智慧感知與決策

311512064 鄧書桓

Task1:

i. Code:

1. Tranning segmentation model

Floor1:

Model of **other scene**:

```
DATASET:
  imgMaxSize: 1000
  imgSizes: (300, 375, 450, 525, 600)
  list_train: ./data/others_training.odgt
  list_val: ./data/data_for_reconstruct.odgt
  num_class: 101
  padding_constant: 8
  random_flip: True
  root_dataset: ""
  segm_downsampling_rate: 8
DIR: ckpt/others
MODEL:
  arch_decoder: ppm_deepsup
  arch_encoder: resnet50dilated
  fc_dim: 2048
  weights_decoder:
  weights_encoder:
TEST:
  batch_size: 1
  checkpoint: epoch_20.pth
  result: ./
TRAIN:
  batch_size_per_gpu: 2
  beta1: 0.9
  deep_sup_scale: 0.4
  disp_iter: 20
  epoch_iters: 650
  fix_bn: False
  lr_decoder: 0.02
  lr_encoder: 0.02
```

```
  lr_pow: 0.9
  num_epoch: 20
  optim: SGD
  seed: 304
  start_epoch: 0
  weight_decay: 0.0001
  workers: 16
VAL:
  batch_size: 1
  checkpoint: epoch_20.pth
  visualize: True
```

我這邊使用 ade20k-resnet50dilated-ppm_deepsup 作為 other scenes 訓練的 model，dataset 部分使用 1300 張照片，其中平均分配在其餘房間(總共 10 個個 130 張)，這邊嘗試過使用較少的房間，但得出來的結果很差。另外，visualize 需要更改為 True 才能將照片存下。

使用的參數如上，主要調整的是 batch_size 為 2， epoch_iter 為 650，這邊是根據相乘為 dataset 大小而選定的，其餘的超參數越調效果越差，故最後用一開始給定的。

MIOU:

這邊的設計為先讀取 excel 文件內的特定資料，並根據給定的 49 個 calss(存在 iou list 中)去做資料讀取，並將讀取到的資料寫入 miou，最後再將 miou 除以 lenth(miou)已得到想要的答案。詳細的做法在下面的 code 中表示:

```
184     miou = []
185     # get the data from exel, data_only wont get the formula in exel
186     wb = openpyxl.load_workbook('apartment0_classes.xlsx', data_only=True)
187     # get sheet1's value
188     s1 = wb['Sheet1']
189     # the data only at 2~50
190     for i in range(49):
191         # get the first colum sheet's value(first value is label)
192         label = s1.cell(i+2, 1).value
193         # add label into miou(original miou has 101 classes, we only need 49 classes)
194         miou.append(iou[int(label)])
195     wb.save('apartment0_classes.xlsx')
196     # 將各個iou除以49
197     print('Mean IoU: {:.4f}, Accuracy: {:.2f}%'
198           .format(sum(miou)/len(miou), acc_meter.average()*100))
199
```

Validation result:

```
[Eval Summary]:
Mean IoU: 0.0294, Accuracy: 40.11%
Evaluation Done!
100%|                                                  | 117/117 [01:12<00:00,  1.60it/s]
(habitat) frank@frank-System-Product-Name:~/PD/work2/semantic-segmentation-pytor
```

Model of **apartment0**:

```yaml
DATASET:
  imgMaxSize: 1000
  imgSizes: (300, 375, 450, 525, 600)
  list_train: ./data/custom_training.odgt
  list_val: ./data/data_for_reconstruct.odgt
  num_class: 101
  padding_constant: 8
  random_flip: True
  root_dataset: ""
  segm_downsampling_rate: 8
DIR: ckpt/apartment0
MODEL:
  arch_decoder: ppm_deepsup
  arch_encoder: resnet50dilated
  fc_dim: 2048
  weights_decoder:
  weights_encoder:
TEST:
  batch_size: 1
  checkpoint: epoch_20.pth
  result: ./
TRAIN:
  batch_size_per_gpu: 2
  beta1: 0.9
  deep_sup_scale: 0.4
  disp_iter: 20
  epoch_iters: 650
  fix_bn: False
  lr_decoder: 0.02
  lr_encoder: 0.02
  lr_pow: 0.9
  num_epoch: 20
  optim: SGD
  seed: 304
  start_epoch: 0
  weight_decay: 0.0001
  workers: 16
```

```
VAL:
  batch_size: 1
  checkpoint: epoch_20.pth
  visualize: True
```

我這邊使用 ade20k-resnet50dilated-ppm_deepsup 作為 apartment0 訓練的 model，dataset 部分使用 1300 張照片，目的為和 other scene 數據集的數量相同，這邊嘗試過使用較少的房間，但得出來的結果很差。

使用的參數如上，主要調整的是 batch_size 為 2， epoch_iter 為 650，這邊是根據相乘為 dataset 大小而選定的，其餘的超參數越調效果越差，故最後用一開始給定的。

MIOU:

此處的 MIOU 與上述的相同，就不再多做贅述。

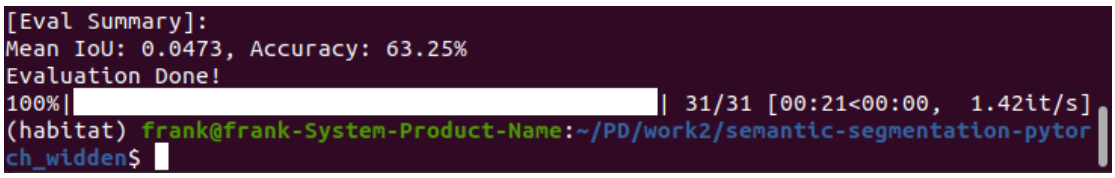
Validation result:

這邊得到的 Accuracy 與 others scene 相比，有著較高的準確率，我認為是因為這邊的 dataset 本就是由 apartment0 收集的，故使用相同環境下的照片經過 model 得到的語意分割圖的結果較佳。

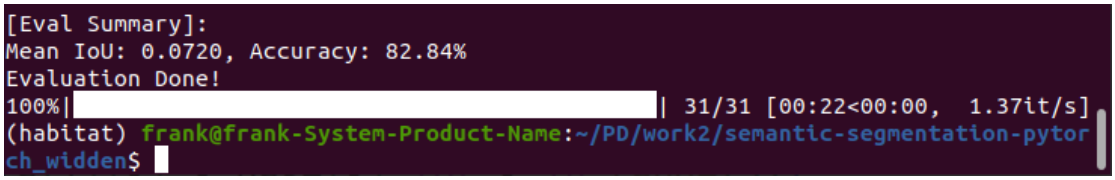
Floor2:

這邊的 model 與 IMOU 的設計都與 floor1 相同，這邊就不做更多的贅述。此處僅表示 IMOU 在 floor2 的效果。

IMOU of others:



IMOU of apartment0:



這邊得到的 Accuracy 與 others scene 相比，有著較高的準確率，我認為是因為這邊的 dataset 本就是由 apartment0 收集的，故使用相同環境下的照片經過 model 得到的語意分割圖的結果較佳。

2.3D semantic map reconstruction:

這邊設計為只要將路徑更改至所要重建的語意照片資料夾，即可自動生成點雲、自動重建與使用 custom voxel down 將最後的重建點雲圖降維。

而 custom voxel down 的部分是我與實驗室的朋友一起討論想出來的，所以會有點相似。一開始是使用 open3D 內的 get_axis_aligned_bounding_box()函式來獲取重建後地圖的 bounding box 與其參數(如質心位置與長寬高)。接著利用利用質心加減長高一半的長度來獲得 bounding box 位於 x, y, z 軸上的起始點與終點。

接著根據 voxel 的大小分別在 x,y,z 軸上取質，想法如下:

1. 先 slice 出 x 軸上的第一個 voxel，因此會得到 x 軸上第一個 voxel 且 y, z 軸都沒被 slice 的部分。
2. 再對 y 軸 slice 出第一個 voxel，會得到 x、y 軸第一個 voxel 且整個 z 軸的部分。
3. 再對 z 軸 slice 出第一個 voxel，這樣將會得到 x、y、z 軸上第一個 voxel 的正方形 box。
4. 接著根據上面所得的部分去 slice 出該位置的 color，可以得到 x、y、z 軸上第一個 voxel 中的 color。
5. 之後計算此 voxel 中哪個 color 最多，並將此顏色變為這個 voxel 的顏色。
6. 將此顯示於此 voxel 的正中央(根據 voxel 的長寬高與質心來實現)。
7. 最後將 x、y、z 軸上各個 voxel 都執行上述步驟(使用 3 個 for 迴圈達成)就能得到 custom_voxel_down 想要的結果。

詳細的做法在下面的 code 中表示:

```
# create bounding box for custom voxel down
def custom_voxel_down(final_pcd, voxel):
    # initialize box and other list
    custom_point = []
    custom_color = []
    num = 1
    box = final_pcd.get_axis_aligned_bounding_box()
    box.color = (1, 1, 1)
    # get array (num_of_point * 3)
    point_all = np.asarray(final_pcd.points)
    color_all = np.asarray(final_pcd.colors)
    # get center, extent of box (box is bounding final pcd)
    box_center = box.get_center()
    box_margin = box.get_extent()
    # get the start, end of x, y, z
```

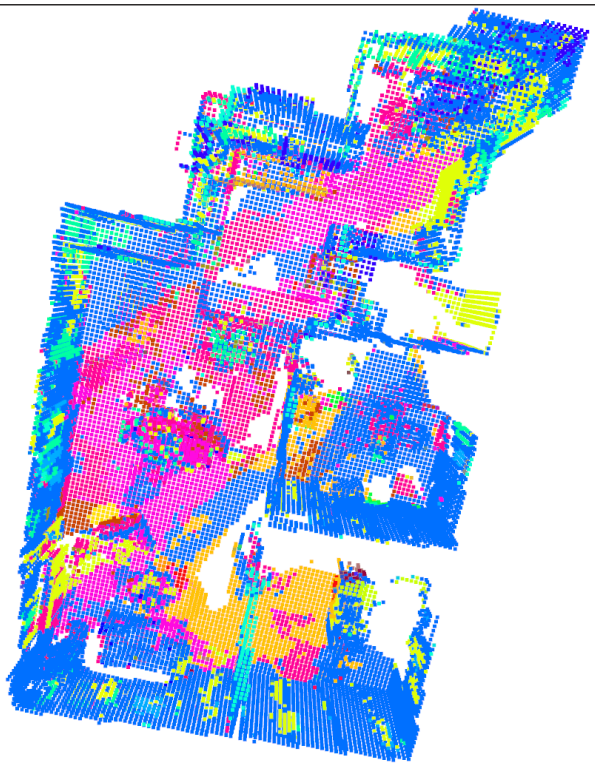
```
    x_start = round(box_center[0] - box_margin[0]/2, 5)
    x_end = round(box_center[0] + box_margin[0]/2, 5)
    y_start = round(box_center[1] - box_margin[1]/2, 5)
    y_end = round(box_center[1] + box_margin[1]/2, 5)
    z_start = round(box_center[2] - box_margin[2]/2, 5)
    z_end = round(box_center[2] + box_margin[2]/2, 5)
    voxel_down_pcd = o3d.geometry.PointCloud()
    total_iteration = int((x_end-x_start)/voxel) * \
        int((y_end-y_start)/voxel) * int((z_end-z_start)/voxel)
    for i in range(int(x_start * (10**5)), int(x_end * (10**5)),
int(voxel * (10**5))):
        for j in range(int(y_start * (10**5)), int(y_end * (10**5)),
int(voxel * (10**5))):
            for k in range(int(z_start * (10**5)), int(z_end *
(10**5)), int(voxel * (10**5))):
                print("iteration for {} ........{}%:".format(
                    str(num), round(num/total_iteration * 100, 2)))
                num = num + 1

                # slice the box in x direction
                slice_x = point_all[np.where((point_all[:, 0] >=
(i/(10**5))) &
                                             (point_all[:, 0] <=
(i/(10**5)+voxel)))]
                # slice the remaining box in y direction
                slice_y = slice_x[np.where(
                    (slice_x[:, 1] >= (j/(10**5))) & (slice_x[:, 1]
<= (j/(10**5)+voxel)))]
                # slice the remaining box in z direction and get the
corresponding color
                slice_z = slice_y[np.where(
                    (slice_y[:, 2] >= (k/(10**5))) & (slice_y[:, 2]
<= (k/(10**5)+voxel)))]

                # get the color with corresponding slices
                # get the color in x slice
                slice_color = color_all[np.where(
```

```
                    (point_all[:, 0] >= (i/(10**5))) & (point_all[:,
0] <= (i/(10**5)+voxel)))]
                # get the color in xy slice
                slice_color = slice_color[np.where(
                    (slice_x[:, 1] >= (j/(10**5))) & (slice_x[:, 1]
<= (j/(10**5)+voxel)))]
                # get the color in xyz slice
                slice_color = slice_color[np.where(
                    (slice_y[:, 2] >= (k/(10**5))) & (slice_y[:, 2]
<= (k/(10**5)+voxel)))]

                #計算出現次數最多的顏色,並新增到 point, color
                # 若沒做四捨五入,會有太多種顏色
                slice_color = np.around(slice_color, 2)
                # return the numbers of each different data
                unique, counts = np.unique(
                    slice_color, axis=0, return_counts=True)
                if counts != []:
                    # find the max number of color and use index to
get it's value
                    major_color =
unique[counts.tolist().index(max(counts))]
                    # get the center of that voxel
                    custom_point.append(
                        [(2*(i/(10**5))+voxel)/2,
(2*(j/(10**5))+voxel)/2, (2*(k/(10**5))+voxel)/2])
                    custom_color.append(major_color)

    voxel_down_pcd.points = o3d.utility.Vector3dVector(custom_point)
    voxel_down_pcd.colors = o3d.utility.Vector3dVector(custom_color)
    return voxel_down_pcd, box
```

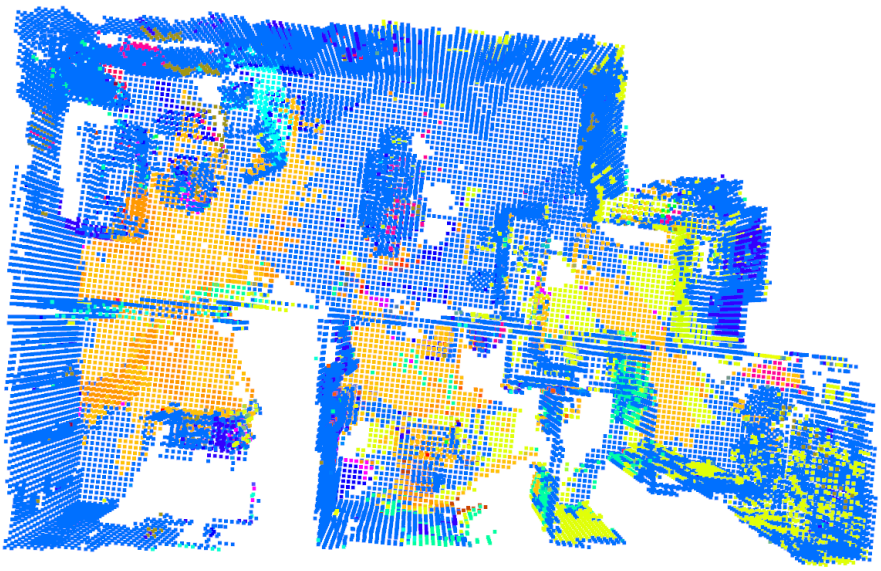
ii.Result and Discussion

Floor1:

Train on other scene:

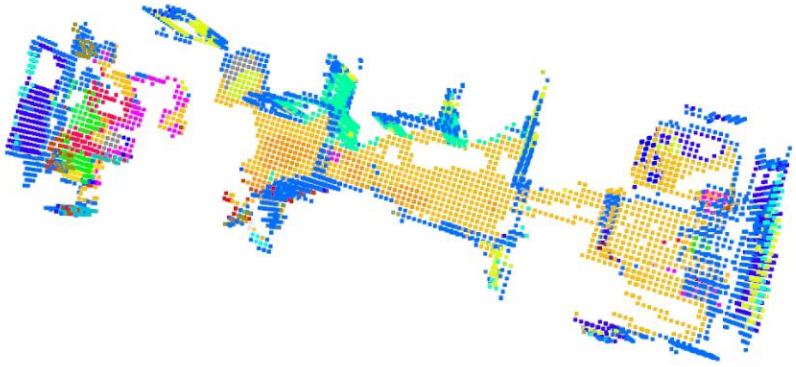


Train on apartment0:

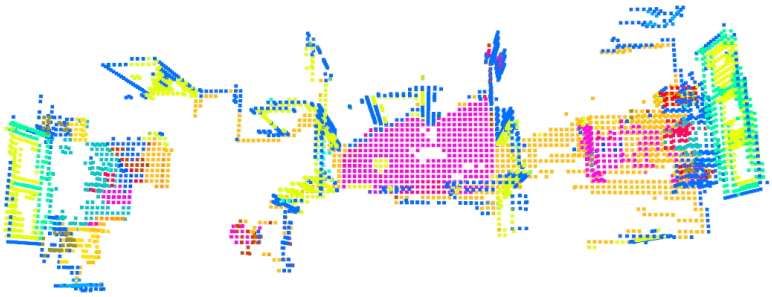


Floor2:

Train on other scene:



Train on apartment0:



雖然已經調整過許多超參數與試過多種 model，但是上完色重建的效果都很差，所以只好推選這組重建效果較好的放在此報告上，我認為是因為本身的顯卡較破爛，訓練出來的成果很差且超級耗時，才會導致這樣。

另外，因為電腦的主機板年代較久遠，因此無法承受較高的 custum_voxel_down，也就是無法使用較小的 voxel 去做降維，我的電腦能接受最低的 voxel 是 0.1，也因此成果圖的顆粒感會看起來這麼重。

除此之外，由於 floor2 的場地較複雜，若是使用太多照片去做重建，我的電腦會不堪負荷而自動關機，因此 floor2 的重建效果較差，這邊僅用 32 張圖去做重建，希望助教見諒。