3D Generation Task Report - CGTrader

Pooya Kabiri

August 9, 2024

1 Approach & Reasoning

Considering the binary nature of the voxels and their relatively low resolution (32) in this task, I conducted a quick review of existing literature on the topic. From this, I concluded that a practical approach, given the limited resources and time, would be to model the probability distribution of the data using a Variational Autoencoder (VAE). I chose this method because it leverages 3D convolutions and transpose convolutions, providing a straightforward yet effective solution. This approach is particularly suited to binary voxel data, as it avoids the need for complex pre- or post-processing steps, allowing the model to work directly with the voxel data in its native form.

Diffusion models are currently a hot topic in Generative AI, but I believed that using one would present some challenges. Voxels are binary, and therefore, not diffusion-friendly. One approach could involve computing a signed distance function (SDF) from the voxels and using that for diffusion. However, this process is time-consuming—something I have done before. It is a very slow process. This method was used in [1].

Another possibility would be to sample points from the surface of the voxel grid and feed them into a diffusion pipeline. However, the unstructured nature of these points poses difficulties for the network, as there are no knowledge priors to guide the learning process, as demonstrated in [2].

In conclusion, I believed that using a diffusion pipeline would be overkill in this scenario.

2 Resources & Training

My network utilizes a simple autoencoder architecture, with four layers in both the encoder and the decoder sections of the network. Given that the voxels are binary, I trained the model using a modified version of the binary cross-entropy loss, as introduced by [3]. This modified loss function places greater emphasis on the non-empty regions of the voxel grid, encouraging the network to focus more on these areas. Additionally, I incorporated a standard KL Divergence loss to help the model learn the underlying normal distribution of the training data.

I conducted training on both specific classes and across the entire dataset. Fortunately, I had access to an RTX 4090 GPU for a limited time (about two hours) from a friend, which I used to train the final versions of the network. Training on a single class took approximately 10 minutes, while training on the entire dataset required around 40 minutes. On a Google Colaboratory T4 GPU, training a single class took 35 minutes. I did not attempt to train on all data on Colab.

I aimed to create a flexible, framework-like codebase that allows for plug-and-play functionality. I structured the project to make it easy to add new methods with minimal effort.

All losses, hyperparameters, gradients, and models were logged using Weights & Biases. You can find all the plots and samples here: Link to WandB

All parts of the code, notebooks, visualizations, and model outputs are also available in this GitHub Repository

In addition to the codebase, I provided two Jupyter notebooks for simple training and evaluation. These are based on the codebase, with some modifications to run in a notebook environment. For the inference notebook, I download the model checkpoints from Google Drive using gdown, from the following folder: Link to Drive.

3 Results

Table 1 shows the quantitative results of the training. I used the mentioned modified BCE loss as the reconstruction loss. These are the performance of the model on a test set (not seen by the model).

Data	Reconstruction	KL Div	Total
ShapeNet-Chairs	0.0003	0.0013	0.0016
ShapeNet-Tables	0.0004	0.0008	0.0012
ShapeNet-All	0.0005	0.0001	0.0006

Table 1: Comparison of Losses for Different Data Subsets

For qualitative evaluation, I sampled randomly from the probability distribution and fed that to the decoder part of the network. Figure 4 demonstrates the direct output of the network, which is represented by a voxel grid. I also extracted the mesh by running marching cubes. Figure 5 is a screenshot from Meshlab.

4 Discussion

As can be seen from the qualitative results, the model does an okay job, given the time and resources. However, when training on all the data, the model does not perform well, as the generated shapes lack diversity. This issue can be addressed by adding some augmentation to the data and introducing some kind of noise, making the model more robust to perturbations and better able to generalize.

One interesting line of work that can be built on top of this is using a VAE architecture alongside a diffusion pipeline. We can use a VAE to encode the voxel grids into a latent space, and then apply a diffusion pipeline to the encoded latent vectors. This approach was recently employed by Nvidia in XCUBE[4]. This method is very promising but resource-intensive.

Another interesting experiment would be to use something other than a VAE to transform the voxels into latent code, such as the approach demonstrated in Neural Wavelet-domain Diffusion[5], which uses a wavelet transform to encode the data and then applies diffusion in that domain.

Finally, a recent method called CLAY[6] uses a VAE architecture alongside Attention to encode and decode the latent code. This work has not been published yet—it's just an ArXiv pre-print for now. The results reported in the paper suggest that this method outperforms the current state-of-the-art.

Should you have any questins regarding any aspect of this work, please don't hesitate to contact me.

References

[1] Y.-C. Cheng, H.-Y. Lee, S. Tulyakov, A. Schwing, and L. Gui, "Sdfusion: Multimodal 3d shape completion, reconstruction, and generation," 2023.

[2] L. Zhou, Y. Du, and J. Wu, "3d shape generation and completion through point-voxel diffusion," 2021.

[3] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," 2016.

[4] X. Ren, J. Huang, X. Zeng, K. Museth, S. Fidler, and F. Williams, "Xcube: Large-scale 3d generative modeling using sparse voxel hierarchies," 2024.

[5] J. Hu, K.-H. Hui, Z. Liu, R. Li, and C.-W. Fu, "Neural wavelet-domain diffusion for 3d shape generation, inversion, and manipulation," 2023.

[6] L. Zhang, Z. Wang, Q. Zhang, Q. Qiu, A. Pang, H. Jiang, W. Yang, L. Xu, and J. Yu, "Clay: A controllable large-scale generative model for creating high-quality 3d assets," 2024.

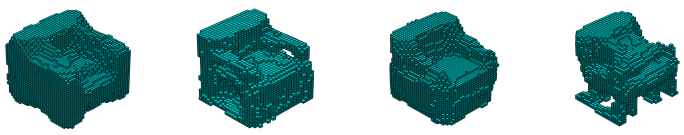

Figure 1: Chairs

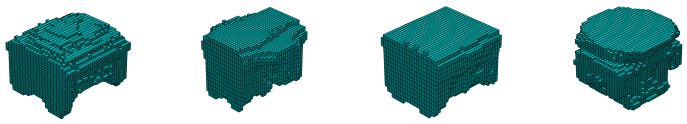

Figure 2: Tables

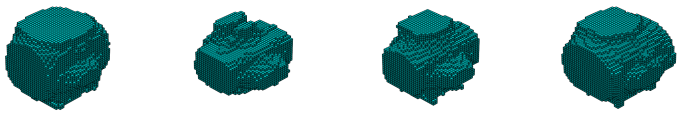

Figure 3: All data

Figure 4: Qualitative sampling

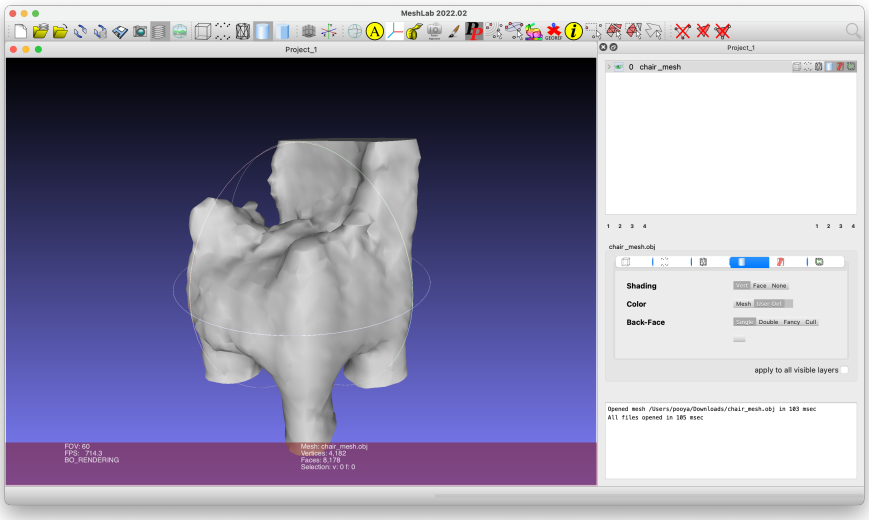

Figure 5: Output mesh of a Chair