Supplementary Material for Beyond Semantic to Instance Segmentation: Weakly-Supervised Instance Segmentation via Semantic Knowledge Transfer and Self-Refinement

Beomyoung Kim[1] YoonJoon Yoo[1,2] Chaeeun Rhee[3] Junmo Kim[4]

NAVER CLOVA[1] NAVER AI Lab[2] Inha University[3] KAIST[4]

1. Detail of Center Clustering Algorithm

For the complementary knowledge between each network output, we employ the center clustering algorithm to extract center points from the offset map when generating the refined label. Here, we describe a detailed algorithm for the center clustering with a Figure 1. First, from the offset map, we create a magnitude map where each pixel represents the magnitude of the 2D vector. In this magnitude map, the pixel near the center of each instance is close to zero. Second, we apply a threshold to the magnitude map. We set the threshold to 2.5. Last, we extract the center point of each mask candidate obtained from the connected component labeling (CCL) algorithm. Here, we observe that the optimal area of the mask candidate is determined according to the threshold. For example, when the threshold is 2.5, the desired area of the mask candidate is near 21. For reliability-check utilizing the above observation, we additionally check whether the area of the mask candidate is between 21-ε and 21+ε; we empirically set the ε to 3. Due to the reliability-check process, we can prevent extracting false center points from the unstable offset map in the early training stage.

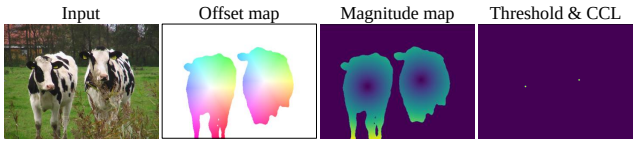

Figure 1. Illustration of the center clustering algorithm. The blue and yellow pixels in the magnitude map indicate that pixel values are close to zero and far from zero, respectively.

2. Additional Ablation Study

Here, we provide some additional ablation studies. First is the class-wise center map. As mentioned in the experiment section, we modify the original Panoptic-DeepLab network; we change the class-agnostic center map to the class-wise center map. This modification yields a 1.0% mAP_{50} improvement due to more accurate instance grouping; compared to the full supervision, our noisy offset vectors in the offset map are sometimes grouped with incorrect center points. To prevent incorrect instance grouping, we restrict the centers of other classes not to be grouped by adopting the class-wise center map.

The second is an additional analysis for the proposed methods. In Table 3 in the main paper, we provided the analysis for the proposed methods. Here, we show an additional study for the model without IAG but with self-refine. As in Table 1, the self-refine without IAG drops the performance because the model without IAG suffers from the semantic drift problem. And the drift iteratively degrades the quality of the refined label, hurting the model.

The third is the effect of hyperparameter α that is a threshold for the PAM module. When the α becomes large, more noisy regions are deactivated. However, due to the IAG and self-refine, mAP_{50} result of BESTIE is robust to the α as in Table 2.

The fourth is the effect of the backbone network in BESTIE. As mentioned in the experiment section, we adopt HRNet-48 [4] as our backbone network. Here, we study the effect of the backbone network by replacing another backbone network, *i.e.*, ResNet-50 [2]. As an experimental result, the HRNet-48 backbone yields about 1% mAP_{50} higher performance than the ResNet-50 (41.8% mAP_{50} for HRNet-48 and 40.9% mAP_{50} ResNet-50 on VOC 2012 validation set). The reason is that the receptive field of the HRNet-48 is much larger than the ResNet-50 and the HRNet-48 is a well-designed network for the key-point representation.

The last is a threshold for extracting instance cues from PAM. Namely, we extract instance cues by obtaining local maximum points from the PAM. Here, we adopt the instance cues whose value is larger than the threshold, which is set to 0.5. When we change the threshold to 0.3 and 0.7, the number of true-positives in pseudo labels changes

Table 1. Additional analysis for the proposed methods.

PAM	IAG	refine	mAP_{50}
✓			29.3
✓	✓		39.2
✓	✓	✓	41.8
✓		✓	27.8

Table 2. Effect of the hyperparameter α.

α	mAP_{50}
0.3	41.76
0.5	41.80
0.7	41.72

slightly, but the mAP_{50} variation is quite small to $\pm 0.1\%$. This is because our *self-refinement* method can progressively refine the pseudo labels and increase the number of true-positives.

3. Details of Peak Attention Module (PAM)

Implementation Details

As described in the main paper, we extract instance cues from the classifier with our PAM. Here, we explain the implementation details of the PAM. We employ VGG-16 [3] classifier and plugin our PAM into the last three convolutional layers of the classifier. The architecture of the classifier with PAM is illustrated in Figure 2. For training the classifier with PAM, we use the binary cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with a weight decay of 0.0005 and a momentum of 0.9. The initial learning rate is set to 0.001 and is decreased by a factor of 10 at epoch 5 and 10. For data augmentation, images are randomly cropped to 321×321, and random horizontal flipping and random color jittering are applied. We use a batch size of 5 and train the classifier for 15 epochs. In the following section, we analyze how the PAM module affects each layer.

Effect of PAM on each layer of Classifier

In this section, we analyze the effect of the PAM on each layer of the classifier. With the classifier described in Figure 2 as our baseline, we plug-in or plug-out the module. For the quantitative comparison as in Table 3, we evaluate the mean average precision (mAP) of our instance segmentation network without the Mask R-CNN refinement step. Since our PAM strengthens the attention on peak regions by deactivating noisy regions, we necessary to accurately distinguish between peak and noise regions. In lower-level layers ($i.e.$, layer-1, layer-2, and layer-3), the classifier captures the local features such as edges, and the definition of the peak region is unclear, so the effect of the PAM is minor. In contrast, in higher-level layers ($i.e.$, layer-4, layer-5, and layer-6), especially in the last layer, the classifier captures the global features, and the distinction between peak regions and noisy regions is more clear; our PAM plays a meaningful role in the last layer. From the results in Table 3, we note that the PAM equipped in only the last layer

Table 3. Effect of the PAM on each layer of the classifier. ✓ means the PAM is equipped.

PAM					
layer4	layer5	layer6	mAP_{25}	mAP_{50}	mAP_{75}
			40.2	34.7	19.6
		✓	**53.5**	**41.8**	**24.2**
	✓	✓	49.6	39.5	23.9
✓	✓	✓	43.7	34.8	21.2
✓		✓	53.6	40.0	23.8
✓			48.3	37.6	22.1
	✓		49.9	39.6	23.4
✓	✓		49.3	38.4	22.7

yields the best performance (41.8% mAP_{50}) but the PAM equipped in the last three layers significantly degrades the performance (34.8% mAP_{50}). We conclude that it is most effective to use the PAM only in the last layer where the definition of peak regions and noisy regions is the most obvious, and excessive modulization of PAM might deactivate the important features, degrading the performance.

4. Qualitative Results of Pseudo Label

In Figure 3, we provide more qualitative results of activation maps and pseudo labels. As in the orange area of Figure 3, the conventional CAMs have a limitation in generating high-quality pseudo labels due to the noisy activation region. However, as in the green area of Figure 3, our PAM produces sparse CAMs that help to extract one instance cue per instance. Therefore, from the *semantic knowledge transfer*, we can obtain more reliable pseudo labels, and the pseudo labels contain more true positive training samples.

5. Qualitative Results of Proposed Method

In Figure 4, we provide more qualitative results of our pseudo labels and network outputs. The pseudo label provides some reliable true-positive samples but contains lots of false-negatives ($i.e.$, missing instances). Due to the proposed *self-refinement* with the *instance-aware guidance*, the network can produce high-quality instance masks including missing instances in pseudo labels. In addition, we compare our instance mask with that of IRN [1], which is the proposal-free method. The comparison results clearly show that our approach can properly segment multiple instances with a high-precision instance mask.

6. Failure Cases for PAM

We provide some failure cases of PAM in Figure 5, and these examples demonstrate the superiority of the point-supervised setting because inaccurate instance cues are replaced by ground-truth points. PAM has trouble in accurately localizing overlapping instances, which leads to the

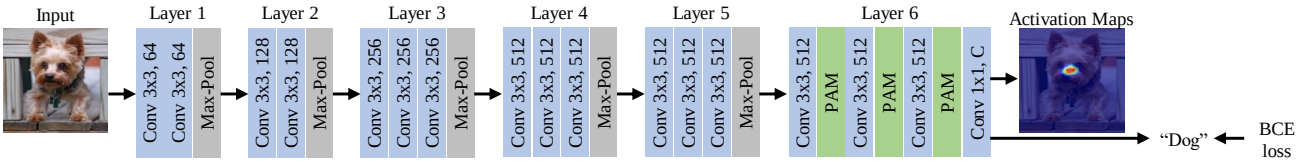

Figure 2. The detailed architecture of our classification network. BCE loss means the binary cross entropy loss function. The PAM is equipped in last three convolutional layers and trained with a self-supervised scheme.

incorrect pseudo label (first row in Figure 5). In addition, missing instance cue or noisy localization increases missing instances in the pseudo label (second and third rows in Figure 5). Last, the WSSS method is trained with only image-level labels, some insufficient semantic segmentation maps yield inaccurate pseudo labels (last row in Figure 5).

7. Failure Cases for Proposed Method

We provide some failure cases of BESTIE in Figure 6. First, when center points of instances are close to each other, we often fail to obtain the proper instance masks (first row in Figure 6); however, the keypoint-based method has suffered this issue even in a fully-supervised setting. In addition, noisy center and offset maps lead to false instance masks (second and third rows in Figure 6). Last, when the semantic segmentation map provides a noisy foreground region, we often fail to obtain the precise instance mask.

8. Pytorch-style Pseudo-code.

To describe the details of each proposed methods, we provide pytorch-style pseudo-code algorithm. Note that our BESTIE is simple and easy to be implemented.

```
for n in range(train_iterations):
    # load data: x (input tensor), seg_map (weakly-supervised semantic map)
    x, seg_map = loader.next()

    # pseudo-label generation
    PAM = classifier(x)
    peak_points = extract_peak_points(PAM)
    pseudo_label = pseudo_label_gen(seg_map, peak_points)

    # network forwarding
    outputs = BESTIE(x)

    # refined-label generation
    refined_label = refined_label_gen(outputs)

    # backward & optimize
    loss_pseudo = objective_function_with_IAG(outputs, pseudo_label)
    loss_refined = objective_function_with_IAG(outputs, refined_label)
    loss = loss_pseudo + loss_refined
    loss.backward()
    optimizer.step()
```

Listing 1. Overview of BESTIE

```
def PAM(x, control_values):
    x = F.relu(x) # [B, K, H, W]

    criteria_points = F.adaptive_max_pool2d(x, 1) # [B, K, 1, 1]
    criteria_points = criteria_points.expand_as(x) # [B, K, H, W]

    noisy_region = (x < criteria_points * control_values)
    x[noisy_region] = 0 # deactivate noisy region
    return x
```

Listing 2. PAM Module

```
def extract_peak_points(heatmap, kernel, threshold, K=40):
    B, C, H, W = heatmap.size()

    heatmap_max = F.max_pool2d(heatmap, (kernel, kernel),
                               stride=1, padding=(kernel - 1) // 2)
    keep = (heatmap_max == heatmap).float()

    local_max = heatmap * keep
    peak_points = torch.nonzero(local_max > threshold)
    return peak_points
```

Listing 3. Point Extraction

```
def pseudo_label_gen(seg_map, peak_points):
    # seg_map : [H, W] , peak points : [N, 2]
    mask_candidates = connected_component_labeling(seg_map)

    for mask in mask_candidates:
        if one_peak_point_in_mask(mask, peak_points):
            center_point = centroid(mask)
            center_map_generation(center_point)
            offset_map_generation(center_point, mask)


def offset_map_generation(center_point, mask): # mask: bindary mask
    cy, cx = center_point
    mask_idx = np.where(mask > 0)

    coord = np.ones_like(mask, dtype=np.float32)
    y_coord = np.cumsum(coord, axis=0)-1
    x_coord = np.cumsum(coord, axis=1)-1

    offset_y_index = (np.zeros_like(mask_idx[0]), mask_idx[0], mask_idx[1])
    offset_x_index = (np.ones_like(mask_idx[0]), mask_idx[0], mask_idx[1])

    pseudo_offset_map[offset_y_index] = cy - y_coord[mask_idx]
    pseudo_offset_map[offset_x_index] = cx - x_coord[mask_idx]
```

Listing 4. Pseudo-Label Generation

```
def refined_label_gen(center_map, offset_map, seg_map, thresh):
    instance_masks = instance_grouping(center_map, offset_map, seg_map,
    thresh)

    for mask in instance_masks:
        center_point = centroid(mask)
        center_map_generation(center_point)
        offset_map_generation(center_point, mask)


def instance_grouping(center_map, offset_map, seg_map, thresh):
    ctr = extract_peak_points(center_map, thresh)
    ctr = center_clustering(ctr, offset_map)
    # center_map : [1, C, H, W], ctr : [N, 2], offset_map : [2, H, W]

    H, W = offset_map.size()[1:]
    y_coord = torch.arange(H).repeat(1, W, 1).transpose(1, 2)
    x_coord = torch.arange(W).repeat(1, H, 1)
    coord = torch.cat((y_coord, x_coord), dim=0)

    ctr_loc = coord + offsets
    ctr_loc = ctr_loc.reshape((2, H*W)).transpose(1, 0) # [H*W, 2]

    dist = torch.norm(ctr.unsqueeze(1)-ctr_loc.unsqueeze(0), dim=-1) # [N,
    H*W]

    # finds center with minimum distance at each location
    instance_mask = torch.argmin(dist, dim=0).reshape((1, H, W)) + 1

    return instance_mask
```

Listing 5. Refined Label Generation

```
def objective_function_with_IAG(pred, gt, eps=1e-4):
    gamma_offset, gamma_center, gamma_semantic = 0.01, 200.0, 1.0
    guidance_region = (gt['offset'] != 0).float() # labeled instance region

    offset_map_loss = F.l1_loss(pred['offset'], gt['offset'], reduction='
    none') * guidance_region
```

```
        offset_map_loss = offset_map_loss.sum() / (guidance_region.sum() + eps)

        center_map_loss = F.mse_loss(pred['center'],gt['center'], reduction='
    none') * guidance_region
        center_map_loss = center_map_loss.sum() / (guidance_region.sum() + eps)

        semantic_map_loss = F.cross_entropy(pred['semantic'], gt['semantic'])

        return (offset_map_loss * gamma_offset) + (center_map_loss *
    gamma_center) + (semantic_map_loss * gamma_semantic)
```

Listing 6. Objective Function with IAG

References

[1] Jiwoon Ahn, Sunghyun Cho, and Suha Kwak. Weakly supervised learning of instance segmentation with inter-pixel relations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2209–2218, 2019. 2, 6

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[4] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5693–5703, 2019. 1

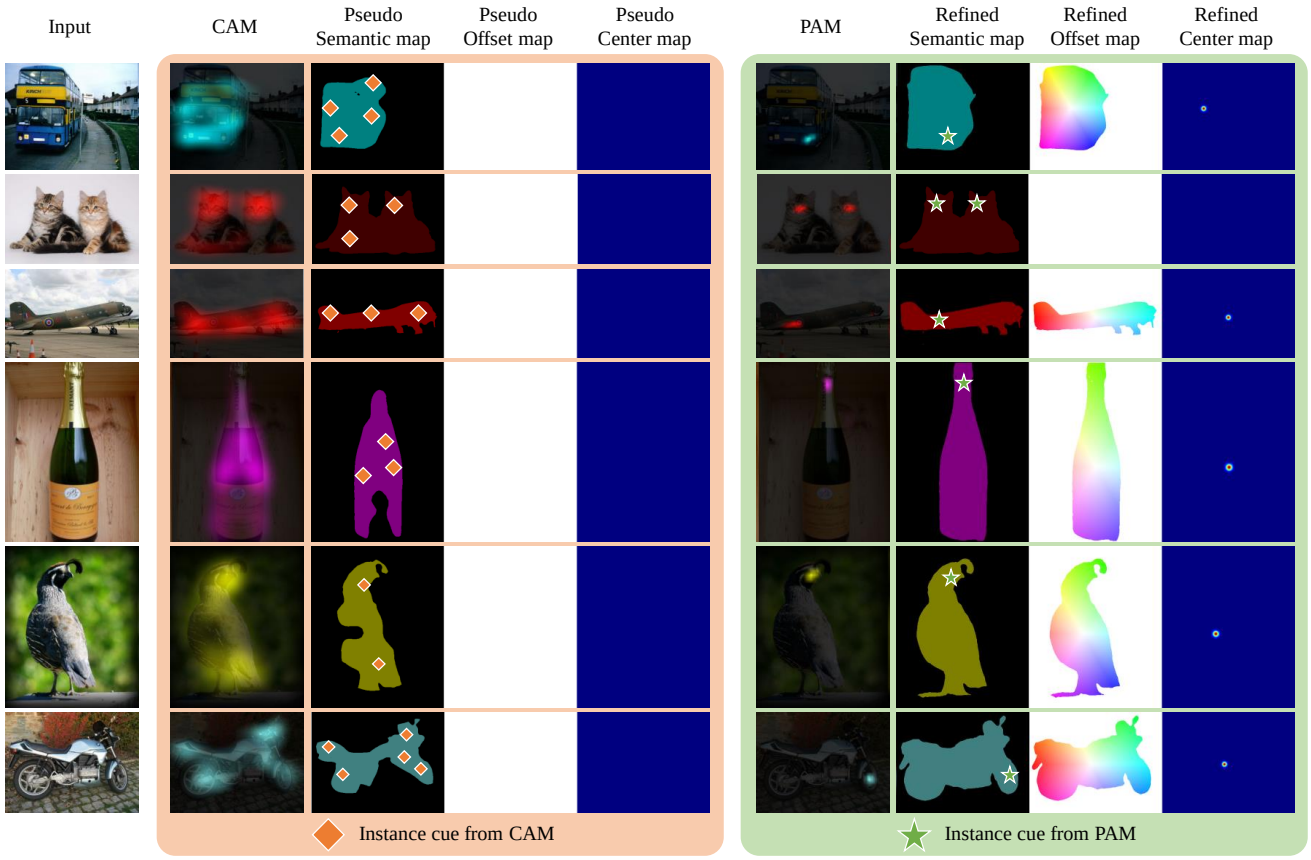

Figure 3. Qualitative results of pseudo labels generated from conventional CAMs (orange region) and pseudo labels generated from our PAM (green region). The PAM can more accurately extract one peak point per instance than the CAM.

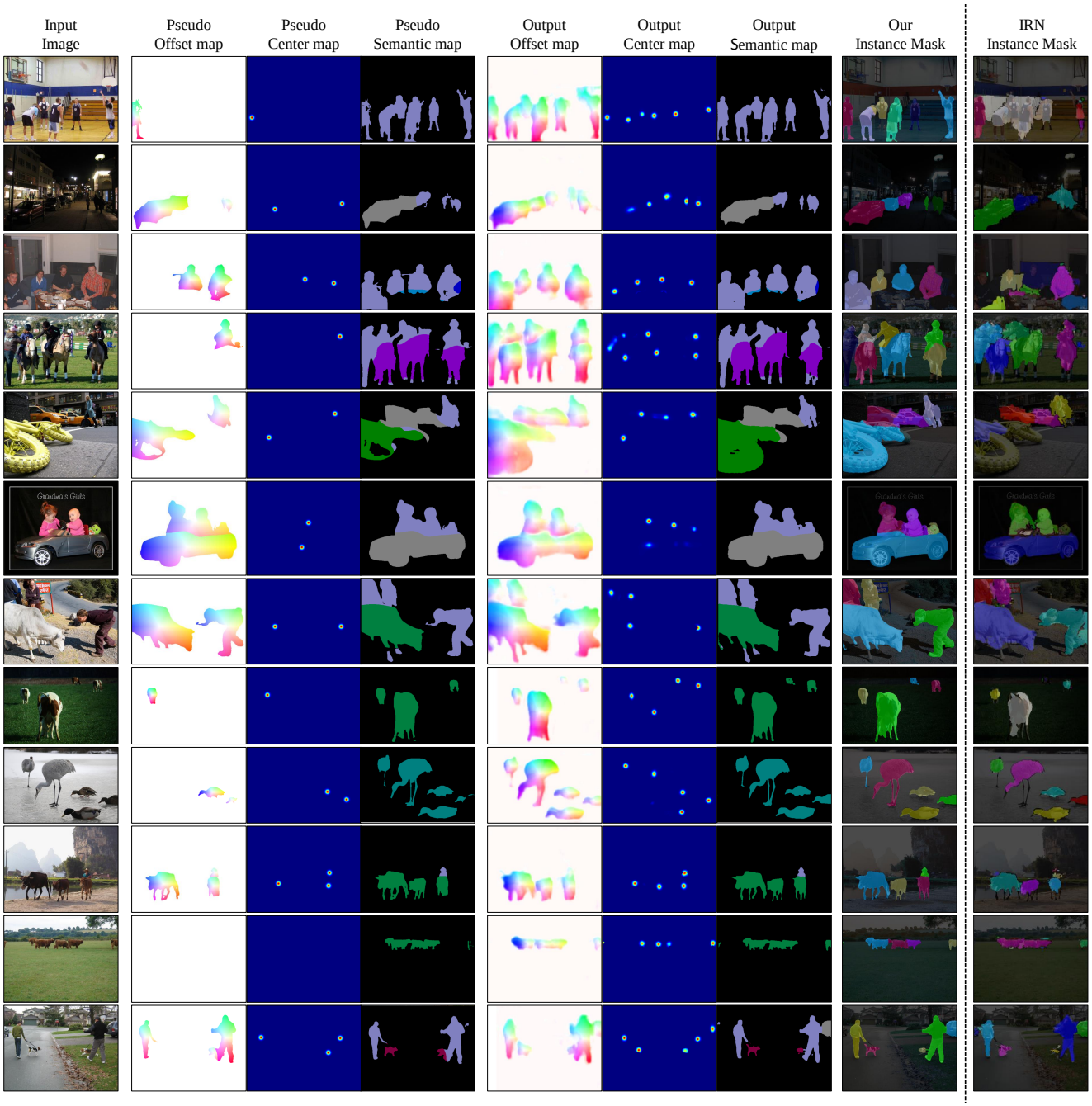

Figure 4. Qualitative results of our pseudo labels and outputs of BESTIE on VOC 2012 dataset. We note that we only use the image-level labels without the off-the-shelf proposal techniques. Compared with IRN [1], which is the proposal-free method, our BESTIE can segment multiple instances more accurately and precisely.

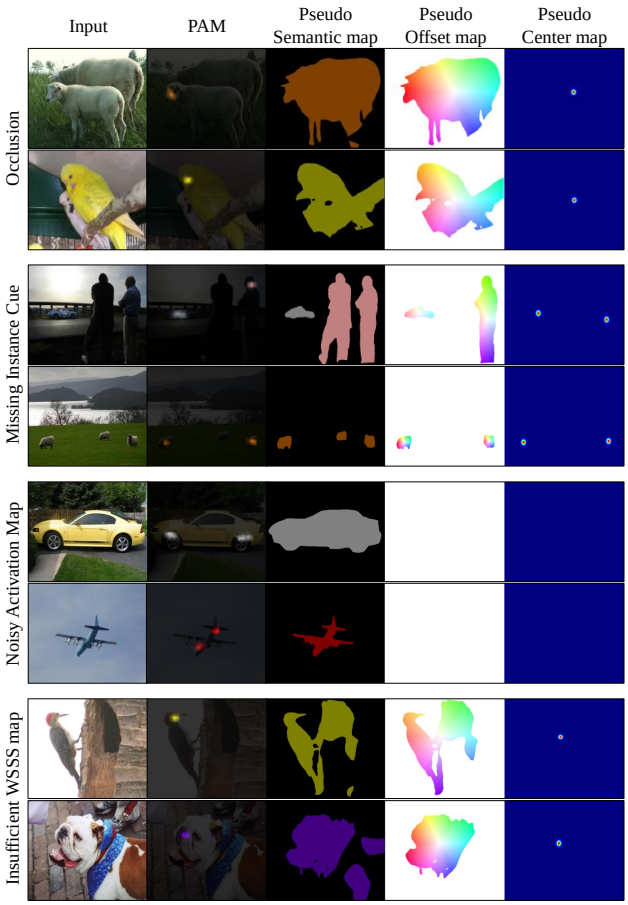

Figure 5. Failure cases of the PAM.

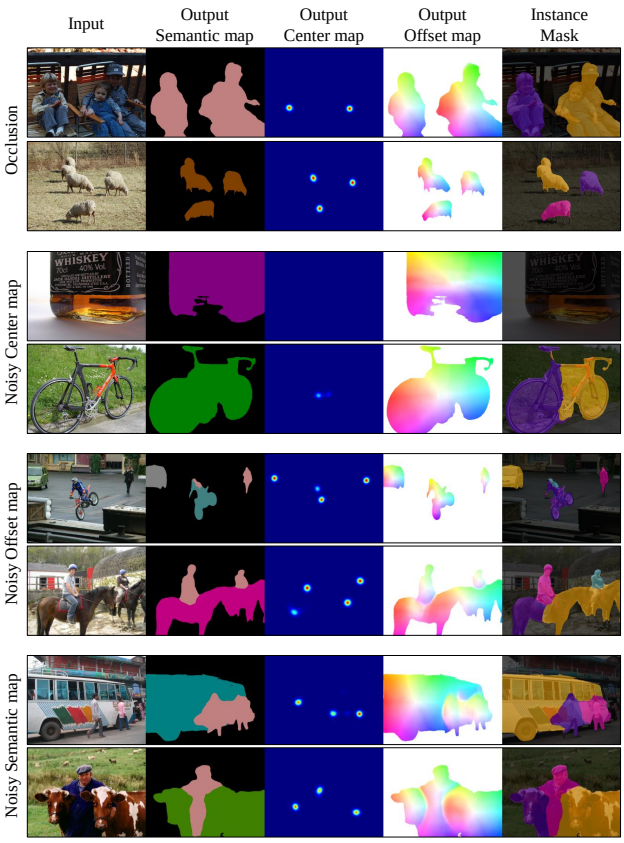

Figure 6. Failure cases of BESTIE.