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 $m_e$ and 0.5 in Eq. 8, there is minor impact to the performance. This indicates our method is robust to hyperparameters. For the balance factor $\beta$ between $\mathcal{L}^\text{FTM}$ and $\mathcal{L}^\text{TT}$, 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 $\mathcal{L}^\text{TT}$ sampled during 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.

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 $\mathcal{L}^\text{TT}$ in Algorithm 1 and focal tree-min loss $\mathcal{L}^\text{FTM}$ in Algorithm 2.

---

*Corresponding author: Wenguan Wang.

**Work done during an internship at Baidu Research.
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


Figure 1. **Hierarchical architecture** of Mapillary Vistas 2.0[3].
Figure 2. **Hierarchical architecture** of Cityscapes[1].

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

Figure 4. **Hierarchical architecture** of LIP[2].
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] va1.
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., $\mathcal{L}^{TT}$) in a PyTorch-like style.

```python
# HEIGHT: height of the semantic tree
def find_triplet_candidates(cur_cls, labels):
    level = random(HEIGHT-1)
    target_parent = FIND_PARENT(level, cur_cls)
    child_cls = FIND_CHILDREN(target_parent)
    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 ===========#
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)):
        margin[cur_trip] += 0.5*(PSI(anc, neg) - PSI(anc, pos))/2*HEIGHT
    return margin

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):
        idx_anc, idx_pos, idx_neg = find_triplet_candidates(cur_cls, labels)
        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]

        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]

        margin = compute_margin(l_anc, l_pos, l_neg)
        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., $\mathcal{L}^{FTM}$) in a PyTorch-like style.

```python
# 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):
    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])

    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])

    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
```