3D GAN Object Generation and Reconstruction

Gendong Zhang
Stanford University
zgdsh29@stanford.edu

Zixuan Zhou
Stanford University
zixuan95@stanford.edu

Liuming Zhao
Stanford University
lxz299@stanford.edu

Abstract

In the report, we demonstrated the ability of Generative Adversarial Networks (GANs) to perform generation and reconstruction tasks. Original GANs suffer from gradient vanishing of generator and collapse mode. We researched different methods, using Wasserstein distance and gradient penalty to target the problems, and compared with the original 3D-GAN. With the same completion of the generated objects, GAN with Wasserstein distance and gradient penalty (WGAN-GP) increased the diversity of the generated samples dramatically. For the reconstruction task, we used variational auto-encoder (VAE) combined with WGAN-GP to successfully recover the 3D object from one single view of the object.

1. Introduction

Generative Adversarial Networks (GANs) is one of promising techniques used in Computer Vision and unsupervised machine learning [5], and it can generate superficially authentic images that is hard to distinguished by human observers. Combined with GANs, 3D object generation and reconstruction gains its popularity in Robotics, tasks for AR/VR and graphics fields. Specifically, reconstruction 3D object can be practical use when dealing with robot grasping and obstacle avoidance. Depth-sensing devices such as Kinect cameras capture RGB-D images with depth information, which can be used to reconstruct 3D model of that object. In addition, training GANs can be difficult and gives undesired behaviors due to training instability, and exploring different types of GANs can be useful to target different problems. In this project, we investigated the training results of GANs with varied GAN architectures and also reconstructed 3D object from one surface of the object.

2. Related Work

2.1. Image generation with an adversarial net

Generative Adversarial Nets (GAN) proposed a new framework for estimating generative models via an adversarial process. With the advance of deep convolution neural network, DC-GAN [11] obtains significant performance in image synthesis. Other vision problems such as style transferring and image editing [10] [17] were also approached using GAN. Training GANs can suffer from training instability, Gulrajani [6] proposed an improved training method using Wasserstein distance to produce a value function to stabilize the training. Our project embedded this method and also incorporated with gradient penalty to solve the undesired behavior.

2.2. 3D object Generation

In recent years, computer visions community pay great attention to 3D object understanding and generation. Huang [8] analyzed 3D shapes using pre-trained templates and at the same time generating structure and surface model of the object, but this model requires supervision during training. Wu [13] proposed a method that required no supervision during training and able to generate objects using DC-GAN from a probabilistic space. We used Wasserstein distance and gradient penalty to re-implement the 3D generation to compare with the vanilla version of the 3D generation.

2.3. 3D object reconstruction from single view

Recovering 3D object from multi-view [4] is the traditional way, and it requires images from different angles and also 3D object label to supervise the training, and y, extra efforts are required to acquire additional information from the camera. Yao[16] proposed a method that can reconstruction 3D object from a single depth view using adversarial learning with auto-encoder and conditional discriminator. In our project, we modified the auto encoder to the variational auto-encoder (VAE) [9], since VAE is preferred in learning complex distributions, this GAN architecture can leverage a single network for both the generator and decoder.

3. Data

For 3D generation task, we use ModelNet10 [15] to train our model, which contains approximately 57000 objects with dimensions of 32x32x32, and each object in the one class is represented in 12 evenly spaced orientations, which consists of 10 classes: bathtubs, beds, chairs, desks, dressers, monitors, nightstands, sofas, tables and toilets. One example of this dataset can be visualized in 3D voxels, shown in Figure 1. The goal of our generation model is to generate a novel object that can trick our discriminator, and also the generated object can be recognized as the specific class even it is not present in the dataset. In this project, 3D GAN generation model was trained on Chair dataset, and all 12 orientations are fed into model simultaneously.

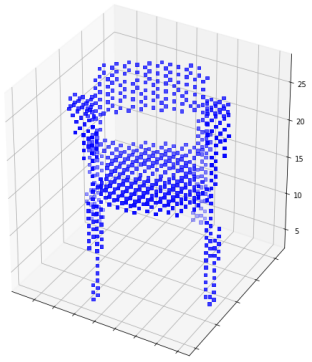

Figure 1: One example from the ModelNet10 Chair class

In 3D reconstruction part, we followed the instruction as described in [12], we used the data pre-processing code as a starter code [1] to generate 900 synthetic RGB-D images from ModelNet10 Chair class, and each sample consists of RGB image and depth image, containing color and distance information. 15 surfaces of each object is generated randomly as input for this model. two examples of the surfaces of the 3D object in Figure 1 can be seen in Figure 2,3. We successfully completed 3D shape with only a single view of RGB-D camera scan. We randomly chose 10% samples as validation set to do fine-tuning, and besides, 10% of the dataset is hidden from the model to evaluate the final system. We expect our reconstruction model can be useful to robotics problem in real world.

Since the it is hard to quantitatively tell whether the generated object looks like the aimed object or not, we monitored loss of our discriminator and show the generated object for each object class.

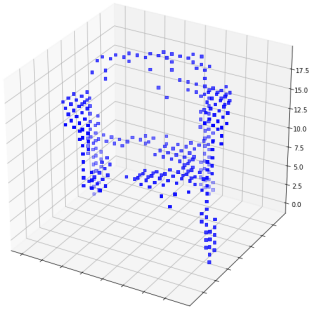

Figure 2: One surface example from the ModelNet10 Chair class

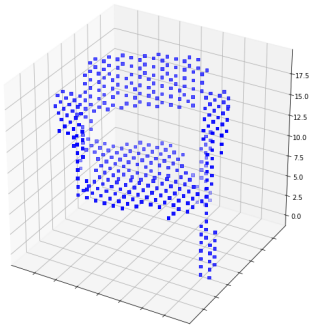

Figure 3: One surface example from the ModelNet10 Chair class

4. Methods

4.1. Baseline Vanilla 3D Generation Model

For the baseline, we utilized the basic 3D generation Model as described in [14], with some modifications on the network architecture. Similar to the 2D Generative Adversarial Network (GAN), our basic 3D GAN also contains one generator and one discriminator, the generator creates a 3D subject and tries to trick the discriminator, while the discriminator tries to identify the synthesized 3D object, and tell if it is true or not. In here, we followed the paper [14] instruction, and let the generator to map a 200-dimensional latent vector z, which was randomly sampled from a Gaussian normal distribution space, to a 32x32x32 3D cube. The generator consists of 5 transposed 3D-convolution layers (also known as deconvolution layers) with different numbers of filters. As can be seen from Figure 4, the leftmost contains 256 filters, and gradually decreasing by factor of 2, the number of filters of the rightmost convolution layer contains 32 filters. The filter size of these layers are all

4, with stride 2. The architecture of discriminator in Figure 5 looks like the counterpart of the generator, with only changing the transposed 3D-convolution layer to the 3D-convolution layer, and the output of the discriminator is the output of a sigmoid function, telling whether the input is real or fake. Note that for generator, we used ReLU as activation function, while for discriminator, leaky ReLU with negative slope 0.2.

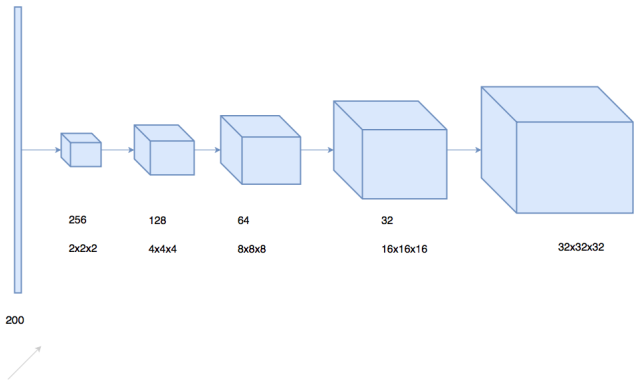

Figure 4: Generator Architecture

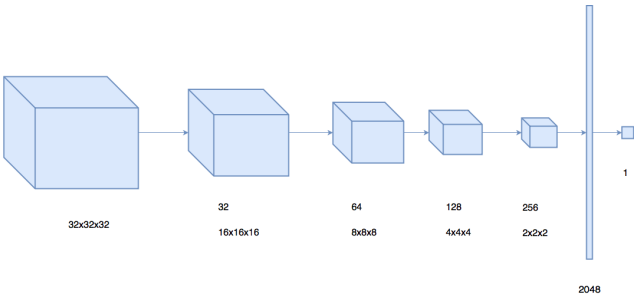

Figure 5: Discriminator Architecture

The loss function for this basic 3D generation model is simple use For Discriminator, we apply binary cross entropy loss, so the loss function for discriminator is:

$$L_D = -E_{x\sim P_r}[logD(x)] - E_{x\sim P_g}[log(1 - D(x))] \quad (1)$$

the loss function for Generator is:

$$L_G = E_{x\sim P_g}[log(1 - D(x))] \quad (2)$$

and total loss function for GAN is:

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

where, x is real object, and z is the 200-dimensional latent vector, following (0, 1) Gaussian distribution, P_r is the distribution of real samples, and P_g is the distribution of generated samples.

4.2. Wasserstein GAN with Gradient Penalty

Basic GAN model Problems. According to [2], original generator loss function can be converted into $2JS(P_r||P_g - 2log2)$, which is JensenShannon divergence between $P_r$and P_g. Along with the training process of discriminator, minimizing the generator loss will get close to JensenShannon divergence between P_r and P_g. However, when there do not exist overlap between P_r and P_g, or the overlap between them can be ignored, JensenShannon divergence of P_r and P_g will be 2log2, and in this case, gradient will become zero. It is very likely to happen, when supports of P_r and P_g is low dimensional manifold in high dimensional space. Since generator samples from a low dimensional random distribution (200-dimensional latent vector z in our case), and besides neural network have effect of mapping dimensional reduction on P_g, the support of P_g is a low dimensional manifold, which is difficult to have overlap with P_r.

For original GAN, if discriminator is over qualified, generator will suffer from gradient vanishing problem, while if discriminator is under qualified, gradient of generator will shuffle around.

In another common way, L_G is replaced with $E_{x\sim p_g}[-logD(x)]$, which we applied in our baseline model, but according to [2], it can be transformed into $KL(P_g||P_r) - 2JS(P_r||P_g)$. While training neural network, the process will minimize KL divergence and maximize JS divergence, which will cause unstable gradient problems.

More importantly, When $P_g(x)$ goes to zero while $P_r(x)$ approaches one, $P_g(x)log\frac{P_g(x)}{P_r(x)}$ goes to zero, meaning the contribution to the KL divergence $KL(P_g||P_r)$ from $P_g(x)$ and $P_r(x)$ is approaching zero. Another case is when $P_g(x)$ goes to one while $P_r(x)$ approaches zero, $P_g(x)log\frac{P_g(x)}{P_r(x)}$ goes to infinity, so the contribution to the KL divergence $KL(P_g||P_r)$ from $P_g(x)$ and $P_r(x)$ is approaching positive infinity. Thus, the generator would like to create repeated but safe samples, rather than various samples, which is called collapse mode phenomenon.

Wasserstein distance. Wasserstein distance is also called Earth-Mover distance, defined as:

$$W(P_r, P_g) = \inf_{\gamma\sim\prod(P_r,P_g)} E_{(x,y)\sim\gamma}[||x - y||] \quad (4)$$

Where, $\prod$ is the collection of all possible joint distribution of P_r and P_g. For each γ, we can sample a real distribution P_r and generated samples distribution P_g. From Equation 4, Wasserstein distance stands for lower bound of expected

value for possible joint distribution. Compared with KL divergence and JS divergence, even if there is little overlap between P_r and P_g, Wasserstein distance can still reveal their relative distance.

According to [3], Wasserstein distance can be transformed as:

$$W(P_r, P_g) = \frac{1}{K} \sup_{||f||_L \leq K} E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)] \tag{5}$$

Where, K is Lipschitz constant of function f. For all possible f, Equation 5 get the upper bound of $E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)]$, and divided by factor K.

When we use a set of parameters W to represent possible f, expressed as:

$$K \cdot W(P_r, P_g) \approx \max_{w:|f_w|_L \leq K} E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)] \tag{6}$$

So far, we can use neural network to fit a set of f. For $|f_w|_L \leq K$, weights w_i are constrained within [-0.01, 0.01], and thus gradient of input is under a certain K, satisfying the Lipschitz condition. In our implementation, after each update, weights will be clipped back into this range.

While the discriminator of original GAN apply sigmoid function deal with binary classification problem, the discriminator of WGAN f_w fits Wasserstein distance, which is regression problem. Thus, we need to remove the sigmoid of last layer, and loss functions for WGAN are Equation 1 and Equation 2. Equation 6 can reflect the training process, which means that the less it is, the smaller Wasserstein distance between P_r and P_g is.

There still exists two problems in weight clipping of WGAN. Since it constrains the range of each parameter in network, the best strategy is to make all parameters extreme, for which almost all parameters are either 0.01 or -0.01, and weaken gradient back to generator.Besides, the value of weight clipping is hard to tune. Since the discriminator has multiple layers, weight clipping will cause that the network is likely to suffer from gradient varnishing or exploding problems.

Gradient Penalty. Since the discriminator attempts to maximize the score distance between real distribution and generated sample distribution, which prefers larger gradient. After long training steps, the norm of gradient will be close to Lipschitz constant K, due to which we introduced a new term of loss function, and it will make the norm of gradient close to K, expressed as, $[||\Delta_x D(x)||_p - K]^2$. When we chose K = 1, the new term of discriminator loss function is defined as, $\lambda E_{x \sim X}[||\Delta_x D(x)||_p - 1]^2$, where, the third term of the loss function comes from whole sample space, which is impossible to carry out. According to [7], we only need to capture centralized region. In more specific, we randomly sampled a real example x_r and generated one x_g, and then do interpolation sampling on them.

$$\hat{x} = \epsilon x_r + (1 - \epsilon) x_g \tag{7}$$

where, $\epsilon \sim Uniform[0, 1]$, and we can obtain probability of $\hat{x}$ $P_{\hat{x}}$.

To conclude, the new loss function of discriminator is defined as:

$$L_D = -E_{x \sim P_r}[D(x)] + E_{x \sim P_g}[D(x)] \tag{8}$$

$$+ \lambda E_{x \sim P_{\hat{x}}}[||\Delta_x D(x)||_p - 1]^2 \tag{9}$$

4.3. Variational Auto-Encoder (VAE)

Rather than randomly sampling from a latent space, mapping from 2D image to latent representation will be much helpful to reconstruct 3D object. We will add image encoder E to transform 2D image x to the latent vector z, which is proposed by [14]. VAE can be used to learn more complex distributions. The encoder E encodes the sample based on its target distribution, producing a vector of mean and variances, and the vector parametrizes by a set of Gaussian spaces, from which we can sample to produce the latent space. In here, the generator of the ordinary GAN serves as the decoder in the VAE, so we can combine the VAE with our GAN model. The decoder/generator attempts to produce the original sample. We used four 3D-convolution layer to extract the mean and variance information of the image, in our project, the surface information (1-D depth view) of the 3D object. The numbers of filters we used are 32,64,128,256 to match the generator. For each 3D-convolution layer, we used filter size 4, stride 2, padding 1. Note that the tanh function was used after the fully-connected layer to constrain the variance between (0,1). The detailed VAE architecture can be seen in Figure 6.

In our implementation, the encoder must be trained from the encoded image as we are only training it using the reconstruction loss. The generator is trained using reconstruction loss and the signal from the discriminator that randomly generator objects appear real. We could also add an additional loss term from the discriminators signal on the models created from encoded images, however it did not seems necessary at the time.

If we were to only use the models produced from encoded images the generator would just memorize the objects . This would mean it would be regular supervised learning, and the discriminator loss would add nothing to the generator's

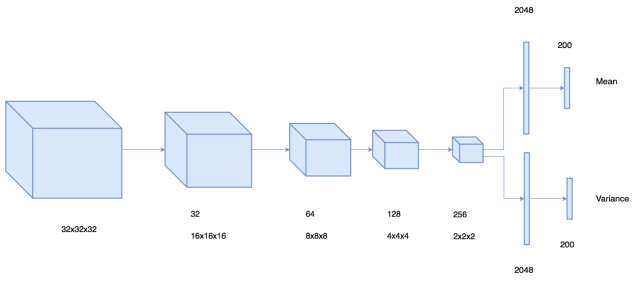

Figure 6: VAE Architecture

learning. Training the generator on randomly sampled latent vectors is a key part of adversarial learning.

Total loss function includes three parts, cross entropy loss for 3D GAN, KL-divergence loss, and reconstruction loss, shown in Equation 10. For KL-divergence loss, it will make variational distribution $q(z|y)$ get close to prior distribution $p(z)$, which Generator can obtain input from same distribution with $p(z)$. Besides, reconstruction loss will push the generated object close to real object in 3D voxel space.

$$L = L_{3D-GAN} + \alpha_1 L_{KL} + \alpha_2 L_{recon}, \qquad (10)$$

where,

$$\begin{aligned} L_{3D-GAN} &= logD(x) + log(1 - D(G(z))), \\ L_{KL} &= D_{KL}(q(z|y)||p(z)), \\ L_{recon} &= ||G(E(y)) - x||_2 \end{aligned}$$

where, y is 2D image, and x is corresponding 3D object.

5. Experiments

5.1. Training Strategies

In this subsection, several effective training strategies of our project will be discussed.

Training Environment. We built our instance on Google Cloud Platform (GCP), with n1-highmem-8 (8 vCPUs, 52 GB memory) and 2 x NVIDIA Tesla K80.

Initialization. The weights of convolution layers and convolution transpose layers (deconvolution layers) are initialized with Xavier initialization.

Batch size. The dataset is divided into mini-batches of size 256 for original GAN, 128 for WGAN-GP, and 32 for VAE-GAN. In order to maximize computation efficiency, a batch size of a power of 2 is preferred. Moreover, since the matrix of our image is of large size and the computation ability available to us is limited, smaller batch size selection for VAE-GAN is further justified.

Learning rate. We fine-tuned the learning rates for each model. During training of baseline model (original GAN), the learning rate of discriminator is 0.00005, while the learning rate of generator is 0.0025, much faster than former. It is easier for discriminator to converge, and if this happens in the early stage, it will be difficult for us to train the generator. Thus, we ultimately chose this pair of learning rate for baseline model, and further experiment shows it works. For WGAN-GP model and VAE-GAN, the learning rate is more easy to search, thanks to the power of Wasserstein distance and gradient penalty. In these two models, 0.0001 works well for both discriminator and generator.

Update Method. Adam Adaptive Moment Estimation is an optimization method that computes adaptive learning rates for each parameter. In practice, when compared to other adaptive learning algorithms, Adam shows the advantages of faster convergence and more efficient learning. It can also correct common problems in other optimization techniques like diminishing of learning rate, slow convergence speed and large fluctuations in the loss function caused by update of high variance parameters. We also tried other optimizers, and Adam outperform other updating methods.

5.2. Results

Baseline Vanilla 3D Generation. For the baseline vanilla 3D GAN, we plotted the loss curve for both generator and discriminator. As can be seen in Figure 7, after approximately 400 epochs, both of the generator and discriminator converges, since we fine-tuned the learning rates of them, the degrees of training for them matched. We generated 256 3D objects in total, Figure 8 shows some of the generated objects, from the plot, we can see that for generation of chairs, which is a comparatively simple task, the baseline 3D GAN performs well. However, we random selected 9 objects to show the results, and it can be seen that there are only 5 distinct 3d objects, meaning that the baseline 3D GAN suffers from collapse mode and lack of diversity.

WGAN-GP. Figure 9 shows the loss curve of the WGAN-GP model, rather than doing binary classification problem, the discriminator of this model is solving the regression problem. As the training goes, the generation loss converges after 1000 epochs. Compared with the baseline GAN model, this model needs more epochs to converge. During experiments, we plotted the intermediate results, and we found that the generated 3D objects were not necessarily clearer than the baseline model, but after convergence, this model can produce equally or even clearer objects. From the plotted 3D objects result shown in Figure 10, however, this model is able to generate more distinct object, in this plot, none of the nine objects are repeated, so WGAN-GP can greatly increase the diversity of generation and dramatically decrease the collapse mode.

VAE-GAN. For the 3D reconstruction task, the VAE-

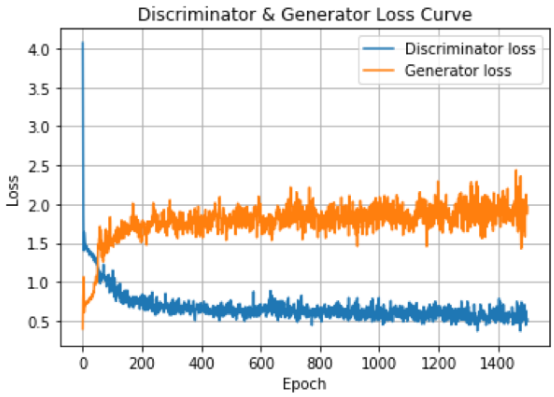

Figure 7: Generator Loss and Discriminator Loss VS Epoch (3D-GAN)

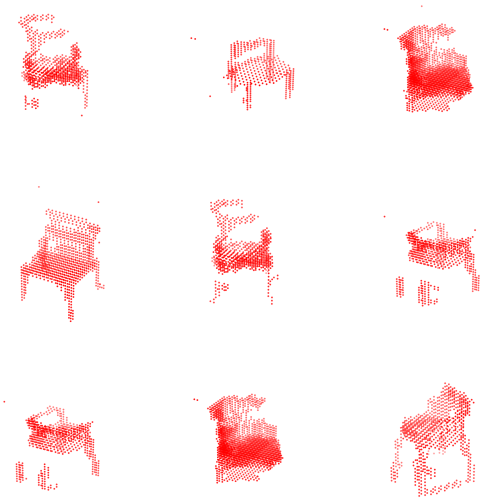

Figure 8: Results of Generated 3D objects (Baseline) at 1500 epoch

GAN achieved our expectation, but due to the Google Cloud connection lost, we lost the beginning 650 epochs, we can only plot the epochs after 650 shown in Figure 11,12. We plotted the loss curves of encoder, generator and discriminator in the figure, the tendency of these three curves have the same behavior, and converge after 900 epochs. Note that the generator loss is the reconstruction loss.

We used Meshlab to plot the reconstructed 3D objects to have a view in 360 degree for a better inspection of the completion of the object. Two example of the chair class were plotted used Meshlab and shown in Figure 13. The

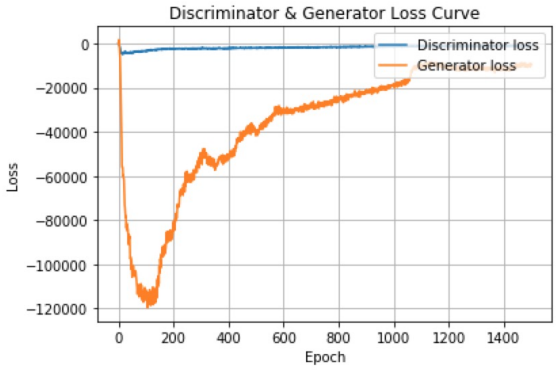

Figure 9: Generator Loss and Discriminator Loss VS Epoch (WGAN-GP)

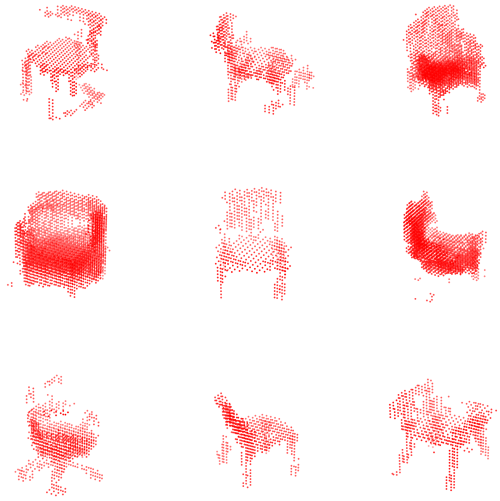

Figure 10: Results of Generated 3D objects (WGAN-GP) at 1500 epoch

first column represents the reconstructed object, and the second column illustrates the original object. From Figure 13, our VAE-GAN model can finish the reconstruction job from one single view, and GAN is a powerful tool to reconstruct object.

6. Conclusion

In this project, we successfully fulfill the task of 3D generation and reconstruction. We compared WGAN-GP with the basic 3D-GAN model, and we found that for the simple task of generating chairs, both of them perform well with respect to the completion of the 3D object generation. However, when it comes to the diversity of the generated objects, WGAN-GP outperforms the basic 3D-GAN signif-

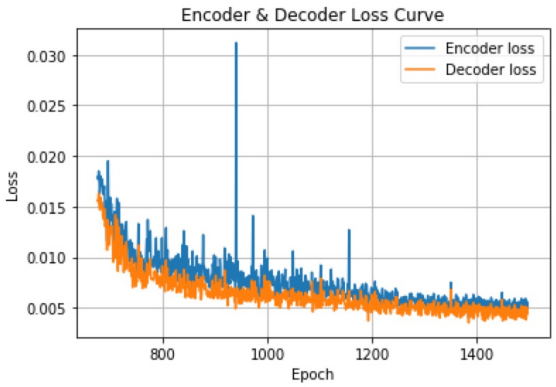

Figure 11: Encoder and Decoder (Reconstruction) Loss

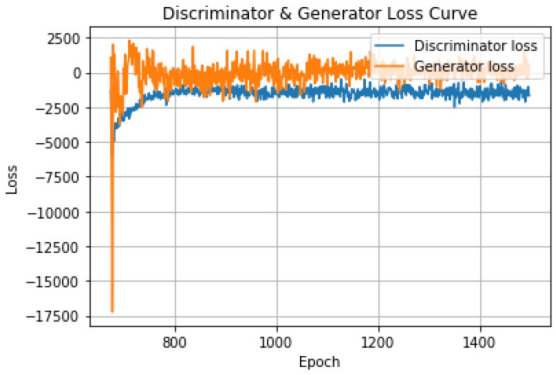

Figure 12: Discriminator and Generator Loss

icantly, due to the benefit of using Wasserstein distance. Besides, for the reconstruction 3D object from one single view, the experiment also demonstrated promising results. Admittedly, compared with the state of art, our models can exhibit reconstructed objects in relatively low resolution due to the limited computational power and the low resolution of the ModelNet dataset itself. For the future work, we can test our model on more classes, and also use real RGB-D image to see the performance of the VAE-GAN.

7. Contribution

Gendong Zhang did data pre-processing, literature review, built model (pytorch) of 3D object generation, reconstruction, plotted graph, trained the model, did documentation. **(Special Contribution)**

Zixuan Zhou did data pre-processing, literature review, trained model, used Meshlab to visualize results, did documentation, management of Google Cloud. **(Special Contribution)**

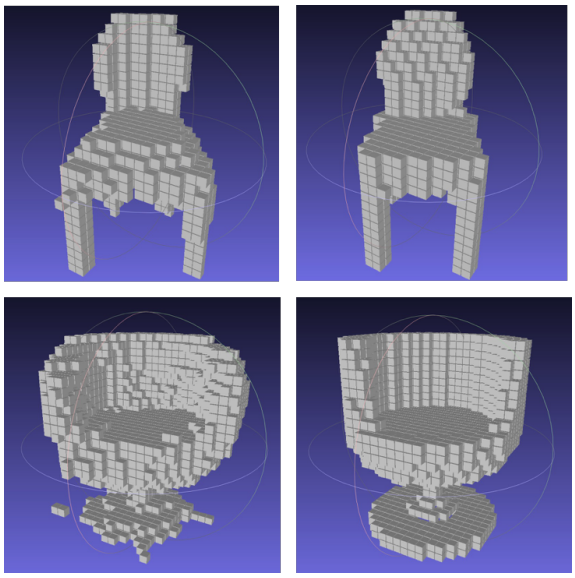

Figure 13: Reconstructed Object VS Original Object

Liuming Zhao did literature review, wrote reconstruction (pytorch) and WGAN-GP (pytorch).

References

[1] https://github.com/EdwardSmith1884. 2

[2] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. 1050, 01 2017. 3

[3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017. 4

[4] M. Gadelha, S. Maji, and R. Wang. 3d shape induction from 2d views of multiple objects. *CoRR*, abs/1612.05872, 2016. 1

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. 1

[6] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017. 1

[7] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017. 4

[8] H. Huang, E. Kalogerakis, and B. Marlin. Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. *Computer Graphics Forum*, 34(5), 2015. 1

[9] A. B. L. Larsen, S. K. Sønderby, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300, 2015. 1

[10] Y. Li, N. Wang, J. Liu, and X. Hou. Demystifying neural style transfer. *CoRR*, abs/1701.01036, 2017. 1

[11] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. 1

[12] E. J. Smith and D. Meger. Improved adversarial systems for 3d object generation and reconstruction. *CoRR*, abs/1707.09557, 2017. 2

[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. 1

[14] 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. 2, 4

[15] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes proceedings of 28th ieee conference on computer vision and pattern recognition (cvpr2015). 2015. Oral Presentation 3D Deep Learning Project Webpage. 2

[16] B. Yang, H. Wen, S. Wang, R. Clark, A. Markham, and N. Trigoni. 3d object reconstruction from a single depth view with adversarial learning. *CoRR*, abs/1708.07969, 2017. 1

[17] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. 1