Supplemental Materials: Self-supervised Pre-training for Semantic Segmentation in an Indoor Scene

Sulabh Shrestha, Yimeng Li, Jana Košecka
George Mason University

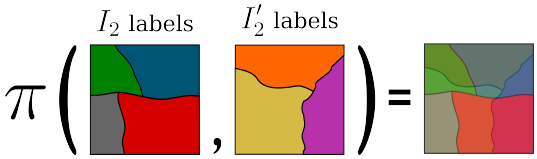

Figure 1. Example of pairing class-agnostic regions using pairing function. Each color represents a unique region. The output of element-wise pairing of input label images is another label image where each unique pair of input labels produces a unique output region label.

1. Class Agnostic Region Overlap

Let I_2 and I_2' be a pair of images/matrices with region labels $\{b_j\}_{j=1}^{R_2}$ and $\{a_i\}_{i=1}^{R_1}$ respectively. Without loss of generality, let N be the total number of elements in I_2, same as in I_2'. For each pair of region labels (a_i, b_j), the task is to find the (pixel) intersection over union (IoU) of their regions.

1.1. Brute-Force Algorithm

As shown in the Brute-Force Algorithm 1, the algorithm iterates over each unique label $b_j \in I_2$ and finds its boolean mask $\bar{b}_j$ that represents locations/mask of elements in I_2 that have label b_j. The boolean mask calculation takes $O(N)$ time per label b_j. For each of these labels, the boolean mask for each label a_i in I_2' needs to be calculated similarly. Element-wise AND operation of these masks gives the mask of intersecting elements. Overall, the brute force method takes $O((R_1 * N) * (R_2 * N)) = O(R_1 R_2 N^2)$ time. Alternatively, the mask for each R_1 and R_2 region labels could be saved once. However, this requires saving a mask that has same size as the input image $R_1 + R_2$ times i.e. $O((R_1 * + R_2) * N)$ space complexity which can be excessive when the input images have high dimensions while the time complexity will still be $O(R_1 R_2)$. In worse case, where every pixel is a separate region this will still be $O(N^2)$.

1.2. Preliminary: Pairing Function

A pairing function $\pi : \mathbb{Z}^* \times \mathbb{Z}^* \rightarrow \mathbb{Z}^*$ is a *reversible bijective* function that maps non-negative integers (x, y) to a unique integer z. We use one of such pairing functions, namely, *Cantor pairing* function which is given by equation (1).

$$\pi(x, y) = z \leftarrow 0.5(x^2 + 2xy + 3x + y + y^2) \quad (1)$$

The inverse of the pairing function, π^{-1} takes the number z and outputs the input pair (x, y) that produced this number. For *Cantor pairing* function, the inverse is given by equations (2)

$$\begin{aligned} c &\leftarrow \lfloor 0.5 * (\sqrt{(8 * z + 1)} - 1) \rfloor \\ x &\leftarrow z - (0.5 * c * (c + 1)) \\ y &\leftarrow c - x \\ \pi^{-1}(z) &\leftarrow (x, y) \end{aligned} \quad (2)$$

One of the desired property is that, for *Cantor pairing* function, both π and π^{-1} takes $O(1)$ time to compute. Another valuable property of any pairing function is that the mapping of unique pair is also unique. Some examples of the *Cantor pairing* of numbers are shown below:

- $\pi(1, 1) = 4$ and $\pi^{-1}(4) = (1, 1)$
- $\pi(1, 2) = 7$ and $\pi^{-1}(7) = (1, 2)$
- $\pi(2, 1) = 8$ and $\pi^{-1}(8) = (2, 1)$
- $\pi(2, 2) = 12$ and $\pi^{-1}(12) = (2, 2)$

Pairing $(1, 2)$ outputs 7 which means no other input pair produces 7. Similarly, the pairing function is not commutative as can be seen from the example that the pairs $(1, 2)$ and $(2, 1)$ produce different outputs.

Hence, the same unique pair of inputs (region labels) is always mapped to the same unique number. A toy example of element-wise pairing of labels from I_2 and I_2' is shown in Figure 1 where each color represents a unique label. The output of this element-wise pairing is equivalent to having

Algorithm 1 Brute Force algorithm for Class Agnostic Region overlap calculation

iouMap $\leftarrow \{\}$ ▷ HashMap to store IoU of labels/regions
for i in R_1 **do**
  $\bar{a}_i \leftarrow (I_2' == a_i)$ ▷ Find boolean mask of a_i in I_2 in O(N)
  for j in R_2 **do**
    $\bar{b}_j \leftarrow (I_2 == b_j)$ ▷ Find boolean mask of a_j in I_2' in O(N)
    intersect = AND($\bar{a}_i$, $\bar{b}_j$)
    union $\leftarrow$ OR($\bar{a}_i$, $\bar{b}_j$) - intersect
    iou $\leftarrow$ intersect / union
    iouMap[(a_i, b_j)] $\leftarrow$ iou
  end for
end for

Algorithm 2 Optimized algorithm for Class Agnostic Region overlap calculation

iouMap $\leftarrow \{\}$ ▷ HashMap to store IoU of labels/regions
$cB \leftarrow \{\}$ ▷ HashMap to store region labels and their pixel counts in I_2
$cA \leftarrow \{\}$ ▷ HashMap to store region labels and their pixel counts in I_2'
$cP \leftarrow \{\}$ ▷ HashMap to store region labels and their pixel counts from paring I_2 and I_2'
for e in N **do** ▷ Iterate over each element in I_2 and I_2' and pair them element-wise in O(N)
  $b \leftarrow I_2[e]$
  $a \leftarrow I_2'[e]$
  $p \leftarrow \pi(a, b)$ ▷ Cantor-pairing
  $cB[b]$ += 1
  $cA[a]$ += 1
  $cP[p]$ += 1
end for
for p in cP **do** ▷ Iterate over unique pairs in cP in O($||cP|| \leq N$)
  $(a, b) \leftarrow \pi^{-1}(p)$
  intersect $\leftarrow cP[p]$
  union $\leftarrow cA[a] + cB[b] - cP[p]$
  iou $\leftarrow$ intersect / union
  iouMap[(a, b)] $\leftarrow$ iou
end for

Algorithm 3 Class Agnostic Region Overlap

$R_1 \leftarrow$ Unique regions and their pixel counts in I_1
$R_2 \leftarrow$ Unique regions and their pixel counts in I_2
$P \leftarrow$ Element wise pairing between I_1 and I_2 using π
$R_p \leftarrow$ Unique regions and their counts in P
for r_p in R_p **do**
  $r_1, r_2 \leftarrow \pi^{-1}(r_p)$
  intersection $\leftarrow P[r_p]$
  union $\leftarrow R_1[r_1] + R_2[r_2] -$ intersection
  pixel-IoU$(r_1, r_2) \leftarrow$ intersection / union
end for

another image, shown on the right-most side of Figure 1, where a unique pairing of labels gives a unique label. For example, the red region on I_2 outputs two unique region labels, one when paired with yellow region and another with purple region in I_2' resulting in an orange label and a red label in the output image respectively.

1.3. Optimized Algorithm

The Optimized Algorithm 2 utilizes *Cantor pairing* function π from Section 1.2 to calculate the IoU of overlapping regions. Although, any pairing function takes $O(1)$ time. The algorithm first iterates over each index e of the N elements in I_2 and I_2' and element-wise pairs them with each other. The count of their respective labels are updated in a HashMap for easier access later. This process takes $O(N)$ time. A unique pair of inputs give the same unique output as explained in Section 1.2. Once all elements have been paired, the unique paired elements (stored in cP in the algorithm), give the number of all overlapping regions. So, by default, non-overlapping regions are omitted from the pairing unlike in the brute force algorithm from Section

```
def find_overlap_iou(img2,img2_from_1):
    # Calc pixel counts in both images O(N)
    A, cA = torch.unique(img2_from_1,
        return_counts=True)
    B, cB = torch.unique(img2,
        return_counts=True)

    # Pair labels using pairing function O(N)
    pairs = pair_func(img1, img2)

    # Calc pixel counts in paired image
    P, cP = torch.unique(pairs, return_counts=
    True)

    ious = {}
    # Iterate over paired regions O(len(P))
    for p in cP:
        a, b = inv_pair_func(p)
        intersect = cP[P == p]
        na, nb = cA[A == a], cB[B == b]
        union = na + nb  - intersect
        iou = intersect / union
        iou[(a, b)] = iou
    return ious
```

Figure 2. Pytorch Implementation of Optimized algorithm for Class Agnostic Region overlap calculation

1.1. Finally, iterating over the unique paired elements (cP) gives us the count of intersecting pixels while π^{-1} gives the original region labels from I_2 and I_2'. From this, the IoU can be easily calculated. The time taken to iterate over the unique paired elements is proportional to the number of overlapping regions which is at most N *i.e.* when each region is only a single element for both I_2 and I_2'. Therefore, the total computation time for this algorithm is $O(N)$.

The PyTorch implementation of the algorithm is given in Figure 2. Although it can be further optimized by creating a dictionary of counts for faster access similar to Algorithm 2, it is omitted in the code for brevity and simplicity.

2. Dataset Generation

2.1. Image generation

Using the habitat simulator [4], we sample and extract images along with their depths, instance segmentation and semantic segmentation from multiple locations in each scene. A grid is formed by horizontal lines parallel to the x-axis and z-axis. The images are captured from this grid *i.e.* from the intersections of the lines. The camera height y is kept fixed. From each navigable location in this grid, images are extracted by rotating the camera around the vertical y-axis separated by 45 degrees from each other resulting in 8 images per location.

Replica The height of the camera is set to -1.0 unit. The parallel lines that form the grid are 0.5 units apart from each other. This results in approximately 300 images in the scene.

Habitat The height of the camera is set to 0.0241 units. The parallel lines that form the grid are 0.5 units apart from each other. This results in approximately 600 images in the scene.

AVD AVD scenes contain images already sampled in a format very much similar to ours. Hence, the image generation step is not required.

2.2. View-Pair Sampling

It is possible to pair each image I_1 in the scene with every image I_2, including itself, to get the image pairs (I_1, I_2). The number of possible pairs is quadratic with respect to the number of images existing in a scene. For example, in Replica, which has the smallest scene with approximately 300 images, this results in 300 x 300 = 90K image pairs. In the AVD scene with approximately 2400 images, the possible image pairs reach 5M. Thus, it is computationally inefficient to use every possible image pairs. Furthermore, not all image pairs contain images that overlap with each other. Therefore, we select the image pairs that satisfy the following criteria:

- The image pairs must be within 3.0 units of each other
- the angle between the viewing directions of the image pairs must be within 90 degrees

We calculate IoU of the image pairs that satisfy both these constraints. Finally, as mentioned in the paper, for the experiments, we select image-pairs (I_1, I_2) whose IoU lies in the range $[IoU_l, IoU_h]$. To ease the training process, we pre-compute these values such that this process only needs to be done once per scene.

3. Pre-Training Details

We use a batch size of 16 image pairs. In each batch, we sample $B_{pix} = 81920$ batch of pixel-pairs for $\mathcal{L}_{pix}$ and $B_{reg} = 81920$ for $\mathcal{L}_{reg}$ while B_{pool} is left unbound to include all region pairs with IoU overlap above $IoU_r = 0.2$. To generate regions, we use the efficient graph-based segmentation method [2] with $scale = 85$ and $\sigma = 2000$. We obtain this value by generating segments with different hyper-parameters and empirically observing the segments on a handful of images from the Replica dataset. We use this default value for all other datasets.

4. Class Mappings

Classes are mapped from other Replica and AVD to ADE20K [6, 7] in a many to one fashion: multiple classes from Replica can be mapped to a single class in ADE20K. If any class is ambiguous such that it can be mapped to more

than one class in ADE20K, it is ignored. ADE20K contains 150 classes. Classes from HM3D [3] are kept intact.

Replica to ADE20K Among the 101 classes in Replica [5], 52 classes are mapped to ADE20K. The exact mapping is shown in Table 1.

AVD to ADE20K Among the 87 classes in AVD [1], 47 classes are mapped to ADE20K. The exact mapping is shown in Table 2.

5. More Examples of Projection

Both RGB and label images are projected from view I_1 to view I_2 to get image I_2'. In Replica, most pixels in the ceiling of the scene do not have depth values. Similarly, HM3D also has missing depth values for 3D poins which lie outside the indoors scene. In AVD, many pixels have missing or erroneous depth values. Hence, some pixels in I_2' are invalid and such pixels are not used during pre-training. Examples from Replica are shown in Figure 3 and from AVD are shown in Figure 4.

6. More Qualitative Results

More qualitative results on the Replica scene, the HM3D scene and the AVD scene are shown in Figures 5, 6 and 7 respectively. The colors for their respective classes have been overlayed on the RGB images to get the segmentation map. Pixels with classes which were not mapped from Replica/AVD to ADE20K have their segmentation shown in black colors. For AVD, we visualize the results from the model fine-tuned on AVD-hard.

	Replica		ADE20K	
s.no.	id	name	id	name
1	2	base-cabinet	11	cabinet
2	3	basket	113	basket;hand
3	4	bathtub	38	bathtub;bat
4	7	bed	8	bed
5	8	bench	70	bench
6	9	bike	128	bicycle;bik
7	10	bin	139	ashcan;tras
8	11	blanket	132	blanket;cov
9	12	blinds	64	blind;scree
10	13	book	68	book
11	14	bottle	99	bottle
12	15	box	42	box
13	18	cabinet	11	cabinet
14	20	chair	20	chair
15	22	clock	149	clock
16	24	clothing	93	apparel;wea
17	29	cushion	40	cushion
18	30	curtain	19	curtain;dra
19	31	ceiling	6	ceiling
20	33	countertop	71	countertop
21	34	desk	34	desk
22	36	desktop-comp	75	computer;co
23	37	door	15	door;double
24	40	floor	4	floor;floor
25	43	handrail	96	bannister;b
26	44	indoor-plant	18	plant;flora
27	47	lamp	37	lamp
28	50	mat	29	rug;carpet;
29	51	microwave	125	microwave;m
30	59	picture	23	painting;pi
31	60	pillar	43	column;pill
32	61	pillow	58	pillow
33	63	plant-stand	126	pot;flowerp
34	64	plate	143	plate
35	67	refrigerator	51	refrigerato
36	69	scarf	93	apparel;wea
37	70	sculpture	133	sculpture
38	71	shelf	25	shelf
39	73	shower-stall	146	shower
40	74	sink	48	sink
41	76	sofa	24	sofa;couch;
42	78	stool	111	stool
43	80	table	16	table
44	84	toilet	66	toilet;can;
45	86	towel	82	towel
46	87	tv-screen	90	television;
47	91	vase	136	vase
48	93	wall	1	wall
49	96	wardrobe	36	wardrobe;cl
50	97	window	9	windowpane;
51	98	rug	29	rug;carpet;
52	100	bag	116	bag

Table 1. Mapping of classes from Replica to ADE20K

	AVD		ADE20K	
s.no.	id	name	id	name
1	2	aunt_jemima	99	bottle
2	4	cholula_chi	99	bottle
3	5	coca_cola_	99	bottle
4	6	crest_compl	99	bottle
5	7	crystal_hot	99	bottle
6	10	honey_bunch	42	box
7	11	honey_bunch	42	box
8	13	listerine_g	99	bottle
9	15	nature_vall	42	box
10	16	nutrigrain_	42	box
11	17	pepto_bismo	99	bottle
12	20	quaker_chew	42	box
13	22	softsoap_cl	99	bottle
14	23	softsoap_go	99	bottle
15	24	softsoap_wh	99	bottle
16	25	spongebob_s	42	box
17	26	tapatio_hot	99	bottle
18	27	vo5_tea_th	99	bottle
19	28	nature_vall	42	box
20	29	nature_vall	42	box
21	30	nature_vall	42	box
22	31	nature_vall	42	box
23	32	paper_plate	143	plate
24	35	wall	1	wall
25	36	floor	4	floor;floor
26	37	door	15	door;double
27	38	ceiling	6	ceiling
28	39	couch	24	sofa;couch;
29	40	pillow	58	pillow
30	42	microwave	125	microwave;m
31	43	oven	119	oven
32	44	sink	48	sink
33	45	refridgerato	51	refrigerato
34	46	dining-table	16	table
35	47	chair	20	chair
36	48	tv	90	television;
37	50	potted-plant	18	plant;flora
38	51	desk	34	desk
39	52	bed	8	bed
40	53	toilet	66	toilet;can;
41	60	book	68	book
42	61	clock	149	clock
43	62	vase	136	vase
44	64	teddy-bear	109	plaything;t
45	68	backpack	116	bag
46	80	plate	143	plate
47	81	bottle	99	bottle

Table 2. Mapping of classes from AVD to ADE20K

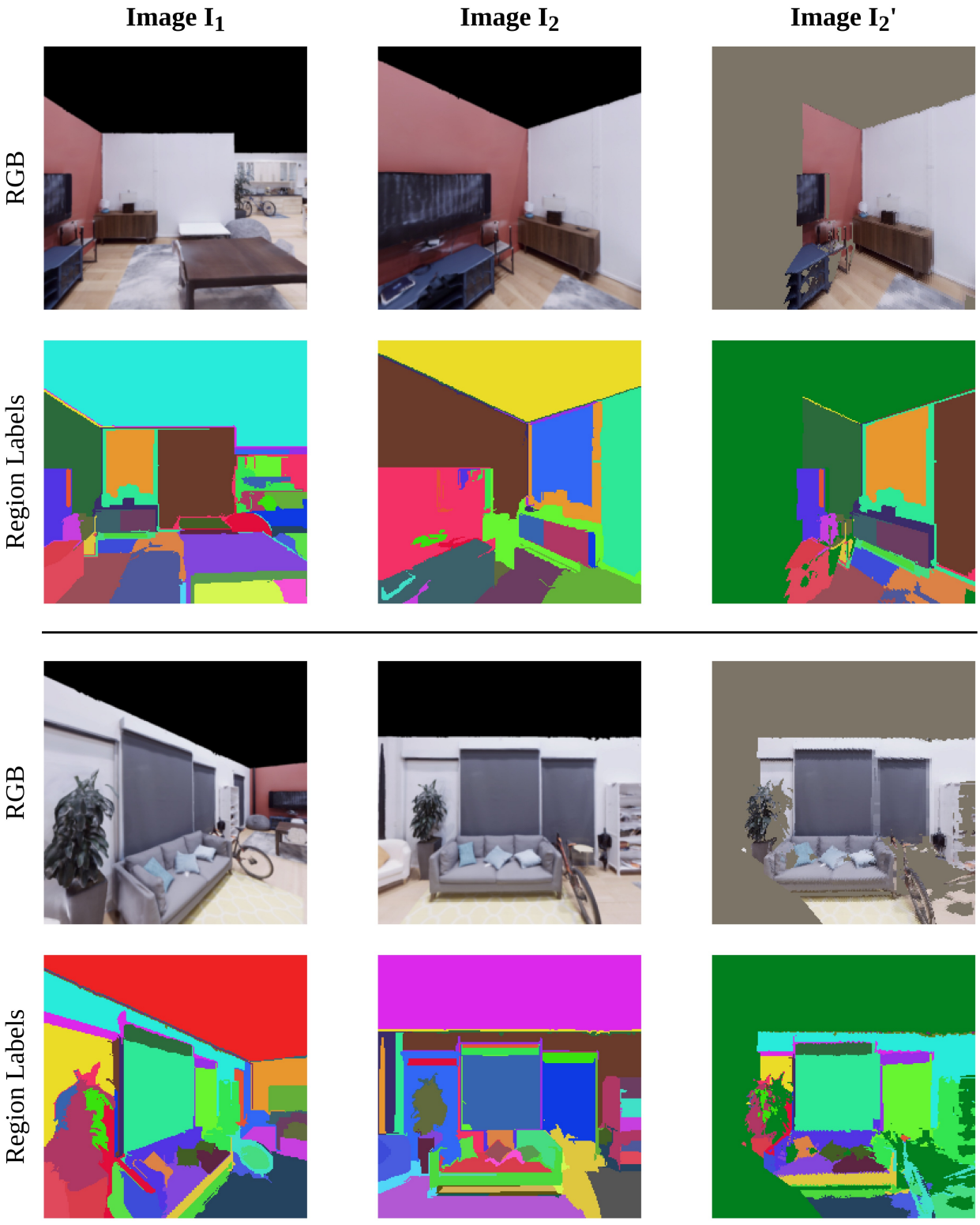

Figure 3. Examples of projection from I_1 to I_2 resulting in I_2' inside Replica scene. Some pixels in I_2' are invalid because of occlusion and missing depth. Such pixels share the same label and color in the visualization in I_2' and are not used during pre-training.

Figure 4. Examples of projection from I_1 to I_2 resulting in I_2' inside AVD scene. Some pixels in I_2' are invalid because of occlusion and missing or erroneous depth. Such pixels share the same label and color in the visualization in I_2' and are not used during pre-training.

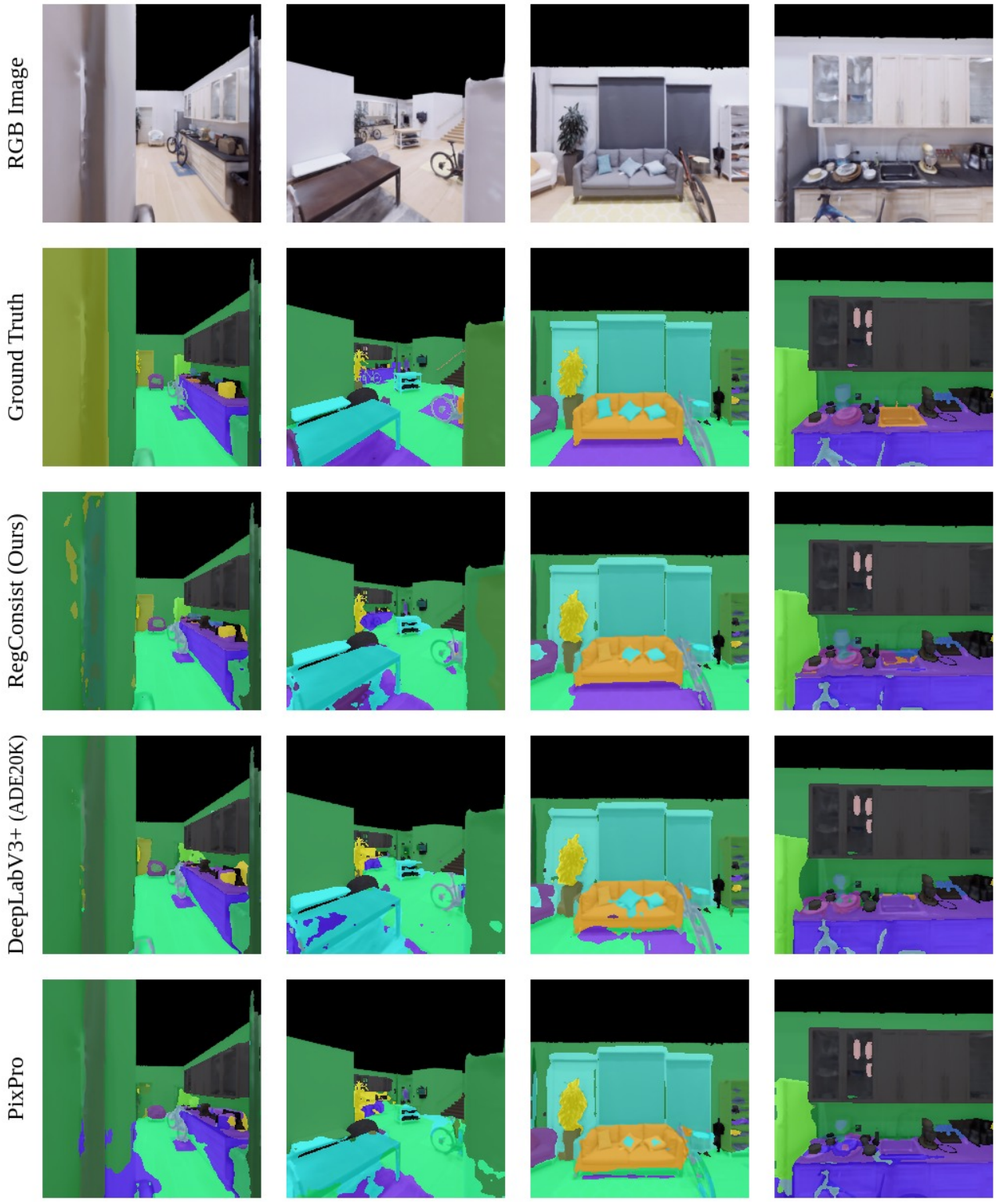

Figure 5. Results on the Replica scene.

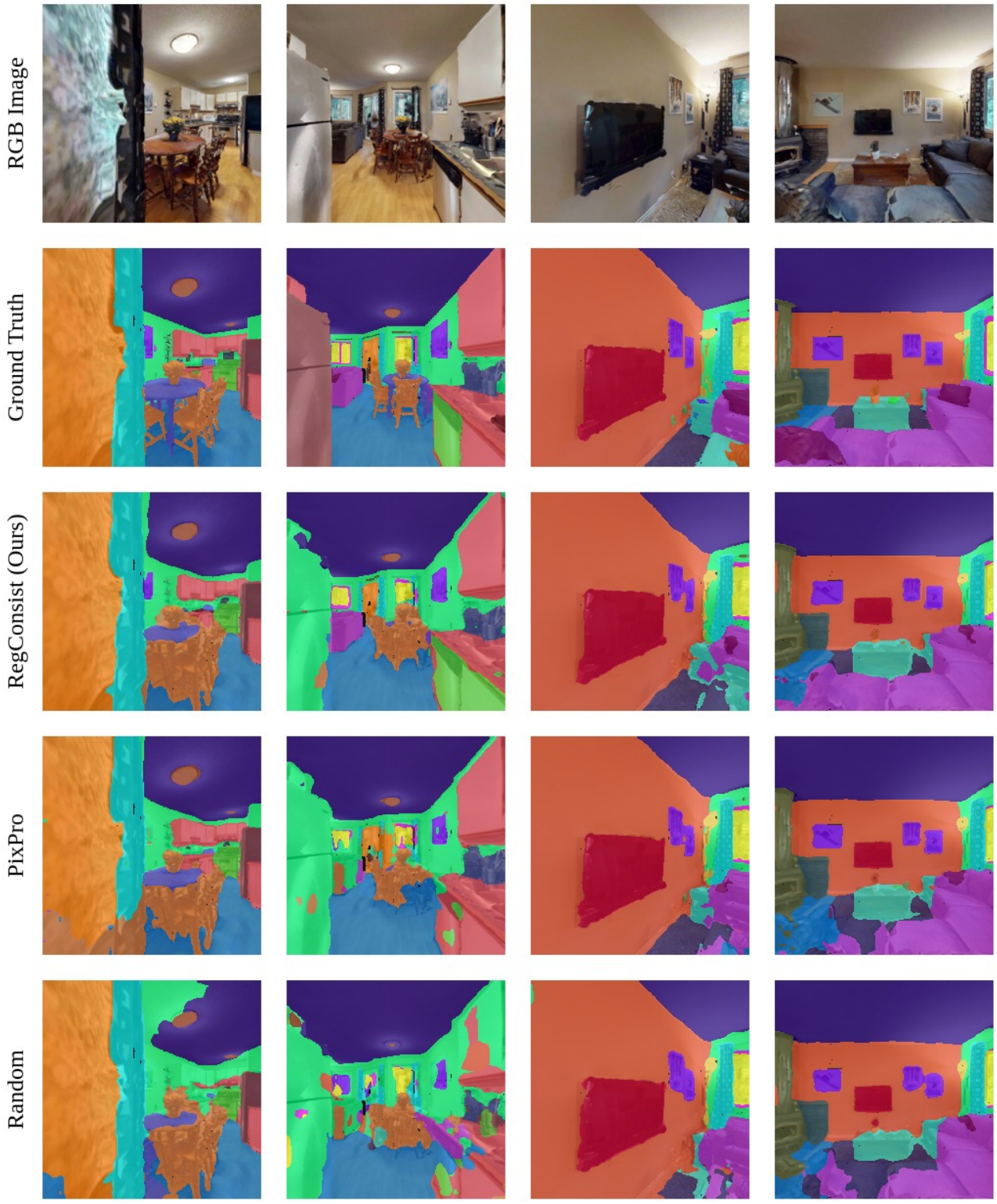

Figure 6. Results on the HM3D scene.

Figure 7. Results on the AVD scene.

References

[1] Phil Ammirato, Patrick Poirson, Eunbyung Park, Jana Kosecka, and Alexander C. Berg. A dataset for developing and benchmarking active vision. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[2] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:167–181, 2004.

[3] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (HM3d): 1000 large-scale 3d environments for embodied AI. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[4] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[5] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[6] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[7] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.