Let's Make Pottery: GANs for Voxel Completion in Ancient Pottery Dataset

Zhuangzhe Wu **Ziyi Liu** **Shaozhe Shan**

Abstract

Our main goal is to generate complete pottery from fragments. We've implemented and refined a GAN model, improved training and evaluation methods for better performance, and finally generated satisfactory results using randomly selected fragments.

1 Introduction

Our topic for the final project is *Let's Make Pottery: GANs for Voxel Completion in Ancient Pottery Dataset.* This problem is significant as it holds great potential in archaeology, cultural heritage restoration, and 3D model generation. By accurately reconstructing pottery, we can better understand historical cultures and craftsmanship. The current progress involves using a GAN network for the task. The raw data in MagicaVoxel format is processed, and the model is trained with specific strategies. The results show that the model performs well, with some metrics even outperforming previous work. However, there are still limitations when dealing with tiny fragments or obscure results.

2 Related Works

Generative Adversarial Networks (GANs), introduced by Goodfellow et al. in 2014 [1], consist of a generator (G) and a discriminator (D) trained through a minimax game. The generator learns to mimic the data distribution, while the discriminator distinguishes real from generated samples. This foundational work introduced alternating training and opened avenues for architectural improvements, such as integrating variational auto-encoders for learning conditional variance. In 2017, Arjovsky et al. proposed the Wasserstein GAN (WGAN) [2], replacing the Jensen-Shannon divergence with the smoother Wasserstein distance to stabilize training. In 2018, Hermoza et al. introduced ORGAN [3], a GAN variant for 3D object reconstruction. Using a 3D CNN with skip-connections as the generator under a Conditional GAN (CGAN) framework, it optimized a combination of mean absolute error (MAE) and Improved Wasserstein GAN (IWGAN) loss. Their work provided valuable insights into 3D GAN architecture and training, though dataset differences require custom implementations. And we referred to a lot of their works.

3 Data Processing

3.1 Input Data

The raw input data follows the MagicaVoxel format and is parsed into 3D NumPy array with pyvox.parser. The converted voxel 3D array has different flags representing different fragments. The size of the training dataset is approximately 10000 and the testing dataset is about 5000.

3.2 Data Processing

First, we ensured the input array was $[64, 64, 64]$ and then resampled it to match the required resolution. In our case, we experimented mainly with $[64, 64, 64]$ and $[32, 32, 32]$.

Then we separately implemented the functions to select a random fragment or select a specific fragment to apply for training and testing scenarios.

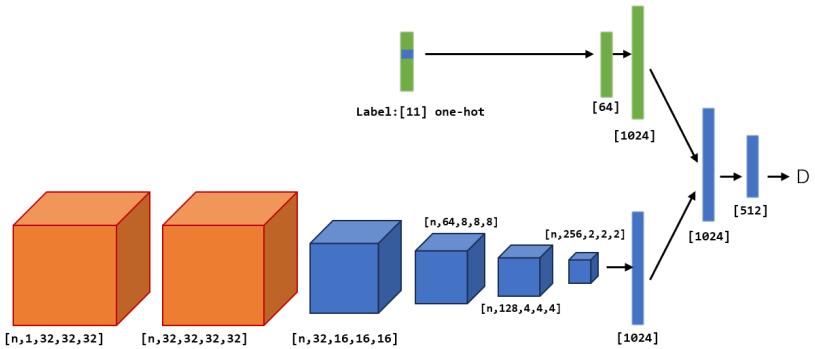


Figure 1: Discriminator32

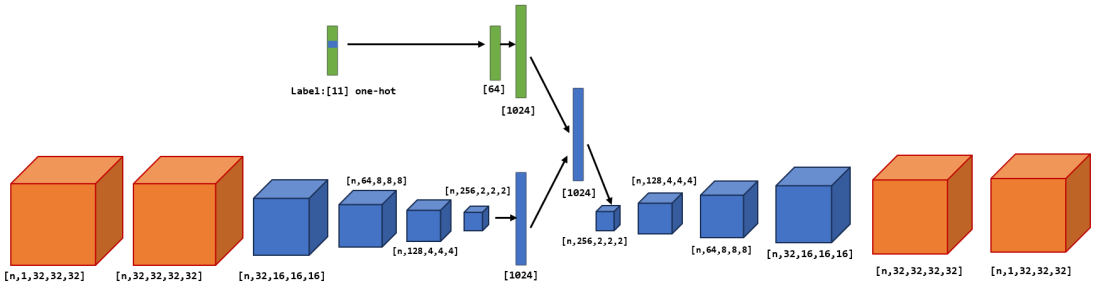


Figure 2: Generator32

During the dataloading process, we first converted the input file into a voxel array and selected a fragment from it. Then we also extracted the label representing the pottery type from the file path. Finally, we returned the selected fragment, the complete original one, and the type label for training.

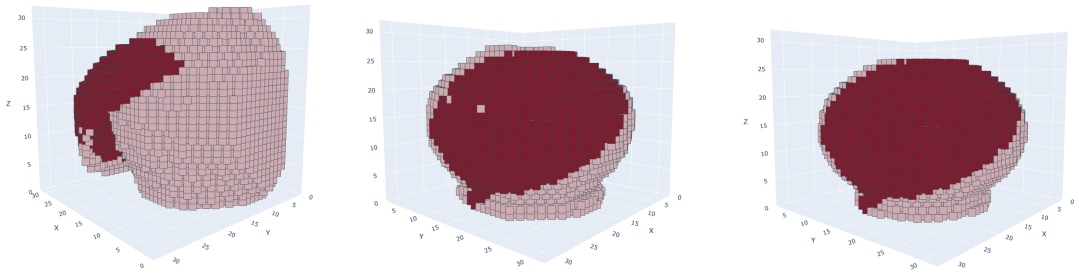
4 Methods

4.1 GAN Model

We utilize a GAN network. Figure 1 and Figure 2 show the architecture of the GAN32 network. For GAN64, we add an encoder to transfer the voxels to $[32, 32, 32]$, run the GAN32, and then output the final results through a decoder which turns the $[32, 32, 32]$ to $[64, 64, 64]$ again. Meanwhile, we utilize a 3D Squeeze-and-Excitation block, which is used in previous work[3].

4.2 Training details

We first train the Generator only and make the loss decrease significantly. We test the model and notice that though the loss is already low, the generated voxels' quality is not perfect and the appearance is unsatisfying (Subfigure initial train 3a). Then we train the Discriminator and Generator together. After a long time, the loss decreases more and its appearance gets more reasonable (Subfigure train final 3b).



(a) initial train (b) train final (c) ground truth

Figure 3: 2 Training stages

5 Experiments

5.1 Evaluation Metrics

We evaluate the quality of generated voxels using several classical functions, including Dice similarity coefficient (DSC), Mean squared error (MSE), and Jaccard distance(JD). When evaluating the true voxels A and generated voxels B, DSC is formulated:

$$DSC = \frac{2|A \cap B|}{|A| + |B|}$$

Actually, A and B are tensors that contain only 0 and 1 (indicates whether the position is occupied). So we can calculate $A \cap B$ as $A * B$, and $|A|$ and $|B|$ is the sum of tensor A and B. The values of the formula are between 0 and 1, where 1 represents the maximum similarity and 0 is the minimum.

Jaccard distance is defined as:

$$JD(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

The values of the formula are between 0 and 1 too, but unlike DSC, 1 represents the minimum similarity, and 0 the maximum.
MSE is defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (d_i - f_i)^2$$

5.2 Results

In Table 1 and Table 2 we present the obtained results across the different size groups of fragments and several pottery types. Table 1 is based on GAN32 and Table 2 is based on GAN64. We observe that the MSE and DSC values are close to the data recorded in previous work [4], and some of them are even better. This means that our model gains great ability to complete ceramics from smaller to larger initial fragments.

Interestingly, in both table, we can observe that specific classes of pottery are easier to reconstruct across all sizes, such as Class 10 and Class 11. We can hypothesize that the improvement, over the averaged performance of the method over these classes, is due to the fact that we have more examples of these classes in our dataset. Furthermore, these classes correspond to a homogeneous type of open shapes (e.g., plates). More test experiment details are recorded in Supplementary Material.

5.3 Visualization

We show some of our generated results in Table 3. More results are accessible and follow the instructions in Supplementary Material or our GitHub page.

5.4 Limitation

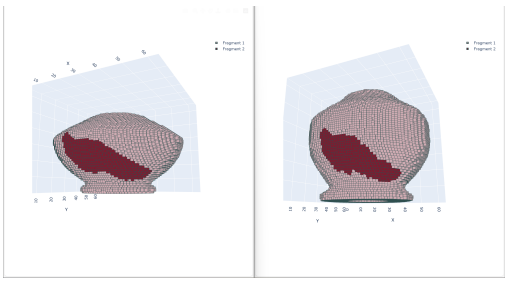
Although the model performs well on most of the fragments, it can't work well when the fragment's size is tiny or the completed results might be obscure. Some of them are listed in figure 4.

Table 1: GAN32: Performance metrics for different classes and fragment sizes.

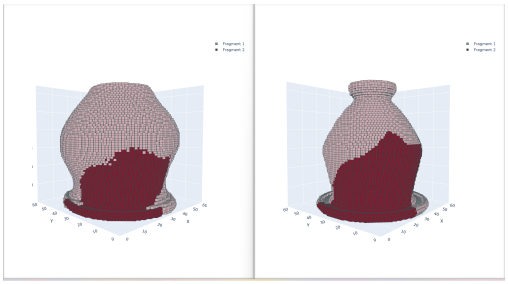
Classes	15-20			20-30			30-100		
	MSE(↓)	DSC(↑)	Jacc.(↓)	MSE(↓)	DSC(↑)	Jacc.(↓)	MSE(↓)	DSC(↑)	Jacc.(↓)
Class 1	0.067	0.700	0.458	0.056	0.717	0.428	0.042	0.809	0.322
Class 2	0.098	0.685	0.490	0.088	0.721	0.423	0.063	0.786	0.350
Class 3	0.094	0.598	0.571	0.083	0.628	0.546	0.063	0.692	0.471
Class 4	0.112	0.545	0.689	0.132	0.602	0.612	0.085	0.604	0.548
Class 5	0.135	0.643	0.512	0.117	0.679	0.481	0.092	0.745	0.392
Class 6	0.076	0.685	0.477	0.071	0.718	0.456	0.051	0.767	0.370
Class 7	0.112	0.661	0.496	0.089	0.674	0.479	0.069	0.733	0.415
Class 8	0.088	0.697	0.453	0.075	0.734	0.415	0.053	0.792	0.338
Class 9	0.069	0.743	0.423	0.057	0.763	0.387	0.036	0.827	0.279
Class 10	**0.032**	**0.774**	**0.359**	**0.025**	**0.804**	**0.330**	**0.019**	**0.848**	**0.267**
Class 11	0.028	0.726	0.438	0.025	0.748	0.396	0.020	0.815	0.332
Average	0.08	0.689	0.488	0.074	0.708	0.450	0.054	0.765	0.371

Table 2: GAN64: Performance metrics for different classes and fragment sizes.

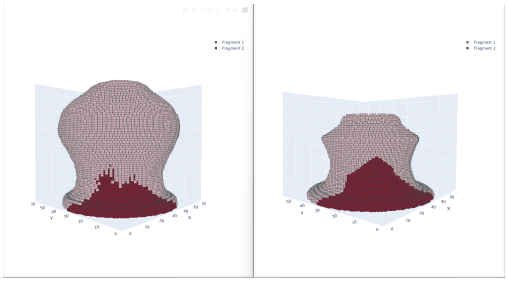
Classes	15-20			20-30			30-100		
	MSE(↓)	DSC(↑)	Jacc.(↓)	MSE(↓)	DSC(↑)	Jacc.(↓)	MSE(↓)	DSC(↑)	Jacc.(↓)
Class 1	0.278	0.709	0.453	0.271	0.727	0.412	0.263	0.801	0.343
Class 2	0.306	0.660	0.496	0.291	0.708	0.448	0.275	0.768	0.372
Class 3	0.302	0.589	0.583	0.291	0.626	0.548	0.279	0.692	0.472
Class 4	0.313	0.631	0.582	0.310	0.570	0.591	0.276	0.617	0.568
Class 5	0.325	0.641	0.522	0.304	0.673	0.489	0.288	0.737	0.420
Class 6	0.289	0.670	0.485	0.278	0.700	0.445	0.269	0.763	0.397
Class 7	0.303	0.662	0.542	0.290	0.678	0.467	0.269	0.735	0.399
Class 8	0.288	0.705	0.460	0.277	0.740	0.405	0.264	0.793	0.341
Class 9	0.277	0.719	0.442	0.274	0.740	0.383	0.254	0.833	0.291
Class 10	**0.262**	**0.752**	**0.396**	**0.258**	**0.790**	**0.355**	**0.252**	**0.843**	**0.265**
Class 11	0.263	0.707	0.447	0.260	0.737	0.407	0.254	0.793	0.326
Average	0.291	0.678	0.492	0.282	0.699	0.450	0.268	0.761	0.381



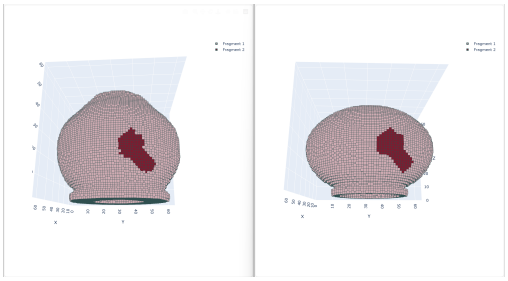
(a) class 2



(b) class 3



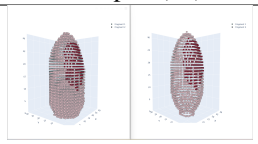
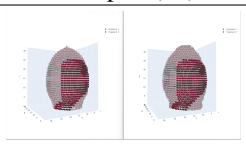
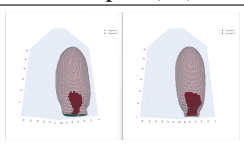
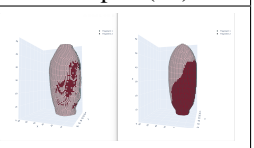
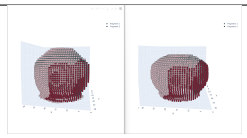
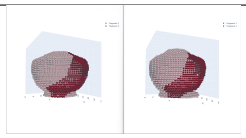
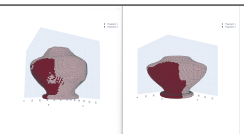
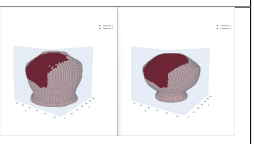
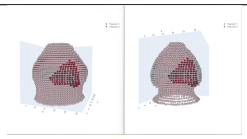
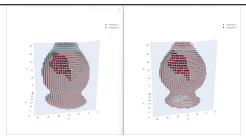
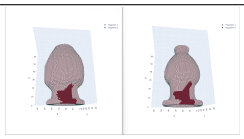
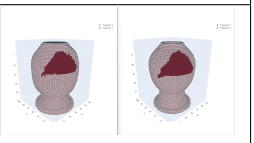
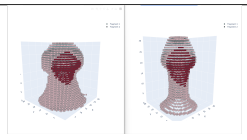
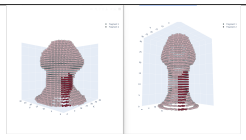
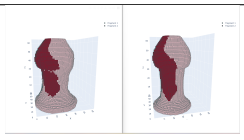
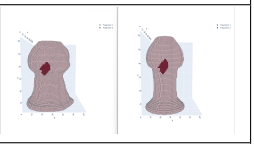
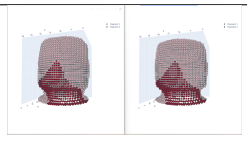
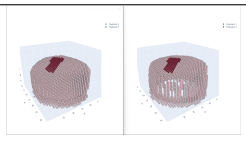
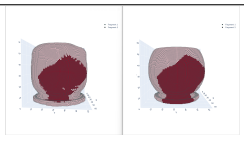
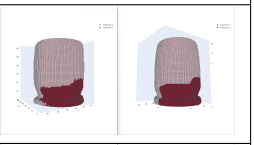
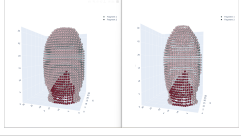
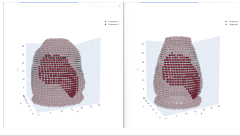
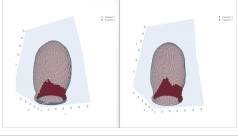
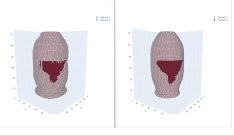
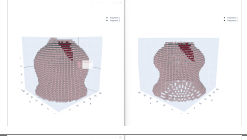
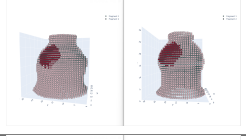
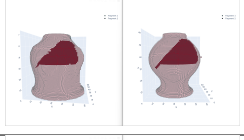
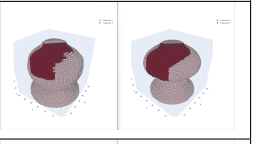
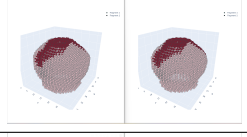
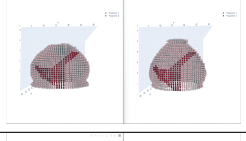
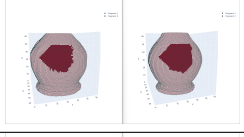
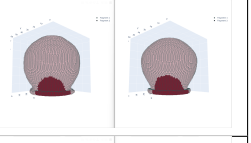
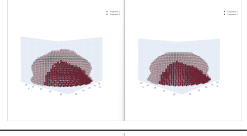
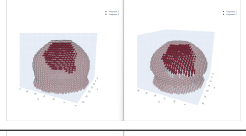
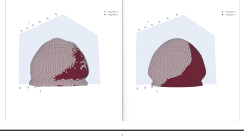
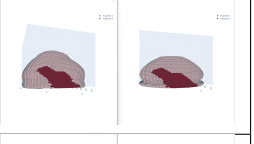
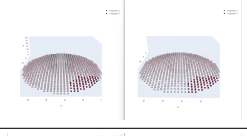
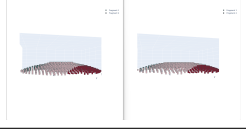
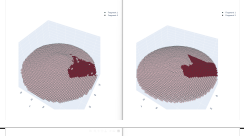
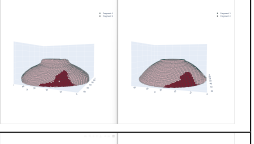
(c) class 4



(d) class 8

Figure 4: Limitations

Table 3: Generated results (left) compared with ground truth (right).

	example1(32)	example2(32)	example3(64)	example4(64)
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				

6 Conclusion

At the final stage, we can generate reasonable complete pottery from fragments, and by using different metrics to evaluate the result, we can see our model performed quite well. During the implementation, we analyzed and established the whole process from data processing to model training and testing. The core difficulty we encountered was the instability of GAN training, and by combining different strategies and using different metrics for evaluation, we were able to get decent results. But still, the training methods were empirical and had room for further improvements. Also, there were some limitations as shown before. So future improvements might try to find methods to enhance the stability of our model in extreme conditions and stabilize the training process. For future extensions, we suggest applying the model for similar generation tasks and adjusting the training strategies and evaluation metrics accordingly.

References

[1] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, volume 27, 2014.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[3] Renato Hermoza, Daniel G. Aliaga, and Rodrigo Quispe. 3d reconstruction of incomplete archaeological objects using adversarial network. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.

[4] Pablo Navarro, Celia Cintas, Manuel Lucena, José Manuel Fuertes, Antonio Rueda, Rafael Segura, Carlos Ogayar-Anguita, Rolando González-José, and Claudio Delrieux. Iberianvoxel: Automatic completion of iberian ceramics for cultural heritage studies. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5833–5841. International Joint Conferences on Artificial Intelligence Organization, 8 2023. AI and Arts.

A Supplementary Material

Source code, running guidance, and model checkpoints are in our GitHub repository at:
https://github.com/weepingguitar/CV2024_PKU
You can also find our result demos at:
https://disk.pku.edu.cn/link/AAB3E7942D487A468FAB3702A4F9A6DBCC