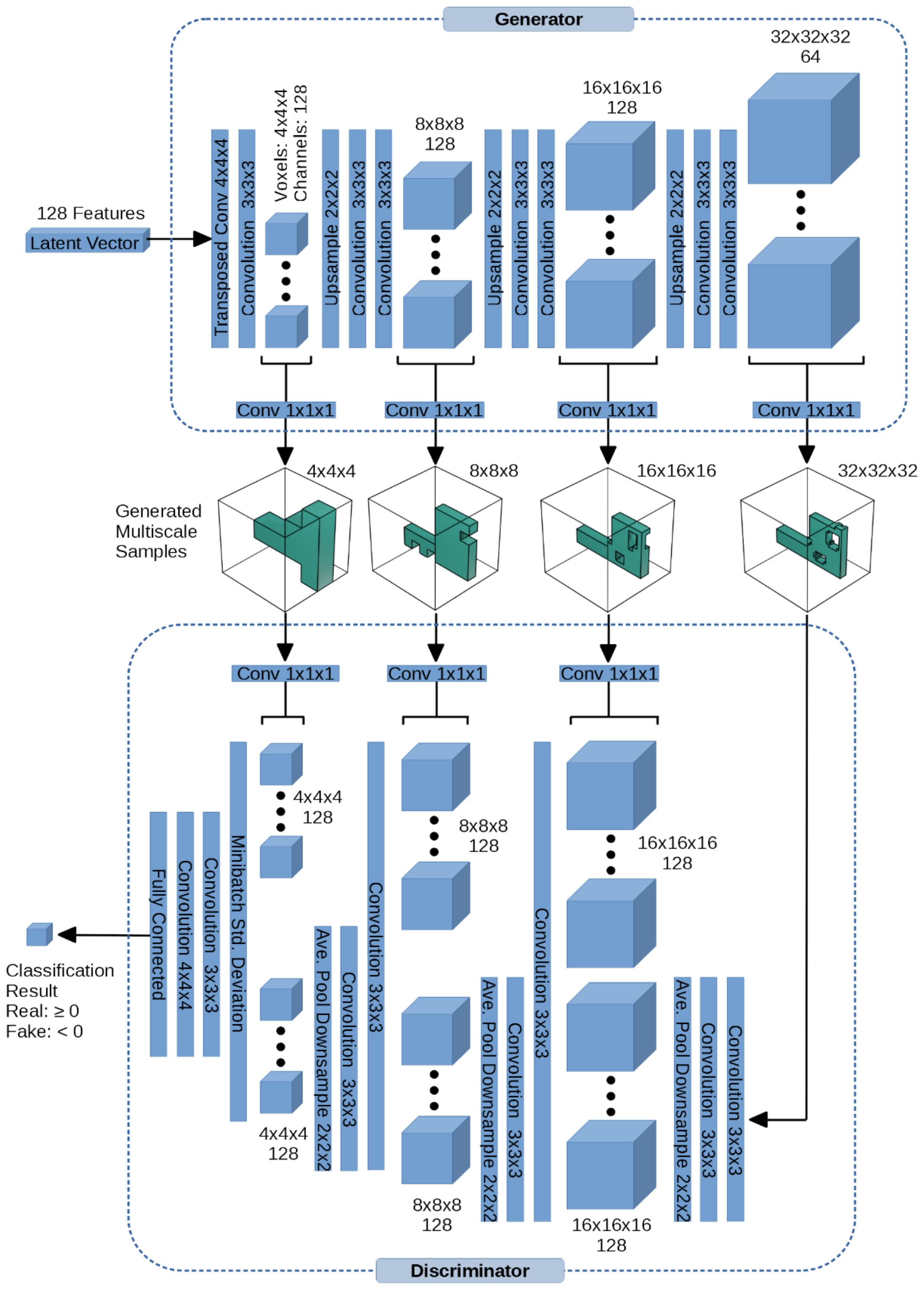

Figure 1: GAN structure is a 3D implementation of MSG-GAN and is used to approximate the distribution of the solid models used to train the GAN.

upsampling layers to create a voxel model. The discriminator, which consists of convolutional and downsampling layers, classifies an input voxel model as either real or fake where fake means that the image was an output of the generator network. The generator and discriminator networks are trained together with the optimization metric for the generator being to fool the discriminator and the optimization metric for the discriminator being to correctly identify the images created by the generator network as fake. Given a sufficient level of complexity of the generator and discriminator networks, GANs have been shown to be able to create synthetic images that appear to be real to the human eye even though they are different from all of the training data that was used. Once a GAN has been trained, the generator creates a mapping between the latent vector space and the distribution of images that it was trained on. It has been shown that the latent vector space can be mapped to certain characteristics in the generated images such as pose or facial expressions for GANs trained on faces.

Training GANs requires a large training data set that represents the distribution to be approximated by the GAN. Obtaining an adequate training set can be challenging. In this paper, we propose using a synthesized training set to represent the distribution of the set of geometries that can be manufactured by a given manufacturing technique. The advantages of this approach are that a large quantity of training data can be generated and desired manufacturability criteria can be precisely specified. For the purposes of demonstrating the technique, the manufacturing method used here is 3-axis CNC machining from a single direction. However, this approach can be extended to other manufacturing processes by generating the appropriate synthetic training set.

Once a GAN is trained that can map from the latent vector space to the manufacturable design space, topology optimization can be performed to find the latent vector that results in an optimal design. Compliance minimization subject to a target volume constraint is the case that will tested here, however, other optimization problems could be solved as well. The performance of the proposed algorithm will be compared to a traditional topology optimization algorithm for compliance minimization subject to a target volume constraint.

This paper will have the following structure. The Methods section will describe the structure of the GAN, the procedure used to generate the training data, the training of the GAN, and the implementation of the traditional topology optimization algorithm and the proposed GAN-based topology optimization algorithm. The Results section will show the performance of the proposed topology optimization algorithm as compared to the traditional topology optimization algorithm. Finally, the Conclusions section will summarize the results and discuss the next steps of this line of research.

2. METHODS

The implementation of the GAN topology optimization algorithm requires implementing a 3D GAN in order to train a generator that is able to map from a latent vector space to a manufacturable 3D model space. In order to train the GAN, a synthetic dataset is generated that spans the manufacturable design space. Finally, the traditional topology optimization algorithm is modified to use the GAN generator latent space as the feature space rather than element densities directly. The remainder of this section describes the methods used to implement the GAN, to generate the training data, and to implement the GAN topology optimization algorithm.

2.1 GAN Structure

Generative Adversarial Networks are relatively new with Goodfellow et al. [4] first introducing the technique in 2014. Since this initial work, much work has gone into extending and improving the initial implementation. The current GANs that generate high quality results leverage some sort of deep convolutional structure that was popularized by Radford et al. [9]. The network structure used for this work is a 3D extension of the 2D multi-scale gradient structure (MSG-GAN) introduced by Karnewar et al. [10] (see Fig. 1). The generator portion of the GAN takes the latent vector as input and uses combinations of upsampling and 3D convolution operations to produce the output voxel set. For the results presented here, a 32x32x32 cube of voxels was used to represent the geometry. The discriminator structure uses a combination of 3D convolution and average pool downsampling operations to reduce the spatial size of the voxel data until a fully connected layer is used on the final layer to produce a scalar result with a positive value indicating that the discriminator has determined the input voxels to be real and a negative value indicating that the discriminator has determined the input voxels to be fake. The MSG-GAN structure introduces connections between the generator and the discriminator at multiple scales (see Fig. 1) to create stronger gradients at all levels of the GAN making training more stable and less sensitive to the choice of hyperparameters.

Numerous error loss functions have been proposed for training GANs [11]. Examples include the minmax loss suggested in the original GAN paper and least squares loss functions [12]. As recommended in the MSG-GAN paper, hinge loss [13] was used for the GAN implementation presented here for training robustness. The hinge loss function can be expressed as:

$$\text{Hinge Loss} = \sum_{i=1}^{n} max(0, 1 - \hat{y_i} \times y_i) \qquad (1)$$

where $\hat{y}$ is the output of the discriminator, y is the target output, and n is the number of training examples. The target value y depends on whether the discriminator or generator is being trained. When training the discriminator, to goal is to classify the solids generated by the generator as fake ($y = -1$) and real solids from the training distribution that is being approximated as real ($y = 1$). When training the generator, the goal is to fool the discriminator into finding the generated solids real, so $y = 1$, even though the solids are actually fake. In general, when training the discriminator, the weights of the generator are held fixed. Likewise, when training the generator, the discriminator weights are held fixed. Training alternates between training the discriminator and the generator.

The training of a GAN can be viewed as a competition between the discriminator and the generator where the generator is trying to generate solid models that fool the discriminator into thinking they are real and the discriminator trying to successfully identify the fake solid models created by the generator as fake. As training progresses, if successful, the discriminator gets better at detecting fakes and the generator gets better at creating fakes. However, the training of GANs can be difficult and it does not always result in a generator that can produce realistic results. One of the issues that can occur is the discriminator getting too good at determining fakes and not allowing the generator to make progress during training. The hinge loss function (1) has the benefit that it saturates at zero once the discriminator can perfectly identify fake and real solid models. Once saturated, the discriminator no longer updates since there is a loss of the gradient from the loss to the discriminator weights during the training. This loss of gradient allows the generator to catch up to the discriminator to increase the stability of the training. It was found that the combination of MSG-GAN and hinge loss resulted in a GAN training approach that was successful without hyperparameter tuning.

For the training results presented below, 20000 training models were used. The GAN was trained for 500 epochs using batch size of $n = 50$. This training took approximately 23 days of computation time using data parallelism on dual Nvidia Tesla K20X GPUs. The PyTorch library was used to implement the GAN [14]. The training time is long but it should be noted that this only needs to be done once for a particular manufacturing method. Once the GAN is trained, the GAN based topology optimization problem can be solved with computational costs that are similar to traditional topology optimization.

2.2 Synthetic Training Data

In order to train the GAN generator to produce manufacturable solid models, a large training set that spans the space of manufacturable designs is required. One means to obtain such a set of manufacturable designs is to collect existing solid models that are manufacturable by the selected manufacturing method. The issue with this approach is that it is difficult to obtain training sets that are large enough to successfully train the GAN. A second issue is that there may be few or no examples of models that are manufacturable by a new or a highly industry specific manufacturing technique. A final limitation may be the issue of intellectual property of the models that are used to train the model. Because of these limitations, the approach that will be used here will be to generate synthetic training data to represent the set of manufacturable designs.

In order to efficiently generate a large number of random designs, a scripting approach is used to generate the manufacturable solid models. The solid models are then converted to a voxel representation compatible with the GAN structure. The open-source CadQuery solid modeling library [15] is used to create the models. The CadQuery library is written in Python and is based off of the PythonOCC library [16], which is a Python wrapper for the Open CASCADE solid modeling library [17]. CadQuery provides a means to create complex solid models directly in Python. The models can then be exported as step or STL files. In order to train the GAN, a voxel representation of the solid is needed. The Trimesh library [18] is used to convert the STL file format to the CGAL OFF file format and the Python bindings to the CGAL [19,20] project are used to generate the voxels. CGAL is used because of its computational efficiency, which is essential for the large number of training examples that need to be generated.

The training examples generated for this present study are intended to be machined by a 3-axis vertical milling machine with a single setup. However, this approach is intended to be able to be applied to any desired manufacturing process. Fig. 2 shows an example of a solid model that includes the features that are used in the randomly generated data sets. These features include blind pockets, through pockets, slots, step downs, and through holes (not show). The existence, location, size, and orientation of these features are randomized. The base part is either an L-shape as shown or a rectangle. For both cases, the length and width proportion and thickness are randomized. In all, there are 17 random binary choices that determine whether certain features exist or not and the properties of those features (through pocket or not, for example). Finally, depending on the features included in each model, up to 51 dimensions, or other parameters such as feature angles or radii, are randomized.

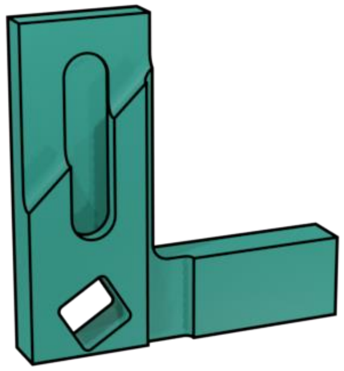

Figure 2: Example of the features used in the randomly generated geometry for the training sets.

After the training solid model examples are generated, they are voxelated at 4x4x4, 8x8x8, 16x16x16, and 32x32x32 scales. The four scales are needed for the multi-scale GAN structure described above. Before voxelization, the models are scaled to fill a unit cube. The position of the model within the cube is randomized in the three x, y, and z axes within the range of values that will ensure that the model will fit entirely within the cube. In addition, the models are randomly mirrored about the x, y, and z planes as well as randomly rotated 90 or 0 degrees about the x, y, and z axes. This ensures that the GAN is able to generate models independent of the machining direction relative to the part. Fig. 3 shows the four voxel scales for the solid model shown in Fig. 2. Fig. 4 shows four examples from the 20000 voxel model representations that were used to train the GAN for the results obtained in this paper.

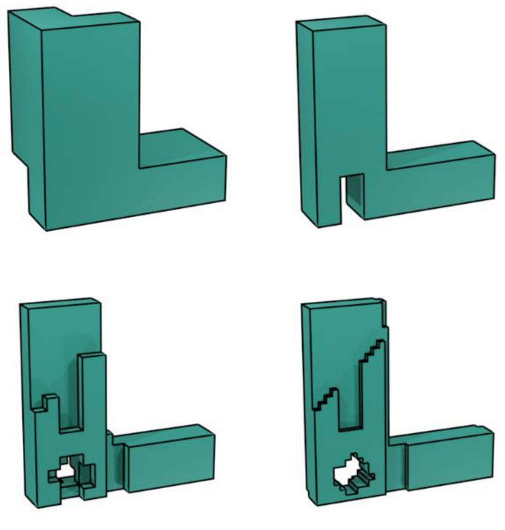

Figure 3: Multi-scale voxel representations generated from the solid model shown in Fig. 2.

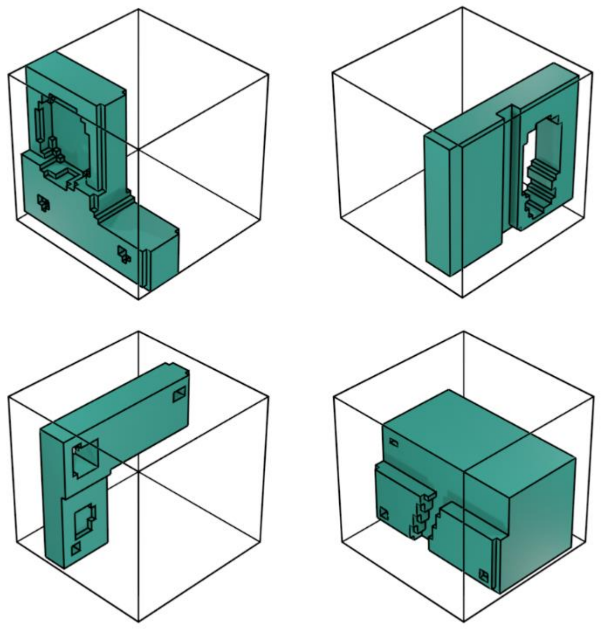

Figure 4: Four examples of training voxel sets used to train the GAN for the results presented in this paper.

After training the GAN using the 20000 training models for 500 epochs with a batch size of 50, 20 random models generated using the generator are shown in Fig. 5. These examples were obtained by randomly sampling the latent vector space of the generator using a normal distribution. These results were not curated and are the first 20 samples randomly obtained. It can be seen from Fig. 5 that the generator is starting to approximate the type of models used for training (see Fig. 4). However, there are some artifacts in some of the models and the models are not as clean as the training models. The results may be improved with further training of the GAN. However, as will be shown below, the model obtained to this point is sufficient to produce manufacturable results for the topology optimization problem.

2.3 Implementation the Generative Network Topology Optimization Algorithm

There are numerous formulations of the topology optimization problem including compliance minimization, maximum stress constraints, and natural frequency maximization (see [1]). In general, the generative approach to topology optimization that is presented here could be applied to any of these problem formulations. For the results presented in this paper, the minimum compliance design formulation is used. In minimum compliance design, the mechanical compliance of the design is minimized while ensuring that the total mass is below a specified target. The mass target is usually specified as a percentage of the mass of the entire design. Minimum compliance design provides a design that maximizes stiffness for a given mass target. The minimum compliance optimization problem can be formulated as [1]:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & c(\boldsymbol{x}) = \tfrac{\mathbf{1}}{\mathbf{2}}\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{K}(\boldsymbol{e})\boldsymbol{u} \\ \text{subject to} \quad & \frac{V(\boldsymbol{x})}{V_0} \leq v_f \\ & \boldsymbol{e} = E(\boldsymbol{x}) \\ & \boldsymbol{K}(\boldsymbol{e})\boldsymbol{u} = \boldsymbol{f} \\ & \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1} \end{aligned} \tag{2}$$

where $\boldsymbol{x}$ is the vector of element densities x_e, $c(\boldsymbol{x})$ is the compliance of the structure ($c(\boldsymbol{x})$ can also be interpreted as the strain energy stored in the structure), $\boldsymbol{u}$ is the vector of nodal displacements of the finite element model and $\boldsymbol{f}$ is the vector of nodal forces, $\boldsymbol{K}(\boldsymbol{e})$ is the global stiffness matrix, $\boldsymbol{e}$ is the vector of element modulus values calculated from the element density vector $\boldsymbol{x}$ using the function E as described below, $V(\boldsymbol{x})$ is the volume of the model and V_o is the volume of the model domain, and v_f is the target volume fraction. In order to construct the global stiffness matrix $\boldsymbol{K}$ the element densities x_e need to be converted to element elastic modulus values e_e. The solid isotropic material with penalization (SIMP) approach, as modified by Sigmund [21], will be used to calculate the element elastic moduli. The SIMP calculation of element elastic moduli can be expressed as:

$$\boldsymbol{e} = E(\boldsymbol{x}) = E_{min} + \boldsymbol{x}^p(E_0 - E_{min}) \tag{3}$$

where E_0 is the elastic modulus of the material, E_{min} is the minimum allowed elastic modulus to provide numerical stability of the finite element analysis, and p is a penalization factor. For the traditional topology optimization results included here, values of $E_{min} = 1 \times 10^{-6}$ and $p = 3$ were used. The SIMP calculation acts as a means to penalize element densities that are not close to 1 or 0, thus minimizing physically unrealizable partial density elements in the optimized model.

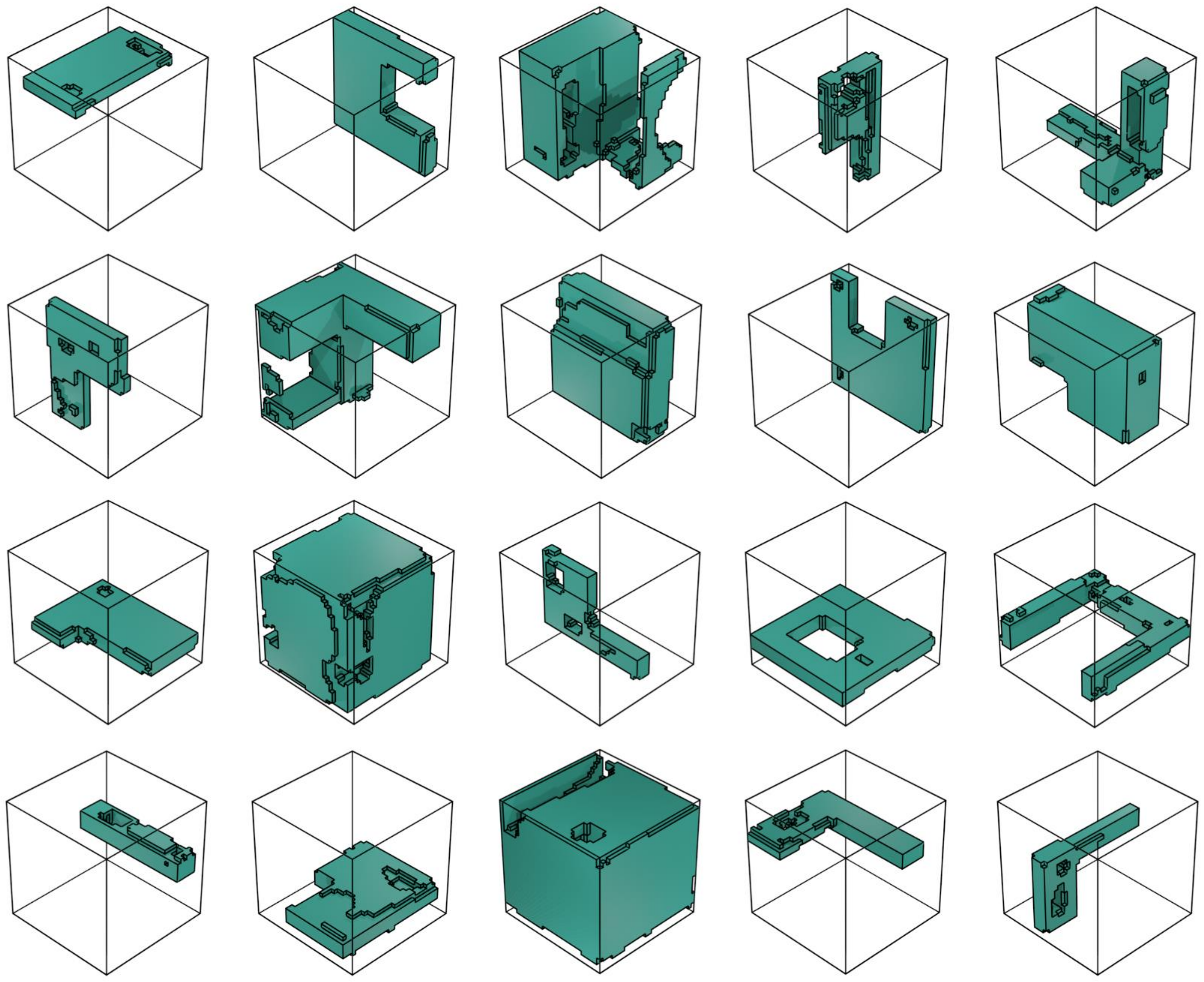
Figure 5: Output of the GAN after 500 training epochs.

In order to solve the minimization problem (2), a numerical approach needs to be used. Typically a gradient based constrained optimization approach is used [1]. The conservative convex separable approximations (CCSA) method of moving asymptotes (MMA) proposed by Svanberg [22] was used since it is well suited to problems with a large number of design variables (32768 variables for the 32x32x32 design space) and a large number of inequality constraints that are characteristic of topology optimization problems. The NLopt implementation of the CCSA algorithm was used for the results presented here [23]. In order to apply a gradient based minimization algorithm, the sensitivity, or gradient, of the compliance function to the element densities needs to be computed with each element of the gradient represented as $\partial c/\partial x_e$. Using the chain rule for multiple dimensions, the elements of the compliance gradient with respect to the element densities can be expressed as:

$$\frac{\partial c}{\partial e_e} = \sum_{i=1}^{3n} \frac{\partial c}{\partial u_i}\frac{\partial u_i}{\partial e_e} \tag{4}$$

where there are n nodes in the finite element mesh (there are $3n$ components of nodal displacements for the three dimensions) and the elements of the gradient of the displacement with respect to the element densities can be expressed as:

$$\frac{\partial u_i}{\partial e_e} = \sum_{j=1}^{m} \frac{\partial K_{ij}^{-1}}{\partial e_e} f_j \tag{5}$$

where there are m elements and since $\boldsymbol{u} = \boldsymbol{K}^{-1}\boldsymbol{f}$ as a solution the linear system of equations that define the finite element problem in (2). Equation (5) assumes that the force vector f is not a function of the element densities. It should be noted that the force vector $\boldsymbol{f}$ may become of a function of the element densities when there are non-zero displacement boundary conditions and the boundary conditions are applied in the standard way [24] to ensure the stiffness matrix $\boldsymbol{K}$ remains symmetric. Throughout this work, only zero displacement boundary conditions are used so (5) will hold. In general, the partial derivative of the matrix inverse can be expressed as [25]:

$$\frac{\partial \boldsymbol{A}^{-1}}{\partial y} = -\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial y}\boldsymbol{A}^{-1} \tag{6}$$

substituting (5) and (6) into (4) yields:

$$\frac{\partial c}{\partial E_e} = \sum_{i=1}^{3n}\frac{\partial c}{\partial u_i}\sum_{j=1}^{m}\left(-\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial e_e}\boldsymbol{K}^{-1}\right)_{ij} f_j \tag{7}$$

Putting (7) into matrix form and noting that $\boldsymbol{K}^{-1}\boldsymbol{f}$ is simply the displacement vector $\boldsymbol{u}$ yields the following expression for the components of the compliance gradient with respect to the element elastic moduli:

$$\frac{\partial c}{\partial e_e} = (\nabla_{\boldsymbol{u}} c)^T(-\boldsymbol{K}^{-1})\left(\frac{\partial \boldsymbol{K}}{\partial e_e}\right)\boldsymbol{u} \tag{8}$$

where the gradient of c with respect to u is defined as:

$$(\nabla_{\boldsymbol{u}} c)^T = [\partial c/\partial u_1 \quad \partial c/\partial u_2 \quad \dots \quad \partial c/\partial u_n] \tag{9}$$

Since the gradient component defined by (8) is a scalar, the right-hand side of (8) can be transposed without changing the result. This resulting form allows for a simplification later:

$$\frac{\partial c}{\partial e_e} = -\boldsymbol{u}^{\boldsymbol{T}}\left(\frac{\partial \boldsymbol{K}}{\partial e_e}\right)^T(\boldsymbol{K}^{-1})(\nabla_{\boldsymbol{u}} c) \tag{10}$$

Notice that in general the inverse of the stiffness matrix is needed to calculate the gradient of the compliance with respect to the element elastic moduli. In order to solve for the nodal displacements, it was required solve the linear equation $\boldsymbol{Ku} = \boldsymbol{f}$. However, in general, the inverse of the stiffness matrix is not directly computed due to the size of the problems involved. For structural problems, $\boldsymbol{K}$ is a sparse matrix. However, $\boldsymbol{K}^{-1}$ is a dense matrix so it is not practical to store the inverse for the size of problems that are typical for structural problems. Because of this, the quantity $(\boldsymbol{K}^{-1})(\nabla_{\boldsymbol{u}} c)$ is computed by solving the equation $\boldsymbol{Ky} = \nabla_{\boldsymbol{u}} c$. The minimum residual iteration method for sparse systems [26], as implemented in the SciPy Python library [27], is used to solve these linear systems. Therefore, two solves of linear systems of equations are required for each iteration of the topology optimization problem.

For the specific case where compliance is used as the objective function as in (2), equation (10) simplifies so that the second solve of the finite element problem is no required. Since c is a quadratic form, and taking advantage of symmetry of $\boldsymbol{K}$ for structural problems, the gradient of the compliance with respect to the nodal displacements becomes [25]:

$$\nabla_{\boldsymbol{u}} c = \nabla_{\boldsymbol{u}}\left(\frac{1}{2}\boldsymbol{u}^T\boldsymbol{K}\boldsymbol{u}\right) = (\boldsymbol{K} + \boldsymbol{K}^T)\boldsymbol{u} = 2\boldsymbol{K}\boldsymbol{u} \tag{11}$$

Substituting (11) into (10) results in the following equation:

$$\frac{\partial c}{\partial e_e} = -2\boldsymbol{u}^{\boldsymbol{T}}\left(\frac{\partial \boldsymbol{K}}{\partial e_e}\right)^T\boldsymbol{u} \tag{12}$$

This elimination of the inverse of the stiffness matrix from the equation for the components for the gradient of compliance with respect the element elastic moduli is one of the reasons that compliance is so commonly used as an objective function for topology optimization problems. Equation (12) will be used for the present work but in general equation (10) needs to be used for the gradient computation for objective functions other than compliance minimization.

Up until this point, equation (12) applies for the traditional topology optimization problem and the GAN topology optimization problem. The two approaches differ in computing the gradient of compliance with respect to the design variables. For the traditional topology optimization problem, the design variables are the element densities x_e. For the GAN topology optimization problem, the design variables are the components of latent vector space vector z_l. Again, using the multi-dimension chain rule, the gradient with respect to the element density design variables is:

$$\frac{\partial c}{\partial x_i} = \sum_{j=1}^{m}\frac{\partial c}{\partial e_j}\frac{\partial e_j}{\partial x_i} \tag{13}$$

where there are m elements in the design volume. For the traditional topology optimization case, the partials of the element elastic moduli with respect to the element densities can be computed from the SIMP calculation (3):

$$\begin{aligned} \frac{\partial e}{\partial x_i} &= p x_i^{p-1}(E_0 - E_{min}) \quad && \text{for } i = j \\ \frac{\partial e_j}{\partial x_i} &= 0 && \text{for } i \neq j \end{aligned} \tag{14}$$

Substituting (14) and (12) into (13) yields:

$$\frac{\partial c}{\partial x_e} = -2p x_e^{p-1}(E_0 - E_{min})\boldsymbol{u}^{\boldsymbol{T}}\left(\frac{\partial \boldsymbol{K}}{\partial e_e}\right)^T\boldsymbol{u} \tag{15}$$

where $\partial \boldsymbol{K}/\partial e_e$ is the element stiffness matrix for element e with unit elastic modulus. Equation (15) is the commonly used compliance minimization sensitivity function that can be found in the literature [28]. If (15) is used directly for the traditional topology problem, the results will exhibit the checkerboard pattern issue where portions of the resulting geometry will have alternating full and zero density elements. This checkerboarding is generally undesirable since it results in geometries that are not manufacturable. To prevent checkerboard patterns in the traditional topology optimization results, a commonly used sensitivity filtering heuristic is used that locally smooths the gradient to prevent the checkboard pattern. The sensitivity filtering equation provided in [28] is used with a filter radius of two times the element size.

To convert the traditional topology optimization algorithm into the proposed GAN-based approach, the optimization problem needs to be performed using the latent feature vector as the design variable rather than the element densities. To this end, the GAN topology optimization problem can be expressed as:

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{minimize}} \quad & c(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{u}^T\boldsymbol{K}(\boldsymbol{e})\boldsymbol{u} \\ \text{subject to} \quad & \frac{V(G(\boldsymbol{z}))}{V_0} \leq v_f \\ & \boldsymbol{e} = E_o G(\boldsymbol{z}) \\ & \boldsymbol{K}(\boldsymbol{e})\boldsymbol{u} = \boldsymbol{f} \\ & -\boldsymbol{6} \leq \boldsymbol{z} \leq \boldsymbol{6} \end{aligned} \qquad (16)$$

where $\boldsymbol{z}$ is the latent feature vector and G is the generator network from the trained GAN that maps from the latent vector to the element densities. For the GAN topology optimization implementation, equation (13) becomes:

$$\frac{\partial c}{\partial z_i} = \sum_{j=1}^{m} \frac{\partial c}{\partial e_j}\frac{\partial e_j}{\partial z_i} = \; E_0 \sum_{j=1}^{m} \frac{\partial c}{\partial e_j}\frac{\partial G(\boldsymbol{z})}{\partial z_i} \qquad (17)$$

For the GAN-based approach, the element densities x_e are output from the GAN generator neural network for a particular latent vector input $\boldsymbol{z}$. The output of the GAN is then multiplied by E_0 and clamped between E_{min} and E_0 to ensure valid elastic modules values. The gradient components of the GAN generator function with respect to the latent vector components z_l, $\partial G(\boldsymbol{z})/\partial z_i$, are obtained using the automatic differentiation capabilities of the PyTorch machine learning framework [14] since PyTorch was used to implement the GAN. Two advantages in using the GAN generator to determine the element densities is that the SIMP equation is not needed since the GAN naturally produces element densities that are near zero or one and the sensitivity filtering to prevent the checker board pattern is not required for the cases tested here.

The NLopt implementation of the sequential least-squares quadratic programming algorithm (SLSQP) proposed by Kraft [26, 27] is used to find the latent feature vector that minimizes the compliance of the structure subject to the target volume fraction constraint. The SLSQP algorithm is used since it has better convergence properties than the CCSA algorithm for optimization problems with fewer variables. In this case, the latent vector is of size 128 compared to 32768 variables for the traditional topology optimization algorithm. Each of the latent design variables is limited to be in the range of -6 to 6 using inequality constraints. Since samples from the standard normal distribution with a standard deviation of one were used to generate the latent vectors while training the GAN, limiting the components of the latent vector to this range is equivalent to limiting the latent vector to be within six standard deviations of the zero latent vector.

Before the GAN topology optimization is performed, the latent space vector components z_l are initialized to create a solid that fills the volume. This is done by maximizing the volume fraction. This optimization is stopped when the volume fraction hits 0.95. This initialization of the latent vector has been found to provide better performance in the GAN topology optimization algorithm as compared to initializing the latent vector to the zero vector.

3. RESULTS AND DISCUSSION

In order to solve the topology optimization problem, a load and fixed boundary conditions need to be applied to the design space volume. Fig. 6 shows the two load cases that were tested. The target volume fraction used for the topology optimization is set to 0.4 for both the traditional topology optimization case and the GAN topology optimization case.

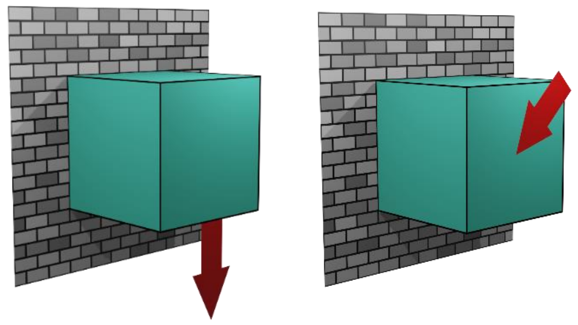

Figure 6: The two load cases used to evaluate the proposed topology optimization problem. Load Case 1 on the left and Load Case 2 on the right.

Fig. 7 shows the results of applying the traditional compliance minimization approach to the load cases shown in Fig. 6. Figs. 8 and 9 show the orthographic views of the solutions obtained using the traditional topology optimization algorithm. The machinability was assessed using an occlusion metric that determines the fraction of unreachable voxels for the machining processes. The Appendix includes a description of how the metric is calculated and a visualization of the unreachable volume during machining for each of the examples. Table 1 gives the compliance number achieved for each of the load cases (lower is better) and the volume fraction of unreachable material during machining (lower is better). The design obtained for Load Case 1 has 18% of the material than cannot be reached for marching due to the shape of the design obtained. There is not an obvious way to modify this design to obtain a design that can easily be machined using a single 3-axis machining setup.

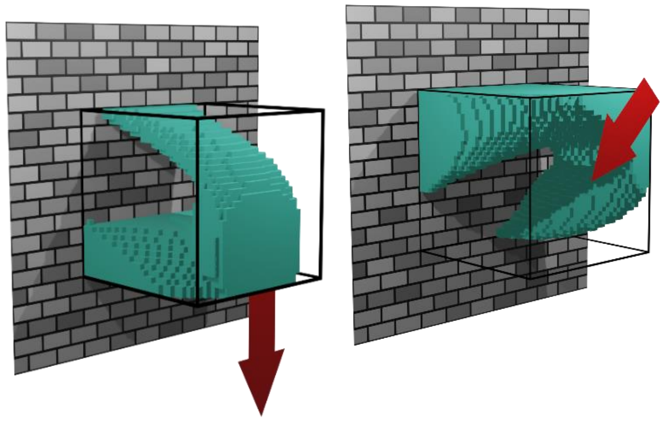
Figure 7: Traditional compliance minimization topology optimization solution for Load Case 1 on the left and Load Case 2 on the right.

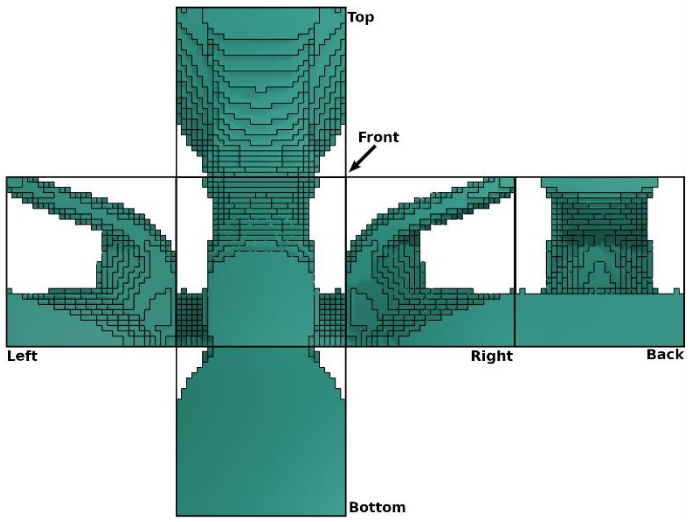

Figure 8: Orthographic views of topology optimization solution for Load Case 1 using the traditional algorithm.

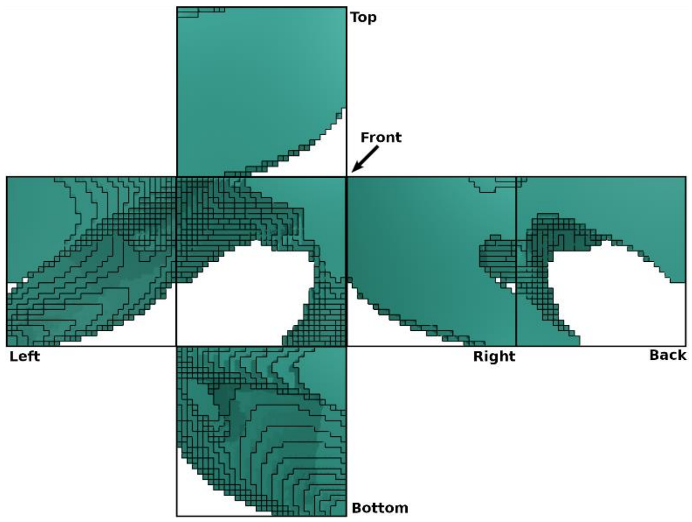

Figure 9: Orthographic views of topology optimization solution for Load Case 2 using the traditional algorithm.

Figs. 10, 11 and 12 show the solution obtained using the GAN implementation of the topology optimization algorithm. For both cases, less than 5% of the volume is unreachable for the machining operation (see Table 1).

The volume constraint for the GAN topology optimization problem (15) requires passing the latent vector through the GAN generator network. This makes the volume constraint more complicated than that of the traditional topology optimization problem (1), which has a linear mapping between the element densities and the total volume. The more complicated volume constraint function does impact the convergence rate of the GAN topology optimization algorithm as compared to the traditional topology optimization algorithm (see Fig. 13). This results in optimization times that are longer for the GAN approach as compared to the traditional approach. This convergence issue of the GAN approach could be eliminating if one of the components of the latent vector could be used to specify the desired target density. With this approach, the volume constraint could more easily be satisfied potentially improving the convergence characteristics of the optimization problem. This approach of appending parameters to the latent vector to constrain the output of the GAN was first introduced by Mirza et al. using an approach called Conditional GANs [31].

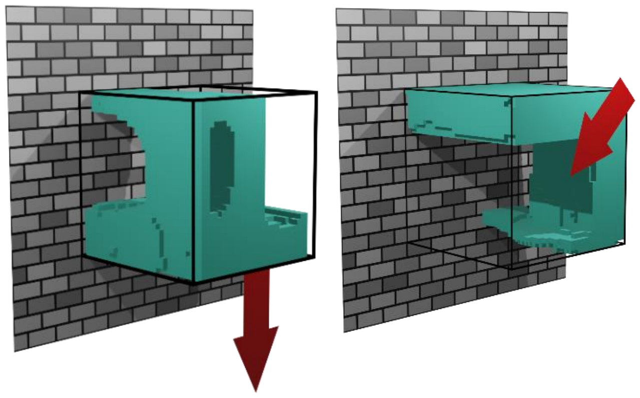
Figure 10: GAN topology optimization solution for Load Case 1 on the left and Load Case 2 on the right.

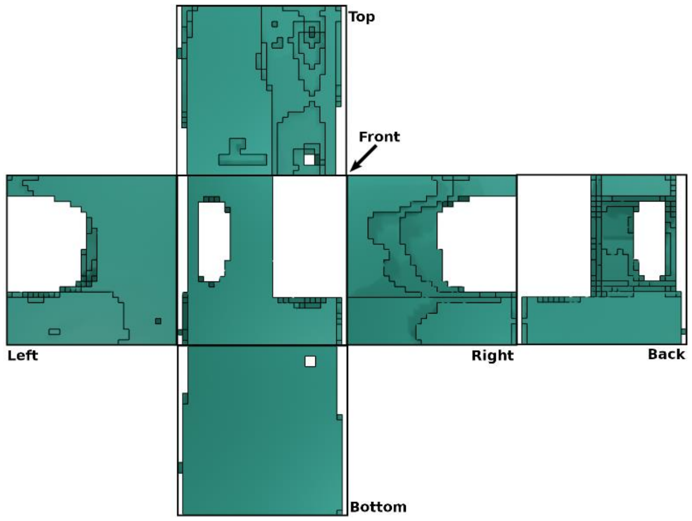

Figure 11: Orthographic views for GAN topology solution for Load Case 1.

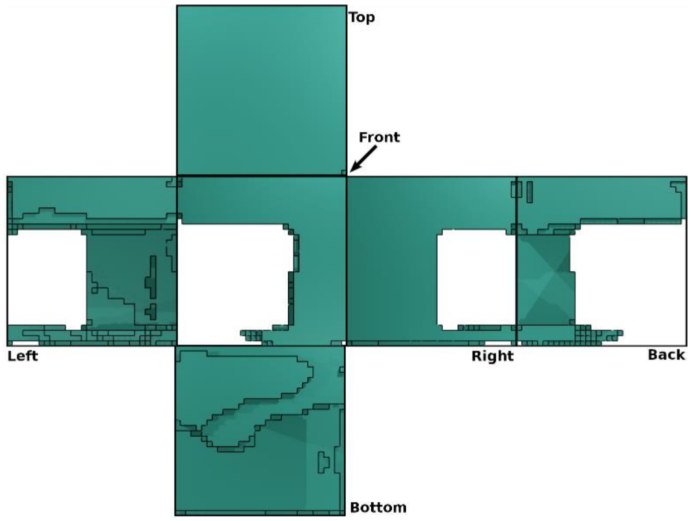

Figure 12: Orthographic views for GAN topology solution for Load Case 2.

Table 1: Quantitative comparison between the traditional approach to topology optimization compared to the proposed GAN approach.

	Compliance		Fraction of Unmachinable Voxels	
Load Case	Traditional	GAN	Traditional	GAN
1	7.63	7.98	.183	.044
2	49.16	49.75	.101	.039

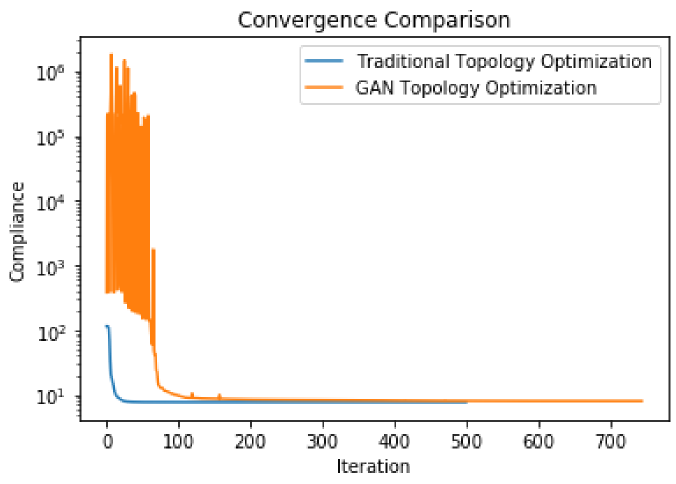

Figure 13: Topology optimization convergence comparison between the traditional topology optimization algorithm and the GAN topology optimization algorithm for Load Case 1.

4. CONCLUSION

The results from the previous section show that a GAN can be successfully trained to generate models that can be manufactured by a specific manufacturing process. The generator from this trained GAN can then be used with the topology optimization algorithm outlined above to ensure that the designs that are generated are manufacturable. The performance of the manufacturable design was only slightly reduced from the design obtained by the traditional algorithm. However, the GAN generated designs shown in Fig. 10 could not be obviously derived from the result of the traditional topology optimization algorithm shown in Fig. 7. In this way, the proposed GAN topology optimization algorithm generates designs that could not be obtained by existing methods.

Future work will include implementing the Conditional GAN algorithm to allow the target volume to be used as an input to the generator, as discussed in the previous section, to improve the convergence characteristics of the GAN-based approach. Additionally, the GAN will be trained for other manufacturing processes such as injection molding in order to show the general applicability of the GAN-based approach. Finally, a longer-term goal will be to speed up the training of the GAN for new manufacturing methods. The GAN presented here took 23 days to train, which will limit the adoption of this approach. One approach to speed up the training may be to use Conditional GANs to not only specify the target volume but to also specify the desired manufacturing process. This would allow one GAN to be trained to generated models for multiple manufacturing processes at once.

ACKNOWLEDGEMENTS

The author acknowledges the Minnesota Supercomputing Institute (MSI) at the University of Minnesota for providing resources that contributed to the research results reported within this paper. URL: http://www.msi.umn.edu

REFERENCES

[1] Bendsoe, M. P., and Sigmund, O., 2003, *Topology Optimization: Theory, Methods, and Applications*, Springer Science & Business Media.

[2] Liu, J., and Ma, Y., 2016, "A Survey of Manufacturing Oriented Topology Optimization Methods," Adv. Eng. Softw., **100**, pp. 161–175.

[3] Zuo, K.-T., Chen, L.-P., Zhang, Y.-Q., and Yang, J., 2006, "Manufacturing- and Machining-Based Topology Optimization," Int. J. Adv. Manuf. Technol., **27**(5–6), pp. 531–536.

[4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014, "Generative Adversarial Nets," *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, eds., Curran Associates, Inc., pp. 2672–2680.

[5] Wu, J., Zhang, C., Xue, T., Freeman, B., and Tenenbaum, J., 2016, "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling," *Advances in Neural Information Processing Systems 29*, D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon, and R. Garnett, eds., Curran Associates, Inc., pp. 82–90.

[6] Shea, K., Aish, R., and Gourtovaia, M., 2005, "Towards Integrated Performance-Driven Generative Design Tools," Autom. Constr., **14**(2), pp. 253–264.

[7] Matejka, J., Glueck, M., Bradner, E., Hashemi, A., Grossman, T., and Fitzmaurice, G., 2018, "Dream Lens: Exploration and Visualization of Large-Scale Generative Design Datasets," *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, pp. 369:1–369:12.

[8] Oh, S., Jung, Y., Kim, S., Lee, I., and Kang, N., 2019, "Deep Generative Design: Integration of Topology Optimization and Generative Models," J. Mech. Des., **141**(11).

[9] Radford, A., Metz, L., and Chintala, S., 2016, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," ArXiv151106434 Cs.

[10] Karnewar, A., Wang, O., and Iyengar, R. S., 2019, "MSG-GAN: Multi-Scale Gradient GAN for Stable Image Synthesis," ArXiv190306048 Cs Stat.

[11] Dong, H.-W., and Yang, Y.-H., 2019, "Towards a Deeper Understanding of Adversarial Losses," ArXiv190108753 Cs Stat.

[12] Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Paul Smolley, S., 2017, "Least Squares Generative Adversarial Networks," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802.

[13] Tran, D., Ranganath, R., and Blei, D., 2017, "Hierarchical Implicit Models and Likelihood-Free Variational Inference," *Advances in Neural Information Processing Systems 30*, I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., Curran Associates, Inc., pp. 5523–5533.

[14] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S., 2019, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, eds., Curran Associates, Inc., pp. 8024–8035.

[15] 2020, *CadQuery/Cadquery*, CadQuery.

[16] Paviot, T., 2020, *Tpaviot/Pythonocc-Core*.

[17] "Overview | OPEN CASCADE" [Online]. Available: https://www.opencascade.com/content/overview. [Accessed: 19-Jan-2020].

[18] Dawson-Haggerty, M., 2020, *Mikedh/Trimesh*.

[19] The CGAL Project, 2020, *CGAL User and Reference Manual*, CGAL Editorial Board.

[20] 2020, *CGAL/Cgal-Swig-Bindings*, The CGAL Project.

[21] Sigmund, O., 2007, "Morphology-Based Black and White Filters for Topology Optimization," Struct. Multidiscip. Optim., **33**(4–5), pp. 401–424.

[22] Svanberg, K., 2002, "A Class of Globally Convergent Optimization Methods Based on Conservative Convex Separable Approximations," SIAM J. Optim., **12**(2), pp. 555–573.

[23] Johnson, S. G., *The NLopt Nonlinear-Optimization Package*, http://github.com/stevengj/nlopt.

[24] Huebner, K. H., Dewhirst, D. L., Smith, D. E., and Byrom, T. G., 2001, *The Finite Element Method for Engineers*, John Wiley & Sons.

[25] Petersen, K. B., and Pedersen, M. S., 2012, "The Matrix Cookbook. Version: Nov. 15 2012."

[26] Paige, C. C., and Saunders, M. A., 1975, "Solution of Sparse Indefinite Systems of Linear Equations," SIAM J. Numer. Anal., **12**(4), pp. 617–629.

[27] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors, S. 1 0, 2019, "SciPy 1.0--Fundamental Algorithms for Scientific Computing in Python," ArXiv190710121 Phys.

[28] Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B. S., and Sigmund, O., 2011, "Efficient Topology Optimization in MATLAB Using 88 Lines of Code," Struct. Multidiscip. Optim., **43**(1), pp. 1–16.

[29] Kraft, D., 1988, "A Software Package for Sequential Quadratic Programming," Forschungsbericht- Dtsch. Forsch.- Vers. Luft- Raumfahrt.

[30] Kraft, D., 1994, "Algorithm 733: TOMP–Fortran Modules for Optimal Control Calculations," ACM Trans. Math. Softw. TOMS, **20**(3), pp. 262–281.

[31] Mirza, M., and Osindero, S., 2014, "Conditional Generative Adversarial Nets," ArXiv Prepr. ArXiv14111784.
[32] Hoefer, M. J., and Frank, M. C., 2018, "Automated Manufacturing Process Selection During Conceptual Design," J. Mech. Des., **140**(3).

APPENDIX

Each of the following figures (Figs. 14-17) show the material that cannot be removed from the optimal designs (highlighted in red) obtained by the traditional topology optimization algorithm (Figs. 14 and 15) and the proposed GAN topology optimization algorithm (Figs. 16 and 17). This unreachable material is shown in red. In the evaluation of the machinability of each design, all six machining directions were considered (±x, ±y, and ±z directions) and the direction with the smallest fraction of unreachable voxels is taken as the machinability metric. A voxel is considered unreachable for a 3-axis machining operation if it is occluded by voxels when viewed from the machining direction. This approach is inspired by the machinability criteria proposed by Hoefer et al. [32].

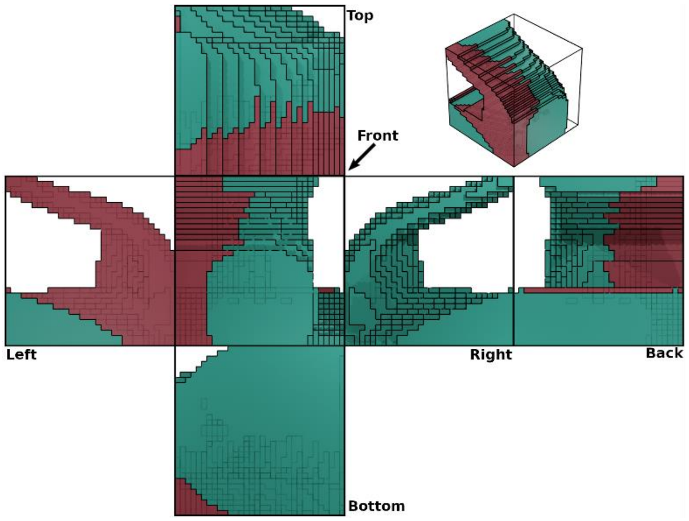

Figure 14: Orthographic views of topology optimization solution for Load Case 1 using the traditional algorithm. 18.3% of the void voxels cannot be removed by a 3-axis machining operation.

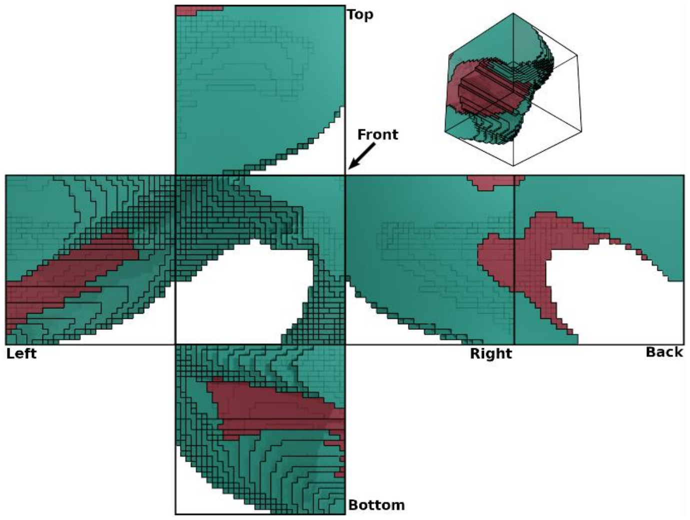

Figure 15: Orthographic views of topology optimization solution for Load Case 2 using the traditional algorithm. 10.1% of the void voxels cannot be removed by a 3-axis machining operation.

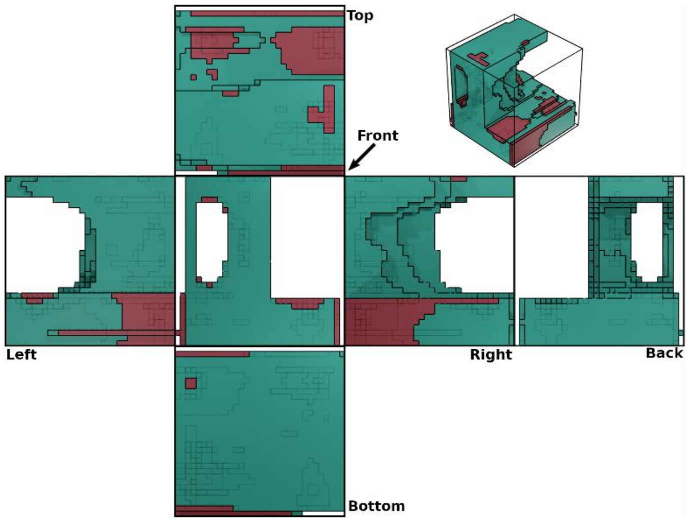

Figure 16: Orthographic views for GAN topology solution for Load Case 1. 4.4% of the void voxels cannot be removed by a 3-axis machining operation.

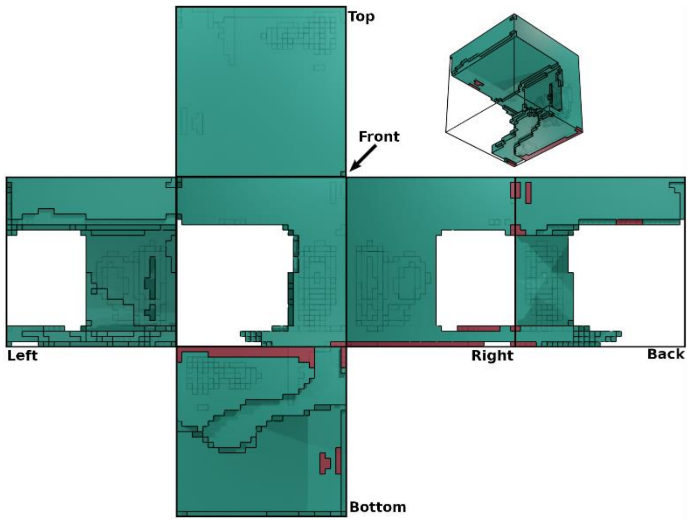

Figure 17: Orthographic views for GAN topology solution for Load Case 2. 3.9% of the void voxels cannot be removed by a 3-axis machining operation.