Supplemental Material: Deep Hierarchical Semantic Segmentation

Liulei Li[1,5*], Tianfei Zhou[2] , Wenguan Wang[3†], Jianwu Li[1] , Yi Yang[4]

[1] Beijing Institute of Technology [2] ETH Zurich [3] ReLER, AAII, University of Technology Sydney [4] CCAI, Zhejiang University [5] Baidu Research

https://github.com/0liliulei/HieraSeg

1. Detailed Hierarchical Architecture

We use the official hierarchical structure provided in each dataset. Detailed semantic hierarchies are provided in Fig. 1 for Mapillary Vistas 2.0 [3], Fig. 2 for Cityscapes [1], Fig. 3 for PASCAL-Person-Part [4] and Fig. 4 for LIP [2]. For Mapillary Vistas 2.0 and Cityscapes, we add a virtual root node (*i.e.*, `All`) to represent the most general concept.

2. Additional Qualitative Result

We provide additional visualization results on four datasets, including Mapillary Vistas 2.0 [3] `val` in Fig. 5, Cityscapes [1] `val` in Fig. 6, PASCAL-Person-Part [4] `val` in Fig. 7 and LIP [2] `val` in Fig. 8. The left column shows results from the baseline model while the right column is the predictions produced by HSSN. We see that HSSN yields consistently better visual effects than the baseline model.

3. Additional Ablative Study

We give extra ablative studies for the hyper-parameters emerged in our approach in Table 1. It can be seen that, for m_ε and 0.5 in Eq. 8, there is minor impact to the perfromance. This indicates our method is robust to hyper-parameters. For the balance factor β between $\mathcal{L}^{\mathrm{FTM}}$ and $\mathcal{L}^{\mathrm{TT}}$, scheduling it in a cosine annealing policy yields better performance. It is reasonable due to the poor recognition capability of the network at the initial stage of training.

4. Discussion on Triplet Number

We further investigate the impact of the number of triplets in $\mathcal{L}^{\mathrm{TT}}$ sampled druing training on performance. It can be seen in Table 2 that the introduce of triplet loss imposes additional computation to the model and slows down the training speed. However, HSSN is able to reach very promising performance using a small number of triplets (*e.g.*, 200) on both datasets. Further increasing the number only brings minor improvements. Based on the results in Table 2, we set the number to 200 for all datasets. This facilitates HSSN to perform triplet sampling at negligible cost and be rewarded with impressive performance boost.

m_ε in Eq. 8	mIoU^2↑	mIoU^1↑
0.05	93.16 (-0.13)	82.81 (-0.18)
0.10 (default)	93.29	82.99
0.15	93.19 (-0.10)	82.86 (-0.13)

(a)

0.5 in Eq. 8	mIoU^2↑	mIoU^1↑
0.45	93.17 (-0.12)	82.84 (-0.15)
0.50 (*default*)	93.29	82.99
0.55	93.13 (-0.16)	82.80 (-0.19)

(b)

β Schedule	Value	mIoU^2↑	mIoU^1↑
Cosine (*default*)	$0 \rightarrow 0.5$	93.29	82.99
Constant	0.5	93.02 (-0.27)	82.68 (-0.31)

(c)

Table 1. **More ablative experiments for hyper-parameters** (§3) on Cityscapes [1] `val`.

#	# Triplets	Mapillary Vistas 2.0 mIoU↑	Mapillary Vistas 2.0 Time↓	PASCAL-Person-Part mIoU↑	PASCAL-Person-Part Time↓
1	–	39.17	0.93	72.89	0.41
2	100	39.82 (-0.34)	1.06 (-0.12)	74.76 (-0.68)	0.52 (-0.20)
3	200 (*default*)	**40.16**	**1.18**	**75.44**	**0.58**
4	500	40.19 (+0.03)	1.58 (+0.40)	75.53 (+0.09)	0.72 (+0.14)

Table 2. **Impact of pixel triplet number** (§4) on Mapillary Vistas 2.0 [3] `val` and PASCAL-Person-Part [4] `test`. **Time** indicates training time (second) for each batch.

5. Broader Impact

Our research offers a novel perspective of modeling hierarchical semantic structures for semantic segmentation. Through directly incorporating semantic hierarchy into the optimization objective, we make a solid step towards a more reliable semantic segmentation algorithm which could enable many practical systems such as autonomous vehicles, robot navigation to make confident decisions.

6. Pseudo Code

To help the understanding of HSSN, we provide pseudocodes for tree-triplet loss $\mathcal{L}^{\mathrm{TT}}$ in Algorithm 1 and focal tree-min loss $\mathcal{L}^{\mathrm{FTM}}$ in Algorithm 2 .

*Work done during an internship at Baidu Research.

†Corresponding author: *Wenguan Wang*.

References

[1] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 1, 4, 6

[2] Xiaodan Liang, Ke Gong, Xiaohui Shen, and Liang Lin. Look into person: Joint body parsing & pose estimation network and a new benchmark. *TPAMI*, 2018. 1, 4, 8

[3] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *ICCV*, 2017. 1, 3, 5

[4] Fangting Xia, Peng Wang, Xianjie Chen, and Alan L Yuille. Joint multi-person pose estimation and semantic part segmentation. In *CVPR*, 2017. 1, 4, 7

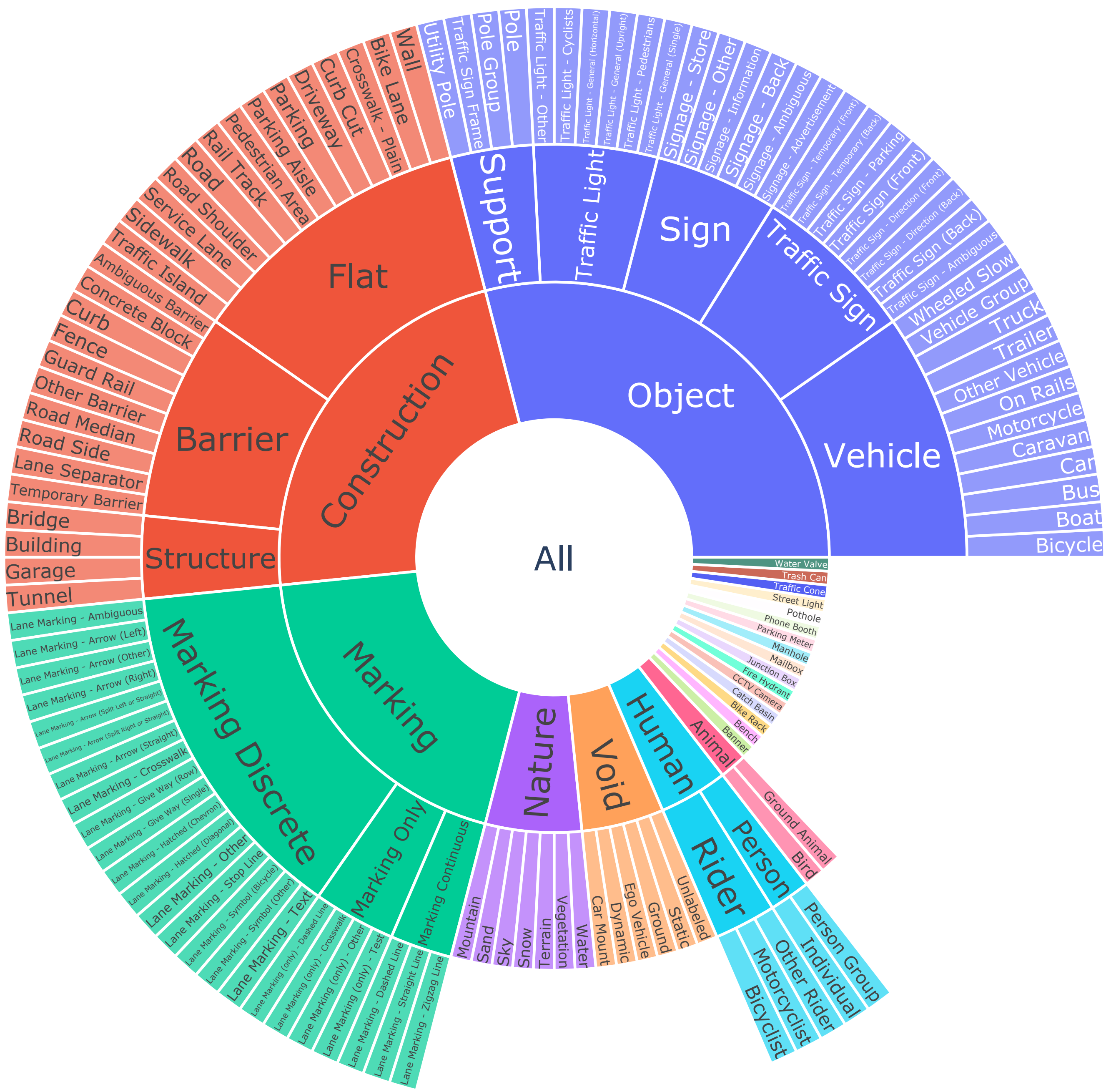

Figure 1. **Hierarchical architecture** of Mapillary Vistas 2.0[3].

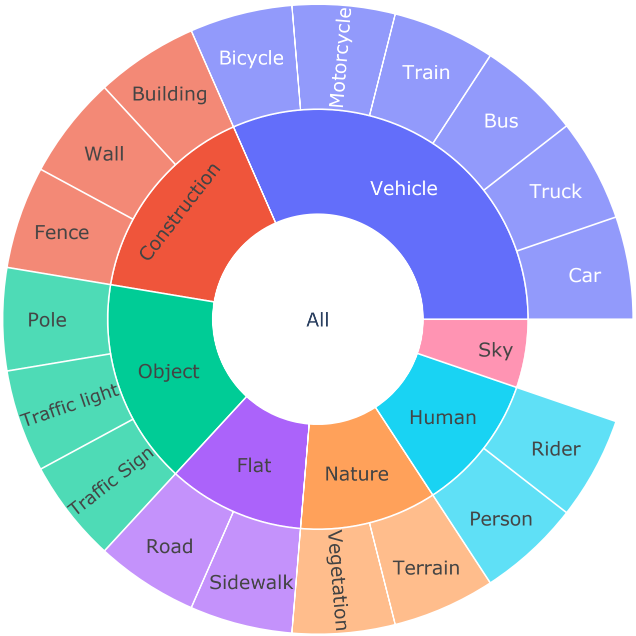

Figure 2. **Hierarchical architecture** of Cityscapes[1].

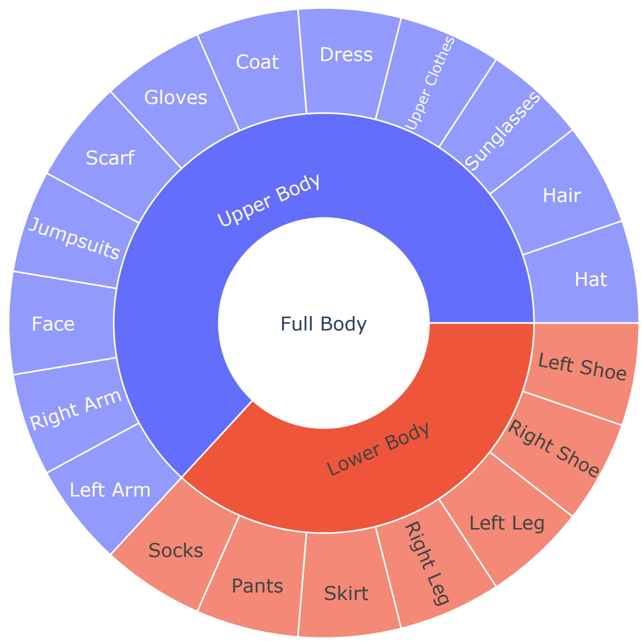

Figure 4. **Hierarchical architecture** of LIP[2].

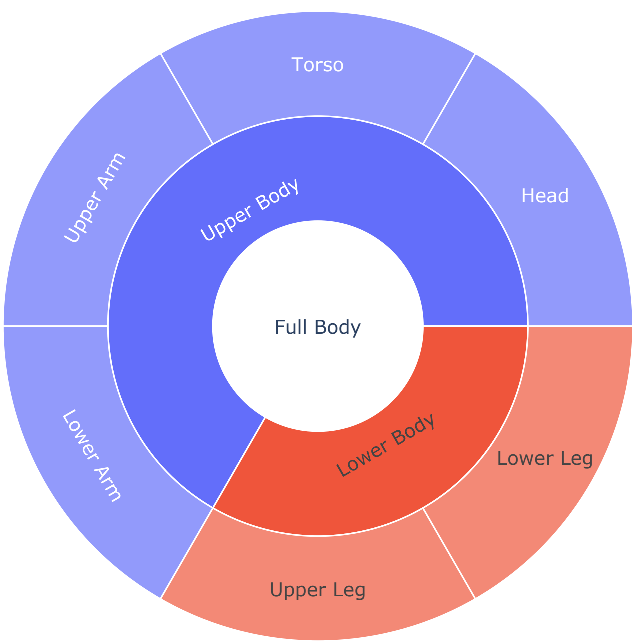

Figure 3. **Hierarchical architecture** of PASCAL-Person-Part[4].

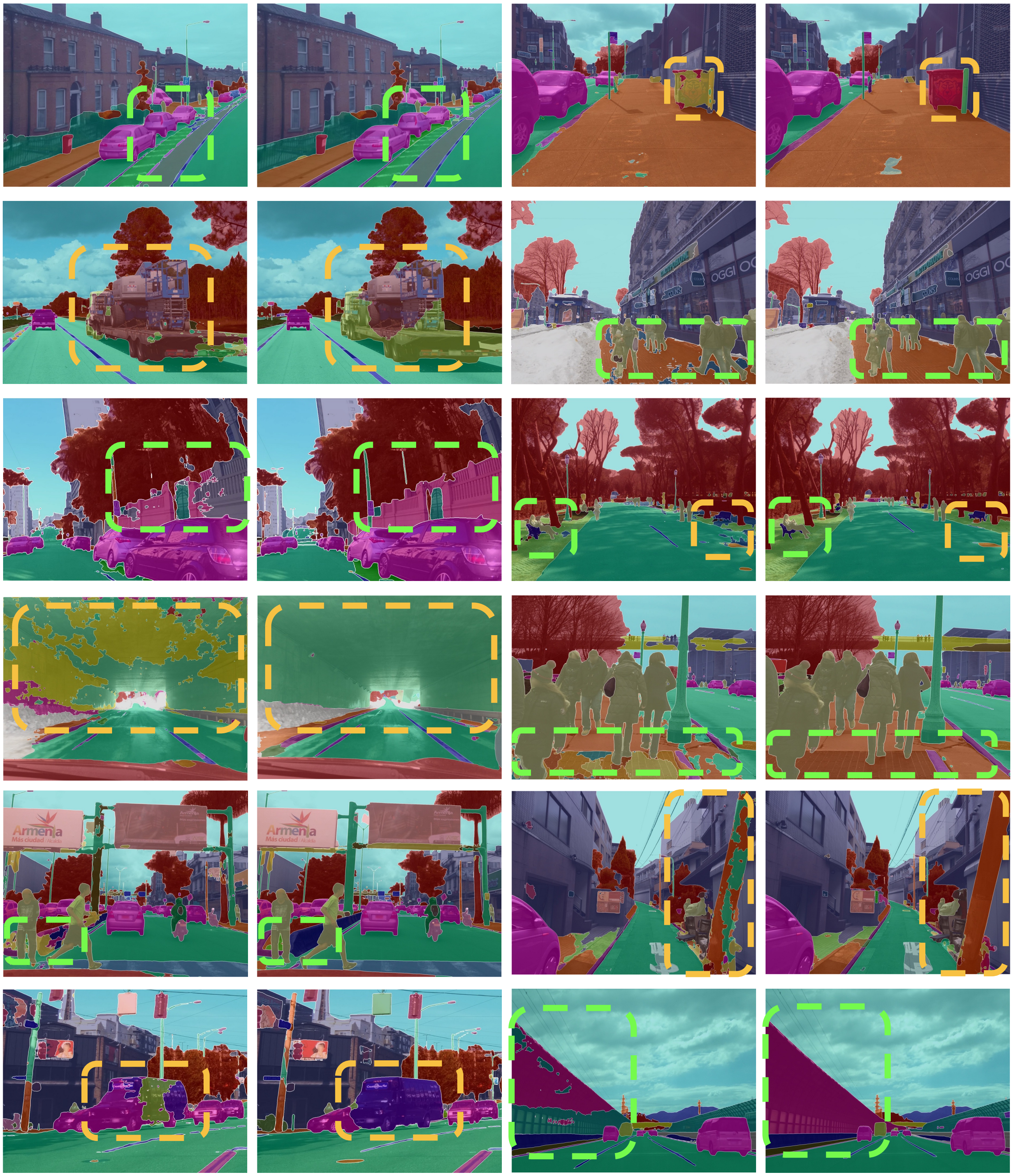

Figure 5. **More visualization results for semantic segmentation** on Mapillary Vistas 2.0[3] `val`.

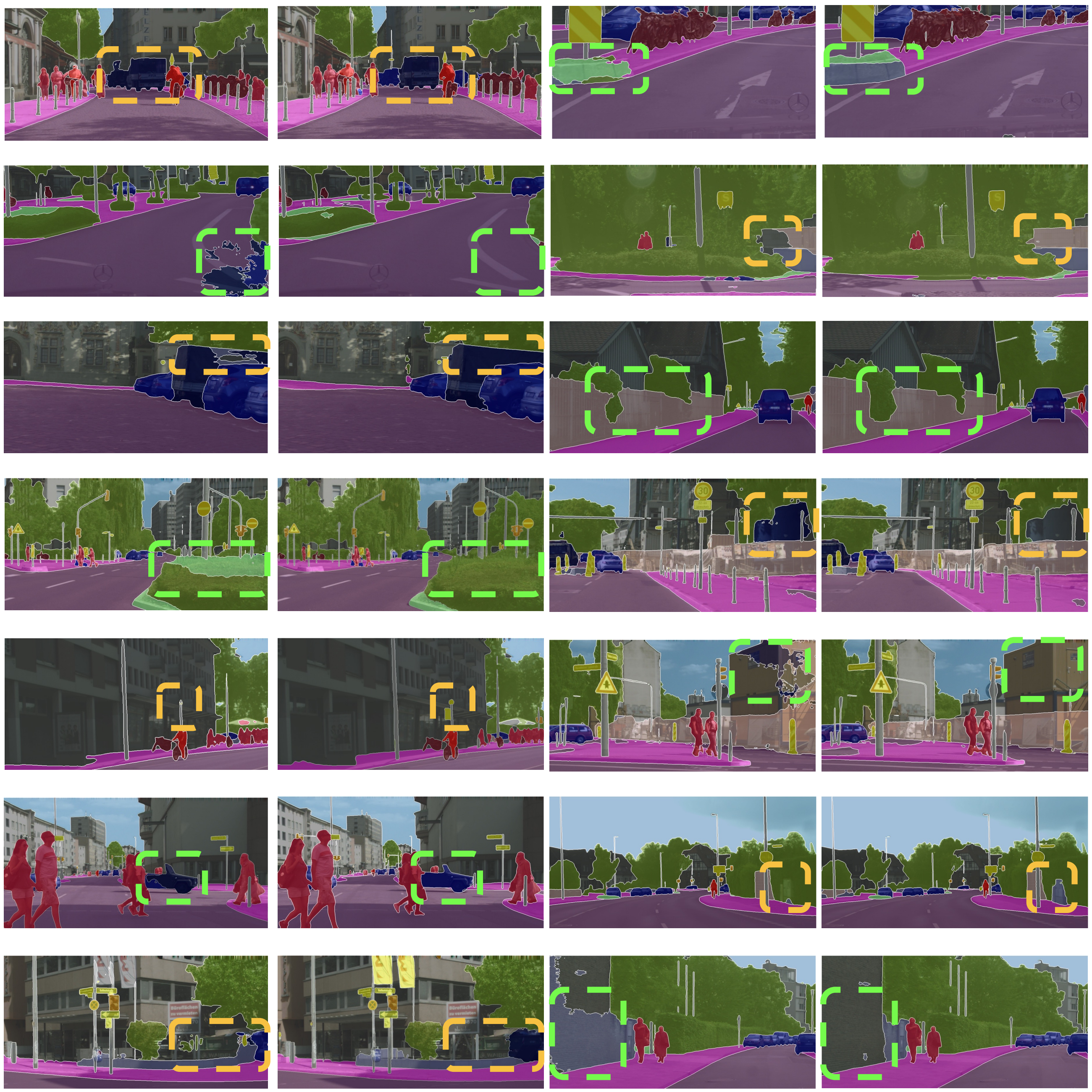

Figure 6. **More visualization results for semantic segmentation** on Cityscapes[1] `val`.

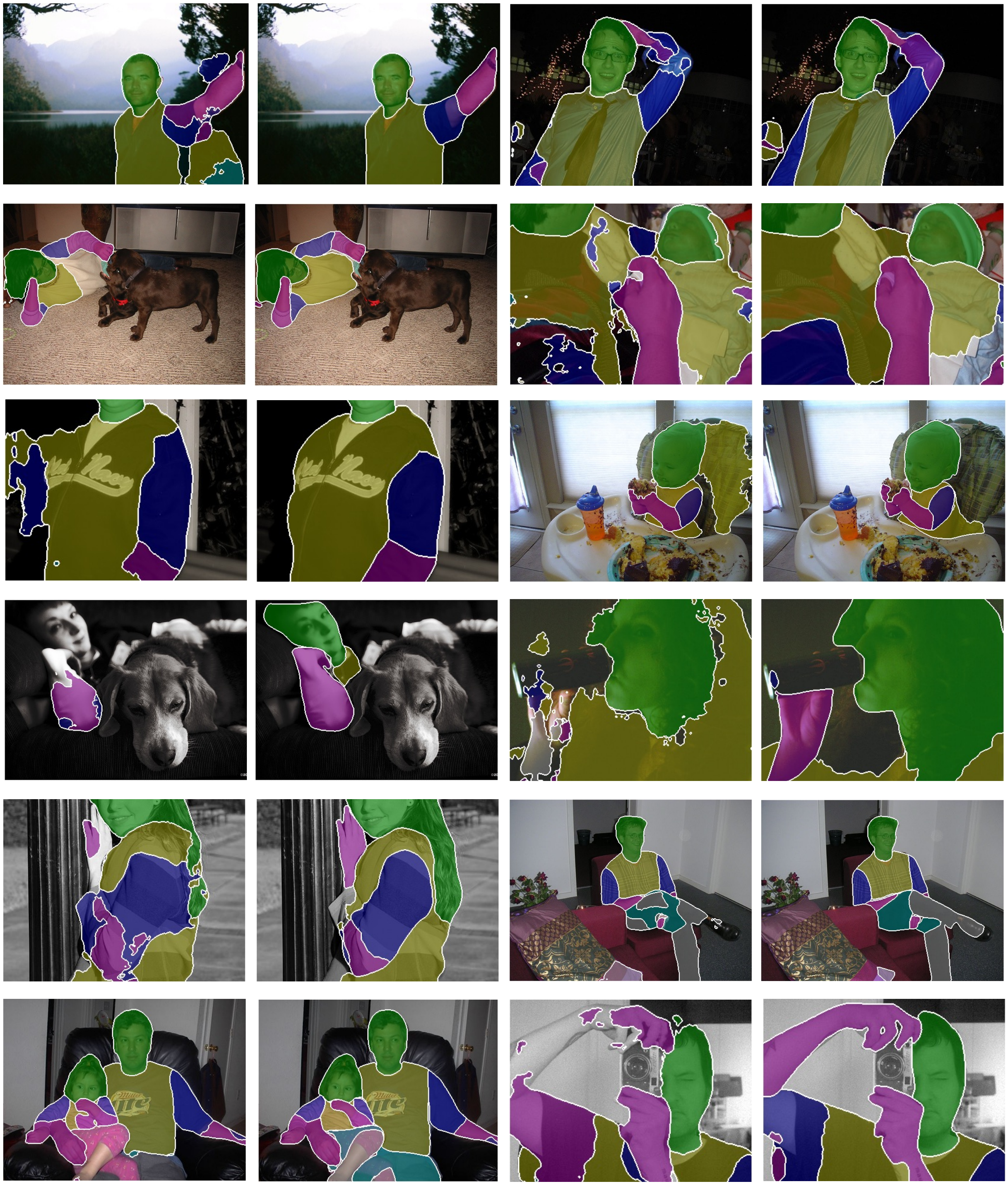

Figure 7. **More visualization results for human parsing** on PASCAL-Person-Part[4] `val`.

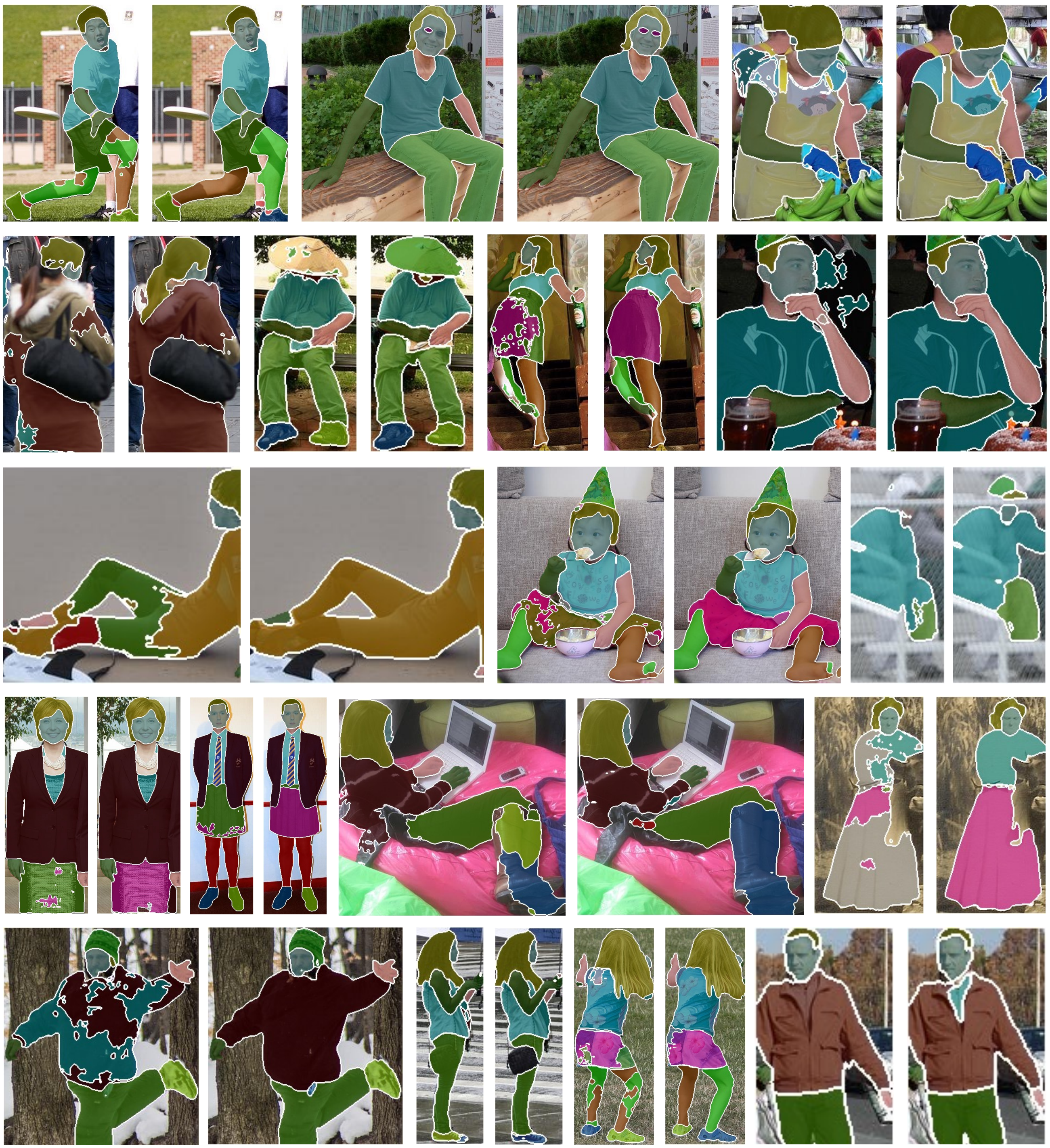

Figure 8. **More visualization results for human parsing** on LIP[2] `val`.

Algorithm 1 Pseudocode of Tree-Triplet loss (*i.e.*, $\mathcal{L}^{\mathrm{TT}}$) in a PyTorch-like style.

```
# HEIGHT: height of the semantic tree
def find_triplet_candidates(cur_cls, labels):
   # find a random level as postive threshold
   level = random(HEIGHT-1)
   # find the parent of current class at given level
   target_parent = FIND_PARENT(level, cur_cls)
   # find all children of the given node
   child_cls = FIND_CHILDREN(target_parent)
   # classes qualified to be positive
   pos_cls = child_cls - cur_cls

   idx_anc = (labels==cur_cls)
   idx_pos = (labels>=pos_cls[0]) & (labels<pos_cls
       [-1])
   idx_neg = (labels<pos_cls[0]) | (labels>=pos_cls
       [-1])

   return idx_anc, idx_pos, idx_neg


#========== compute margin m, Eq.8 ===========#
# l_*: label of anchor, pos and neg
# epsilon: constant for the tolerance of intra-class
    variance
# PSI: function to get the tree distance between two
    classes
def compute_margin(l_anc, l_pos, l_neg, epsilon=0.1):
   margin = torch.ones_like(l_anc) * epsilon
   for cur_trip, (anc, pos, neg) in enumerate(zip(
       l_anc, l_pos, l_neg)):
      # dynamic margin according to tree distance
      margin[cur_trip] += 0.5*(PSI(anc, neg)-PSI(anc,
          pos))/(2*HEIGHT)

   return margin


#====== compute tree triplet loss, Eq.7 =======#
# embedding: l2-normed features (H x W x C)
# labels: ground truth (H x W)
# max_triplet: max number of triplets sampled in a
    batch
def tree_triplet_loss(embedding, labels, max_triplet):
   labels = labels.view(-1)
   embedding = embedding.view(-1, embedding.size(-1))

   triplet_loss = 0

   for cur_cls in torch.unique(labels):
      # for each class, find anchor, positive and
          negative candidates
      idx_anc, idx_pos, idx_neg =
          find_triplet_candidates(cur_cls, labels)

      # maximum number of sampled triples
      max_num = min(torch.sum(idx_anc), torch.sum(
          idx_pos), torch.sum(idx_neg), max_triplet)

      f_anc = embedding[idx_anc][:max_num]
      f_pos = embedding[idx_pos][:max_num]
      f_neg = embedding[idx_neg][:max_num]

      # compute cosine distance
      distance = torch.zeros((max_num, 2))
      distance[:,0] = 1-(f_anc*f_pos).sum(dim=1)
      distance[:,1] = 1-(f_anc*f_neg).sum(dim=1)

      l_anc = labels[idx_anc][:max_num]
      l_pos = labels[idx_pos][:max_num]
      l_neg = labels[idx_neg][:max_num]

      # compute margin
      margin = compute_margin(l_anc, l_pos, l_neg)

      # compute triplet loss
      loss = distance[:,0] - distance[:,1] + margin
      loss = F.relu(loss)
      triplet_loss += loss.sum()

   return triplet_loss
```

Algorithm 2 Pseudocode of Focal Tree-Min loss (*i.e.*, $\mathcal{L}^{\mathrm{FTM}}$) in a PyTorch-like style.

```
# predict: predicted score map (H x W x N)
# target: ground-truth label map (H x W x N)
# gamma: focusing hyper-parameter
# CLASS_RANGE(int: level): an utility function to get
     indices of classes in a specific level
def focal_tree_min_loss(predict, target, gamma):
   # compute hierarchy-coherent scores for positive
       classes (Eq.4)
   pos_score = [predict[:,:,CLASS_RANGE(HEIGHT-1)]
   for ii in range(HEIGHT-2, -1, -1):
      pos_score.append(torch.min(
         torch.cat([predict[:,:,CLASS_RANGE(ii)],
             pos_score[-1]], dim=-1), dim=-1, keepdim=
             True)[0])

   # compute hierarchy-coherent scores for negative
       classes (Eq.4)
   neg_score = [predict[:,:,CLASS_RANGE(0)]]
   for ii in range(1, HEIGHT):
      neg_score.append(torch.max(
         torch.cat([predict[:,:,CLASS_RANGE(ii)],
             neg_score[-1]], dim=-1), dim=-1, keepdim=
             True)[0])

   # accumulate losses for all levels
   loss = 0
   for ii in range(HEIGHT):
      loss +=(-target*torch.pow(1-pos_score[ii],gamma)
           *torch.log(pos_score[ii])
           -(1-target)*torch.pow(neg_score[ii],gamma)
           *torch.log(1-neg_score[ii])).sum()

   return loss
```