Semi-supervised Semantic Segmentation with Directional Context-aware Consistency — Supplementary Material

Introduction

This is the supplementary material, which is divided into the following sections.

- In Sec. 1, we elaborate the details of our implementation with PyTorch [9] gradient checkpoint function that significantly saves the training memory.
- In Sec. 2, we also demonstrate the details of the datasets used in our experiments.
- To show that more confident features are generally more accurate, the accuracy curve of the predictions with different confidence values is shown in Sec. 3.
- More illustrations of Grad-CAM [10] are presented in Sec. 4, which demonstrates that the model trained with only labeled data is easy to overly use the contexts and lose self-awareness.
- More examples of visual comparisons are demonstrated in Sec. 5 to show the superiority of our method.

1. Implementation Details

The gradient checkpoint function is provided in PyTorch [9] to reduce the cost of training memory. Normally, when we forward the tensors to the network, the intermediate activations will be automatically stored. In this case, when computing the gradients in later backward gradient propagation, we do not need to recompute the intermediate activations any more, thus saving time and accelerating the training process. However, in our case, we need to increase the negative samples from 500 to 19.2k, which consumes much more memory if the intermediate activations are stored. Therefore, we could use the gradient checkpoint function provided by PyTorch not to store the intermediate activations. More details about the gradient checkpoint function can be found in PyTorch official documentation[1].

Specifically, we split the loss function computation process into several parts, and inside each part, we do not store the intermediate activations but recompute them in the backward process. Meanwhile, all other forward processes except calculating the loss function remain the same, *i.e.*, storing the intermediate activations to accelerate training. In this case, our implementation significantly reduces memory consumption, while incurring negligible extra training time. Because calculating the loss function as normal consumes a lot of memory but a relatively short time. The implementation details are shown in Algorithm 1.

Algorithm 1 Pseudo code of calculating the loss function with gradient checkpoint in a PyTorch-like style

```
# calculate the negative logits of proposed loss function
def calc_neg_logits(feats, pseudo_labels, neg_feats, neg_pseudo_labels):
    pseudo_labels = pseudo_labels.unsqueeze(-1)
    neg_pseudo_labels = neg_pseudo_labels.unsqueeze(0)

    # negative sampling mask (Nxb)
    neg_mask = (pseudo_labels != neg_pseudo_labels).float()
    neg_scores = (feats @ neg_feats.T) / temp # negative scores (Nxb)
    return (neg_mask.float() * torch.exp(neg_scores)).sum(-1)

# feats1: features of the overlapping region in the first crop (NxC)
# feats2: features of the overlapping region in the second crop (NxC)
# neg_feats: all selected negative features (nxC)
# pseudo_labels1: pseudo labels for feats1 (N)
# pseudo_logits1: confidence for feats1 (N)
# pseudo_logits2: confidence for feats2 (N)
# neg_pseudo_labels: pseudo labels for neg_feats (n)
# gamma: the threshold value for positive filtering
# temp: the temperature value
# b: an integer to divide the loss computation into several parts

pos1 = (feats1 * feats2.detach()).sum(-1) / temp # positive scores (N)
neg_logits = torch.zeros(pos1.size(0)) # initialize negative scores (n)

# divide the negative logits computation into several parts
# in each part, only b negative samples are considered
for i in range((n-1) // b + 1):
    neg_feats_i = neg_feats[i*b:(i+1)*b]
    neg_pseudo_labels_i = neg_pseudo_labels[i*b:(i+1)*b]
    neg_logits_i = torch.utils.checkpoint.checkpoint(calc_neg_logits,
            feats1, pseudo_labels1, neg_feats_i, neg_pseudo_labels_i)
    neg_logits += neg_logits_i

# compute the loss for the first crop
logits1 = torch.exp(pos1) / (torch.exp(pos1) + neg_logits + 1e-8)
loss1 = -torch.log(logits1 + 1e-8) # (N)
dir_mask1 = (pseudo_logits1 < pseudo_logits2) # directional mask (N)
pos_mask1 = (pseudo_logits2 > gamma) # positive filtering mask (N)
mask1 = (dir_mask1 * pos_mask1).float()

# final loss for the first crop
loss1 = (mask1 * loss1).sum() / (mask1.sum() + 1e-8)
```

2. Details of Datasets

Following previous works [4, 5, 7, 6, 8], our method is evaluated on PASCAL VOC [2] augmented with annota-

[1]https://pytorch.org/docs/1.6.0/checkpoint.html

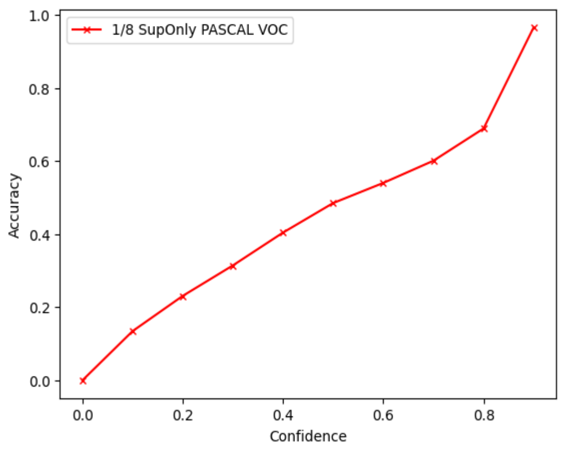

Figure 1. The accuracy curve of the predictions with different confidence values. The predictions are made by the SupOnly model on the validation set of PASCAL VOC. Best viewed in zoom.

tions from SBD [3], which consists of 10582 training images and 1449 validation images with one background and 20 foreground classes. Note that both labeled and unlabeled images are selected from the training images. Also, our method generalizes well to Cityscapes [1] that contains 2975 and 500 finely annotated images with 19 classes for training and validation respectively.

3. Confident Features Are More Accurate

As shown in Fig. 1, we observe that for a prediction, higher confidence generally corresponds to higher accuracy. This demonstrates that if a feature has higher confidence, then it generally will be more accurate.

4. Grad-CAM Visualization

Grad-CAM [10] has been proposed to visualize the high-level features of classification networks. Firstly, it obtains the gradients regarding high-level features at a specific layer by back propagating the highest activation of the last layer. After that, it globally averages the spatial gradients as the weights, which is further used to get a channel-wise weighted sum with the features at that layer. After min-max normalization, the result is the heat map of the so-called Grad-CAM, where the red region corresponds to high contribution.

In our case of the segmentation network, we replace the last layer feature of the original classification network with the feature of interest. And then, we calculate the gradients regarding the last layer features by back propagating the highest activation of the logits of the feature that we are interested in. All other operations remain the same.

Fig. 2 demonstrates more Grad-CAM visualizations. These illustrations show that features of the SupOnly model are easy to overly use the contexts but overlook themselves,

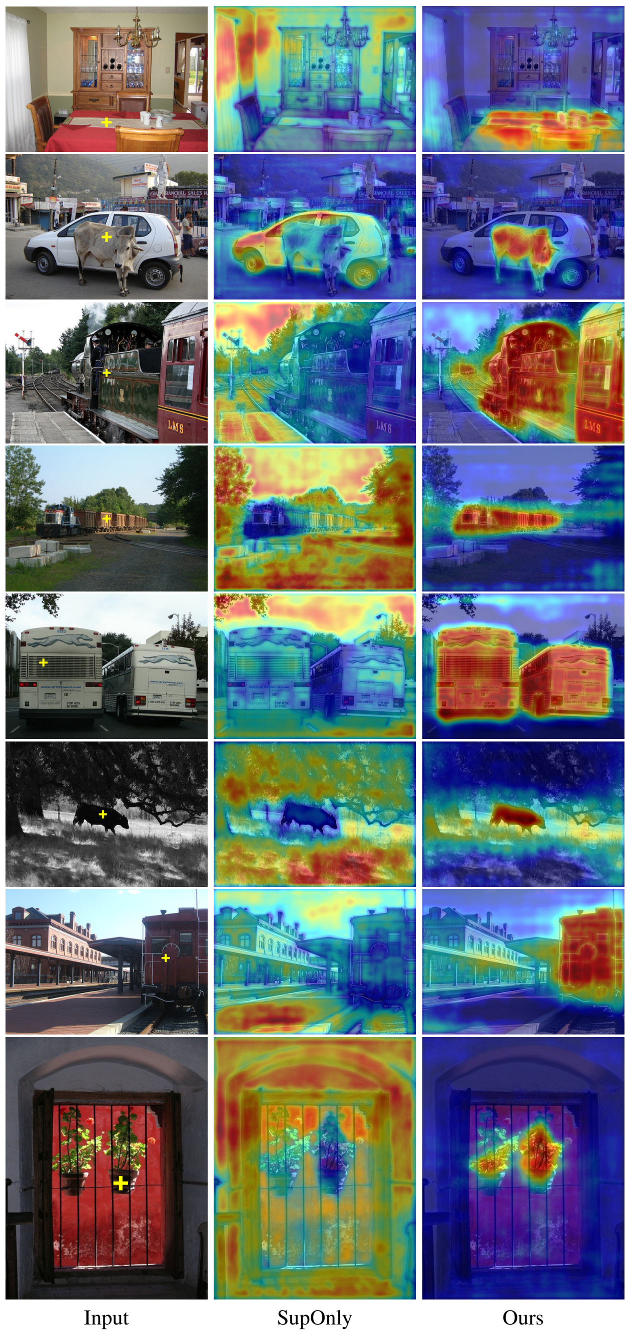

Figure 2. More Grad-CAM [10] visualizations of the regional contribution to the feature of interest (*i.e.*, the yellow cross shown in the input). The red region corresponds to high contribution. SupOnly: the model trained with only 1/8 labeled data.

while those of ours can focus on themselves and also make use of the contexts more reasonably.

5. Visual Comparison

In Fig. 3, we demonstrate more examples of the visual comparisons. They show that our method is generally superior to others.

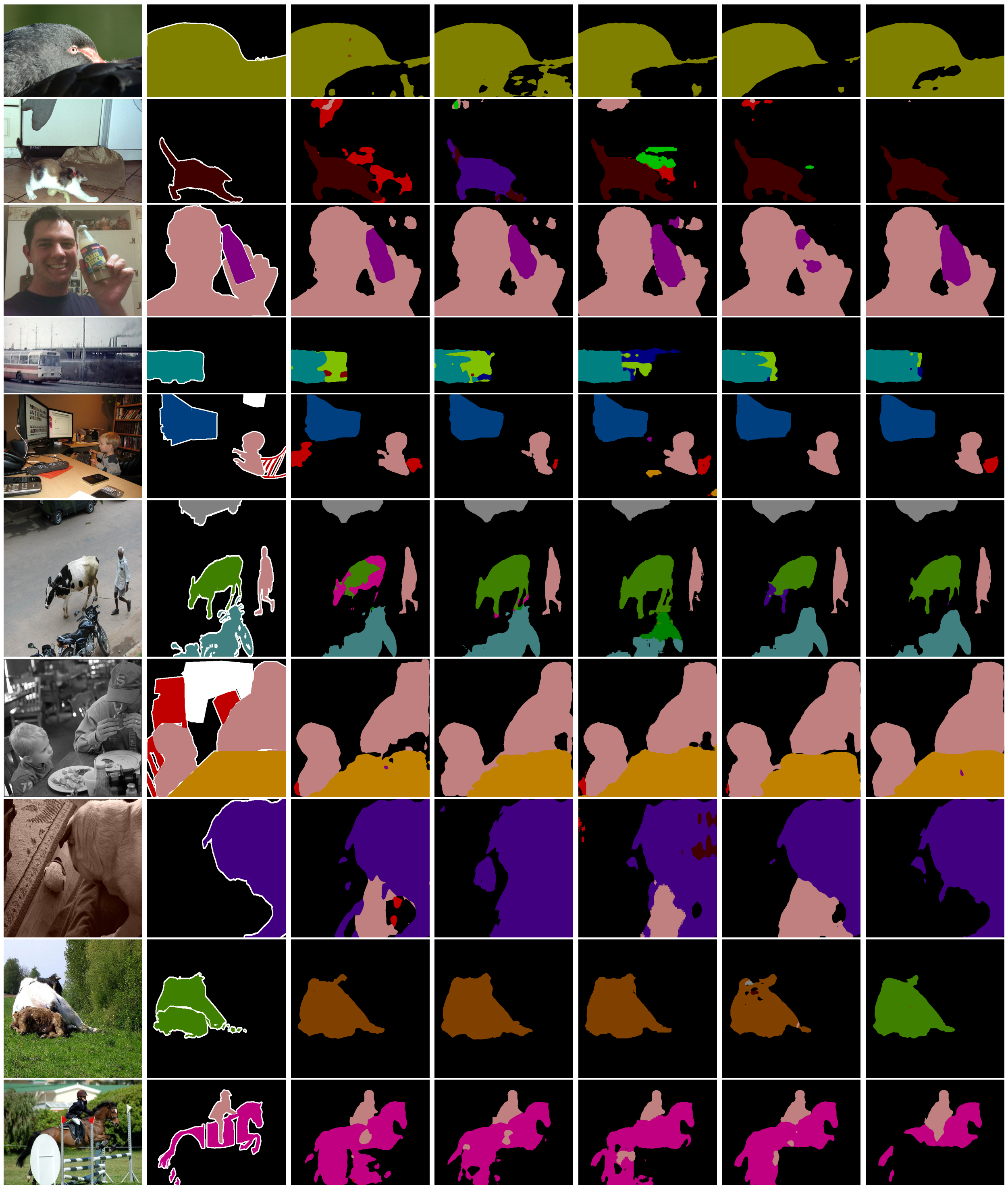

Figure 3. More visual comparisons with SupOnly (*i.e.*, trained with only supervised loss) and other current state-of-the-art methods.

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. 2

[2] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object

classes (voc) challenge. *IJCV*, 2010. 1

[3] Bharath Hariharan, Pablo Arbelaez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *ICCV*, 2011. 2

[4] Wei-Chih Hung, Yi-Hsuan Tsai, Yan-Ting Liou, Yen-Yu Lin, and Ming-Hsuan Yang. Adversarial learning for semi-supervised semantic segmentation. In *BMVC*, 2018. 1

[5] Zhanghan Ke, Di Qiu, Kaican Li, Qiong Yan, and Rynson W.H. Lau. Guided collaborative training for pixel-wise semi-supervised learning. In *ECCV*, 2020. 1

[6] Robert Mendel, Luis Antonio, De Souza Jr, David Rauber, and Christoph Palm. Semi-supervised segmentation based on error-correcting supervision. In *ECCV*, 2020. 1

[7] Sudhanshu Mittal, Maxim Tatarchenko, and Thomas Brox. Semi-supervised semantic segmentation with high- and low-level consistency. *TPAMI*, 2019. 1

[8] Yassine Ouali, Celine Hudelot, and Myriam Tami. Semi-supervised semantic segmentation with cross-consistency training. In *CVPR*, 2020. 1

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 1

[10] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *IJCV*, 2020. 1, 2