311551051 曾正豪 HW2

A. Code

I. Training segmentation model

The model I used:

encoder: mobilenetv2dilated decoder: c1_deepsup

I choose this architecture since it run fastest with not bad accuracy.

The content of the dataset:

In training apartment0, I use 1000 images that takes in 10 different rooms.

In training other scene, I use 1000 images that takes in 10 different scenes and rooms.

Training setting:

In this part, I almost didn't modify the setting from "*ade20k-mobilenetv2dilated-c1_deepsup.ymal*" file since I use the pretrained weight to do the fine-tuning. I only modify the "*num_epoch*" and "*start_epoch*" to let the model to start from the pretrained weight. And the "*epoch_iters*" is also set to 1000 since the size of training set is 1000.

Training result:

Trained on apartment0:

```
Epoch: [30][980/1000], Time: 0.05, Data: 0.02, lr_encoder: 0.000028, lr_decoder:
 0.000028, Accuracy: 97.29, Loss: 0.125442
Saving checkpoints...
Training Done!
```

Mean IoU: 0.5926, Accuracy: 96.75%

Trained on other scene:

```
Epoch: [30][980/1000], Time: 0.05, Data: 0.02, lr_encoder: 0.000028, lr_decoder:
 0.000028, Accuracy: 96.19, Loss: 0.200042
```

Mean IoU: 0.0701, Accuracy: 55.15%

II. 3D semantic map reconstruction

How to run:

Just run it with "python 3d_semantic_map.py", then use keyboard to control the agent. In the end, press "G" to see the reconstructed 3d semantic map.

```python
def custom_voxel_down(pcd, voxel_size):
    pcd_color = np.asarray(pcd.colors)

    min_bound = pcd.get_min_bound() - voxel_size * 0.5
    max_bound = pcd.get_max_bound() + voxel_size * 0.5
    ret = pcd.voxel_down_sample_and_trace(voxel_size, min_bound, max_bound)
    pcd_down = ret[0]
    cubics_ids = ret[1]

```

In the "*custom_voxel_down*" function, I would take the input point could and the voxel size as its input. Then, take out the point cloud's color to a numpy array. And use "*voxel_down_sample_and_trace*" to do the voxel down sample. This function can indicate what points are in the same voxel in the input point cloud.

```python
    new_pcd_color = []
    for cubic_ids in cubics_ids:
        cubic_ids = cubic_ids[cubic_ids != -1]
        cubic_color = pcd_color[cubic_ids]

        unqc, C = np.unique(cubic_color, axis=0, return_counts=True)
        index = np.argmax(C)
        new_pcd_color.append(unqc[index])

```

After that, I would iterate all the voxel cube that has points. In each iteration, I would collect all the point colors in a voxel before down sampling. Then, use numpy's "*unique*" function to count the number of occurrences and choose the largest one. In the end, use that color to fill that voxel's point.

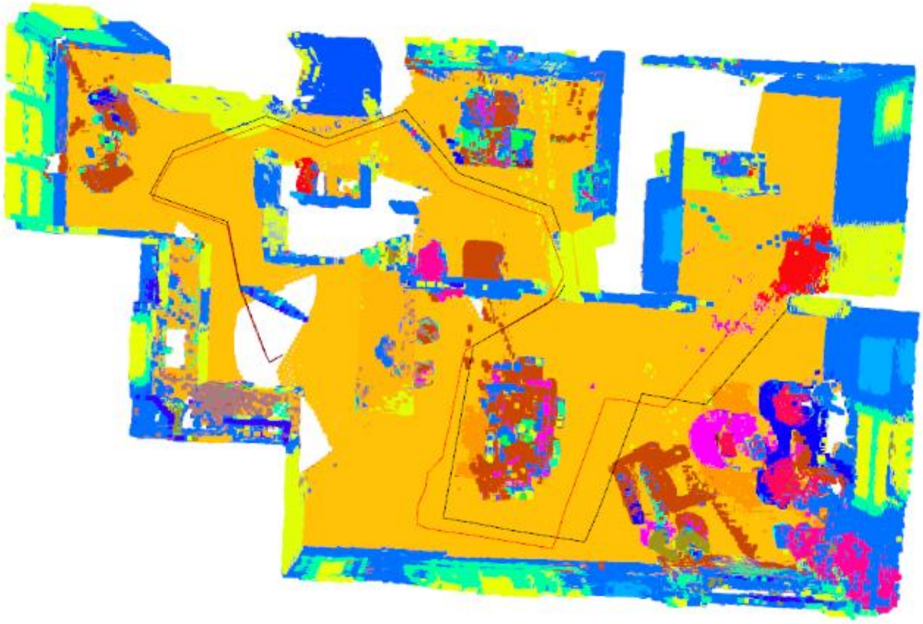
```python
    new_pcd_color = np.array(new_pcd_color)
    pcd_down.colors = o3d.utility.Vector3dVector(new_pcd_color)

    return pcd_down
```

Finally, fill all the collected color into that down-sampled point cloud.

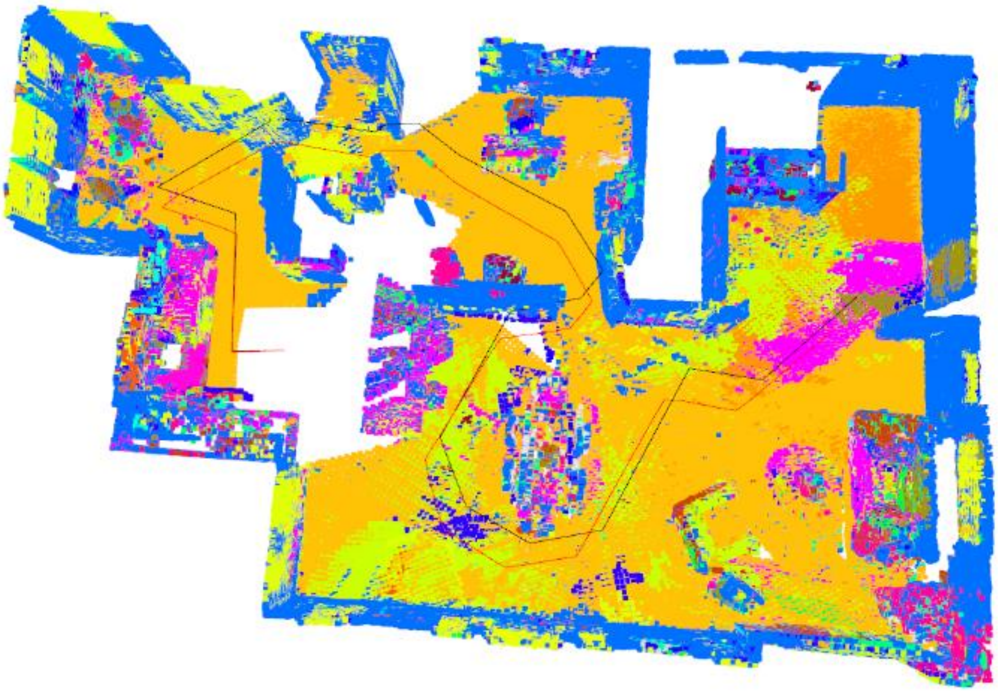
B. Result and Discussion

I. Result of your semantic map

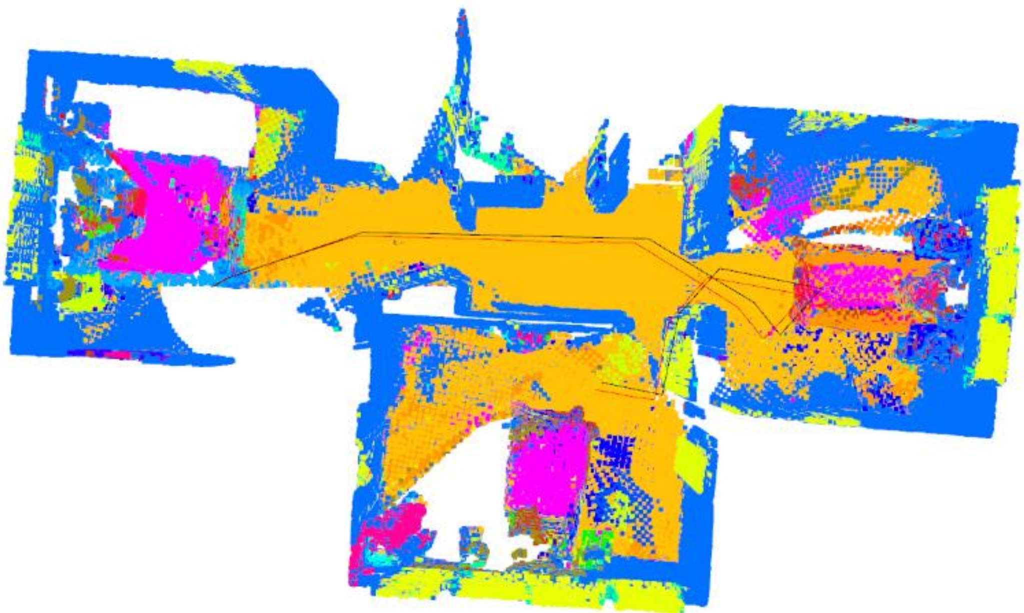


▲ Floor1, trained on apartment_0

▲ Floor2, trained on apartment_0



▲ Floor1, trained on other scenes



▲ Floor2, trained on other scenes

II. Anything you want to discuss

From the result, we can observe that the model trained in apartment_0 would have a much better performance than the one trained in other scenes. The model trained in other scenes would have many noises on objects. However, it can still roughly segment the floor and wall part. I think it's because that floor and wall are the common object in different scenes.

III. Any reference you take

CSAILVision/semantic-segmentation-pytorch
https://github.com/CSAILVision/semantic-segmentation-pytorch
Implementation of semantic-segmentation-pytorch - HackMD
https://hackmd.io/wNGlmMq2RC-lY3l8JhO4SA?view
Open3D-PointNet2-Semantic3D/downsample.py at master · isl-org/Open3D-PointNet2-Semantic3D · GitHub
https://github.com/isl-org/Open3D-PointNet2-Semantic3D/blob/master/downsample.py
python - Find Top10(n) RGB colors in Numpy - Stack Overflow
https://stackoverflow.com/questions/61992049/find-top10n-rgb-colors-in-numpy