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

Quizmaster: 罗宇轩

Motivation

Archaeological studies require the accurate reconstruction of fragmented artifacts, such as pottery, to unveil civilizations and document history. Manual reconstruction of fragmented ceramics is time-consuming and involves physical manipulation. To address this challenge, for those of you who are enthusiastic and thoughtful, you will utilize your knowledge in computer vision and artificial intelligence to construct a generative model that assists archaeologists in reconstructing complete shapes from ceramic fragments.

In this task, archaeologists provide a pottery dataset `vox_pottery.zip`. It is a voxel dataset describing 11 types. We will guide you through this dataset in the following description and progressively teach you how to build a voxel completion model based on a 3D Autoencoder Generative Adversarial Network. Your goal is to successfully train and test its reconstruction effectiveness.

1. Dataset at a Glance

You can download the dataset from the link:

https://disk.pku.edu.cn:443/link/6B902375BB5D488F5BB8B0FF51512F12

有效期限：2024-01-31 23:59

The ceramics are classified into eleven different classes based on their shape. These classes consider the forms of the lip, neck, body, base, and handles and the relative ratios between their sizes. Nine of these classes correspond to closed pottery shapes, and two others belong to open ones.

The directory tree is:

```
data_voxelized
└── data
    ├── Test
    |   ├── 1
    |   ├── 2
    |   ├── 3
    |   ...
    |   └── 11
    └── Train
        ├── 1
        ├── 2
        ├── 3
        ...
        └── 11
```

Each file is made up of the pottery ID and the number of fragments:

data/(test or train)/CLASS/**POTTERY_ID**-n**NroFragments**-*.vox

For example: **data/test/9/AL_11D-n005-t1649436904.vox**

Notice:

- The dataset is formatted with a '.vox' postfix, indicating that it follows the MagicaVoxel format, allowing for compact representation and easy usage with the `pyvox.parser` library.
- Each file contains a complete ceramic utensil in 64×64×64 resolution. After reading, you can receive a `numpy.ndarray` type of data, with each position recording the fragment labels.
- The dataset consists of at most 10 types of fragments, numbered from 1 to 10, with 0 denoting an "empty" position.

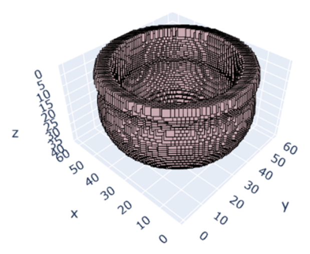
1) Try visualize the dataset (10 pts.)

Thus, for your first attempt, please complete `utils/visualize.py` to:

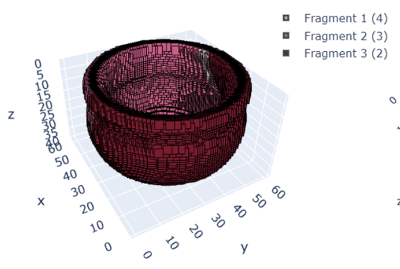
- Successfully read a voxel and familiarize with the data in the function `__read_vox__` under a given resolution (perhaps 32×32×32). Considering the heavy memory consumption, we may initially implement our GAN based on a lower resolution.

- Implement `__read_vox_frag__` to read the designated fragment from one voxel data at `fragment_idx`. One simple way is to set the voxels that don't match the given label to zero.
- Revise the `plot` function and implement `plot_frag` functions to visualize the entire or one specific fraction of the pottery, respectively.
- Implement `plot_join` for final analysis. This function receives two voxels and combines them together, useful for visualizing your model's generated fragment and matching it with the input data.

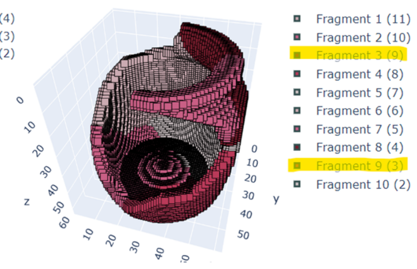
If you succeed in completing the above code implementation, the results and functionalities you should obtain are as follows (**Note:** you need to implement a unit test yourself; it is efficient to create an `.ipynb` file to check the validity of your implementations):



(a) Visualization of plot function



(b) Visualization of plot_frag function



(c) You can interact with each fragments if you use plotly.graph_objects for visualization

2. Implement FragmentDataset.py

Through the above attempts, you've familiarized yourself with the dataset format (and if you find it uninteresting or challenging, it's not too late to switch tasks now 🤣). To leverage the data for training the network you're about to design, the first step is to build a Python `class` responsible for handling data reading and processing, preparing it for the `DataLoader` class.

Considering that our objective is to **reconstruct the entire shape of pottery from ceramic fragments**, let's aptly name it `FragmentDataset`. This name suggests that the dataset will undergo further processing within this class, including random or specific extraction of fragments for training. The class inherits from the `torch.utils.data.Dataset` class. You can refer to its source code to identify the parts you need to implement.

2）Implement the Dataset class (15 pts.)

Follow the instructions in `utils/FragmentDataset.py` and implement the functions:

- **init function:** You must implement `__init__` function and initialize properties that may be useful in future processes.
- `__getitem__` **method:** This method will be called in the `DataLoader`. Implement it accordingly.
- `__getitem_specific_frag__` **function:** For evaluation purposes, you may need to specify the fragment type you are going to select. Implement this function to achieve that.
- Utilize `__select_fragment__` and `__select_fragment_specific__` within the above functions, calling them respectively.

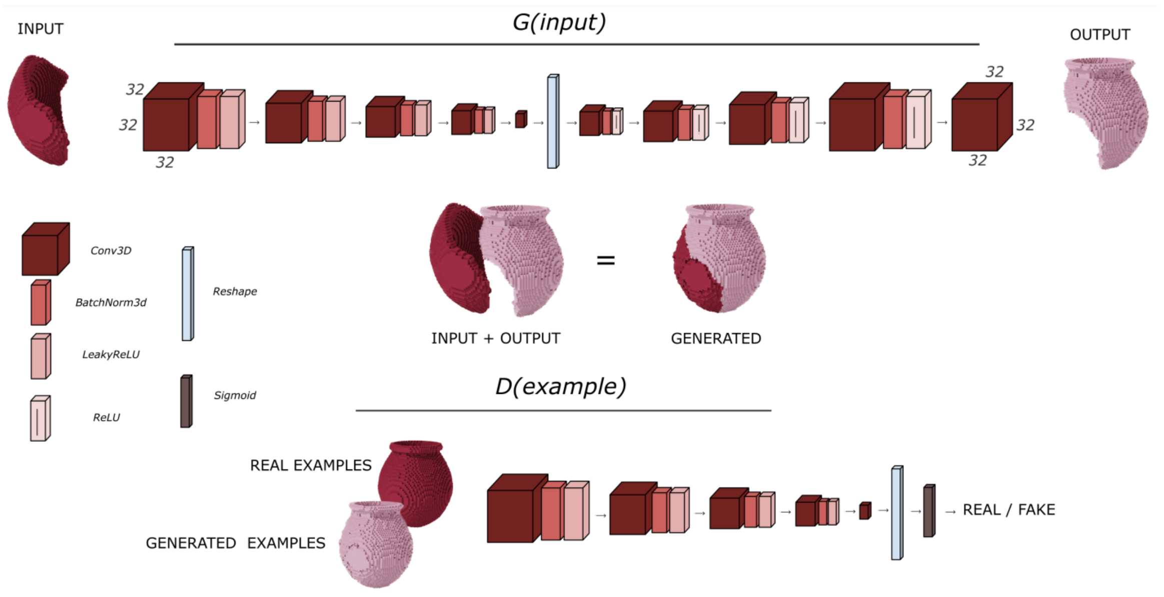
3. Constructing a naive GAN

A typical GAN framework consists of a generator **(G)** and a discriminator **(D)**. G aims to generate realistic artifacts, while D learns to discriminate between real data $p_d(x)$ and generated samples. In our context, z is a voxelized input fragment (**we recommend you start with 32 × 32 × 32 volumes**). The generator function $G(z)$ maps this fragment to the data space of a complete voxelized pottery, creating an Autoencoding GAN (AE-GAN).

In AE-GAN, G includes *encoders* trained to map fragments to latent space and decoders map points in latent space (w) back to complete pottery. Iterative adjustments inform G on generating misclassifications in D. Here, x is a three-dimensional binary array representing voxelized pottery geometry. $D(G(z))$ is the probability that G's output is a real artifact from the 3D dataset.

D maximizes $log(D(x))$, the probability of correctly classifying real ceramics, while G minimizes $log(1 - D(G(z)))$, the probability of D recognizing generated outputs. the model you are going to implement is an autoencoding GAN, where the generator is an Autoencoding network $Encode(z) \rightarrow w$, $Decode(w) \rightarrow x\prime$. To train the discriminator, we use $D(y)$, where $y = z + x\prime$ for generated examples.

A figurized explanation lies below:



3) Implement 3D AE-GAN (20 pts.)

To implement the blocks in GAN, we recommend you follow the popular `“Conv-BatchNorm-Activate”` or `“Conv-BatchNorm-Maxpooling-Activate”` paradigm among auto-encoder structures. You can easily search for numerous 2D AE repos for reference.

Before you start, you may wonder the kernel size, stride and padding numbers. Take 2D data as example, the dimension of a convolved image follows:

$$new\_len = \frac{len - kernel\_size + 2 \times padding}{stride} + 1$$

thus for 32 resolution or 64 resolution, you can pick proper kernel size, padding and stride.

And for the following instructions is what you are going to conduct in `utils/model.py`:

- Pick one paradigm and derive the kernel_size, padding, and stride. Further you should be aware of the dimension of bottleneck latent z.
- Build `__init__` to initialize hyper-parameters. Your model should be compatible with data in either 32 dimension or 64 dimension resolution.
- Build the blocks in `__init__` following the guidance in handout and python file.
- Conduct `__forward__` functions in `Generator` and `Discriminator`.

4. Complete the training process

To alleviate the burden of parameterization-tuning, we provide a set of possible hyper-params:

```
epochs = 100
loss_function = 'BCE'
generator_learning_rate = 2e-3
discriminator_learning_rate = 2e-4
initial_data_resolution = (32, 32, 32)
optimizer = 'ADAM'
beta1 = 0.9
beta2 = 0.999
batch_size = 64  # modify according to your device capability
```

Note that the training objective for GAN is displayed left-hand-side.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim \overset{\text{Completed Voxelized Space}}{p_d(x)}}\left[log(D(\overset{\text{Completed Pottery}}{x}))\right] + \mathbb{E}_{z\sim p_z(z)}\left[log(1 - D(z + \overset{\text{Generative Autoencoder}}{G}(\overset{\text{Input Fragment}}{z})))\right]$$

We provide several possible ealuation metrics for this voxel-completion task: **Dice similarity coeffcient (DSC)**[4]; **Jaccard distance**[5] and **Mean squared error (MSE)**.

- Given A: the generated set of voxels, and B: the real ceramic, **DSC** is formulated as

  $DSC = \frac{2|A \cup B|}{|A|+|B|}$, where $|A|$ and $|B|$ is the size in voxels of the pottery. The values of the formula are between 0 and 1, where 1 represents the maximum similarity, and 0 is the minimum.

- **Jaccard distance**, like DSC, is a statistic used to estimate the dissimilarity of a dataset, but unlike DSC, Jaccard distance satisfes the triangle inequality. Jaccard distance is defned as:

  $JD(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$, where A is a three-dimensional binary array containing the
  generated pottery voxelized geometry, and B is the real ceramic. The values of the formula are between 0 and 1 too, but unlike DSC, 1 represents the minimum similarity and 0 is the maximum.

- **MSE** is a common quality estimator in 2D and 3D classifcation and reconstruction problems, which we will not go into detail.

- **If you find it hard to implement the above metrics, you can design your own metric and prove its effectiveness.**

4) Implement training.py (25 pts.)

- Implement the whole training process following the instructions in `training.py`.
- Make sure you calculate the score functions and update gradients correctly for Discriminator and Generator.
- Also, try to implement a test function, utilizing test datasets and test dataloader to validate current training quality. You may have to implement several metrics and finally present both quantitative and qualitative analysis about your endeavors.

5. Move to higher resolutions

If you've moved to this step, congratulations! You've succeeded in recovering pottery fragments using your constructed generator! Now let's move on and try higher resolutions in 64×64×64!

5) Train and test your model on 64^3 resolution (5 pts.)

- Re-train your model on higher-resolution datasets.
- Conduct comprehensive quantitative anaysis.
- Compare the visualized results between labels and fragment percentages.

6. (Optional) Improvements

You may have noticed that at higher resolutions, the generated fragments may suffer from deformation and distortion. It is pointed out here that introducing normal vectors is often advantageous for improving the quality of the generated model reconstruction. Designing loss functions that focus on normal vectors or other aspects in more detail can also be beneficial. Additionally, switching to a more powerful model backbone can help.

Therefore, if you are willing to challenge yourself, you can complete the following three parts:

6) Modifications to the naive GAN (25 pts.)

- Try to introduce surface norm based on the voxelized dataset.
- Derive effective loss functions besides basic GAN loss.
- Introduce more effective GAN structure.

Report & Presentation:

Please write a detailed report summarizing all the efforts of your group and presenting all completed experiments. The report must clearly state the names and student IDs of the team members, along with the tasks assigned to each student. In the final stage, we will organize two class presentations, so please ensure that each group of students prepares early.

For more basic information, you can refer to the papers listed in the references and conduct internet searches. We encourage in-group discussion, so please try to minimize the frequency of asking the teaching assistant for assistance and aim to independently complete tasks within the group.

References:

[1] Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems* 27 (2014).

[2] Hermoza, Renato, and Ivan Sipiran. "3D reconstruction of incomplete archaeological objects using a generative adversarial network." *Proceedings of Computer Graphics International 2018*. 2018. 5-11.

[3] Cintas, Celia, et al. "Automatic feature extraction and classification of Iberian ceramics based on deep convolutional networks." *Journal of Cultural Heritage* 41 (2020): 106-112.

[4] Sorensen, Thorvald. "A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons." *Biologiske skrifter* 5 (1948): 1-34.

[5] Kosub, Sven. "A note on the triangle inequality for the Jaccard distance." *Pattern Recognition Letters* 120 (2019): 36-38.

[6] Wang, Bingxu, Jinhui Lan, and Feifan Li. "MSG-Voxel-GAN: multi-scale gradient voxel GAN for 3D object generation." *Multimedia Tools and Applications* (2023): 1-18.

[7] Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

[8] Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020.