Homework 2 Report

Name: 謝元碩
Student ID: 311512015

A. Code

1. Training segmentation model

(1) 收集資料

首先使用 data_generator.py 收集訓練時所需的資料集。此處我希望 apartment_0 和其他環境(後面稱作 others)的訓練集總數相同，因此作了以下修改：

- Apartment0 的 create_room 有 13 個，frames_per_room=100→13*100=1300 張
- Others 的 create_room 有 20 個，frames_per_room=65→20*65=1300 張

```
self._scene_to_rooms = {
    "apartment_0": [
        create_room(-0.3, 1.4, 2.8, 1.68, 0.28, 3.04), # bedroom
        create_room(2.25, -1.38, 3.12, 3.52, -2.0, 2.19), # bathroom
        create_room(-1.65, -2.84, 3.36, -0.08, -5.12, 2.36), # bedroom
        create_room(0.86, -6.93, 2.93, 3.0, -7.85, 2.39), # bedroom
        create_room(2.53, -3.9, 3.15, 3.19, -2.98, 2.81), #bathroom
        create_room(0.95, -3.20, 3.09, 1.76, -4.96, 2.10), #corridor
        create_room(3.92, -4.79, 2.23, 5.07, -4.90, 1.30), #staircase
        create_room(3.7, -8.03, 0.69, 3.5, -6.28, -0.04), #entrance hall
        create_room(1.44, -8.22, 0.37, -1.75, -2.88, -0.27), #livingroom
        create_room(0.12, -0.65, 0.48, 0.89, 1.43, -0.39), #livingroom
        create_room(2.21, 1.98, 0.00, 3.43, 3.63, -0.31), #small table room
        create_room(4.16, 0.07, 0.09, 2.96, -0.24, -0.17), #bathroom
        create_room(4.25, -1.24, 0.52, 2.32, -3.73, -0.46) # workroom
    ],
```

Apartment0

```
"apartment_1": [
    create_room(-1.20, 0.42, 0.53, 0.40, 5.59, -0.22), # livingroom
    create_room(2.76, 5.69, 0.49, 6.5, 4.1, -0.22) # meetingroom
],
"apartment_2": [
    create_room(-1.25, 1.03, 0.79, -0.09, -0.94, -0.26), # workroom
    create_room(1.90, -0.76, 0.35, 5.84, 0.67, -0.50), # bedroom
    create_room(5.69, 3.0, 0.38, 3.97, 4.08, -0.32), # livingroom
    create_room(3.11, 5.86, 0.38, 5.35, 7.75, 0.04) # eatingroom
],
"frl_apartment_0": [
    create_room(0.55, -4.08, 0.31, 4.58, -1.98, -0.24),
    create_room(4.89, -5.53, -0.31, 3.24, -7.87, 0.36),
    create_room(0.019, 1.89, 0.77, 0.51, 0.45, -0.29)
],
"frl_apartment_1": [
    create_room(-1.96, 0.09, -0.66, -0.79, 0.65, 0.46),
    create_room(1.27, 0.56, 0.78, 3.74, 4.14, 0.04),
    create_room(5.45, 4.85, 0.06, 7.45, 3.51, 0.53)
],
"hotel_0": [
    create_room(-1.56, -0.26, 1.08, 1.4, 0.95, 0.113),
    create_room(2.64, 1.4, 1.0, 4.51, 0.98, -0.07),
    create_room(4.71, -0.15, 0.90, 3.69, -0.65, 0.34)
```

Others

```
# adjust [frames_per_room] to collect arbitrary number of  images
generator = Generator(path=args.dataset_folder)
generator.generate(out_folder=args.output,
                   split_name='train',
                   frames_per_room=100)
generator.generate(out_folder=args.output,
                   split_name='val',
                   frames_per_room=20)
generator.generate(out_folder=args.output,
                   split_name='test',
                   frames_per_room=20)
```

Frames per room setting

切換收集資料集的位置如下：

```
class Generator:
    """Generator for replica dataset, rgb, depth, and semantics.
    """
    def __init__(self, path):
        self._dataset_path = os.path.normpath(path)
        # fill the scene name you want to collect data from, and set the rooms
        # self._scenes = ["apartment_1", "apartment_2",
        #                 "frl_apartment_0", "frl_apartment_1",
        #                 "hotel_0", "office_0", "office_1",
        #                 "room_0", "room_1", "room_2"]
        self._scenes = ["apartment_0"]

        self._height = 512
        self._width = 512
```

(2) 建立 odgt 路徑檔

這裡主要把 images 路徑轉成 annotations 路徑，以用來做訓練。程式碼如下：

```
def odgt(img_path):
    seg_path = img_path.replace('images','annotations')
    seg_path = seg_path.replace('.jpg','.png')

    if os.path.exists(seg_path):
        img = cv2.imread(img_path)
        h, w, _ = img.shape

        odgt_dic = {}
        odgt_dic["fpath_img"] = img_path
        odgt_dic["fpath_segm"] = seg_path
        odgt_dic["width"] = h
        odgt_dic["height"] = w
        return odgt_dic
    else:
        print('the corresponded annotation does not exist')
        # print(img_path)
        return None


if __name__ == "__main__":
    # for training
    modes = ['train','val']
    saves = ['others_training.odgt', 'others_validation.odgt']

    for i, mode in enumerate(modes):
        save = saves[i]
        dir_path = f"./data/output/images/{mode}"
        img_list = os.listdir(dir_path)
        img_list.sort()
        img_list = [os.path.join(dir_path, img) for img in img_list]

        with open(f'data/{save}', mode='wt', encoding='utf-8') as myodgt:
            for i, img in enumerate(img_list):
                a_odgt = odgt(img)
                if a_odgt is not None:
                    myodgt.write(f'{json.dumps(a_odgt)}\n')
```

(3) 設定 yaml 參數及路徑

- Model：無論是 apartment0 還是 others，我選擇的的都是 resnet50dilated，才能比較結果
- 路徑：DATASET 部分改成前面步驟做出來的 odgt(吃資料集)，最下面 DIR 改成訓練結果存放位置(權重、語意分割結果)
- 參數修改：
  - ✓ Num_class=101(類別數)
  - ✓ imgMaxSie=1000
  - ✓ batch_size_per_gpu=2
  - ✓ epoch_iters=650
  - ✓ batch_size_per_gpu* epoch_iters 必須等於訓練集張數(1300)
- visualize：改成 True，如此才能顯示圖片結果

```yaml
DATASET:
  root_dataset: ""
  list_train: "./data/others_training.odgt"
  list_val: "./data/others_validation.odgt"
  num_class: 101
  imgSizes: (300, 375, 450, 525, 600)
  imgMaxSize: 1000
  padding_constant: 8
  segm_downsampling_rate: 8
  random_flip: True

MODEL:
  arch_encoder: "resnet50dilated"
  arch_decoder: "ppm_deepsup"
  fc_dim: 2048

TRAIN:
  batch_size_per_gpu: 2
  num_epoch: 20
  start_epoch: 0
  epoch_iters: 650
  optim: "SGD"
  lr_encoder: 0.02
  lr_decoder: 0.02
  lr_pow: 0.9
  beta1: 0.9
  weight_decay: 1e-4
  deep_sup_scale: 0.4
  fix_bn: False
  workers: 16
  disp_iter: 20
  seed: 304

VAL:
  visualize: True
  checkpoint: "epoch_20.pth"

TEST:
  checkpoint: "epoch_20.pth"
  result: "./"

DIR: "ckpt/others"
```

(4) 執行 train.py 訓練

分別對 apartment0 和 others 進行訓練，訓練結果會存放在 ckpt 資料夾中。此處訓練完的權重，兩種環境以測試集做測試，皆擁有 90~95%的正確率。

(5) 訓練結果

使用 eval_multipro.py 觀察訓練結果。首先修改讀取的 class 檔：

```
colors = loadmat('data/color101.mat')['colors']
```

接著是修改 mIOU 計算的部分，由於在 apartment0 中只有 49 種特定類別須納入考量，因此只要將那 49 種 class 取平均就好，程式新增如下：

```
new_iou = []
wb = openpyxl.load_workbook('apartment0_classes.xlsx', data_only=True)
s1 = wb['Sheet1']
for i in range(2,51):
    label = s1.cell(i,1).value
    print("add  " + str(int(label)) + "  " +  str(iou[int(label)]))
    new_iou.append(iou[int(label)])
wb.save('apartment0_classes.xlsx')

print('[Eval Summary]:')
# print('Mean IoU: {:.4f}, Accuracy: {:.2f}%'
#       .format(iou.mean(), acc_meter.average()*100))
print('Mean IoU: {:.4f}, Accuracy: {:.2f}%'
      .format(sum(new_iou)/len(new_iou), acc_meter.average()*100))
```

我使用 openyxl 讀取 HW2 spec 中給予的 excel 檔，直接去讀取特定類別，再將原本 iou 中的特定類別取出來計算即可。這裡分別使用 apartment0 和 others 的訓練權重對 1F、2F 做 mIOU 及 accuracy 計算，結果如下面的截圖。

```
[Eval Summary]:
Mean IoU: 0.1033, Accuracy: 61.82%
Evaluation Done!
```

1F apartment0

```
[Eval Summary]:
Mean IoU: 0.0402, Accuracy: 40.16%
Evaluation Done!
```

1F others

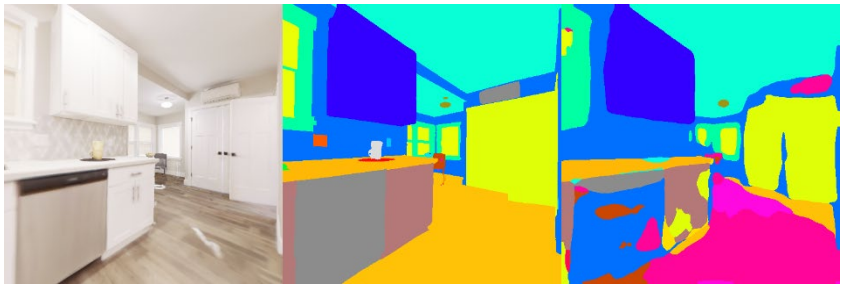
```
[Eval Summary]:
Mean IoU: 0.0757, Accuracy: 82.98%
Evaluation Done!
```

2F apartment0

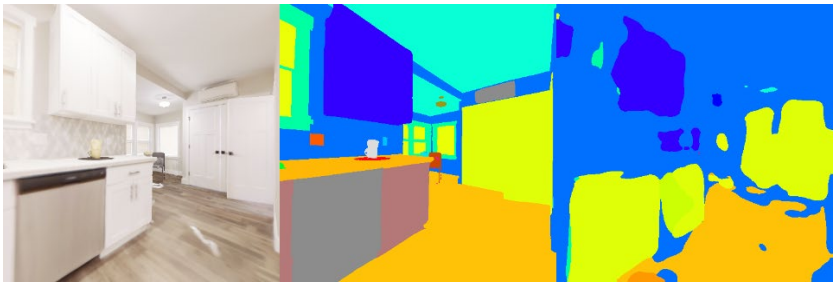
```
[Eval Summary]:
Mean IoU: 0.0436, Accuracy: 55.42%
Evaluation Done!
```

2F others

此檔案所讀取的 yaml 一樣為訓練前的檔案(因為只是要看結果)，語義分割結果如下：



1F apartment0



1F others

2F apartment0

2F others

由於顯示卡性能較為不佳，導致結果沒有那麼理想。後續 3D 重建即使用這些訓練結果。

2. 3D semantic map reconstruction

(1) 收集照片

在 eval_multipro.py 中，有多個針對語義分割任務所需的函式，包括讀取 json、label 標記、執行語義分割、存 annotation 圖片等，如下圖：

```
def filename_from_frame_number(frame_number):…
def load_scene_semantic_dict(self, scene):…
def fix_semantic_observation(self, semantic_observation, scene_dict):…
def save_color_observation(self, observation, frame_number, out_folder, scene):…
def save_semantic_observation(self, observation, frame_number, out_folder, scene_dict, scene):…
def save_depth_observation(self, observation, frame_number, out_folder,scene):…
def save_observations(self, observation, frame_number, out_folder, split_name, scene_dict, scene):…
```

我將這幾個函式中幾個重要的部分整理後，放進 HW1 的 load.py 做使用，新增的位置在 navigateAndSee 函式中，如下圖：

```
if save_count %2 == 0:
    scene = "apartment_"
    out_folder = "./semantic-segmentation-pytorch/data/apartment0_new"
    with open(os.path.join(os.path.normpath("dataset"), "apartment_0", 'habitat', 'info_semantic.json'), 'r') as f:
        scene_dict = json.load(f)
    if not os.path.exists(out_folder):
        os.makedirs(out_folder)
    instance_id_to_semantic_label_id = np.array(scene_dict["id_to_label"])
    semantic = instance_id_to_semantic_label_id[observations["semantic_sensor"]]

    semantic_img = Image.new("L", (semantic.shape[1], semantic.shape[0]))
    semantic_img.putdata(semantic.flatten())
    frame_number = f"{count:05d}.png"
    semantic_img.save(os.path.join(out_folder, "annotations", scene + frame_number))
    count = count+1
    save_count = 0

    cv2.imwrite(os.path.join(out_folder, "images", "apartment_" + frame_number),transform_rgb_bgr(observations["color_sensor"]))
    cv2.imwrite(os.path.join(out_folder, "depths", "depth_" + frame_number),transform_depth(observations["depth_sensor"]))
    print("Save image " + str(count))

    # 記錄當下座標記錄當下座標
    with open('ground_xyz.txt', 'a') as outfile:
        outfile.write(str(sensor_state.position[0]) + " " + str(sensor_state.position[1]) + " " + str(sensor_state.position[2]) + ' \n')

    # 更新圖像張數
    with open('count.txt', 'w') as outfile:
        outfile.write(str(count))
```

最後執行 load.py，即可存下 annotation、depth、rgb 的圖片。

(2) 建立 odgt 檔

由於執行 eval_multipro.py 需要讀取 yaml 檔中的 odgt 檔，因此我們還必須修改製作 odgt 程式碼的部分。我將讀取圖片的路徑改成前面 load.py 所收集的圖片路徑，並直接執行，如下圖：

```
if __name__ == "__main__":
    # for reconstruct
    save = 'reconstruct.odgt'
    dir_path = f"./data/apartment0_new/images"
    img_list = os.listdir(dir_path)
    img_list.sort()
    img_list = [os.path.join(dir_path, img) for img in img_list]
    with open(f'./data/{save}', mode='wt', encoding='utf-8') as myodgt:
        for i, img in enumerate(img_list):
            a_odgt = odgt(img)
            if a_odgt is not None:
                myodgt.write(f'{json.dumps(a_odgt)}\n')
```

(3) yaml 檔修改

這裡跟訓練時用到的內容大同小異，僅需修改 odgt 的路徑以及 visualize=True 的部分就好，如下圖：

```
DATASET:
  root_dataset: ""
  list_train: "./data/reconstruct.odgt"
  list_val: "./data/reconstruct.odgt"
  num_class: 101
  imgSizes: (300, 375, 450, 525, 600)
  imgMaxSize: 1000
  padding_constant: 8
  segm_downsampling_rate: 8
  random_flip: True
```

```yaml
VAL:
  visualize: True
  checkpoint: "epoch_20.pth"

TEST:
  checkpoint: "epoch_20.pth"
  result: "./"

DIR: "ckpt/others"
```

(4) 生成語義分割圖片

同前面的 eval_multipro.py，讀取上個步驟的 yaml 檔，接著將 visualize_result 函式中存取圖片的部分改成分別存取 ground truth 及 segmentation 的照片，就能順利存取 3D 重建所需的資料集，如下圖：

```python
def visualize_result(data, pred, dir_result):
    (img, seg, info) = data

    # segmentation
    seg_color = colorEncode(seg, colors)

    # prediction
    pred_color = colorEncode(pred, colors)

    # aggregate images and save
    # im_vis = np.concatenate((img, seg_color, pred_color),
    #                         axis=1).astype(np.uint8)
    im_vis = pred_color.astype(np.uint8)
    gt_vis = seg_color.astype(np.uint8)

    img_name = info.split('/')[-1]
    # Image.fromarray(im_vis).save(os.path.join(dir_result, img_name.replace('.jpg', '.png')))
    Image.fromarray(im_vis).save("reconstruct/" + img_name)
    Image.fromarray(gt_vis).save("reconstruct_gt/" + img_name)
```

(5) Reconstruct and custom voxel down

這裡針對 HW1 的 reconstruct,py 做修改，讀取前一個步驟存取的照片集，即可生成重建地圖，如下圖：

```python
def get_pointcloud(img2pcd_count):

    save_rgb.clear()
    save_pixel.clear()

    frame_number = f"{img2pcd_count:05d}.png"
    # img_sem = cv2.imread('semantic-segmentation-pytorch/reconstruct/apartment_' + frame_number)
    img_sem = cv2.imread('semantic-segmentation-pytorch/reconstruct_gt/apartment_' + frame_number)
    img_depth = cv2.imread('semantic-segmentation-pytorch/data/apartment0_new/depths/depth_' + frame_number)

    fov = 90
    f = float(512/2*(1/math.tan(fov/180*math.pi/2)))

    for i in range(0,512):
        for j in range(0,512):
            x = (img_depth[i,j][0]/25.5)*(j-256)/f
            y = (img_depth[i,j][0]/25.5)*(i-256)/f

            if y>(-0.6):  #去除天花板
                save_pixel.append([x,y,img_depth[i,j][0]/25.5])
                save_rgb.append([img_sem[i][j][2]/255,img_sem[i][j][1]/255,img_sem[i][j][0]/255])

    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(save_pixel)
    pcd.colors = o3d.utility.Vector3dVector(save_rgb)
    o3d.io.write_point_cloud('pcd/' + str(img2pcd_count) +  '.pcd',pcd)
    print("Save Point Cloud " + str(img2pcd_count))
```

接著介紹我是如何完成 custom voxel down。程式碼如下：

```
######################custom voxel down#####################
xyz = np.asarray(final_pcd.points)
xyz_color = np.asarray(final_pcd.colors)

# 建立bounding box
box = final_pcd.get_axis_aligned_bounding_box()
box.color = (1,0,0)
center = box.get_center() # 取質心
margin = box.get_extent() # 取長寬高
print("center: ", center, ", margin: ", margin)
x_min = round(center[0] - margin[0]/2, 2)
x_max = round(center[0] + margin[0]/2, 2)
y_min = round(center[1] - margin[1]/2, 2)
z_min = round(center[2] - margin[2]/2, 2)
z_max = round(center[2] + margin[2]/2, 2)
print("total point: ", len(xyz))
print("x_min:", x_min, "x_max:", x_max, "y_min:", y_min, "y_max:", y_max, "z_min:", z_min, "z_max:", z_max)

# # 顯示被voxel down的方格(觀察用可不打開)
# box_z = [[x_min+0.2, y_min, z_min+1],[x_min+0.4, y_min, z_min+1],[x_min+0.2, y_min+0.2, z_min+1],[x_min+0.2, y_min, z_min+1.2]]
# pcd_voxel = o3d.geometry.PointCloud()
# pcd_voxel.points=o3d.utility.Vector3dVector(box_z)
# box_voxel = pcd_voxel.get_axis_aligned_bounding_box()
# box_voxel.color = (0,1,0)

# 宣告降維後的pcd
pcd_custom = o3d.geometry.PointCloud()
point = []
color = []

voxel = 0.1
count = 1
```

```
# 最主要的voxel down函式，i,j,k分別代表x,y,z軸
for i in range(int(x_min*100), int(x_max*100), int(voxel*100)):
    for j in range(int(y_min*100), int(y_max*100), int(voxel*100)):
        for k in range(int(z_min*100), int(z_max*100), int(voxel*100)):
            # print("step:", str(count))
            print("Success: ", round(count/(margin[0]*margin[1]*margin[2])*(voxel**3)*100, 1), "%")
            count = count + 1

            #在x軸取box範圍
            box_x = xyz[np.where((xyz[:,0]>=(i/100)) & (xyz[:,0]<=(i/100+voxel)))]
            box_color_voxel = xyz_color[np.where((xyz[:,0]>=(i/100)) & (xyz[:,0]<=(i/100+voxel)))]
            # print("After voxel x: ", box_x.shape[0])

            #在y軸取box範圍
            box_y = box_x[np.where((box_x[:,1]>=(j/100)) & (box_x[:,1]<=(j/100+voxel)))]
            box_color_voxel = box_color_voxel[np.where((box_x[:,1]>=(j/100)) & (box_x[:,1]<=(j/100+voxel)))]
            # print("After voxel y: ",box_y.shape[0])

            #在z軸取box範圍
            box_z = box_y[np.where((box_y[:,2]>=(k/100)) & (box_y[:,2]<=(k/100+voxel)))]
            box_color_voxel = box_color_voxel[np.where((box_y[:,2]>=(k/100)) & (box_y[:,2]<=(k/100+voxel)))]
            # print("numbers of picked points: ",box_z.shape[0])

            #計算出現次數最多的顏色,並新增到point, color
            box_color_voxel = np.around(box_color_voxel, 2)
            unique, counts = np.unique(box_color_voxel, axis=0, return_counts=True)
            if counts != []:
                major_color = unique[counts.tolist().index(max(counts))]
                # print("major color:", major_color)
                point.append([(2*(i/100)+voxel)/2, (2*(j/100)+voxel)/2, (2*(k/100)+voxel)/2])
                color.append(major_color)

pcd_custom.points = o3d.utility.Vector3dVector(point)
pcd_custom.colors = o3d.utility.Vector3dVector(color)
```

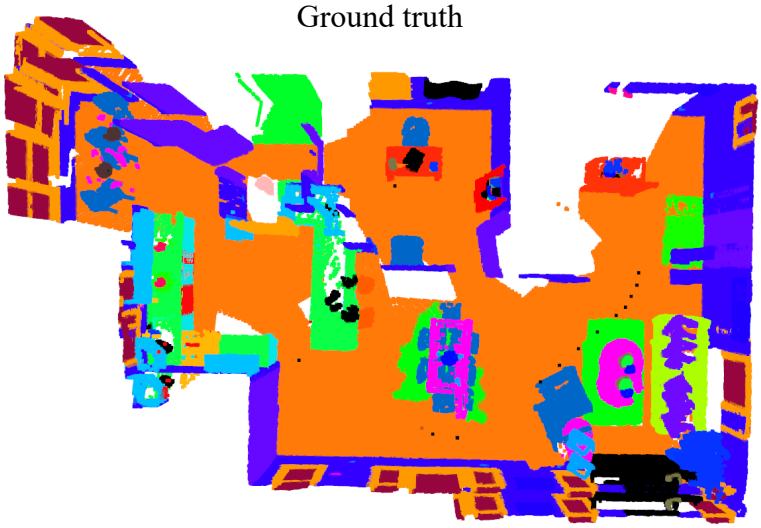
- 首先在最終疊合起來的 point cloud 資料中，將 point 和 color 資料分別存成陣列型態，以方便做運算
- 接著使用 bounding box 將整個點雲框起來，這個 box 的資料型態可以分別使用 get_center、get_extent 取出中心點座標及長寬高資訊。擁有這些資訊，即可計算出這個 box 的 x,y,z 範圍在哪，如我在程式碼中寫的式子。
- 利用這些 x,y,z 的上下限範圍，就可以切出每一塊 voxel，做降維的處理。這個部分使用三個迴圈執行所有需要被計算的 voxel，裡面則是演算法。
- 演算法中的每個步驟主要使用了 numpy 這個函式庫，來加速運算速度(當初全用 for 迴圈做，跑不動)。一開始先依序對 x,y,z 的 bound，使用 np.where 找出符合範圍內的點(在 where 裡面設上下限，會回傳 index)，依序做的目的，是為了減少這邊的計算量，實測過快速許多。這個部分 color 的陣列資料也必須跟著一起計算，不然會產生 color 的 index 跟 point 的找出的 index 不對稱的問題。
- 接著進行範圍內點雲的 color 數量計算，尋找出現最多次的顏色。這裡使用到 np.unique，這個函式可以將陣列中所有資料做數量計算，並回傳每組陣列(每個顏色)出現的次數，最後將出現次數最多的顏色給予 major_color 變數。進行 np.unique 之前，必須先對資料做四捨五入的動作，才能將幾近相同的顏色分在同一類(若不做會分類失敗)。
- 最後則是將這個顏色給予新宣告的點雲，同時也將這個 voxel 的中心座標新增進去，custom voxel down 完成。

B. Result and discussion

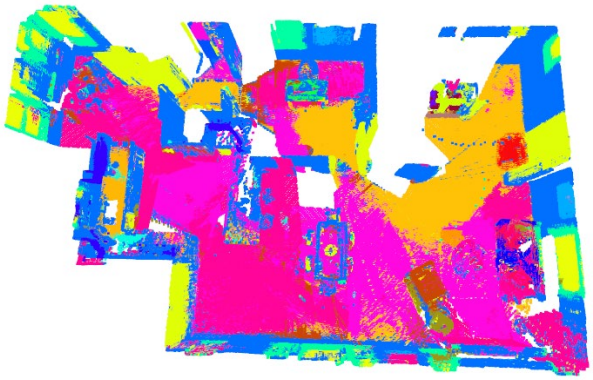
1. Result of your semantic map

(1) 1F

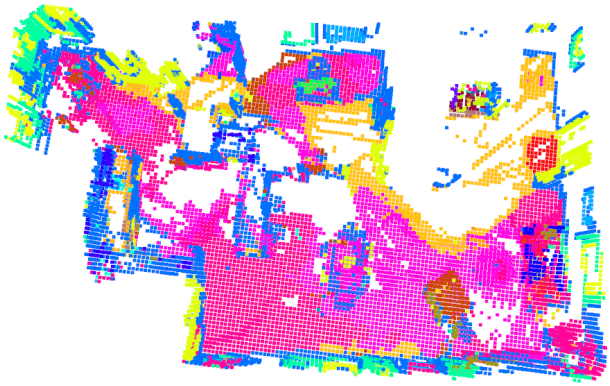
Ground truth



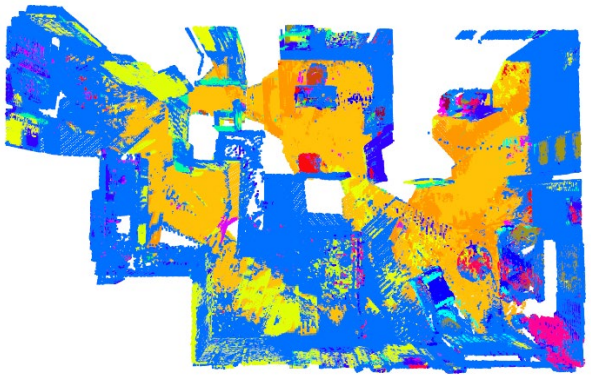
Trained on apartment0(no voxel down)



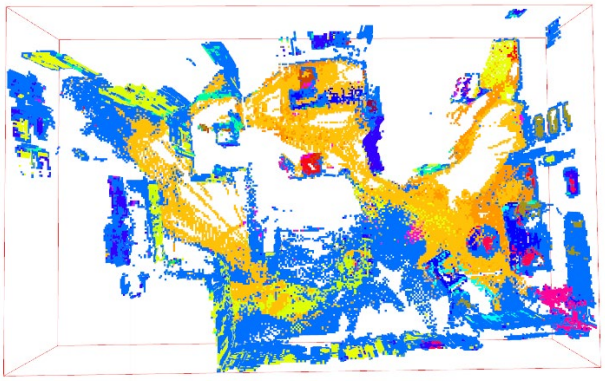
Trained on apartment0(voxel=0.1)



Trained on other scenes (no voxel down)

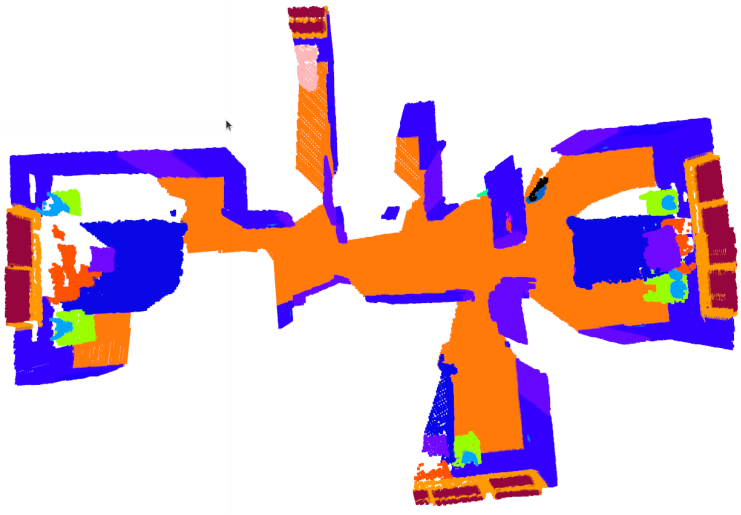


Trained on other scenes (voxel=0.05)

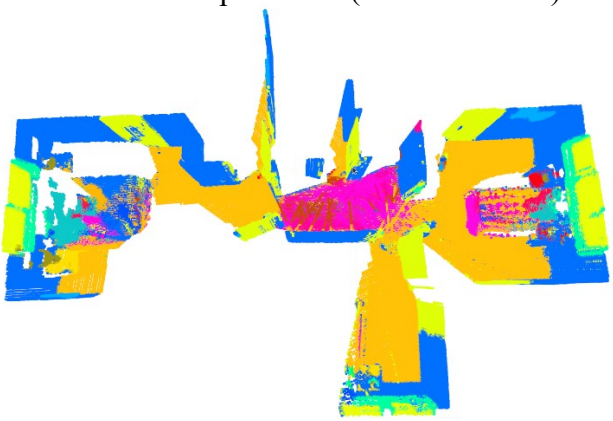


(2) 2F

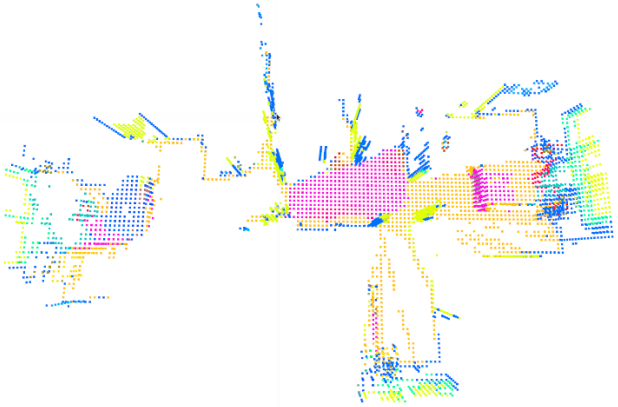
Ground truth



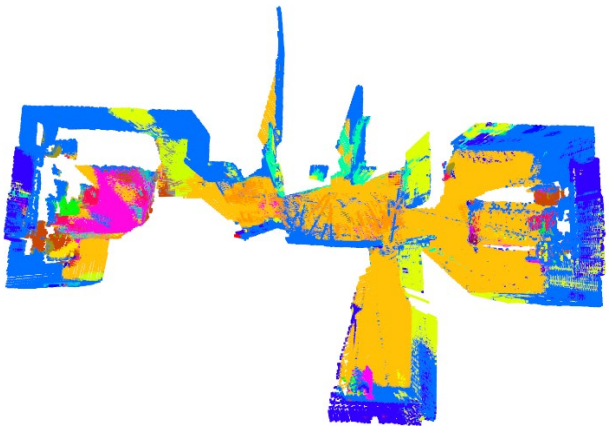
Trained on apartment0(no voxel down)



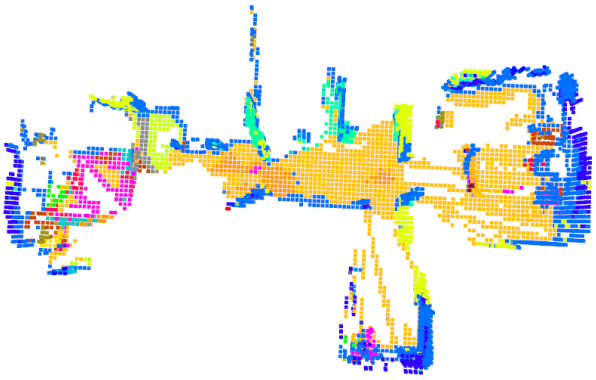
Trained on apartment0(voxel=0.1)



Trained on other scenes (no voxel down)



Trained on other scenes (voxel=0.1)



2. Discussion

- 在計算 mIOU 的時候，雖然把特定的那 49 種類別取出來做平均，但數據還是非常低。這個部分有可能受電腦性能跟 model 參數影響，但另一方面我認為可能是我自己的計算仍有 bug。當我在 print 每一個 IOU 時，發現這 49 種類別中，有非常多數值仍是 0，這些 0 仍然也被拿來取平均，而這有可能是導致 mIOU 數值極低的原因。IOU 為 0，代表準確率真的是 0，還是這個類別並沒有出在場景中，值得討論，或許應該也把 0 的數據拿掉。
- Custom voxel down 的部分，我一開始先想好邏輯後，全部使用 for 迴圈完成，執行後發現計算量太大，執行太久或是出現 segmentation fault 的錯誤(記憶體爆掉)，只有當 voxel 設非常大的時候才能在幾小時內跑完。後來詢問助教後嘗試使用 numpy 做做看，使用 library 的執行效率勢必比自己寫 for 還快，研究 numpy 許久後試著修改原本的程式，發現效率確實高出非常多，程式行數也變得非常少(原本 100 多行)，效果也不會太差。不過由於將整個樓層的地圖做 voxel down 點雲範圍仍非常龐大，voxel 設 0.1 仍須跑至少 20 分鐘，若為 0.05

同樣可能出現 segmentation fault 或電腦當機的狀況，因此報告中的 voxel down 點雲看起來粒粒分明。也許還是有簡化的方法可以加速這個演算法的運算，這個部分仍然值得討論。