2020

An Adversarial Framework for Deep 3D Target Template Generation

Walter E. Waldow
*Wright State University*

Follow this and additional works at: [https://corescholar.libraries.wright.edu/etd_all](https://corescholar.libraries.wright.edu/etd_all)

Part of the [Computer Engineering Commons](https://corescholar.libraries.wright.edu/etd_all/computer-engineering), and the [Computer Sciences Commons](https://corescholar.libraries.wright.edu/etd_all/computer-sciences)

**Repository Citation**

[https://corescholar.libraries.wright.edu/etd_all/2350](https://corescholar.libraries.wright.edu/etd_all/2350)

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.
AN ADVERSARIAL FRAMEWORK FOR DEEP 3D TARGET TEMPLATE GENERATION

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science

by

WALTER E. WALDOW
B.S., Colorado State University, 2016

2020
Wright State University
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Walter E. Waldow ENTITLED AN ADVERSARIAL FRAMEWORK FOR DEEP 3D TARGET TEMPLATE GENERATION BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Derek Doran, Ph.D.
Thesis Director

Mateen Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

Derek Doran, Ph.D.

John Gallagher, Ph.D.

Fred Garber, Ph.D.

Barry Milligan, Ph.D.
Interim Dean of the Graduate School
This paper presents a framework for the generation of 3D models. This is an important problem for many reasons. For example, 3D models are import for systems that are involved in target recognition. These systems use 3D models to train up accuracy on identifying real world object. Traditional means of gathering 3D models have limitations that the generation of 3D models can help overcome. The framework uses a novel generative adversarial network (GAN) that learns latent representations of two dimensional views of a model to bootstrap the network’s ability to learn to generate three dimensional objects. The novel architecture is evaluated using two different types of evaluation. The two dimensional views are evaluated using a combination of an Inception Score and Hausdorff Distance, compared against the two dimensional views of the real 3D models used in training. The three dimensional object are evaluated using the Hausdorff Distance compared against the real 3D models used in training. Experimental results demonstrate that the novel generative adversarial network that is being proposed generated realistic looking models faster, and with higher fidelity than a basic 3D generative adversarial network would produce with the same training structure. The thesis illustrates the promise of GAN bootstrapping with two dimensional perspective codes to create higher fidelity three dimensional models.
# Contents

1 Introduction and Motivation 1

2 Background and Related Work 7
   2.1 Background ......................................................... 7
   2.2 Related Work .................................................... 9

3 Preliminaries 12

4 Methodology 17
   4.1 Dataset description ............................................... 18
   4.2 Neural Networks .................................................. 22
      4.2.1 Latent Code Generator ....................................... 26
      4.2.2 3D Generative Adversarial Network ......................... 30
   4.3 Model Training ................................................... 35
      4.3.1 Latent Code Generator ....................................... 36
      4.3.2 3D Generative Adversarial Network ......................... 38

5 Evaluation 39
   5.1 Toilet Object Class ............................................... 41
      5.1.1 Latent Code Generator ....................................... 41
      5.1.2 3D Generative Adversarial Network ......................... 44
   5.2 Bed Object Class .................................................. 49
      5.2.1 Latent Code Generator ....................................... 49
      5.2.2 3D Generative Adversarial Network ......................... 52
   5.3 Airplane Object Class ............................................ 58
      5.3.1 Latent Code Generator ....................................... 58
      5.3.2 3D Generative Adversarial Network ......................... 60
   5.4 Comparative Analysis ............................................ 62

6 Conclusion 68

Bibliography 70
List of Figures

3.1 Octree Example ................................................. 13
3.2 GAN Example .................................................. 16

4.1 Framework with Perspective ................................. 18
4.2 Example of Airplane Models ................................. 20
4.3 Example of Bed Models ........................................ 21
4.4 Example of Toilet Models ................................. 23
4.5 Visual Representation of the 2D Discriminator .......... 26
4.6 Visual Representation of the 2D Generator ............... 29
4.7 Visual Representation of the 3D Discriminator .......... 31
4.8 Visual Representation of the 3D Generator ............... 33

5.2 Generated Output vs. Real Images ......................... 45
5.7 Generated Output vs. Real Images ......................... 53
5.12 Generated Output vs. Real Images ....................... 61
I would like to extend my heartfelt thanks to my thesis advisor, Dr. Derek Doran. When I first met Dr. Doran, I had no knowledge about machine learning. With his help and advice I have learned a great deal. He has always been open to me asking questions and encouraging my development and my growth. Without his help and his guidance, I would not be where I am today. He has not only been a great advisor but he has also been an amazing mentor. I will be forever grateful that he accepted me as his graduate student and convinced me to attend Wright State University.

Next, I would like to thank one of my committee members, Dr. Fred Garber. Dr. Garber was the first member of Wright State Faculty that I met several years ago during an internship at Wright State University. Since that time, Dr. Garber has provided stability that has encouraged my growth throughout my Masters program. I would also like to thank my other committee member, Dr. John Gallagher. Dr. Gallagher has always taken the time to answer any questions I’ve had. He also has provided me with opportunities to talk through any ideas or thoughts related to machine learning. Both of my committee members have encouraged me to learn and grow.

Lastly, I would like to extend a thank you to all of my coworkers that shared a lab with me these past two years. I’d like to thank Jameson Morgan and Matt Barrett in particular. They have taken time from their busy schedules to help me with any problems that I have had or just take time while I talked through my work. Jameson has always been willing to help me do technical setup and help with any problems. Matt has spent countless hours listening to me talk about and allowing me to share my work with him. Also, my gratitude goes out to all the people that I have met and become friends with throughout my time at Wright State. They have made these two years pass like a blink of the eye.
Finally, I have to thank my family. Without their constant support throughout the years, I would have never been able to get to where I am today. Whether it was just listening to me talk about my work and schooling or providing me with motivation to finish, I could not have done it without them. This achievement would not have been possible without them.

This work is supported by industrial and government partners at the Center for Surveillance Research, a National Science Foundation I/UCRC. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the sponsoring organization.
1 Introduction and Motivation

Introduction

This thesis is designed to provide an overarching description of current 3D modeling techniques, problems with the current 3D modeling techniques, and a description of a novel framework, our generative adversarial network (GAN) framework that is designed to alleviate the problems.

Throughout this thesis, we take a look at 3D models and the variety of methods employed in their creation. Focusing on past and current methods is important as is looking at possible future methods. Looking at the positive and negative aspects of prior and active techniques, allows for new and different way to develop 3D models to be explored.

3D models are import tools in many different ways. Thus the ability to gather and make a collection of accurate 3D models is just as important. 3D models come from a variety of sources. Some are manual scanning through various methods such as LIDAR (Light Detection and Ranging) and LADAR (LAser Detection And Ranging), or even low cost option that can sense depth, such a the XBOX Kinect Camera. Other methods involve the creation of 3D models by hand through various modeling programs. Each of the sources has its own drawbacks. There has even been attempts at incorporating the use of machine learning methods in efforts to create 3D models.

3D models are useful tools in many practical applications. One such practical application that 3D models are useful for is as tools in training recognition systems. The Air
Force Research Laboratory’s Automatic Target Recognition Center uses 3D models to help develop target recognition systems. Such as it is, the accuracy of 3D models is of great importance. An inaccurate model can lead to errors in many of the practical applications, that then can trickle through to the end results.

There are a litany of approaches to produce 3D models that are physically sensed. Each of these methods have their advantages and disadvantages. Many of the manually scanning approaches take an excess of time to render the 3D model. The more detail that is captured by the scanning, the more time that is involved. The higher resolution manual approaches, like LIDAR and LADAR, take far more time than that of a lower resolution manual approach, the the XBOX Kinect Camera. The uses of a manual scanning option such as the XBOX Kinect Camera, has an upside of faster rendering time but for the trade off of lower resolution. All forms of manual scanning approaches share some common disadvantages. A common disadvantage is there exist a limit on the scope that is able to be manually scanned. For example, an XBOX Kinect Camera, is limited to object that can typically fit in a room. While it could be used for larger objects, either resolution would have to be greatly sacrificed, or render time would have to be greatly increased. Another common disadvantage of manual scanning, is the requirement that the object has to be an actual real world object, which can limit that amount of models produced.

There are drawbacks to the approach of creating models by hand through various modeling programs. Firstly, when creating models by hand using a modeling program, expertise in whichever modeling program must be cultivated. Secondly, creation of 3D models by hand through the various programs, is in itself time consuming. The creation of a collection of 3D models can take an abundant amount of time, when a single 3D model takes plenty of time to finish. Another downside of the creation of 3D models by hand through modeling programs, is that the realism of the created 3D model can be off compared to an actual representation of the object would look like.

Approaches that looked to apply machine learning methods in an effort to create 3D
models, have disadvantages and pitfalls. A disadvantage is the requirement for extremely large repositories of 3D models. Usually, a sufficient data set for machine learning consists of thousands of objects. Another disadvantage is in regards to resources, the more complex the problem is, the more computing resources are required. If insufficient resources are available, machine learning training time skyrockets or is unable to be completed. With regards to pitfalls, typically the pitfalls happen in one of two categories, or sometimes both categories. Either the time involved in these methods is exuberant or the approach does not converge to produce realistic 3D models. When referencing exuberant time involved, it has to do with programmed training time, not physical time to train. The pitfall of the approach not converging, can be a result of a variety of different causes. These different causes are an inherit risk in any machine learning approach. In the end, any combination of these disadvantages and pitfalls can cause machine learning approaches to be plagued with problems.

Generative adversarial networks, a type of machine learning algorithm, is a promising approach to mitigate some the problems with other approaches to the creation of 3D models. In a generating adversarial network, two separate networks compete to outperform each other. The underlying purpose of generative adversarial networks, is the generation of data. This underlying purpose makes a generative adversarial network based approach ideal for the creation of 3D models.

Like any approach to the creation of 3D model, generative adversarial networks have potential advantages and disadvantages, but the advantages outweigh the disadvantages. Generative adversarial networks employ unsupervised learning, meaning that the labeling of the data set is unimportant. This is an advantage because there is no consequences if any object in the data set is mislabeled. It also lends itself to wider applications and object classes because the network learns from the implicit patterns in the data rather than explicit identifiers. The design of generative adversarial networks allow one of the networks inside of the generative adversarial network to produce output that mimic the distribution of the
real data that is inputted, and the other network becomes a classifier. The advantage of this, is that the networks compete to become more adept at their job and produce high quality output. Since a generative adversarial network has two networks and it learns the distribution of the input data, a generative adversarial network can produce results with smaller data sets than other machine learning methods.

The major pitfall of using generative adversarial networks is that they are known for being difficult to train. This pitfall is very apparent, because as stated before, generative adversarial networks contain two competing networks. There needs to some equality between the two networks, through the training. If one of the networks becomes an expert at their job at a much faster rate, the other network does not have the chance improve and training fails. A usual cause of this is the vanishing gradient problem. This happens when the generator network does not get enough information and cannot learn. Generative adversarial networks are also prone to experiencing mode collapse. If the generator produces the same output each time, and eventually the other network gets stuck and experience mode collapse. The generative adversarial network may also not converge during training. Some of this pitfalls, are common with other types of machine learning approaches as well.

In this thesis we show how our novel framework for 3D template generation will be structured. The motivation behind this thesis is to develop a way to provide 3D models that are unique while still conforming to a generalized shape structure. The idea is that the 3D models that are generated can then manipulated slightly so that they can then be useful in many different training scenarios. In this thesis we show how it reduces or eliminates the pitfalls of a more traditional generative adversarial network. The proposed framework makes use of the concept of generative adversarial networks. It makes uses of both two dimensional generative adversarial networks and three dimensional generative adversarial networks. The basis of the framework is training a two dimensional generative adversarial network and then using the generator from that network to bootstrap a generator for the three dimensional generative adversarial network. The two dimensional generative adver-
sarial network is trained in the same manner as other two dimensional generative adversarial networks, without suffering the pitfalls that befall generative adversarial networks. By using the two dimensional generator to bootstrap the three dimensional generator, the problem of vanishing gradient and the disparity in the performance of the generator and the discriminator, is reduced and eliminated. Our framework, will also be able to perform faster and with a smaller data set than other traditional generative adversarial networks, due to the bootstrap of the three dimensional generator. Dealing with the common pitfalls that afflict generative adversarial networks boils down to tuning the hyper-parameters of the networks.

Our work has to be evaluated on two levels. First we have to evaluate how our two dimensional generative adversarial network performs. We evaluate our two dimensional generative adversarial network using standard metrics to evaluate two dimensional output of generative adversarial networks as well as qualitative evaluations of the output. Secondly, we have to evaluate the three dimensional generative adversarial network. We use a Hausdorff Distance metric to have a quantitative evaluation as well as using qualitative evaluations of the output.

There have been many takeaways from this thesis. One of the key takeaways, is that the idea of using the two dimensional generator to bootstrap the three dimensional generator, provides multiple benefits. One of the benefits is that the training time required for the three dimensional generator to produce anything slightly resembling the generalized distribution of the input data is greatly reduced. In a short amount of training time, the bootstrapping technique produces a rough approximation of the underlying distribution of the data. In the same amount of time, with the same hyper-parameters, a three dimensional generator without the bootstrap produces an output with little difference from randomized noise. Another benefit, is an increase in the fidelity of the models that are outputted. Comparing models produced by the two methods, the bootstrapped generator produced models with a greater fidelity than the models produced by the generator without the bootstrap
applied. Another key takeaway is the notion that a generative adversarial network performs better, when the generator of that generative adversarial network has some sort of pre-training before the initial generative adversarial network training.

**Chapter Overview**

The rest of the thesis is organized as follows.

**Chapter 2: Related Work**  This section describes work related to generative adversarial networks and work related to 3D models.

**Chapter 3: Preliminaries**  This section outlines the preliminary ideas and theories needed for the methodology.

**Chapter 4: Methodology**  This section outlines the methods used, the theories behind the methods. It will also detail the current approach to machine learning convergence.

**Chapter 5: Results and Evaluation**  This section details the methods used for evaluation of the output, as well as the results of the evaluation methods.

**Chapter 6: Conclusions and Future Work**
2 Background and Related Work

2.1 Background

Machine Learning has evolved throughout its history, from simple perceptrons to the complex networks that exist today. Machine learning has been able to evolve because of a constant inflow of new ideas as well as advancements in technology. One of these new ideas was the concept of a generative adversarial network (GAN). The concept of a generative adversarial network was proposed by Goodfellow [10], in which two networks compete for better and better performance. These networks work against each other to produce results that are a generalization of the input data. Since his proposal, the field of machine learning in regards to GANs, has exploded with many new ideas and exciting work. These advancements have led to quicker and more exact generation of data.

There has been an interest in generative adversarial networks since they were first proposed by Goodfellow [10]. Training of GANs take a large amount of time and there are issues with stability. Either the GANs do not reach convergence or they suffer from mode collapse. The interests in GANs have led to advancements in the field of generative adversarial networks, whether those advancements led to increase in the stability of training or decrease in time taken to train the networks to convergence. One of the first major advances was the introduction of the deep convolutional generative adversarial network (DCGAN) [21]. The proposed framework of the DCGAN has led to an increase in the stability of training. Another advancement was the introduction of the Wasserstein GAN
WGAN drastically reduces mode collapse in training and increases training stability.

There has been a rising interest in 3D modeling and 3D model generation, whether its reconstruction or creation. When it comes to 3D modeling there are three different approaches. Each approach can have its own advantages and disadvantages. There is part-wise based, where parts are generated and then combined. This allows for higher resolution models to be produced. Another approach is view based methods. View based methods generate models from a single image or multiple images. The last approach is voxel based. Voxel based generation are most commonly used with generative adversarial networks. There has been a constant flow of new work being pushed out in regards to each of the approaches.

Part-wise based modeling is advantageous in a variety of ways. Modeling part-wise allows for the models to produce a much higher resolution than the other approaches. While using a part-wise approach, the different parts of the model can be rendered at a much higher resolution. The reason why this is possible is that a single part of the overall object, for example an airplane wing, can be rendered without consideration for the rest of the object. A part-wise approach allows a given framework more resources to use on any given part, compared to other methods that need to consider the entire object and the subsequent resources are diminished. The disadvantages of a part-wise approach is that it requires parts from a 3D object that are labeled, which are unique and often unobtainable for many objects. It is of course possible to obtain a dataset and disassemble the objects into label parts, but this method would be time consuming, regarding the fact that a dataset could contain many thousands of objects.

View based modeling has its advantages as well. View based modeling strongly lends itself to object reconstruction and to object classification. Recent works that make use of view based modeling tend to take a single image and use it to reconstruct a 3D object. These images are most often 2D images or 2.5D sketches. Often when view based modeling is
implemented, autoencoders are heavily involved. In the realm of object classification, view based modeling is useful. View based modeling is very similar to object classification in 2D images. The downside of view based modeling is that it does not lend itself strongly to generation of 3D objects. The reason behind this is that fact that view based modeling does not have access to the total information about a given 3D object.

With voxel based modeling the advantages are more easily seen. Voxel based modeling strongly lends itself to 3D object generation. Voxel based modeling includes point cloud, oct-tree, or other voxel bases. Voxel based modeling can naturally take advantage of volumetric convolutions and deconvolutions. It also allows for laxer labeling requirements. Voxel based modeling has access to the entire amount of information of the given 3D object, unlike view based modeling or part-wise modeling. The major disadvantage of using voxel based modeling is the resource requirements. When modeling, this approach has a much higher dimension of data. With a higher dimensions of data, the detail that can be rendered has to be at a lower resolution.

The framework that is being proposed mixes view-based methods and voxel based approaches.

### 2.2 Related Work

The work of Wu et. al. [32] was among the first to propose a 3D generative adversarial network. The methods proposed (3D-GAN) were a combination of generative adversarial networks with 3D convolutional networks. Other work proposed dealt with 3D object reconstruction Wu et. al [32] and Soltani et. al. [3]. MarrNet [32] took a 2D image and performed a 2.5D sketch estimation. Then maps the output to a 200 dimensions vector. That vector then goes through a decoder network, which is similar to the generator of 3D-GAN. The work of Soltani et. al. was similar to the work of Wu et. al. although with more of a focus on generation of 3D objects versus just the reconstruction of objects from
images. In their work, the inputs were depth maps that were then fed into their networks that would learn the generalized distribution of the input and generate depth maps that were randomly formed from the generalized distribution. They mapped the depth maps back to the 3D space to get 3D objects.

There are many different ways to analyze the accuracy of a model. Different methods have different strengths and weaknesses. Common methods, involve taking two point clouds representing a 3D model and computing distance measurements between the two of them, like the Hausdorff distance, which is the Euclidian norm between two point clouds. Another common method involves reducing the model to features. Then distances between the features are calculated and finally k-nearest neighbors are used to compare similarity of models. There are other, different methods can be used to compare one model to another or compare sets of models. The basics of these methods rely on measuring the similarities between the models. The different approaches can be separated into groups. The different kinds of groups are statistic characteristic based methods, geometry based methods, projection based methods, and topology based methods [5]. Statistic characteristic methods deal with 3D models in meshes and polygon based models. A feature vector is created that is comprised of three components, average surface distance from center, variation in surface distance from center, and inertia moment. The feature vectors of models are compared using Euclidean distance between the vectors and elastic match distance [20]. An example of geometry based methods take 3D models and converts them to weighted point sets, based on spatial location. The similarity is computed by taking 3D models weighted point sets and calculating the Earth Mover’s Distance, also known as the Wasserstein metric [27]. Projection based methods transforms a 3D model into a series of 2D projections, which are then used to find similarities for comparison [5]. Lastly topology based methods involve skeleton trees, which are attribute graphs. In Research on Similarity Measurements of 3D Models Based on Skeleton Trees by Chen et. al. [5], a method is proposed that takes 3D models and converts them to skeleton trees while retaining topological features. The simi-
larities between two models are defined similarities in branches features and nodes. There are methods that are less traditional and those methods are machine learning approaches. Networks can be trained up to recognize 3D models. After training, 3D model analysis can be handled by passing in the 3D model and using the network output to compare similarity to the models that the network was trained on.

There has been work on generating 3D models using various machine learning approaches. Both voxel based 3D models and 3D shape surface models are been generated using various approaches. Octree Generating Networks use a convolutional architecture to generate high resolution 3D models [29]. The octree generating network outputs 3D models in an octree representation. The network consists of custom layers, that operate on octree levels, and yields feature maps of individual levels. After a run through of the network, a full octree representation is produced that is then converted into a dense 3D voxel grid. An advantage of this method is that, unlike dense voxel layers, this method is much less memory intensive and allows for high resolutions to be generated. SurfNet, by Sinha et. al. [26], generates 3D shape surface models. These 3D shape surfaces are generated by deep residual networks. To accomplish the generation of the 3D shape surfaces, several architectures are employed. One network is used to learn the geometry of the 3D surface shape from an image, using down and up residual layers, another architecture to learn features from class labels and view angles, as one-hot encodings. A final network learns the x, y, and z, residuals and combines them into a 3D shape surface.

All of these works were import in forming the framework in this thesis.
3 Preliminaries

We next introduce fundamental concepts and mathematics used in our methodology in this thesis.

Octrees

There are many ways to represent 3D objects. Octrees are one such way. The basis of an octree model is that it is a cell structure with adaptive cell sizes. The way to define an octree is through levels and function values. Function values can be discrete or some continuous value, but most often the values are one or zero. The way to determine that levels is to take a single cell that defines represents the space entirely. Then through recursively partitioning to the desired resolution, the levels get defined. From a single cell, it gets divided into eight octants. Those octants are either labeled with a one or zero, or they remain undefined. The way to determine the label of the octant is to subdivide it into eight octants and look at the labels of those octants. At the desired final resolution, if any part of the octant is filled it is given the label of one. If all the subdivided octants contain the same label, either 1 or 0, then the octant does not get subdivided. All non-divided octants at a given resolution are on the same level. A label of one means that all sub-octants contain a label of one, and a label of zero means that all sub-octants contain a label of zero.

A label of one indicates that the entire voxel is filled and a label of one indicates that the entire voxel is empty. This way the octree is recursively built. There are different ways
to store octree data. One way is to record pointers from the parent voxels to the children. Tatarchenko et. al [29] used hash-tables to store octree data, which is more efficient than the pointers from parents to children. In their method, a given octree cell at level $L$ with spacial coordinates $X = \{x, y, z\}$ can be represent as a tuple $(m, v)$, where $v$ is the value of the cell and $m$ is given by applying a z-order curve, $m = Z(X, L)$. Then an octree can be given by the set of all the tuples $(m, v)$. Lastly an octree can be represented by a 3D matrix with values of any given $(x, y, z)$ being (0,1), indicating at the desired resolution that octant is filled or empty.

![Figure 3.1: Octree Example](image)

Figure 3.1 is an example of an octree object. The octree starts as a single ”cube”. Then that cube is subdivided into 8 separate ”sub-cubes”. The next step, either divides one of the ”sub-cubes” into 8 smaller ”sub-cubes”, or no division occurs. If no division occurs, one of two things happens to that ”sub-cube”. Either that ”sub-cube” is left alone, as can be seen in the forward-top-right ”sub-cube” or that ”sub-cube” is removed, as can be seen in the forwar-bottom-right ”sub-cube”. If the ”sub-cube” is left alone, it means that the entire ”sub-cube” is filled in the model, or if it is removed, it means that the entire ”sub-cube” is empty.
Generative Adversarial Network

A generative adversarial network (GAN) consists of two competing neural networks. These two networks, a generator and a discriminator, are in competition to achieve better performance. Goodfellow et. al. [10] described framework of the generative adversarial network as two adversaries pitted against each other. The generator is akin to counterfeiters and the discriminator is akin to police. The generator (counterfeiter) is trying to produce fake currency and the discriminator (police) is trying to detect the fake currency. The two networks, generator and discriminator, compete, each trying to outperform the other until the fake currency is indistinguishable from real currency, in the proposed example. The discriminator in a generative adversarial network is a classifier network, typically consisting of a convolutional neural network, that outputs a probability \( D(x) \) that the input is real \((1)\) or fake \((0)\). The generator is a generating network, typically consisting of a deconvolutional layers, that outputs generated data that has the same dimensionality of the real data. The two networks participate in a minimax game with the value function \( V(G, D) \):

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim P_{data}}(\log(D(x))) + \mathbb{E}_{z \sim P_z}(\log(1 - D(G(z)))) \tag{3.1}
\]

This equation is stating that \( D \) needs to be maximized for \([\log(D(x))] + \log(1 - D(G(z)))\] and \( G \) needs to minimized for \([\log(D(x))] + \log(1 - D(G(z)))\]. The first term is the probability that \( x \) is from the distribution of real data which would be label one for true and the second term is the probability that \( x \) is from the distribution of fake data, which would be label zero for false. Since the discriminator wants to be correct the terms want to be maximized, and since the generator wants to trick the discriminator, it needs to be minimized. Since the terms are for a single data point, \( \mathbb{E}_{x \sim P_{data}}(x) \) represents the expected values of points coming from the distribution of real data and \( \mathbb{E}_{z \sim P_z}(z) \) represents the expected values of points coming from the noise distribution.
Throughout training, both the discriminator and the generator have different goals. During training the discriminator is trying to maximize the probability of assigning correct labels to real and generated data. Originally the idea was for the generator during training to minimize $\log(1 - D(G(\hat{z})))$. The problem with this was that during the early stages of training the generated data is easily distinguishable from the real data and in turn $\log(1 - D(G(\hat{z})))$ gets saturated. Instead, during training the generator is trying to maximize $\log(D(G(\hat{z})))$. The method proposed for backpropagation in this framework propagates derivatives using the following:

$$\lim_{\sigma \to 0} \nabla_x \mathbb{E}_{\epsilon \sim \mathcal{N}(0, \sigma^2 I)} f(x + \epsilon) = \nabla_x f(x)$$

(3.2)

This equation shows that as $\sigma$ approaches zero, the gradient in respect to $x$, with an expected $\epsilon$ from the distribution applied to $f(x + \epsilon)$ is equivalent to the actual gradient of $f(x)$ which allows for the gradient to be backpropagated through the network.

At the time, others were working a different methods of backpropagation. Other methods of backpropagation were more general stochastic backpropagation, that align closer to the current methods employed in current generative adversarial networks. In current backpropagation, the loss obtained through a loss function or criterion function allows backpropagation through all layers of the network. At the end of training, an ideal situation would one where the generator is producing output that fits the probability distribution of the input data so well that the discriminator is unable to differentiate between generated data and real data. In this case the discriminator would only be correct 50% of the time, effectively guessing whether if what it was given was real or generated.

Figure 3.2 is an example of the structure of a generative adversarial network. The figure shows the flow of the network and the learning. It shows that the real samples and that the generator generates fake samples from the latent space. Both are fed into the discriminator that outputs a real/fake score that is fed back into the generator and the
Figure 3.2: GAN Example

discriminator.
4 Methodology

In this section we present our GAN architecture to generate three dimensional models with two dimensional latent codes.

When developing the architecture to generate three dimensional models, a concern was how to increase fidelity of the generated three dimensional models while at the same time decrease overall training time. The idea started as finding a way to jump start the three dimensional generator. Two dimensional latent codes were decided on as a way to accomplish this task. By using the two dimensional latent codes, the generator of the 3D generative adversarial network is provided with information about the 3D models, that a network without the two dimensional latent codes lacks. The two dimensional latent codes provide information about how the 3D models look along one of the given axes. Originally, the only two dimensional latent codes that were considered were the two dimensional perspectives along the x-axis, y-axis, and z-axis. Additional two dimensional latent codes were added based on the fact that basic but import model characteristics would not be included with only the original two dimensional latent codes. In the framework, one generative adversarial network is tasked with learning these two dimensional latent codes. After the network has learned the two dimensional latent codes, the other network is tasked with learning the three dimensional models, by using the learned aspects from the two dimensional latent codes.

Figure 4.1 is a visual representation of the proposed framework. It shows how the
real data is separated into the 7 perspectives that are fed into the 2D discriminator as well as the latent space is fed into the 2D generator, which generates images that are fed into the 2D discriminated. The 2D discriminator outputs a score that indicates real or fake, and that is fed back into the discriminator and generator respectively. After the training, the latent space is fed into the 2D generator which generated images that are fed into the 3D generator. The 3D generator generates 3D models that are fed into the 3D discriminator along with the real data. The 3D discriminator outputs a score that indicates real or fake, and that is fed back into the discriminator and generator respectively.

4.1 Dataset description

A robust dataset is an important part of any machine learning project. Princeton’s ModelNet is a comprehensive collection of 3D CAD models [34]. ModelNet is comprised of 127,915 CAD models. Those 127,915 CAD models are separated into 662 object categories. A
clean subset of these models, ModelNet10, is comprised of ten popular object categories. From these ten object categories, the models were manually aligned so that all models face the same direction. Another subset of the total data set, ModelNet40, is comprised of 40 object categories. Unlike ModelNet10, ModelNet40 is not aligned. Work done by [24] provide aligned versions of ModelNet40. Two versions are provided, a manually aligned version and an auto-aligned version.

All the object classes that are included in ModelNet10 are also included in ModelNet40, along with 30 other object classes. The object classes included in ModelNet10 were bathtubs, beds, chairs, desks, dressers, monitors, night stands, sofas, tables, and toilets. The 30 object classes added to ModelNet40 are airplanes, benches, bookshelves, bottles, bowls, cars, cones, cups, curtains, doors, flower pots, glass boxes, guitars, keyboards, lamps, laptops, mantels, people, pianos, plants, radios, range hoods, sinks, stairs, stools, tents, tv stands, vases, wardrobes, and xboxes.

The CAD models are provided in Object File Format (OFF). To convert the files from Object File Format to octree models, the code provided for the paper ‘OctNet: Learning Deep 3D Representations at High Resolutions’ [23] proved to be useful. The code reads in the Object File Format and converts the object contained in the file to an octree format. It allows for the depth to be changed to whatever resolution desired. The models can be rotated any number of degrees. After the conversion, the octree models are saved out as MAT files. MAT files are binary files that are the MATLAB default files, which can be also be read into python, which is necessary for our framework.

For the purpose this thesis, we look at three object categories. The object categories that we look at are airplanes, beds, and toilets. The airplane object category is comprised of 726 models. There is some diversity in the airplane object category. Airplanes consist of multiple types of planes, such as, jet planes, passenger planes, prop planes, a few gliders, and one helicopter, seen in Figure 4.2. We remove the helicopter object for a total of 725
Figure 4.2: Example of Airplane Models
Figure 4.3: Example of Bed Models
The bed object category is comprised of 615 models. There is considerable diversity in the bed object category. The bed objects come in a variety of shapes and sizes. The beds have a range of different headboards and footboards. Some of the beds are lifted off the ground and some are on the ground. There are also different types of beds, such as bunk beds. Figure 4.3 illustrates the variety in the bed object category.

The toilet object category is comprised of 444 models. The is substantial diversity in the toilet object category. Toilets come in different shapes. Some toilets are the typical model that are common in most houses. Some are toilets without visible tanks, some don’t have bases, and some have the tanks separated and high above the bowl. Figure 4.4 illustrates the variety in the bed object category.

For each of the data sets, airplanes, beds, and toilets, we convert the CAD models to octree models. For the octree models, we use a resolution of 64. Then we rotate the models 45 degrees clockwise and 45 degrees counter-clockwise. The models were rotated to gain an advantage for the two dimensional latent codes. Without the rotation, important information that would otherwise be missing from the two dimensional latent codes, is included. Since the latent codes coming from the perspectives along the three axis, x-axis, y-axis, z-axis, are basically silhouettes of the model, information about the model that would be seen from an angle, are not included. This is an import because non conformity in models needs to be included in the latent codes. The final output for each object in each class is a $3 \times 64 \times 64 \times 64$ 4D tensor.

### 4.2 Neural Networks

There are a total of four networks in our framework. The framework consists of two generative adversarial networks. One of the generative adversarial networks is a 2D generative adversarial network and the other is the 3D generative adversarial network. The 2D gen-
Figure 4.4: Example of Toilet Models
erative adversarial network is to learn the perspectives of the 3D models and produce a random sample of the 3D perspectives. The 3D generative adversarial network is to learn the shape of the 3D models and produce random models that are in the general shape of the model data set. Each of the generative adversarial networks consists of two networks, a generator and a discriminator.

The types of layers that are used in the networks are convolutional layers, deconvolutional or convolutional transpose layers, batch normalization layers, rectified linear units (ReLU) layers, leaky rectified linear units (leakyReLU), and sigmoid layers. Convolutional layers take input \((N, C_{in}, \text{dim}_{in})\) and output \((N, C_{out}, \text{dim}_{out})\) by

\[
\text{out}(N_i, C_{out,j}) = \text{bias}(C_{out,j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out,j}, k) \ast \text{input}(N_i, k)
\] (4.1)

where \(N\) is the batch size, \(C_{in}\) is the number of input channels, \(\text{dim}_{in}\) is the dimensions of the input, \(C_{out}\) is the number of output channels, and \(\text{dim}_{out}\) is the dimensions of the output. In the equation \(\ast\) is a valid cross-correlation operator. The calculation of \(\text{dim}_{out}\) from \(\text{dim}_{in}\) is

\[
\text{dim}_{out} = \left\lfloor \frac{\text{dim}_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel} \times \text{size} - 1)}{\text{stride}} + 1 \right\rfloor
\] (4.2)

where padding, dilation, kernel size and stride are explicitly stated when the layer is called or a default value is used. Convolutional layers are used to decrease size while increasing dimensionality to learn feature maps.

A deconvolutional, or convolutional transpose layer, is a fractionally strided convolution, where input is \((N, C_{in}, \text{dim}_{in})\) and output \((N, C_{out}, \text{dim}_{out})\). Deconvolutional layers are used to increase size while decreasing dimensionality to combine features, almost an inverse of convolutional layers. The calculation of \(\text{dim}_{out}\) is
\[ \text{dim}_{\text{out}} = (\text{dim}_{\text{in}} - 1) \times \text{stride} - 2 \times \text{padding} + \text{dilation} \times (\text{kernel}_\text{size} - 1) + \text{output}_\text{padding} + 1 \] (4.3)

where stride, padding, dilation, kernel size, and output padding are explicitly stated when the layer is called, or a default value is used.

Batch normalization layers are used to normalize inputs, for each batch, which helps reduce training. Batch normalization uses the equation

\[ y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} \times \gamma + \beta \] (4.4)

where \( E[x] \) is the mean of the input and \( \text{Var}[x] \) is the variance of the input, while \( \gamma \) and \( \beta \) are learnable and are vectors of the same size of the input.

ReLU layers and leakyReLU layers are very similar, but there is a key difference. A ReLU layer applies the element wise function

\[ \text{ReLU}(x) = \max(0, x) \] (4.5)

while a leakyReLU layer applies the element wise function

\[ \text{leakyReLU}(x) = \max(0, x) + \text{negative}_\text{slope} \times \min(0, x) \] (4.6)

Both types of layers are non-linear activation functions, with the difference between the two is that a leakyReLU layer has a negative slope for negative values. A sigmoid layer is another type of non-linear activation functions, that applies the element wise function

\[ \text{Sigmoid}(x) = \frac{1}{1 + \exp(-x)} \] (4.7)
4.2.1 Latent Code Generator

Our framework for the 2D latent code generative adversarial network contains a basic structure. It contains two dimensional convolutional layers, two dimensional batch normalization layers, leaky rectified linear units (leakyReLU) layers, and a sigmoid activation layer at the end. Convolutional layers are common in discriminators for generative adversarial networks. They are used as feature extractors, reducing the size of the input while increasing the dimensionality. In the discriminator, we use the variable \( ncd \) to indicate the factor for channels to increase the dimensionality. For the first convolutional layer, the input is a \( 7 \times 64 \times 64 \) tensor, so the number of input channels is 7 and the number of output channels is \( ncd \). After running through the first layer, the data is of size \( ncd \times 32 \times 32 \). This process continues until the final layers. In the final convolutional layer the data is reduced to \( 1 \times 1 \times 1 \) and passed to a sigmoid layer that return values between zero and one. The returned value, is a prediction from the discriminator on whether the input data consisted of real data or generated data. All the layers of the 2D discriminator are detailed below, in order:

![Figure 4.5: Visual Representation of the 2D Discriminator](image)

Figure 4.5 show a visual representation of the discriminator of the 2D generative adversarial network. The colored blocks represent the layers of the network, with the other blocks representing the input as it goes through the network. The red blocks represent
the 2D convolutional layers, the blue blocks represents the batch normalization layers, the yellow blocks represents LeakyReLU layers, and the green layer representing a sigmoid layers. The discriminator takes the input, passes it through all the layers and returns a single value that is a representation of the probability that the input is real or generated.

class DNet(nn.Module):
    def __init__(self, ncd):
        super(DNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(7, ncd, 2, 2, 0, bias=False),
            nn.LeakyReLU(0.2,True))
        self.conv2 = nn.Sequential(
            nn.Conv2d(ncd, ncd * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ncd * 2),
            nn.LeakyReLU(0.2, True))
        self.conv3 = nn.Sequential(
            nn.Conv2d(ncd * 2, ncd * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ncd * 4),
            nn.LeakyReLU(0.2, True))
        self.conv4 = nn.Sequential(
            nn.Conv2d(ncd * 4, ncd * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ncd * 8),
            nn.LeakyReLU(0.2, True))
        self.conv5 = nn.Sequential(
The generator of the 2D generative adversarial network consists of two dimensional deconvolutional layers, or convolutional transpose layers, two dimensional batch normalization layers, rectified linear units (ReLU) layers, and a sigmoid layer. Deconvolutional layers, or convolutional transpose layers, are common component of generator networks inside of a generative adversarial network. They work in an opposite manner than that of a convolutional layer. The purpose of a deconvolutional layer is to increase input size while reducing dimensionality.

In the generator we use a variable $ncg$ to indicate the factor for channels to decrease the dimensionality. For the first deconvolutional layer, or convolutional transpose layer, the input is a random noise vector. The variable $zdim$ is used to indicate the dimensionality of the noise vector. The layer increases the size and the dimensionality, but only in the first deconvolutional layer. The process is continued in following layers, increasing the size of the input and decreasing the size of the input. In the final deconvolutional layer, the input is increased to match the size of real data as well as the dimensionality of real data, in our case seven. The final sigmoid layer is used to fit the generated data between zero and one.

All the layers of the 2D generator are detailed below, in order:

Figure 4.6 show a visual representation of the generator of the 2D generative adversarial network. The colored blocks represent the layers of the network, with the other blocks representing the input as it goes through the network. The red blocks represents the 2D
deconvolutional layers, the blue blocks represents the batch normalization layers, the yellow blocks represents ReLU layers, and the green layer representing a sigmoid layers. The generator takes a latent vector, passes it through all the layers and returns an output that is the same shape of the real data.

class GNet(nn.Module):
    def __init__(self,ncg):
        super(GNet, self).__init__()

        self.convt1 = nn.Sequential(
            nn.ConvTranspose2d(zdim, ncg * 16, 2, 2, 0, bias=False),
            nn.BatchNorm2d(ncg * 16),
            nn.ReLU(True))

        self.convt2 = nn.Sequential(
            nn.ConvTranspose2d(ncg * 16, ncg * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ncg * 8),
            nn.ReLU(True))

        self.convt3 = nn.Sequential(
            nn.ConvTranspose2d(ncg * 8, ncg * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ncg * 4),
            nn.ReLU(True))
4.2.2 3D Generative Adversarial Network

The framework for the 3D generative adversarial network contains structure that would be expected in a generative adversarial network expanded to be applied to 3D objects. The discriminator of the 3D generative adversarial network is similar to the discriminator of the 2D generative adversarial network. It contains the same type of layers, with the exception that the layers of three dimensional. The discriminator contains three dimensional convolutional layers, three dimensional batch normalization, leakyReLU layers, and a sigmoid layer. The three dimensional convolutional layers behave exactly like two dimensional convolutional layers, except that applies the convolutions over a three dimensional space rather than a two dimensional space. The variable \( ncd \) represents the same idea as it does in the 2D discriminator. The discriminator takes input in the shape of \( 1 \times 64 \times 64 \times 64 \)
and after it passes through all the layers, returns a probability score, between zero and one, which represents whether the discriminator believes that the input is from the real data set or if it is from the generated data. All the layers of the 3D discriminator are detailed below, in order:

![Figure 4.7: Visual Representation of the 3D Discriminator](image)

Figure 4.7 show a visual representation of the discriminator of the 3D generative adversarial network. The colored blocks represent the layers of the network, with the other blocks representing the input as it goes through the network. The red blocks represents the 3D convolutional layers, the blue blocks represents the batch normalization layers, the yellow blocks represents LeakyReLU layers, and the green layer representing a sigmoid layers. The discriminator takes the input, passes it through all the layers and returns a single value that is a representation of the probability that the input is real or generated.

class DNet_3D(nn.Module):
    def __init__(self, ncd):
        super(DNet_3D, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv3d(1, ncd, 2, 2, 0, bias=False),
            nn.LeakyReLU(0.2,True))

        self.conv2 = nn.Sequential(
            nn.Conv3d(ncd, ncd * 2, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ncd * 2),
            nn.LeakyReLU(0.2,True))

        self.conv3 = nn.Sequential(
            nn.Conv3d(ncd * 2, ncd * 4, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ncd * 4),
            nn.LeakyReLU(0.2,True))

        self.conv4 = nn.Sequential(
            nn.Conv3d(ncd * 4, ncd * 8, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ncd * 8),
            nn.LeakyReLU(0.2,True))

        self.conv5 = nn.Sequential(
            nn.Conv3d(ncd * 8, ncd * 16, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ncd * 16),
            nn.LeakyReLU(0.2,True))

        self.conv6 = nn.Sequential(
            nn.Conv3d(ncd * 16, ncd * 32, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ncd * 32),
            nn.LeakyReLU(0.2,True))

        self.conv7 = nn.Sequential(
            nn.Conv3d(ncd * 32, ncd * 64, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ncd * 64),
            nn.LeakyReLU(0.2,True))

        self.conv8 = nn.Sequential(
            nn.Conv3d(ncd * 64, ncd * 128, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ncd * 128),
            nn.LeakyReLU(0.2,True))

        self.conv9 = nn.Sequential(
            nn.Conv3d(ncd * 128, ncd * 256, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ncd * 256),
            nn.LeakyReLU(0.2,True))

        self.conv10 = nn.Sequential(
            nn.Conv3d(ncd * 256, 1, 4, 2, 1, bias=False),
            nn.Sigmoid())
The generator of the 3D generative adversarial network is different from the generator of the 2D generative adversarial network. The generator contains two dimensional deconvolutional, or convolutional transpose, layers, three dimensional deconvolutional, or convolutional transpose layers, two dimensional batch normalization layers, three dimensional batch normalization layers, ReLU layers, and potential two sigmoid layers. Since the 3D generator is bootstrapped by the 2D generator, it is necessary to include both the 2D and 3D deconvolutional, or convolutional transpose layers. The two dimensional deconvolutional, or convolutional transpose, layers are used to bring the input of shape $7 \times 64 \times 64$, which is
the shape of the data outputted from the 2D generator, to data with the shape of $64 \times 64 \times 64$ which then is expanded to the shape $1 \times 64 \times 64 \times 64$ which allows the data to be passed through the 3D deconvolutional or convolutional transpose layers. After a complete pass of the generator, the outputted data is in the same shape as data from the data set. All the layers of the 3D generator are detailed below, in order:

**Figure 4.8: Visual Representation of the 3D Generator**

Figure 4.8 show a visual representation of the generator of the 2D generative adversarial network. The colored blocks represent the layers of the network, with the other blocks representing the input as it goes through the network. The red blocks represents the 3D deconvolutional layers, the blue blocks represents the batch normalization layers, the yellow blocks represents ReLU layers, and the green layer representing a sigmoid layers. The blue circle represent the 2D generator. The generator takes a latent vector, passes it through the 2D generator and passes the output through all the layers and returns an output that is the same shape of the real data.

```python
class GNet_3D(nn.Module):
    def __init__(self, ncg):
```


super(GNet_3D, self).__init__()

self.convt1 = nn.Sequential(
    nn.ConvTranspose2d(7, ncg, 3, 1, 1, bias=False),
    nn.BatchNorm2d(ncg),
    nn.ReLU(True))

self.convt2 = nn.Sequential(
    nn.ConvTranspose2d(ncg, ncg * 2, 3, 1, 1, bias=False),
    nn.BatchNorm2d(ncg * 2),
    nn.ReLU(True))

self.convt3 = nn.Sequential(
    nn.ConvTranspose2d(ncg * 2, ncg * 4, 3, 1, 1, bias=False),
    nn.BatchNorm2d(ncg * 4),
    nn.ReLU(True))

self.convt4 = nn.Sequential(
    nn.ConvTranspose2d(ncg * 4, 64, 3, 1, 1, bias=False),
    nn.Sigmoid())

self.convt5 = nn.Sequential(
    nn.ConvTranspose3d(1, ncg * 4, 3, 1, 1, bias=False),
    nn.BatchNorm3d(ncg * 4),
    nn.ReLU(True))

self.convt6 = nn.Sequential(
    nn.ConvTranspose3d(ncg * 4, ncg * 2, 3, 1, 1, bias=False),
    nn.BatchNorm3d(ncg * 2),
    nn.ReLU(True))
4.3 Model Training

Training is an important part in any creation of a framework. Missteps in training can render a framework unstable, unreliable and unusable. Training can take much time and careful tweaking to make it work. Generative adversarial networks require more attention to detail and careful tweaking compared to some of the other machine learning networks.

Both of the networks, the 2D generative adversarial network and the 3D generative adversarial network, have the same basic training structure. The training data set is separated into training mini-batches. For each minibatch the discriminator is trained and then the generator is trained.

The minibatch is fed into the discriminator network. The output from the discriminator is used for the loss function, with a label of one applied, indicating the output is from the real data. The loss function returns a value that indicates the performance of the discriminator, on the real data. The loss is then back propagated through the discriminator. A random sample, of the same size is then eventually fed into the discriminator. The random sample comes from a normalized distribution. It is fed into the generator. Output from the generator is fed back into the discriminator. From there the output is used in the same
loss function, but this time the labels applied is zero to indicate that the data is generated data. The loss function returns another value, that again indicates the performance of the discriminator. This value is then also back propagated through the discriminator. The discriminator optimizer is then stepped, which updates the parameters, based on the backward passes.

After the discriminator has been trained, its time to train the generator. Training the generator involves less steps than training the discriminator. The same random sample that was used in the discriminator training is used again. It is once again fed into the generator. The output of the generator feeds into the updated discriminator. The discriminator is fed into the loss function, with the labels applied being one. The reason the labels are one, is because the generator is trying to trick the discriminator into believing that the output from the generator is real data. The loss is then back propagated through the generator. After the back propagation, the generator optimizer is stepped, again updating the parameters.

This process continues through all the minibatches that the data set was separated into. After cycling through all the minibatches, the training has complete one epoch of training. The process continues for as many epochs has explicitly stated or until a certain predetermined threshold has been met.

4.3.1 Latent Code Generator

The data set that is used for our 2D generative adversarial network is saved as mat files, in a $3 \times 64 \times 64 \times 64$ array. The first $1 \times 64 \times 64 \times 64$ representing the model with no rotation, the second representing the model with 45 degree clockwise rotation, and the third representing the model with 45 degree counter-clockwise rotation. From each of the those we take the max along the x-axis and y-axis, along with the max along the z-axis for the non-rotated model. The purpose of taking the max along each of these axis is to obtain a perspective of the models along these axis. The results of this is that the training data is a $7 \times 64 \times 64$ array. The network is able to uses this data like a regular image, but instead
of 3 color channels, there are 7 color channels. To fit the size of the input data, we need the generator to output data of the shape $7 \times 64 \times 64$. We use a minibatch size of only 8. The reason for this is that our data set is on the small size. There is also disagreement within the machine learning community on whether smaller or larger minibatches are more desirable [16]

The loss function that we use for the 2D generative adversarial network is binary cross entropy with loss, which can be described by

$$l(x, y) = L = \{l_1, l_2, \ldots, l_n\}^T$$

$$l_n = -w_n[y_n \cdot \log(x_n) + (1 - y_n) \cdot \log(1 - x_n)]$$

where $w_n$ represents the weight, that get updated, $y_n$ are the label, and $x_n$ are the probability scores.

The same type of optimizer is used for both the discriminator and the generator. The optimizer in use is the ADAM optimizer, which implements the ADAM algorithm, proposed in the paper, ADAM: A Method For Stochastic Optimization, by Kingma and Ba [17]. The learning rate used for the discriminator is $1e^{-6}$ and the learning rate used for the generator is $2e^{-5}$. As pointed out in the above sections, the variables $ncg$ and $ncd$ were used to indicate the channel factors that were used. In the 2D generative adversarial network $ncg = 128$ and $ncd = 64$

The number of epochs used for the 2D generative adversarial network was 250. Originally training was set to 300 epochs, but after multiple tests, the network converged after 250 epochs no further training resulted in any qualitative increases. The state of the generator after training ends, is saved and passed to the 3D generative adversarial network.
4.3.2 3D Generative Adversarial Network

The data set that the 3D generative adversarial network uses is the same as the 2D generative network. The difference comes when the arrays are loaded in, the 3D generative adversarial network only takes the first $1 \times 64 \times 64 \times 64$ tensor, which is the non-rotated model. The 3D generative adversarial network has the same minibatch size, loss function, and optimizers as the 2D generative adversarial network. The learning rates for the optimizers also match the learning rates of the 2D optimizers. There is a difference in the $ncg$ and $ncd$ values used in the 3D generative adversarial network. In this case, $ncg = 16$ and $ncd = 8$.

During training, several variations of the generator are experimented with. The above depiction of the generator contains all the layers that have been used. Three variations were used in the experimentation. The first variant, only consisted of 2D deconvolutions, to take the input to the 3D generator, which is the output for the 2D generator, of size $7 \times 64 \times 64$, to the desired size of $64 \times 64 \times 64$. The second variant, consisting of one 2D deconvolution, that takes the size from $7 \times 64 \times 64$ to $64 \times 64 \times 64$. Then the output from that deconvolution has its dimensions expanded to $1 \times 64 \times 64 \times 64$. Then the 3D deconvolutional layers are applied. The third variant is a combination of the first and second variants. It contains all of the layers that are listed above. Several 2D deconvolution layers are applied, then the dimensions are expanded, and the 3D deconvolution layers are applied.
5 Evaluation

In this chapter, the results of both the 2D generative adversarial network and the 3D generative adversarial network will be provided. The proposed GAN is evaluated over three different classes of objects: toilets with the most diversity, beds, with the next most diversity, and planes, with the least diversity. Diversity of the object class was measured by the amount of variance in the data set. The airplanes object class had the least amount of variance, while the bed object class had the next amount, and the toilet object class had the most variance. The Hausdorff Distance metric also provides evidence of the diversity of the object classes. By taking the Hausdorff Distance between all the models in each object class, it also show the amount of variance, with lower values indicating more similarity. Again, the airplanes object class had the lowest Hausdorff Distance, the bed object class has the next lowest Hausdorff Distance, and the toilet object class has the highest Hausdorff Distance. The amount of variance in the data set is important because it allows insight into how much difference is in the object class, which is useful in determining how easily the network can generalize object features. Evaluation was performed on each of the object classes. Evaluations hold great importance. It allows for reflection on the work and provides guidance on how to proceed. The evaluation will be done on the outputs from the 2D generative adversarial network and the outputs from the 3D generative adversarial networks, i.e. the results of the noise vectors being passed through the generators at the end of training. For the 3D generative adversarial network, there will be evaluation of the output from a network trained with the bootstrapped generator and a network trained without the
This evaluation will serve to illustrate the performance of both networks and provide a comparative analysis of the networks. The evaluation will show that the networks that were provided with the 2D latent codes perform better, with a higher fidelity, and perform faster than the networks that are not provided the 2D latent codes. The results will be shown in both a quantitative and qualitative manner. There are two different methods that were used as a quantitative evaluation. For the 2D generative adversarial network, the evaluation comes from an Inception Score, which uses a pretrained Inception V3 network. It evaluates the quality of generated images similar to a human evaluation. The output of the Inception V3 network returns a probability distribution of labels, which sums to 1 and sums the label distribution to create a marginal distribution, which indicates the variety of the generators output. By comparing the two distributions, using KL divergence, the Inception Score is created. The more the two distributions differ, the higher the Inception Score. The best scores are obtained when the probability distribution of the labels are narrow, one label more likely than the others, and the marginal distribution are uniform over all the labels. The Inception Score gives a high value when the Inception V3 can clearly identify an object in the input, by this score high quality images are those that the network can identify an object in the image and classify it right. There are some downsides to this metric. The Inception Score will always be low if the network is evaluating output from a generator that the network was not trained on, as is the case in this evaluation. For that reason, this metric is useful for comparison to a baseline value \cite{19}. This provides a decent metric, but only when compared to the real images and not taking into account other data sets. The other metric used is the Hausdorff Distance metric. The Hausdorff Distance metric is defined as the Euclidian Norm between the difference of two data sets. This is the metric used to measure the generated models in the 3D generative adversarial network. The Hausdorff Distance is also shown for the generated perspectives to show continuity in the evaluation.

In the following sections, the evaluation and results of each object class will be dis-
played and investigated. The methods stated above, are uninformative without explanation and a proper baseline to compare the values with. To achieve the baseline for the Inception Score, the real data set was fed through the Inception V3 Network, each perspective by itself. It must be stated, that comparing the results with other data sets provides skewed values that misrepresent the actual performance of the 2D generative adversarial network. This Inception Score is useful for comparing the generated images and the real images. The baseline for the Hausdorff Distance was computed in another manner. Every model is compared against every other model in the data set and the metric is calculated. The mean value of the metric is baseline, which is used to measure performance.

For the evaluation of the performance, a 512 random sampling is used. The random sample is drawn from a normalized distribution between zero and one. The random sample is fed into the generator to acquire a $512 \times 7 \times 64 \times 64$ which is then used to get the training Inception Score and also is used to find the Hausdorff distance. In the case for the 3D models, the random sampling is fed into the 2D generator and that output is fed into the 3D generator. The final product is what is then used to calculate the Hausdorff distance for the 3D models. The same random sampling is used throughout training to ensure that the metrics and evaluation are meaningful.

5.1 Toilet Object Class

5.1.1 Latent Code Generator

The Inception Score baseline for the Toilet object class is 1.9802. This value was obtained by taking the entirety of the data set, all the models in the bed object class, and running them through the Inception V3 Network. The actual value show the ability of the network to recognize the objects in the data set, but since the Inception V3 Network was not trained on the object class the Inception Score values are low. This means that very little actual
information is obtained from the value itself, but the baseline value is useful for comparing how similar the generated perspectives are to the real perspectives. The Hausdorff Distance baseline for the Toilet object class 2D perspectives is 77.2478. This value was obtained by taking the entirety of the data set and computing the Hausdorff Distance between every object and the rest, and then taking the mean of the values. The baseline value, shows a similarity of the data set, with values closer to zero representing a data set that is similar and values farther from zero representing a data set that is dissimilar. The value can be seen as a partial similarity metric but it is also useful for comparison. By using the baseline value as a comparison against the values of the generated data, it is possible to show similarity between the real data and the generated data.

In Figure 5.1c the graphs represent the results of our evaluation for the training of the 2D generative adversarial network on the Toilet object class. By defining $B_{IS}$ as the baseline value for the Inception Score, $IS$ as the Inception Score of the generated images, $B_{HD}$ as the baseline value for the Hausdorff Distance, and $HD$ as the Hausdorff Distance of the generated images, the values represented by the graphs are defined by $\text{abs}(B_{IS} - IS)$ and $\text{abs}(B_{HD} - HD)$. When analyzing the results of this class, there are differences compared to the evaluation of the other classes. In this object class, the Inception Scores, appear to increase, but if the several of the first epochs are ignored, the progression of the Inception Scores behaves in the same manner as the Inception Scores in the other two cases. The Hausdorff distance behaves in the same way. If several of the first epochs are ignored, the progression of the Hausdorff distance have the same progression of the other object classes. The values for the first several epochs are disparate with the visual output from the network. After the disparate epochs, the visual output matches with the values from the evaluation metrics. The two graphs show the general performance of the generator from the 2D generative adversarial network trained on the Toilet object class. Ignoring the several of the first training epochs, the values of the Inception Scores remain high but there values do decrease and in the case of the Hausdorff Distance the values do approach zero. This
(a) Inception Scores

(b) Hausdorff Distance

(c) Evaluations for 2D GAN trained on Toilet object class
indicates that the networks performance increases. By comparing the scores with the output from the network, it shows how well the scores represent the output from the network.

Figure 5.2 shows a sample of the generated perspectives on top of a sample of the real perspectives, for the toilet object class. By comparing the output perspectives and the real perspectives, it shows that the scores are a good indicator of the output.

5.1.2 3D Generative Adversarial Network

The Hausdorff Distance baseline for the Toilet object class 3D models is 149.9582. This value was obtained by taking the entirety of the data set and computing the Hausdorff Distance between every object and the rest, and then taking the mean of the values. The baseline value, shows a similarity of the data set, with values closer to zero representing a data set that is similar and values farther from zero representing a data set that is dissimilar. The value, can be seen as a partial similarity metric, but it is also useful for comparison. By using the baseline value as a comparison against the values of the generated data, it is possible to show similarity between the real data and the generated data.

In Figure 5.3d three graphs are used to indicate the Hausdorff Distances of the generated 3D models. By defining $B_{HD}$ as the baseline value for Hausdorff Distance and $HD$ as the Hausdorff Distance for the generated models, the values are calculated by $\text{abs}(B_{HD} - HD)$. The first graph, fig 5.3a, show the Hausdorff Distance for training the 3D generative adversarial network, with the use of the latent codes. Figure 5.4a is a sample output for the 3D generator, with the latent codes, from when the 3D network performed the best. The second graph, fig 5.3b, show the Hausdorff Distance for training the 3D generative adversarial network, without the use of the latent codes. Figure 5.10a is a sample output for the 3D generator, without the latent codes, from when the 3D network performed the best. The last graph, fig 5.3c, shows the comparison of the two types of
(a) Hausdorff Distance with Perspective

(b) Hausdorff Distance with No Perspective

(c) Compared Hausdorff Distance

(d) Evaluations for 3D GAN trained on Toilet object class
(a) Generated Octrees with Latent Codes
(a) Generated Octrees without Latent Codes
training. The reason that there exists a point in the training with both types of networks, is because during training, the Hausdorff Distance passes the baseline that was established. As with the training of the 2D network, several of the first epochs of both networks have increasing values. By looking at Fig. 5.3a, the Hausdorff Distance is increasing overall. Several times the Hausdorff Distance drops. These drops correlate with improvements in the 3D models generated by the network. As the training continues, the Hausdorff Distance keeps increasing and this correlates to the generated 3D models becoming more noisy. By looking at Fig 5.3b, the Hausdorff Distance increases and basically levels out. The output of the 3D models generated by the network without latent codes, are very sparse, with most of the 3D model being empty space. The third graph show that even if the models generated by the network trained with latent codes have a higher fidelity than the models generated by the network that was trained without the latent codes.

It's important to note that ignoring the dip towards zero in both training cases, the rest of the training data shows that the training of the network with the latent codes performs better than the network without the latent codes. By comparing the output of the networks to the Hausdorff Distance metric, it shows that the metric does correlate with the performance of the networks. Comparing the output of the two networks to each other, it corroborates with the idea that the network trained with the latent codes perform better than the networks with the latent codes.

5.2 Bed Object Class

5.2.1 Latent Code Generator

The Inception Score baseline for the Bed object class is 1.5902. This value was obtained by taking the entirety of the data set, all the models in the bed object class, and running them through the Inception V3 Network. The actual value show the ability of the network
to recognize the objects in the data set, but since the Inception V3 Network was not trained on the object class the Inception Score values are low. This fact means that very little actual information is obtained from the value itself, but the baseline value is useful for comparing how similar the generated perspectives are to the real perspectives. The Hausdorff Distance baseline for the Bed object class 2D perspectives is 65.6947. This value was obtained by taking the entirety of the data set and computing the Hausdorff Distance between every object and the rest, and then taking the mean of the values. The baseline value, shows a similarity of the data set, with values closer to zero representing a data set that is similar and values farther from zero representing a data set that is dissimilar. The value, can be seen as a partial similarity metric, but it is also useful for comparison. By using the baseline value as a comparison against the values of the generated data, it is possible to show similarity between the real data and the generated data.

In Figure 5.6c the graphs represent the results of our evaluation for training of the 2D generative adversarial network on the Bed object class. The values represented by the graphs are defined by $\text{abs}(B_{IS} - IS)$ and $\text{abs}(B_{HD} - HD)$. With this representation, the indication of the best performance is when the values are the closest to zero. Inspection of the two graphs, does indicate that the performance of the 2D generative adversarial network improves from start to finish. In Figure 5.6a, towards the end of training there is some oscillation in the Inception Scores, but the fact that there exist a downward trend indicates improvement. In Figure 5.6b during training there exists ups and downs until after the 200th training epoch, which from that point the network improves greatly. Eventually reaching the baseline of the real models, around the end of training. As stated before, the closer the values are to zero, the better performance that is exhibited by the 2D network. The output from this 2D generator reflects the outputs from the evaluation metrics. The two graphs show the general performance of the generator from the 2D generative adversarial network trained on the Bed object class. Since both of the metric approach zero, which is the goal of the network, this indicates that the generator is performing well, i.e. generating
(a) Inception Scores

(b) Hausdorff Distance

(c) Evaluations for 2D GAN trained on Bed object class
perspectives that appear realistic. While the scores look good, it is important to also examine the output of the generator alongside the scores. When comparing the scores with the generated output, it is easy to see that the generated output looks similar to the real data set.

Figure 5.7 shows a sample of the generated perspectives on top of a sample of the real perspectives, for the Bed object class. By comparing the output perspectives and the real perspectives, it shows that the scores are a good indicator of the output.

5.2.2 3D Generative Adversarial Network

The Hausdorff Distance baseline for the Bed object class 3D models is 138.3105. This value was obtained by taking the entirety of the data set and computing the Hausdorff Distance between every object and the rest, and then taking the mean of the values. The baseline value, shows a similarity of the data set, with values closer to zero representing a data set that is similar and values farther from zero representing a data set that is dissimilar. The value, can be seen as a partial similarity metric, but it is also useful for comparison. By using the baseline value as a comparison against the values of the generated data, it is possible to show similarity between the real data and the generated data.

In Figure 5.8d three graphs are used to indicate the Hausdorff Distances of the generated 3D models. By defining $B_{HD}$ as the baseline value for Hausdorff Distance and $HD$ as the Hausdorff Distance for the generated models, the values are calculated by $abs(B_{HD} - HD)$. The first graph, figure 5.8a, depicts the Hausdorff Distance for training of the 3D generative adversarial network, with the latent codes used, i.e. the generator is bootstrapped by the generator from the 2D generative adversarial network. Figure 5.9a is a sample output for the 3D generator, with the latent codes, from when the 3D network performed the best. The second graph, figure 5.8b, depicts the Hausdorff Distance for training
Figure 5.7: Generated Output vs. Real Images
(a) Hausdorff Distance with Perspective

(b) Hausdorff Distance with No Perspective

(c) Compared Hausdorff Distance

(d) Evaluations for 3D GAN trained on Bed object class
(a) Generated Octrees with Latent Codes
(a) Generated Octrees without Latent Codes
of the 3D generative adversarial network, without the latent codes used, i.e. the generator is
not bootstrapped by the generator from the 2D generative adversarial network. Figure 5.10a
is a sample output for the 3D generator, without the latent codes, from when the 3D network
performed the best. The last graph, figure 5.8c, is the training of the bootstrapped network
vs. the training of the non-bootstrapped network. As the values get closer to zero, the 3D
model should be a higher fidelity. In this case, the Hausdorff Distances never get closer
to zero than 70. This indicates that the fidelity of the 3D models do not have high fidelity.
This is in part because of the high diversity in the real data set. The 3D models generated
by the network with the latent codes and the 3D models generated by the network without
the latent codes are have a similar Hausdorff Distance. Despite the fact that the network
without the use of latent codes having a smaller minimum value, the 3D models generated
by the network with the use of latent codes, have a higher fidelity.

It’s important to note that ignoring the dip towards zero in both training cases, the rest
of the training data shows that the training of the network with the latent codes performs
slightly better than the network without the latent codes. The two networks have a similar
range in the Hausdorff Distance metric. This is potentially because almost every object in
the object class is a general rectangular cube. This allows the network trained without the
latent codes, to be performing relatively on par with the network trained with the latent
codes. By comparing the output of the networks to the Hausdorff Distance metric, it shows
that the metric does correlate with the performance of the networks. The comparison of the
output of the two networks with each other, does not readily show that the network with
the latent codes performs better than the network without the latent codes.
5.3 Airplane Object Class

5.3.1 Latent Code Generator

The Inception Score baseline for the Airplane object class is 1.5864. This value was obtained by taking the entirety of the data set, all the models in the bed object class, and running them through the Inception V3 Network. The actual value shows the ability of the network to recognize the objects in the data set, but since the Inception V3 Network was not trained on the object class the Inception Score values are low. This fact means that very little actual information is obtained from the value itself, but the baseline value is useful for comparing how similar the generated perspectives are to the real perspectives. The Hausdorff Distance baseline for the Airplane object class 2D perspectives is 42.8525. This value was obtained by taking the entirety of the data set and computing the Hausdorff Distance between every object and the rest, and then taking the mean of the values. The baseline value shows a similarity of the data set, with values closer to zero representing a data set that is similar and values farther from zero representing a data set that is dissimilar. The value can be seen as a partial similarity metric, but it is also useful for comparison. By using the baseline value as a comparison against the values of the generated data, it is possible to show similarity between the real data and the generated data.

In Figure 5.11c the graphs represent the results of our evaluation for the training of the 2D generative adversarial network on the Airplane object class. The values represented by the graphs are defined by $\text{abs}(B_{IS} - IS)$ and $\text{abs}(B_{HD} - HD)$. With this representation, the indication of the best performance is when the values are the closest to zero. Inspection of the two graphs does indicate that the performance of the 2D generative adversarial network improves from start to finish. In Figure 5.11a, towards the end of training there is a good deal of oscillation after a number of epochs in the Inception Scores, but the fact that there exist a downward trend indicates improvement. In Figure 5.11b during training there exists
(a) Inception Scores

(b) Hausdorff Distance

(c) Evaluations for 2D GAN trained on Airplane object class
ups and downs in the training, mostly a downward trend. After around the 150th epoch, the values for the Hausdorff distance are very close to zero, which means close to the baseline. The combination of these scores, are how to interpret the performance of the 2D network. The best performance of the network occurs when the lowest values of both measurements are at the lowest point. The outputs of this network match the evaluation metrics. The two graphs have a certain correlation that show how well the network performs overtime. Both of the graphs trend towards zero, which indicates that the networks performance is increasing. The scores look good but it is important to correlate the scores with the actual output from the network. Figure 5.12 shows a sample of the generated perspectives on top of a sample of the real perspectives, for the Airplane object class. By comparing the output perspectives and the real perspectives, it shows that the scores are a good indicator of the output.

### 5.3.2 3D Generative Adversarial Network

The Hausdorff Distance baseline for the Airplane object class 3D models is 54.2953. This value was obtained by taking the entirety of the data set and computing the Hausdorff Distance between every object and the rest, and then taking the mean of the values. The baseline value, shows a similarity of the data set, with values closer to zero representing a data set that is similar and values farther from zero representing a data set that is dissimilar. The value, can be seen as a partial similarity metric, but it is also useful for comparison. By using the baseline value as a comparison against the values of the generated data, it is possible to show similarity between the real data and the generated data.

The training data shows that the training of the network with the latent codes performs better than the network without the latent codes. By comparing the output of the networks to the Hausdorff Distance metric, it shows that the metric does correlate with the performance of the networks. Comparing the output of the two networks to each other, it
Figure 5.12: Generated Output vs. Real Images
corroborates with the idea that the network trained with the latent codes perform better than the networks with the latent codes.

In Figure 5.13d three graphs are used to indicate the Hausdorff Distances of the generated 3D models. By defining $B_{HD}$ as the baseline value for Hausdorff Distance and $HD$ as the Hausdorff Distance for the generated models, the values are calculated by $\text{abs}(B_{HD} - HD)$. The first graph, fig 5.13a, show the Hausdorff Distance for training the 3D generative adversarial network, with the use of the latent codes. Figure 5.14a is a sample of the output from the generator, with latent codes, wherein the network performs the best. The second graph, fig 5.13b, show the Hausdorff Distance for training the 3D generative adversarial network, without the use of the latent codes. Figure 5.15a is a sample of the output from the generator, without latent codes, wherein the network performs the best. The last graph, fig 5.13c, shows the comparison of the two types of training. Looking at the first graph, its easy to see that the Hausdorff Distance decreases, and then starts to slightly oscillate after about halfway through training, but with a downward trend. The lowest value from the first graph, correlates to the highest fidelity models. By analyzing the second graph, the Hausdorff Distance decreases quickly, and then levels out. The reason behind this is because the network learns the empty space quickly, but does not learn the generalized shape of the real data set. The graph that contains the comparison between the network trained with the latent codes and the network trained without the latent codes, shows the difference in the fidelity of the output of the two networks. It shows that the network with latent codes produces 3D models with a higher fidelity.

5.4 Comparative Analysis

A comparative analysis of the three classes is important to examine the validity of the proposed framework. The framework produces much better results for the airplane object
(a) Hausdorff Distance with Perspective

(b) Hausdorff Distance with No Perspective

(c) Compared Hausdorff Distance

(d) Evaluations for 3D GAN trained on Airplane object class
(a) Generated Octrees with Latent Codes
(a) Generated Octrees without Latent Codes
class, than the bed object class or the toilet object class. There exist significant variation in the output from the 3D generative adversarial networks trained on the different object classes. The network trained on the airplane object class produces better output than the other two classes. This can be seen in both the evaluation metrics as well as the sample of generated models shown. There could be several reason that the output from the network trained on the airplane object class is better than the output from the networks trained on the bed object class and the toilet object class. The difference could be a result of a relatively small amount of training or the small amount of real data in the respective data sets. The airplane object class has the most amount of models in the data, followed by the number of models in the bed object class, followed by the number of models in the toilet object class. The difference in the number of models in the data set correlates with the difference in performance. The airplane object class also has a smaller amount of variance in the data set compared to the variance in the other two object classes. In fact, the variance in the airplane data set is an order of magnitude less than the variance in the other two object classes. Several methods could be implemented to improve the training and the output from the other two object classes. Implementing more training time for the other two classes would yield an opportunity for the networks to better generalize the basic structures of the object classes. Since there is also difference in the size of the data sets, more models for the two classes might help alleviate the difference in performance. Lastly, finding a way to decrease the variance in the two data sets would most certainly decrease the difference in performance between the three networks. Changing aspects of the networks, different layers, changing learning rates, and loss functions, for example, could also increase performance for any of the networks.

The evaluation has shown that the network architecture is promising. For each object class, it is clear to see that the networks trained with the 2D perspective codes perform better than the networks trained without the 2D perspective codes. Although the output is far from perfect, the models generated by the framework have a much high fidelity than
models generated by a basic 3D generative adversarial network. The framework is able to generate the realistic looking models with much less training time than a basic 3D generative adversarial network. Based on the performance of the framework, it shows that this framework is a promising step to generate high fidelity models as well as larger models.
6 Conclusion

This thesis proposed an adversarial framework for deep 3D target template generation. It has shown the reason why such a framework is needed and has merit. This paper has delved into past and current methods of 3D modeling and background on generative adversarial networks. The explicit structure of the adversarial framework has been shown and explained. Evaluations of the outputs have shown that this method has ample potential and deserves further pursuit.

This framework deserves further inspection. As machine learning techniques advance and generative adversarial networks expand, this method can wield fantastic results. There are current methods that can be experimented with that could produce better results. One of the most important ideas that should be looked at to be implemented in the future, is an end to end framework. Instead of separately training the 2D generative adversarial network and then training the 3D generative adversarial network, an end to end framework trains the 2D and the 3D generative adversarial networks at the same time. The next step in research is to change the architecture of the 3D generator. The idea is to take the three variants of the generator and instead of using deconvolutional layers, exchange all of them, both the 2D deconvolutional layers and 3D convolutional layers, and replace them with 2D convolutional layers and 3D convolutional layers. After experimenting with that architecture, the next step would be to experiment with a combination of deconvolutional layers and convolutional layers, using 2D deconvolutional layers and 3D convolutional layers, since the size of the input doesn’t change when using either 3D convolutional or deconvolutional
layers. After those 2 experiments, the next step is to take the best performing framework and combine the training of the latent code network and the training of the 3D model generation network into one end to end network. The end to end framework would consist of the 2D discriminator being updated, then the 3D discriminator being updated, then the 2D generator being updated, and finally the 3D generator being updated. A final future experiment would be to implement a progressive GAN with the current framework. A progressive GAN [15] generates low resolution output and progressively increase the size of the output.
Bibliography


[35] Bo Yang, Hongkai Wen, Sen Wang, Ronald Clark, Andrew Markham, and Niki Trigoni. 3d object reconstruction from a single depth view with adversarial learn-