Introduction to Computer Vision (Spring 2022)
Assignment 3

Release date: May 2, due date: May 22, 2022 11:59 PM

This assignment includes 5 tasks: transforming a depth image to a point cloud, sampling a point cloud from a mesh, implementing Marching Cube, implementing and training a PointNet for classification and segmentation, and training a Mask R-CNN, which sums up to 150 points and will be counted as 15 points towards your final score of this course. This assignment is fully covered by the course material from Lecture 7-10.

The objective of this assignment is to get you familiar with processing 3D data and coding the basic deep learning-based algorithms of 3D vision and object detection. We offer starting code for all the tasks and you are expected to implement the key functions using Python, Numpy, and PyTorch (for neural networks).

Policy on using for loop/while: for some questions that don't allow for loop/while , using them will be penalized (2 point for 1 use). Some useful Numpy functions are included in Appendix for your information.

Submission: please compress your code along with your results to **Name_ID.zip** following the original path structure, then submit to course.pku.edu.cn. Feel free to post in the discussion panel for any questions and we encourage everyone to report the potential improvements of this assignment with a bonus of up to 5 points.

1. **Backprojection: Transform a Depth Image to a Point Cloud (15 points):**

   In this question, you are required to transform a depth image to a point cloud. The depth image is synthesised by a standard perspective camera.

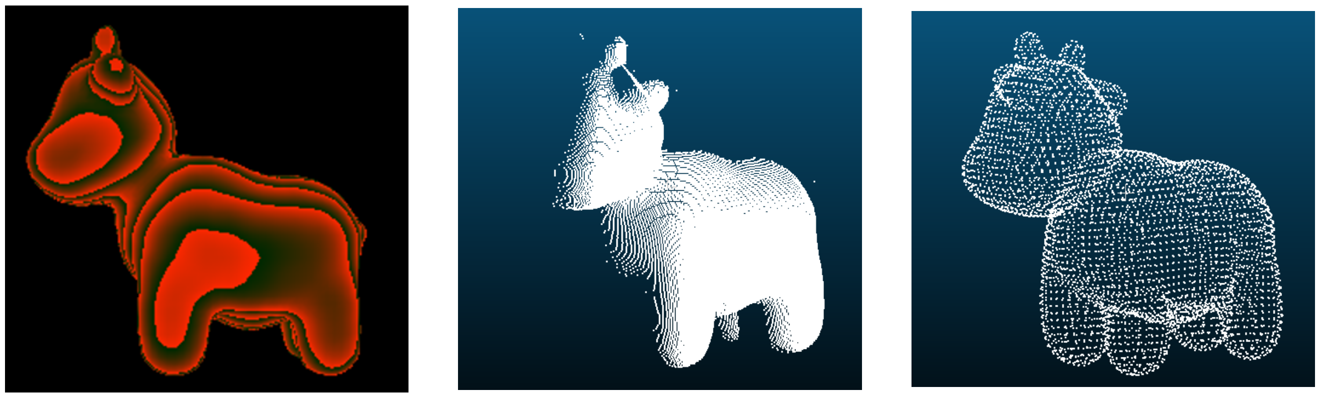

Figure 1: Left: raw depth image. Middle: transformed partial point cloud with another viewpoint. Right: ground truth complete point cloud.

After backprojection, compute the one-way Chamfer distance from your generated partial point cloud to the ground truth complete point cloud. You are required to only use Numpy to finish this task. Note that for-loop is not allowed.

2. **Sample a Point Cloud from a Mesh (35 points):**
Please refer to the newest slides (Lecture 9 P8-P20) from course.pku.edu.

a)[15 points] Uniform Sampling

An easy and fast way to sample a point cloud from a mesh is uniform sampling. First, you need to compute the area of each individual face and use it to compute the probability of each face to be sampled. Then, independent identically distributed (i.i.d.) sample faces according to the probabilities. Finally, for each sampled face, uniformly sample one point inside the triangle.

In this question, you are required to implement this algorithm and you should expect a result as Figure 2. Note that for-loop is not allowed.

b)[15 points] Farthest Point Sampling

Though uniform sampling is easy and fast, this method often results in irregularly spaced sampling as shown in Figure 2. Farthest point sampling (FPS) is another sampling strategy, which can ensure that the sampled points are far away from the others.

Under the greedy approximation, this algorithm will first uniformly sample a large number of points U, and randomly choose one point p_0 as the initialization of a point set S. Then, for each iteration, pick up the point $p_i \in U$, which is the farthest to the current point set S, and add p_i to S. This pick-and-add process is repeated until the point set S have enough points as we want. The result is shown in Figure 2.

Note that for-loop is allowed here, since the sampled point of each iteration relies on the results of previous iterations.

c)[5 points] Metrics

Finally you are required to compute the Chamfer distance (CD) and earth move distance (EMD) of those two point clouds sampled by different methods. To give you a feeling about which metric is more sensitive to sampling, you are required to repeat sampling and computing metrics for 5 times, and save the mean and variance of CD and EMD for submission. For EMD, you can directly use this repository.

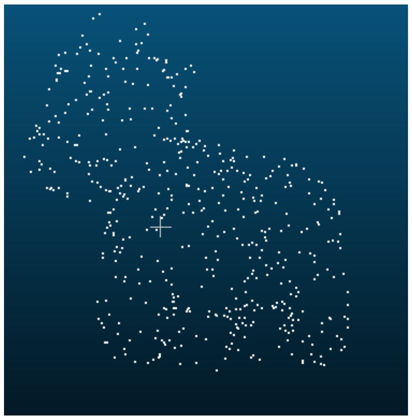
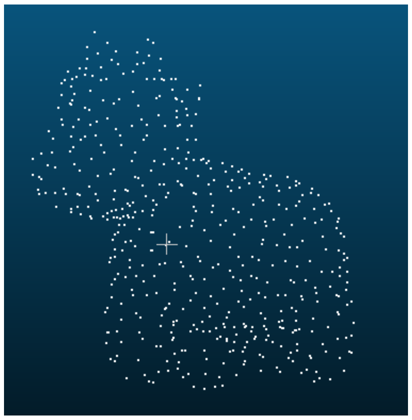

Figure 2: Left: result of uniform sampling. Right: result of farthest point sampling.

3. **Marching Cube (25 points):**

   Marching Cube is one of the most classic and famous algorithm to transform a signed distance field (SDF) to a mesh. In this question, you are expected to implement this algorithm. To make it easier to start, we provide the lookup table and some useful function, as well as a detailed demonstration of how to use them. Please see the hints in *lookup_table.py* for more details.

   You are required to only use Numpy to finish this task. To help you debug and get a feeling of this algorithm, we also provide two SDFs for you. The meshes generated from Marching Cube algorithm should look like Figure 3. Note that for-loop is allowed here to reduce the difficulty.

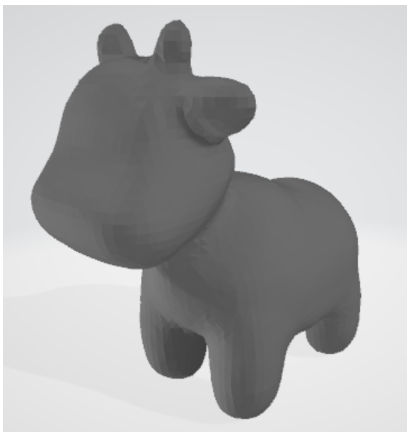
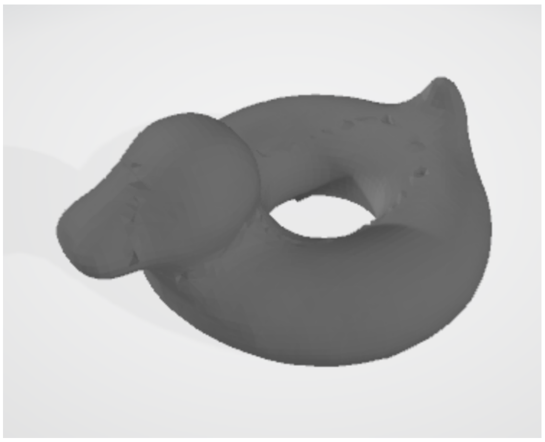

Figure 3: Generated meshes from Marching Cube algorithm. Left: Spot. Right: Bob.

4. **PointNet (35 points):**

   PointNet is the most widely adopted neural network for point cloud learning. In this question, you are requested to implement the pipeline of PointNet for both classification and segmentation tasks on ShapeNetPart[1] and then visualize the features. Please read and follow the README to prepare the environment and dataset.

   a)[20 points] Points Classification and Segmentation.

   First of all, you need to build the PointNet by following the architecture in Fig.4. In implementation, you will build separate networks for different tasks with different feature dimensions. We provide the off-the-shelf data-loaders of ShapeNetPart for both classification and segmentation tasks. The training process may take you about 20 minutes. The content of this question can be found in *model.py*.

   a.1) **For segmentation task**, your network should predict the part labels of the given point cloud. Specifically, we consider the "airplane" category. Please refer to *train_segmentation.py* for more details.

   a.2) **For classification task**, your network should predict the category of given point clouds. Besides, we want you to investigate the effect of the dimensions of the global feature. So in this classification part, you are requested to train two separate PointNets. One is the original PointNet with $1024D$ global feature. And another one uses $256D$ global feature. You will have a similar training curve as Fig.5. Please check *train_classification.py* for more details and submit the screenshot.

[1] Yi L et al. A scalable active framework for region annotation in 3d shape collections[J]. ACM Transactions on Graphics (ToG), 2016

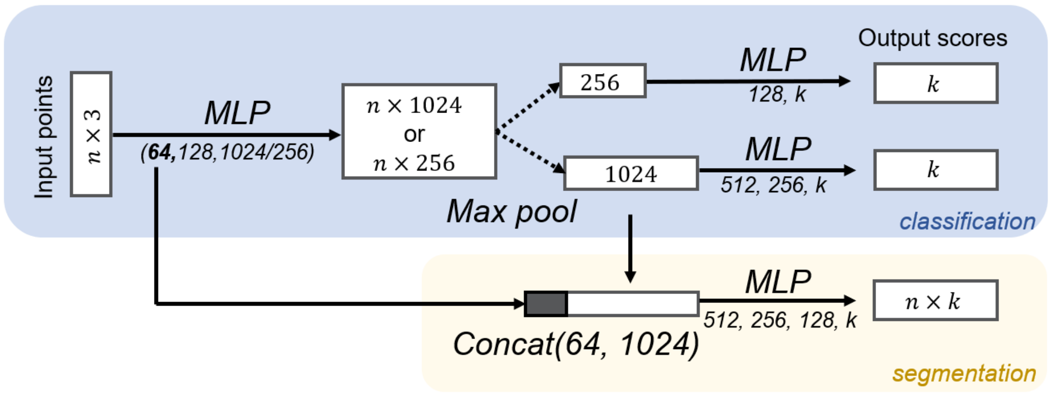

Figure 4: PoinNet architecture for both classification and segmentation tasks.

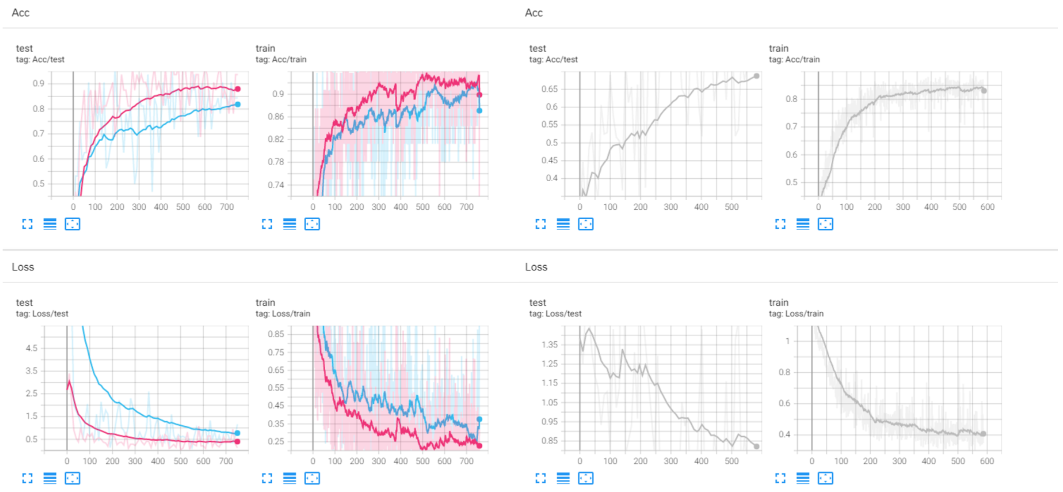

Figure 5: Classification task with 1024D (pink) and 256D (blue) global feature. Segmentation task with 1024D global feature (gray)

b) [15 points] Point Feature Visualization

We have already trained some networks and draw lots of curves. But sometimes, we want a more comprehensive way to understand what the network actually learned. So in this question, you are required to visualize the *cruciality* of $n \times 1024$ points feature before max-pooling in the classification network(1024D). And the *cruciality* is simply defined as the maximum value along the point feature dimensions. The colormap and points-to-ply functions are provided and you can obtain similar colored point clouds as Fig.6. please see *classficiation1024D_feat_vis.py* for more details.

5. **Mask RCNN (40 points):**

Mask RCNN is the milestone of the 2D instance segmentation. Even now, it's still considered as one of the most important baselines in the field. In this question, you will have the chance to play with a small Mask RCNN by predicting the instance segmentation of simple shapes, *e.g.* triangular, sphere and rectangle. We build the Mask RCNN with *torchvision*[2] and have replaced the backbone with

[2]https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html

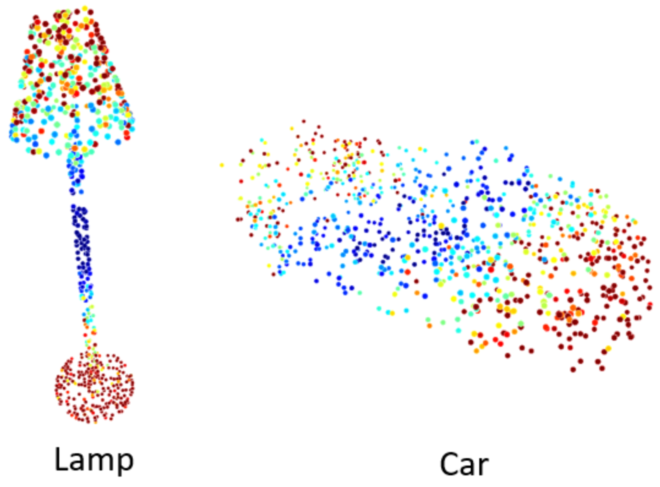

Figure 6: Feature visualization with colormap. Your results may not be exactly the same.

lightweight mobilenetV2. Some settings have been modified to make it easy to run on pure CPUs machines. The loop is allowed in this question and you can create new functions for your convenience. The content of this question can be found in *MaskRCNN*.

a)[20 points] Prepare the dataset

Data preparation is always the first if you want to get a network work on your own data. In this question, you are requested to generate a dataset that contains rectangles, spheres and triangles. Our requirements are:

1. The number, position and size of the shapes should be random. And the color for both shapes and backgrounds should be random.

2. We do not want large occlusions between shapes. So you need to implement a non-max suppression (NMS) to remove the shapes with large overlap.

3. The detailed parameters are listed in *dataset.py*. We also encourage any creative ideas to make the data preparation more interesting.

Several examples can be found in Fig.7. Please complete the *MaskRCNN/dataest.py*.

b)[10 points] Train the Mask RCNN

Now, with all data prepared, we are ready to train a Mask RCNN. You can obtain and load the pre-trained weight to boost your training process by following the README. And the log will be automatically saved in *results/maskrcnn/train.log*. Please check the *MaskRCNN/train.py* (this could cost you 1 $\sim$ 2 hours for a good laptop with CPUs).

c)[10 points] Evaluate the Mask RCNN

Finally, we'd like to evaluate the performance of Mask RCNN. So You are required to implement a function which is able to compute the mAP0.5 for both the detection and segmentation tasks. However, the implementation of data preparation may vary by student. We, therefore, provide a separate dataset called *SingleShapeDatast* which provides only one shape in each image. You will evaluate your Mask RCNN with this specific dataset. Please complete the *MaskRCNN/test.py*.

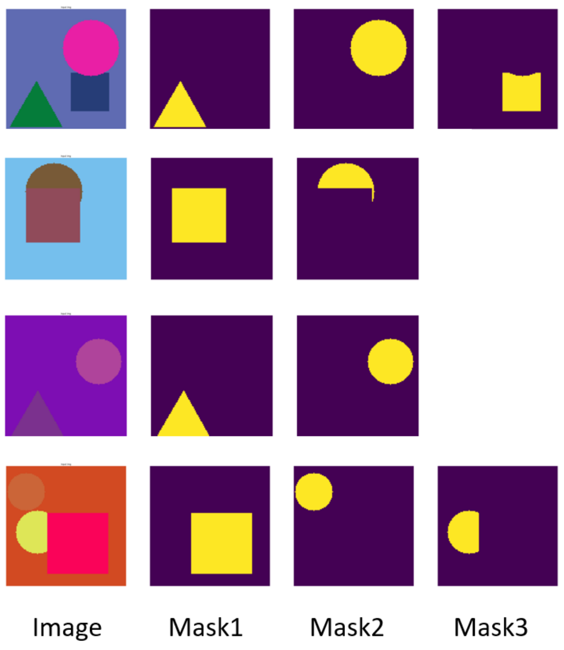

Figure 7: Input data for Mask RCNN

Similar results could be found in Fig.8. Note that, due to the limited training steps, the performance would not be perfect. The points would be counted as long as the results are reasonable :)

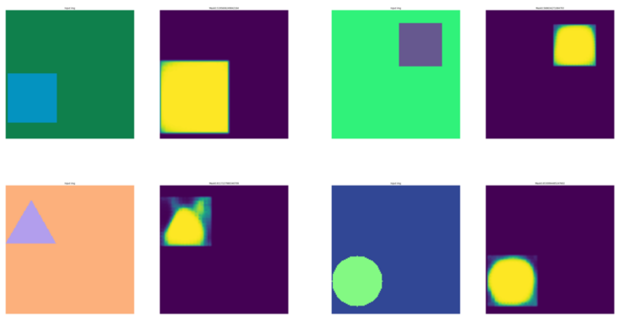

Figure 8: Results of the Mask RCNN

Appendix

1. We recommend some handy Numpy functions which may help your tensor-style coding.

   - meshgrid, https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html
   - concatenate, https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html
   - where, https://numpy.org/doc/stable/reference/generated/numpy.where.html
   - argmax, https://numpy.org/doc/stable/reference/generated/numpy.argmax.html
   - linalg.svd, https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html

2. We recommend some useful software for visualize point clouds and meshes.

   - CloudCompare, https://www.danielgm.net/cc/
   - MeshLab, https://www.meshlab.net/