CSE 291I Project Report
Generative 3D Compression

Vamsidhar Kamanuru
A53249571
UC San Diego
vkamanur@eng.ucsd.edu

Techit Limtiyakul
A53237647
UC San Diego
tlimtiya@eng.ucsd.edu

1. Introduction

We propose a Generative 3D Compression model for compressing 3D volumetric objects. The model has an advantage over traditional compression method using linear transformation in a sense that the model can learn an arbitrary encoding function via backpropagation. While a simple autoencoder models can easily achieve that, the reconstructed object are usually lossy and quite unpleasant. Our model uses generative network(3D GAN) to guide the direction of the decoder to further improve the quality of the reconstructed object.

We worked on implementing 3D-GAN, with the methodologies developed by Wu et al.[13], for 3D object generation. The 3D-GAN model is then extended to a Generative 3D Compression network by using the adversarially trained genenerator network as a decoder. In addition, we incorporate perceptual loss into the training of the network to help the network capture low-level features. The model is able to compress volumetric objects by over 164 times and is still able to recover most of the important features of the objects. Finally, we compare the object reconstructions produced by this network with the traditional auto-encoder reconstructed models to compare the quality of the reconstruction.

1.1. Contributions

Techit Limtiyakul: Contributed in creating the structure for training the Generative 3D Compression model. For the 3D-GAN, he experimented with some of the tips to improve GAN performance, fine tuning the hyperparameters, as well as adapting the network architecture so the model works well for the dataset. In addition, he also contributed to the Training process and output visualization.

Vamsidhar Kamanuru: Contributed to the modelling of parts of Generative 3D compression network and also with the training. Also the working of several utilities required for loading the training set, 3D visualization of the models and debugging of the network architecture to identify to the bottlenecks during training process. He also contributed to improving in the generation of better quality 3D models by the GAN.

Himanshu Jaiswal: Himanshu contributed in 3D GAN network implementation part. This project is an extension of 3D GAN

2. Background

Goodfellow et al. [4] proposed the Generative Adversarial Network using the generator and discriminator for generative modeling problem. Santurkar et al. [10] applied Generative Adversarial Network model to a compression problem, using the generator in GAN as a pretrained decoder as well as using a classifier network to capture low-level features.

Maturana and Scherer [8] proposed VoxNet, a 3D Convolutional Neural Network for Volumetric object which is capable of labeling 3D objects in real time. Brock et al. [1] used variational autoencoder to build a generative and discriminative voxel models. Wu et al. [13] combined the approach of GAN and VoxNet, called 3D-GAN, to generate 3D objects from probabilistic space. In addition, the author extended a Variational Autoencoder [7] to map 2D images to the latent space of 3D-GAN. This allows the model to reconstruct 3D objects from 2D images. Smith and Meger [11] extended 3D-GAN using Improved Wasserstein distance [5] normalized with gradient penalization as loss function.

Fan [3] et al. proposed a point set generation model for 3D reconstruction from 2D image Riegler et al. [9] proposed OctNet, which use Octree representation of 3D objects to train deep 3D convolution network with high resolution. Tatarchenko et al. [12] built a volumetric 3D generative model based on octree structure. Huang et al. [6]'s study used pretrained templates to generate 3D object which include both the object structure and the surface geometry.

Choy et al. [2] studied using Recurrent Neural Network to reconstruct 3D object from one or more 2D images.

3. Method

3.1. Dataset

We used objects from 3DShapeNet[14] dataset to train the 3D GAN. The dataset contains `32x32x32` CAD models of volumetric objects from 40 classes including airplane, chair, table, vase, toilet, etc. The dataset is already processed with 3D-objects being centered in the space of `32x32x32`. So, we did not perform any pre-processing, and used the data as is.

Here we will first discuss about the architecture and training of 3D GAN. Then extend to the construction and training of 3D Generative Compression network.

3.2. 3D-GAN

3.2.1 Loss Function

Generative Adversarial Network (GAN) [4] consists of a generator and discriminator, where they act adversarially such that discriminator tries to classify real objects and objects synthesized by the generator, and the generator attempts to confuse the discriminator. The generator of 3D-GAN of our task, maps a 200-dimensional latent vector z, randomly sampled from a probabilistic latent space, to a 32 `x` 32 `x` 32 cube, representing an object G(z) in 3D voxel space; while the discriminator D outputs a confidence value D(x) of whether a 3D object input x is real or synthetic. [13].

The binary cross entropy is used as the classification loss. The overall adversarial loss function that incorporates for generation and discrimination is:

$$L_{3D-GAN} = log(D(x)) + log(1 - D(G(z))) \quad (1)$$

where, x is a real object in a `32x32x32` space, and z is a randomly sampled noise vector from a distribution p(z). Each dimension of z is an i.i.d. sampled from a multivariate normal distribution with zero mean and 0.33 standard deviation $N(0,0.33^2)$.

If 3D-VAE-GAN is incorporated in the 3D-GAN for reconstructions of 3D objects from 2D images, then 3D-VAE-GAN will have 3 components in loss function as follows:

$$L_{3D-VAE-GAN} = L_{3D-GAN} + \alpha_1 L_{KL} + \alpha_2 L_{recon} \quad (2)$$

where,

$$L_{3D-GAN} = log(D(x)) + log(1 - D(G(z))), \quad (3)$$
$$L_{KL} = D_{KL}(q(z|y)||p(z)), \quad (4)$$
$$L_{recon} = ||G(E(y)) - x||_2 \quad (5)$$
$$(6)$$

3.2.2 Network Structure

3D-GAN has convolutional neural network (CNN) for both generator and discriminator. It consists of five volumetric convolutional layers of kernel sizes `4x4x4` and strides 2, with batch normalization layers, and leaky ReLU layers added in between convolutional layers, while Sigmoid layer is there at the end of the network.

To match the input data's dimension, we adjusted the network so the network outputs a `32x32x32` volumetric object instead of `64x64x64`. The last convolutional layer of 3D-GAN[13] is removed, and a fully connected layer of 2048 unit is inserted between the input layer and the first convolutional layer. Note that for this layer, we use a fully connected layer since a convolutional layer in this place would be too small to capture any expressive features.The network for the generator is as shown in Figure 1.

The discriminator basically mirrors the generator, except that it has fully-connected layer at the end with sigmoid activation.

3.2.3 Training of 3D GAN

We experimented with the network architectures of generator and discriminator of 3D-GAN to achieve better results than that published by the authors [13] for generation of 3D objects. In particular, we tried the so called `GAN hacks` to further improve performance of 3D-GAN for object generation.

- **Modified loss function:** We modified the loss function of 3D-GAN as per equ. 1 to maximize `log(D)` instead of minimizing `log(1-D)`. The reason for doing this is to remove vanishing gradients in the latter case, which happens early on during training the network.
- **Batch Normalization:** We performed batch normalization after each convolutional layer in Generator and Discriminator.
- **Label Smoothing:** We performed label smoothing, which is basically to use soft and noisy labels instead of using hard labels: `1` for real and `0` for Fake. For each incoming sample, we replaced the label with a random number between 0.7 to 1.2, if it is a real sample, and a random number between 0.0 to 0.3, if it is a fake sample.

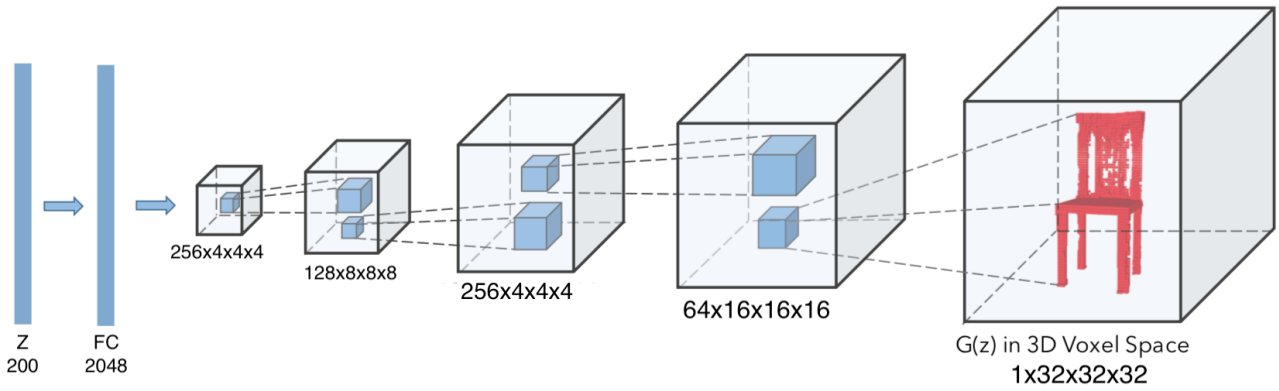


Figure 1: The generator of a modified 3D-GAN[13]

- **Adaptive learning rate:** We had adaptive learning rates for both Generator and Discriminator networks. Since, discriminator learns very fast in comparison to the generator, and achieves 100% accuracy very soon, so we had higher initial learning rate for generator (0.0025), while lower learning rate (0.0001) for the discriminator and varied this learning rate depending on loss dynamics of the GAN.

- **Optimizers for Discriminator and Generator:** We used ADAM optimizer for Generator while SGD for Discriminator. The reason for doing so is again because Discriminator learns very fast in comparison to Generator. So, in addition to having different learning rates for the generator and discriminator networks, we can have different optimizers, since SGD takes time to reach the optima of loss function so it will further reduce training speed of the discriminator, while ADAM will enhance training speed of the generator. For Adam optimizer we used $\beta = 0.5$.

- **Adaptive training strategy:** One of the most difficult problems in training GAN is that either the generator could not catch up with the discriminator or the other way around. It is common to use different learning rate for the generator and the discriminator, or train one of the network more often than the other. However, simply adjusting the learning rate is usually not enough. One of the techniques that we found helpful in training the network is to use the adaptive training strategy as employed by [13], where the discriminator only gets updated when its accuracy is below 80%, since most of the time the discriminator will learn faster and reach over 90% accuracy despite the generator having higher learning rate.

In addition, we found that, changing the weight initialization is also important. We initialize the weights from a normal distribution with zero mean and very small standard deviation. Using high values of standard deviation for initialization could make the network not converge at all. Weights are initialized from normal distribution with zero mean and 0.01 standard deviation.

Generated objects are sampled for inspection every few epochs since loss function and accuracy of the network are not very good indicators of the generated object's quality. For the training, we used batch size of 100.

3.3. Generative 3D Compression Model

3.3.1 Network Structure

We build a voxel compression network that has fully learnable encoding function. Instead of applying handcrafted linear transformation to the input voxel, the model learns the mapping function f from an input x to a latent space $f(x)$, as well as reconstruction function g that recreates voxel from the latent space.

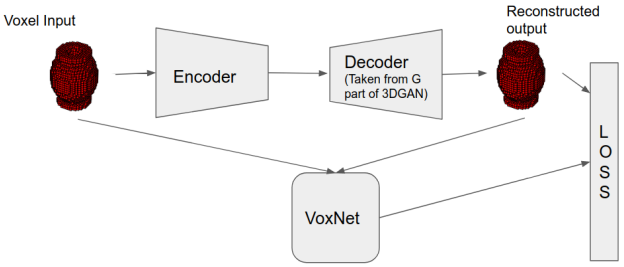


Figure 2: Model architecture for the Generative 3D Compression Network

The encoder network is a transpose of the Generator part of 3D GAN Network and the decoder is the trained generator from the 3D GAN. The VoxNet has the classical VoxNet architecture with two convolution layers and one fully connected layer. The VoxNet is trained until an accuracy of

94% is achieved on 5 object classes. The novel part of the 3D generator network comes from how these reusable and simple architectures are integrated to achieve a compression of 165 times and reconstructed to reflect finer details of the 3D object. This integration of networks is shown Fig 2 and explained further in the Loss Function section.

3.3.2 Loss Function for Generative 3D Compression Model

The model is trained to minimize the sum of two different loss functions. The first loss function is grid-level loss function which is used by every autoencoder. Another loss function is the perceptual loss function, where we take advantage of the VoxNet classifier's ability to understand some low level features of the input, such as edges and textures. For the classifier we built a VoxNet classifier which is trained on 5 objects among which one of them is the object that we intend to reconstruct. The perceptual loss function can be computed by taking the average of the L2 norm of the difference in the intermediate representations for the input object x and the reconstructed object $\hat{x}$, within one of the hidden layers of a classifier network. We trained VoxNet [8] as a classifier network and use the output of the second convolutional layer to measure the difference. Given the reconstructed object $\hat{x} = g(f(x))$, the loss function is as follows.

$$L(x, \hat{x}) = \lambda_1 ||x - \hat{x}||_2 + \lambda_2 ||conv_2(x) - conv_2(\hat{x})||_2$$

Where $conv_2$ is the output of the second convolutional layer of a trained VoxNet[8]

3.3.3 Training

We first train the generator model from section 3.2 with a discriminitor. The generator is later used as a pretrained decoder network. We also trained a VoxNet to use as a feature extractor for the objects. We then used Adam optimizer with learning rate 0.001 that decays by half every 30 epochs and $\beta = 0.5$. For training, we use batch size of 100.

We evaluate the models by passing the unseen object from the test set and compare with the original object.

4. Results

4.1. 3D-GAN

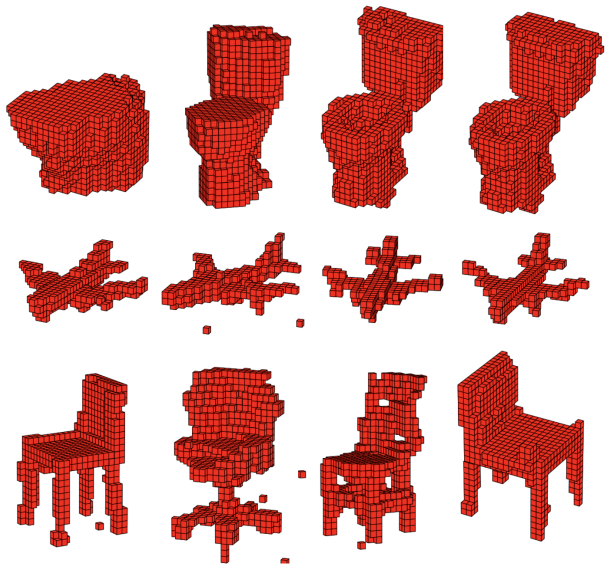
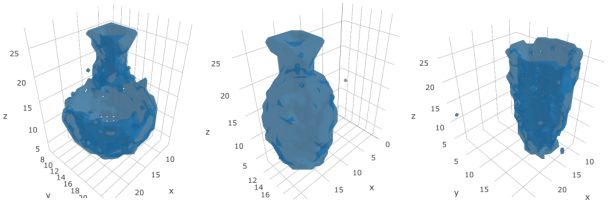


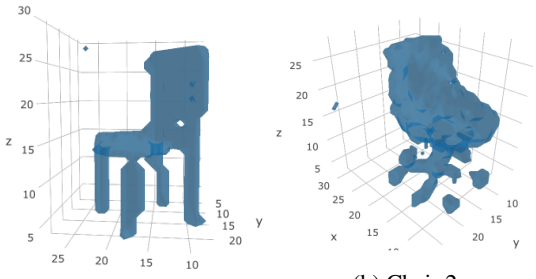
Figure 3: Some of the objects generated by our model. They are of class toilet (top), airplane (middle) and chair (bottom)

Figure 3 show the volumetric objects generated by our model. The models are trained separately for each object, with some classes easier to train. The GAN for these classes are able to produce sensible objects in smaller number of epochs. Fig 6 and 7 shows the loss function for GAN that generates vase objects and GAN that generates chair objects, respectively. The vase are easier to train most likely due to its symmetry and having less complicated features. Notice that the loss of both generator and discriminator of vase GAN converged very early, i.e. around 2000 batches (100 epochs), whereas chair GAN takes much longer. The quality of the generated objects are also different. Generated chairs have more imperfections than the vases despite taking much longer time to train, as shown in fig. 17 and fig. 4.



(a) Vase 1 (b) Vase 2 (c) Vase 3

Figure 4: Generated Vase objects with 3D GAN



(a) Chair 1 (b) Chair 2

Figure 5: Generated Chair objects with 3D GAN

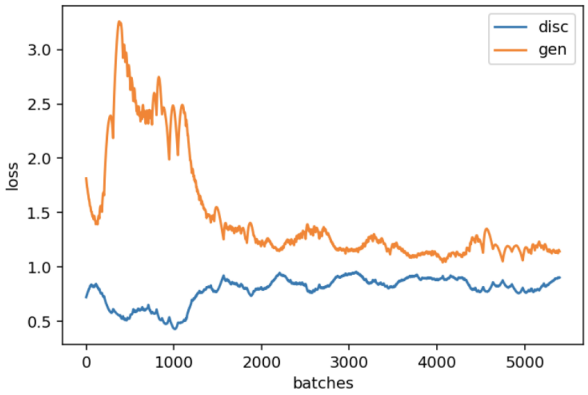


Figure 6: Loss function for GAN that generates vase objects

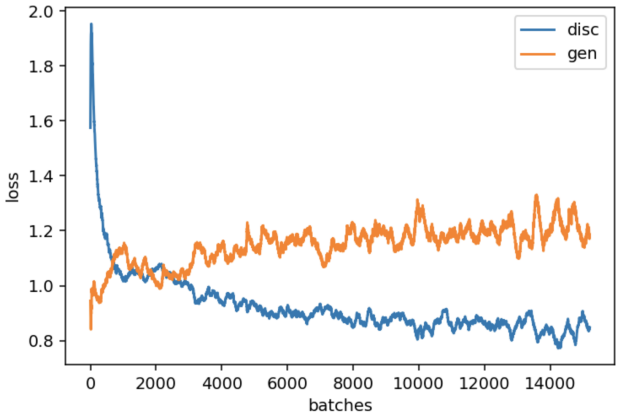


Figure 7: Loss function for GAN that generates chair objects

4.1.1 64*64*64 GAN

We have experimented with the model whose output is 64 x 64 x 64, we follow the hyperparameters provided by [13]. However, since the input data is only 32 x 32 x 32, we have to perform up sampling to generate the input for the model. The model is not ideal for our training data as it takes longer time to train and requires more memory, which limits the size of batch that we can use per training step. We are unable to produce sensible results after over 10 hours of training as the generator could not catch up with the discriminator. The generated objects are all cubes that fill up almost the entire space. Fig. 8 shows the loss function of the model training on a subset of chair objects and Fig. 9 shows a sample of generated objects from the latest epoch.

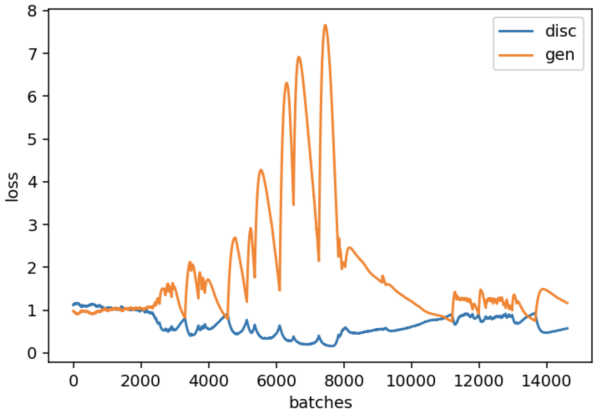


Figure 8: Loss function for 64 x 64 x 64 GAN training on chair objects. The spikes in loss function shows the instability of the model

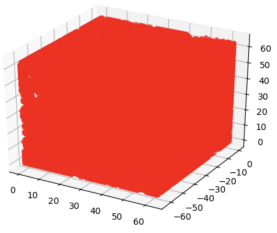


Figure 9: Sample generated object from unsuccessfully trained GAN

4.2. Generative 3D Compression

Our model is capable of reconstructing objects after encoding them into 200 dimension (about 1/165 times original size).

4.2.1 Chair objects reconstruction

Figure 10, 11, 12, 13 are some decent results of chair object compression. In figure 10, the object is quite simple

and the model can reconstruct almost 100% of the original. This is probably due to the chair of this style is common in the training set and the model can remember their basic structures. Fig.

11 shows how the model tries to reconstruct the texture of the object. There are also examples where the model incorrectly predict some parts of the chair, resulting in a valid, but different style of chair, these are demonstrated in Fig. 12 and 13. This is probably due to the model remembers the features of nearby objects in the training set and apply to an unseen object.

For some uncommon styles of objects, the model does not reconstruct them very well, the result is often an object with missing parts. We also notice that the generator network of the GAN that is used for training the Generative Compression model is not very good at generating this style of chair as well.

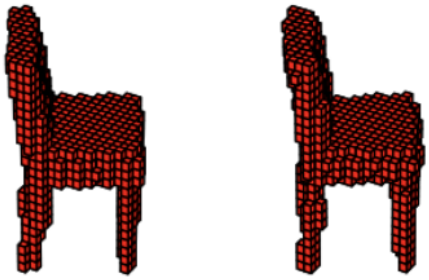


Figure 10: Image of an original chair object (left) vs reconstructed chair (right). The model is capable of compressing and reconstructing easier objects really well.

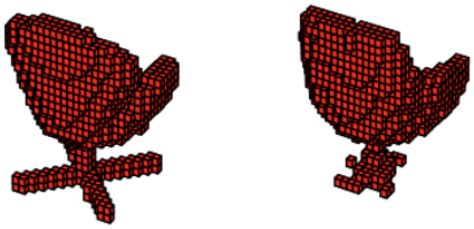


Figure 11: The model reconstructs the texture in the back of the chair

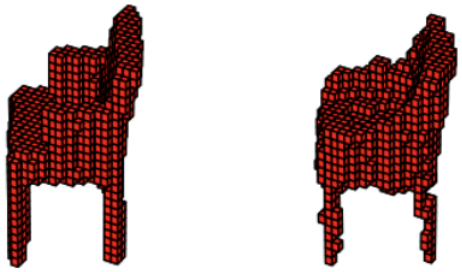


Figure 12: The reconstruct object contains a part that is different from the original model.

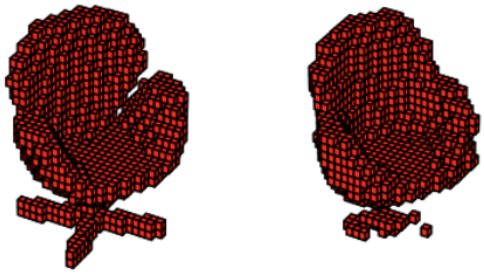


Figure 13: The reconstruct object contains a part that is different from the original model.

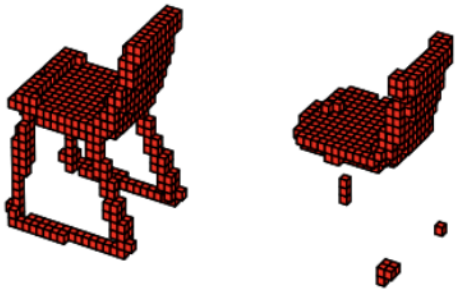


Figure 14: The model fails to reconstruct the legs of the chair

4.2.2 Object reconstruction vs traditional autoencoder

We also trained the model on an easier class of objects like Vase. For this class, the model is able to recover almost 100% of the original objects. For giving a comparision of quality of reconstructed models, we trained a traditional encoder decoder network and the Generative 3D compression

network on the Vase dataset and below are the results for Vase object reconstruction from both the networks.

(a) Vase compression reconstruction. Original object (left) vs reconstructed (right) using Traditional encoder decoder network

(b) Vase compression reconstruction. Original object (left) vs reconstructed (right) using Generative 3D compression network. We can see that the surface details are constructed more realistic here.

Figure 15: Comparison of object reconstruction using Autoencoder(top) vs Generative Compression network (bottom)

(a) Vase compression reconstruction. Original object (left) vs reconstructed (right) using Traditional autoencoder network. The reconstructed is very lossy.

(a) Vase compression reconstruction. Original object (left) vs reconstructed (right) using Generative 3D compression network.

Figure 17: Comparison of object reconstruction using Autoencoder(top) vs Generative Compression network(bottom)

4.2.3 Training loss

The model using pretrained GAN as a decoder network trains rather slowly in comparison to traditional autoencoders. Since the generator network is pretrained on the same training dataset, it starts off generating meaningful object right away. This leads to small amount of gradient being passed to the encoder network. Fig. 18 shows the loss function of training set and test set over time.

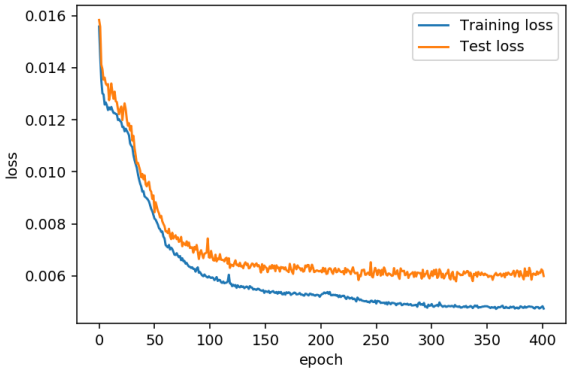


Figure 18: Training loss vs test loss during training of Generative 3D Compression model

5. Conclusion

The quality of the output for the Generative Compression depends largely on how GAN is trained. As we observe, our model does not reconstruct chair objects very well as some parts are missing from the generated object. On the other hand, some of the smaller details are well preserved. We can also try using a deeper classifier model as a feature extractor instead of the VoxNet, which has only 2 3D Convolutional

layers.

We also have seen the capability of 3D GAN model to produce novel 3D objects. Training a 3D GAN requires a very careful selection of hyperparameters. Evaluation of GAN outputs is also not a simple, there is no easy way to determine whether the model has converged besides manually inspecting the generated outputs, which has to be done by human.

6. Future Direction

We implemented 3D Generative Compression of models using a prelearnt decoder from a DC-GAN type architecture and this achieved a compression ratio of approximately 165 times. We believe that the compression ratio can be improved with the use of IWGAN architecture and this part we would like to experiment further.

References

[1] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Generative and discriminative voxel modeling with convolutional neural networks. *CoRR*, abs/1608.04236, 2016.

[2] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. *CoRR*, abs/1604.00449, 2016.

[3] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. *CoRR*, abs/1612.00603, 2016.

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[5] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[6] H. Huang, E. Kalogerakis, and B. Marlin. Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. *Comput. Graph. Forum*, 34(5):25–38, Aug. 2015.

[7] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, Dec. 2013.

[8] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, Sept 2015.

[9] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. *CoRR*, abs/1611.05009, 2016.

[10] S. Santurkar, D. M. Budden, and N. Shavit. Generative compression. *CoRR*, abs/1703.01467, 2017.

[11] E. J. Smith and D. Meger. Improved adversarial systems for 3d object generation and reconstruction. *CoRR*, abs/1707.09557, 2017.

[12] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *CoRR*, abs/1703.09438, 2017.

[13] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *CoRR*, abs/1610.07584, 2016.

[14] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. *CoRR*, abs/1406.5670, 2014.