Single Stage Weakly Supervised Semantic Segmentation of Complex Scenes Supplementary Material

Peri Akiva
Rutgers University
peri.akiva@rutgers.edu

Kristin Dana
Rutgers University
kristin.dana@rutgers.edu

1. Ablation Study

In Tab. 1 we investigate the effects of each component in our proposed method and show its impact on overall performance. It can be seen that thresholding refined features alone is not enough, and that spatially accurate features obtained by the expanding distance fields are essential in generation of better pseudo-masks and performance. Additionally, point blots are shown to provide significant utility compared to points, providing additional contextual information not available otherwise. Note when point blots are not used, points are used instead. If PAC Refiner or Expanding Distance Fields is used, then pseudo-mask is generated from thresholded output (refined or not) features.

Expanding Distance Fields	Point Blot	PAC Refiner	mIoU (%)
	points only		15.2
	✓	✓	24.7
	✓		49.1
✓			38.3
✓	✓		48.9
✓		✓	54.5
✓	✓	✓	**60.7**

Table 1: **Ablation study** on Pascal VOC 2012 validation set [9].

2. Annotation Collection

As mentioned in the main paper, we consider the following datasets: Pascal VOC 2012 [9], Cranberry from Aerial Imagery Dataset (CRAID) [2], CityPersons [22], Inria Aerial Dataset (IAD) [13], ADE20K [24], and CityScapes [6]. Full details about the datasets is provided in Tab. 2.

Given a fully annotated dataset, we obtain points for objects by selecting the center points of bounding boxes or segmentation mask, and points for backgrounds by uniformly sampling four points per object outside of all boxes in a given scene, given background is available (not applicable to ADE20K, CityScapes, or similar). CRAID [2], a computational agriculture dataset, provides 2,835 images with point annotations, and 231 with pixel-wise annotations. CityPersons [22], a pedestrian detection dataset subset of Cityscapes [6], provides 2115 training and 391 testing image with bounding boxes (processed to points similar to Pascal VOC). IAD [13], a remote sensing dataset, provides 180 images (cropped to 29239 images) with pixel-wise annotations (processed to points).

3. Implementation Details

To highlight the contribution of our method, we choose to adopt a standard fully convolutional network (***untrained*** ResNet50 backbone encoder) that is trained from scratch. Note that this is not typical of other baseline methods, in which

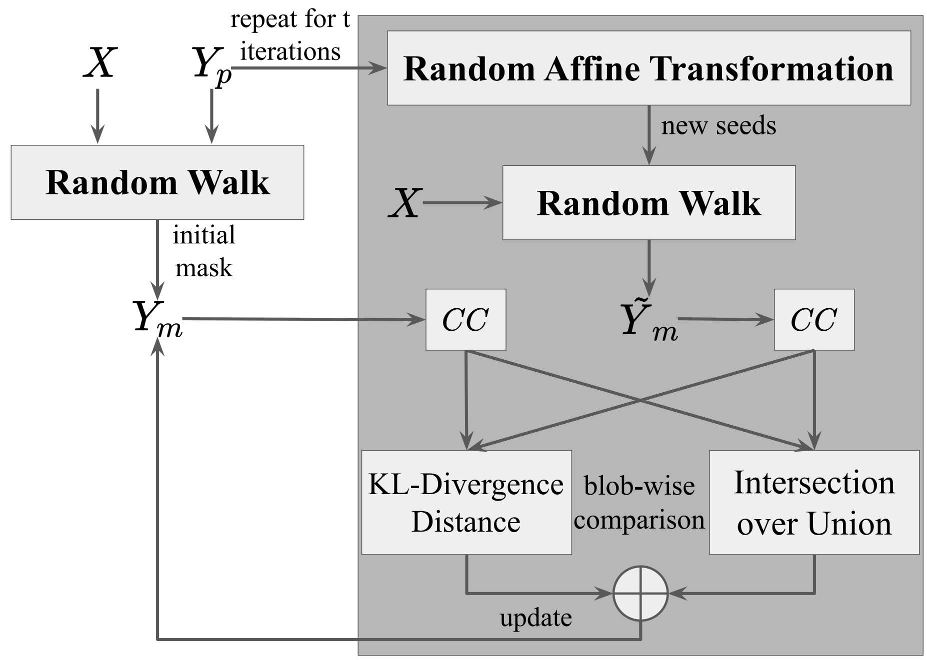

Figure 1: **Point Blot Generator** pipeline. The module generates initial point blots using input image, X, and ground truth points, Y_p. Initial point blots are then iteratively updated conditioned to coverage matching and underlying color distribution similarity of current and candidate blobs. Candidate blobs are generated through perturbations of initial points followed by random walks in color space, which are separated into candidate blobs using the connected component (CC) algorithm [8].

Dataset	Annotations	Complexity	# training images	# validation images	Domain
Pascal VOC 2012 [9]	$\mathcal{F}$	Diversity $\sim$ Count $\downarrow$ Scale $\uparrow$	10,582	1,449	Benchmark
ADE20K [24]	$\mathcal{F}$	Diversity $\uparrow$ Count $\uparrow$ Scale $\downarrow$	20,210	2,000	Complex indoor and outdoors
CityScapes [6]	$\mathcal{F}$	Diversity $\sim$ Count $\uparrow$ Scale $\downarrow$	22,977	500	Autonomous vehicles
CRAID [2]	$\mathcal{P}$	Diversity $\downarrow$ Count $\uparrow$ Scale $\downarrow$	2,835	231	Precision agriculture
IAD [13]	$\mathcal{F}$	Diversity $\downarrow$ Count $\uparrow$ Scale $\sim$	27,777	1,462	Remote sensing
CityPersons [22]	$\mathcal{F}$	Diversity $\downarrow$ Count $\sim$ Scale $\downarrow$	2115	391	Pedestrian detection

Table 2: Datasets explored in this work with corresponding complexity parameters, dataset details, and domain. $\sim\downarrow$, and $\uparrow$ correspond to average, lower end of the parameter range, and upper end of the parameter range.

pre-trained, complex networks (often pre-trained on the benchmark or similar dataset) are used to achieve SOTA performance. Our network is trained using the SGD optimizer, with starting learning rate of 1e-5 and cosine annealing scheduler [12]. We use weight standardization [20] and group normalization layers [16] with group size of 32. Training data is augmented with normalization transformation, color jittering, and random vertical and horizontal flips. We use Cross Entropy loss for training, with "0" labels ignored (background points, labeled as $\mathcal{C}+1$, are considered instead). For the PAC Refinement Network, we use 10 layers with kernel sizes $(5,5,3,3,3,3,3,3,3,3)$, dilations $(1,1,2,2,4,4,8,8,16,24)$, and strides $(2,2,2,2,1,1,1,1,1,1,1,1)$. We use $-0.025, 0.025$ for lower and upper limits for the Expanding Distance Fields, and 0.75 for pseudo-mask thresholding. The Point Blot generation has two sets of parameters, depending on the system pipeline. For run-time generation, we use $k=2$, $\lambda=0.5$, $\phi=0.2$, with random walk parameters $beta=90$, $tol=0.01$, and $prob=0.9$. Those parameters ensure faster execution of the Point Blotter, with faster random walk convergence and small number of iterations. This can also be done as a deterministic data pre-processing step (which is different than pre-training steps), in which case more constraining parameters can be used at the cost of longer processing time. In our implementation, we use $beta=200$, $tol=0.0001$, and $prob=0.85$ at a significant time cost increase (For performance evaluation, we report mean Intersection over Union (mIoU) for both validation and test sets. Note that all experiments reported in the main paper are done in a single stage, without pruning or eliminating output predictions. Baseline method for real world datasets was trained in accordance with the method's reported procedure. Further implementation details and pseudo-code is available in the following sub-sections. Full code will be released upon publication.

3.1. Pixel Adaptive Convolution Refinement Network

```python

class PACRN(nn.Module):

    def __init__(self, num_iter=10, dilations=[1]):
        super(PACRN, self).__init__()

        self.num_iter = num_iter
        self.pac_x = PACL2(dilations)
        self.pac_m = LaplacianBaseKernel(dilations)
        self.pac_std = PACStd(dilations)
        self.pac_mean = PACMean(dilations)

    def forward(self, x, mask):
        # x: [B,3,H,W]
        # mask: [B,C,H,W]
        B,K,H,W = x.size()

        x_std = self.pac_std(x)
        mask_mean = self.pac_mean(x)

        x = -(self.pac_x(x) * mask_mean) / (1e-8 + 1.0 * x_std)

        x = x.mean(1, keepdim=True)
        x = F.softmax(x, 2)

        for _ in range(self.num_iter):
            m = self.pac_m(mask)
            mask = (m * x)
            mask = mask.sum(2)

        return mask
```

Listing 1: **Pixel Adaptive Convolution Refinement Network Simplified Pseudo-Code**

3.2. Point Blot Generator

```python


class PointBlotter(object):
    def combine_masks(self, image, current_pmask,
                      labels_from_image, padding: int =10):
        """Combine current and proposal masks based on IoU and KL Div. Distance thresholds

        Args:
            image: input image
            current_pmask: current mask
            labels_from_image: proposal mask from perturbed points
            padding (int, optional): padding around blobs. Defaults to 10.

        Returns:
            np.array: mask which is either current or combined
        """
        mask = current_pmask.copy()
        height, width = current_pmask.shape[0], current_pmask.shape[1]
        blobs_from_perturbed_image, nblobs_image = ndimage.label(labels_from_image)
        blobs_from_pmask, nblobs_pmask = ndimage.label(current_pmask)

        for label in range(1, nblobs_pmask+1):

            comparable_region_label_image = collect_pixels()
            comparable_region_label_pmask = collect_pixels()

            comparable_region_rgb_image = collect_pixels()
            comparable_region_rgb_pmask = collect_pixels()

            # blob IoU calculations
            iou_tmp = IoU(comparable_region_label_image,
                          comparable_region_label_pmask,
                          num_classes=self.num_classes)

            # KL Divergence Distance
            kl_dist = entropy(comparable_region_rgb_image, comparable_region_rgb_pmask)

            if iou_tmp > self.iou_thresh and kl_dist < self.kl_dist_thresh:
                tmp_mask = np.add(comparable_region_label_image, comparable_region_label_pmask)
                tmp_mask[tmp_mask != 0] = real_label
                mask[overlap_locations] = tmp_mask

        return mask

    def generate(self, image, points_mask):
        """Given points, perturb and combine proposal regions

        Args:
            image: input image
            points_mask: initial point annotations
        """

        ### Generate initial mask under strict constraints
        mask = random_walker(image, points_mask,
                             beta=self.beta, mode=self.mode,
                             multichannel=True, tol=self.tol,
                             return_full_prob=self.return_full_prob)

        # remove background regions
```

```
        if self.consider_background:
            mask[mask == mask.max()] = 0

        ### Iterate over number of perturbations
        for index in range(self.num_of_perturbations):

            # define increasing ranges of perturbations
            base_translation = 2*index
            base_angle = 2*index
            random_translation_x = random.randint(-base_translation, base_translation)
            random_translation_y = random.randint(-base_translation, base_translation)
            random_angle = random.randint(-base_angle, base_angle)

            # apply affine transformation of points
            perturbed_points_mask = affine_transformation(points_mask, angle=random_angle,
                                                          translate=(random_translation_x, random_translation_y),
                                                          scale=1.0, shear=0, fillcolor=0)

            # generate a mask proposal using a random walk
            mask_proposal = random_walker(image, perturbed_points_mask, beta=self.beta, mode='cg_mg',
                                          multichannel=True, tol=self.tol, return_full_prob=False)

            # remove background proposals
            mask_proposal[mask_proposal == mask_proposal.max()] = 0

            # check if proposal mask should be added to the current mask
            mask = self.combine_masks(image=image, current_pmask=mask,
                                      labels_from_image=mask_proposal)

        if self.consider_background:
            mask[background_points_inds[:, 0], background_points_inds[:, 1]] = self.num_classes-1

        return mask
```

Listing 2: **Point Blot Generator Simplified Pseudo-Code**

3.3. Expanding Distance Field

```


class ExpandingDistanceMapper(object):
    """Given set of points, generate distance fields"""

    def get_distance_map(self, points_mask, image, labels_logits,
                         dm_confidence, bg_dm_confidence):
        """generate expanding distance map for batch images

        Args:
            points_mask: points mask [b,c,h,w]
            image: input image [b,3,h,w]
            labels_logits: logits one hot encoding of labels
            dm_confidence (float, optional): distance map confidence score for objects with confidence. Defaults to 0.
            bg_dm_confidence (float, optional): distance map confidence score for background. Defaults to 0.

        Returns:
            distance maps of batch for each class with shape [b,c,h,w]
        """

        batch_size, height, width = mask.shape
        distance_maps = torch.zeros((batch_size, self.num_classes, height, width))
        for b in range(batch_size):
            mask_b = points_mask[b,:,:]
            labels_logits_b = labels_logits[b,:]
            classes_in_mask_b = torch.where(labels_logits_b==1)[0]

            # If background class exists, use those points
            # If background class does not exists, we use all other points as background
            if self.background_class_label:
                background_points = (mask_b==self.background_class_label).nonzero()
                neg_distance_map = distanceTransform(background_points)
                neg_distance_map = utils.normalize_dm(neg_distance_map,
                                                      confidence_score=bg_dm_confidence)
                neg_distance_map[neg_distance_map>1] = 1
                neg_distance_map[neg_distance_map<0] = 0

            for label in classes_in_mask_b:
                Y_1 = torch.zeros((height, width))
                label_points = (mask_b==label).nonzero()

                if label_points.shape[0]==0:
                    # This covers the case in which a random crop is applied,
                    # and a class is now not visible in the crop.
                    pos_distance_map = np.ones((height, width))/3
                else:


                    # Generate distance map for points
                    pos_distance_map = distanceTransform(label_points,)

                    pos_distance_map = utils.normalize_dm(pos_distance_map,
                                                          confidence_score=dm_confidence)

                    pos_distance_map[pos_distance_map>1] = 1
                    pos_distance_map[pos_distance_map<0] = 0

                if self.background_class_label:
                    combined = neg_distance_map*pos_distance_map
                else:
                    background_condition = ((mask_b!=label) & (mask_b != 0))
                    background_points = (background_condition).nonzero()
                    neg_distance_map = distanceTransform(background_points,)
                    neg_distance_map = utils.normalize_dm(neg_distance_map,
```

```
                                                    confidence_score=bg_dm_confidence)
                    neg_distance_map[neg_distance_map>1] = 1
                    neg_distance_map[neg_distance_map<0] = 0
                    combined = neg_distance_map*pos_distance_map

                distance_maps[b, label, :, :] = torch.from_numpy(combined)

        return distance_maps
```

Listing 3: **Expanding Distance Field Simplified Pseudo-Code**

4. Additional Qualitative and Quantitative Results

This supplementary material provides class-wise mIoU for validation (Table 3) and test (Table 5) sets. It also presents additional qualitative results for Pascal VOC 2012 validation set (Figure 2), epoch-by-epoch pseudo-mask progression (Figure 5), and additional qualitative results for CRAID [2], IAD [13], and CityPersons [22] datasets (Figure 3).

Figure 2: Additional qualitative results of our method on Pascal VOC 2012 [9]. Best viewed in color and zoomed. Dark gray pixels represent background class.

Method	bkg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbk	person	plant	sheep	sofa	train	tv	mIoU
Multi Stage																						
FickleNet [11]	89.5	76.6	32.6	74.6	51.5	71.1	83.4	74.4	83.6	24.1	73.4	47.4	78.2	74.0	68.8	73.2	47.8	79.9	37.0	57.3	64.6	64.9
AffinityNet [1]	88.2	68.2	30.6	81.1	49.6	61.0	77.8	66.1	75.1	29.0	66.0	40.2	80.4	62.0	70.4	73.7	42.5	70.7	42.5	68.1	51.6	61.7
SSDD [18]	89.0	62.5	28.9	83.7	52.9	59.5	77.6	73.7	87.0	34.0	83.7	47.6	84.1	77.0	73.9	69.6	29.8	84.0	43.2	68.0	53.4	64.9
SEAM [19]	88.8	68.5	33.3	85.7	40.4	67.3	78.9	76.3	81.9	29.1	75.5	48.1	79.9	73.8	71.4	75.2	48.9	79.8	40.9	58.2	53.0	64.5
Single Stage																						
MIL+LSE [15]	79.6	50.2	21.6	40.9	34.9	40.5	45.9	51.5	60.6	12.6	51.2	11.6	56.8	52.9	44.8	42.7	31.2	55.4	21.5	38.8	36.9	42.0
CRF-RNN [17]	85.8	65.2	29.4	63.8	31.2	37.2	69.6	64.3	76.2	21.4	56.3	29.8	68.2	60.6	66.2	55.8	30.8	66.1	34.9	48.8	47.1	52.8
Araslanov *et al*. [3]	87.0	63.4	33.1	64.5	47.4	63.2	70.2	59.2	76.9	27.3	67.1	29.8	77.0	67.2	64.0	72.4	46.5	67.6	38.1	68.2	63.6	59.7
Ours	88.1	69.6	22.0	57.8	55.6	59.4	59.4	59.6	78.1	30.9	76.7	59.2	73.8	69.7	51.6	59.2	47.1	75.8	54.7	69.8	56.8	60.7
Single Stage + CRF																						
Araslanov *et al*. + CRF [3]	88.7	70.4	35.1	75.7	51.9	65.8	71.9	64.2	81.1	30.8	73.3	28.1	81.6	69.1	62.6	74.8	48.6	71.0	40.1	68.5	64.3	62.7
Ours + CRF	88.9	69.8	24.0	66.4	58.2	62.4	61.1	64.1	78.6	31.3	78.0	59.3	74.3	71.2	55.3	61.6	51.1	76.1	57.8	71.0	59.6	62.9

Table 3: Class-wise Mean Intersection over Union (%) accuracy (higher is better) on Pascal VOC 2012 validation set [9].

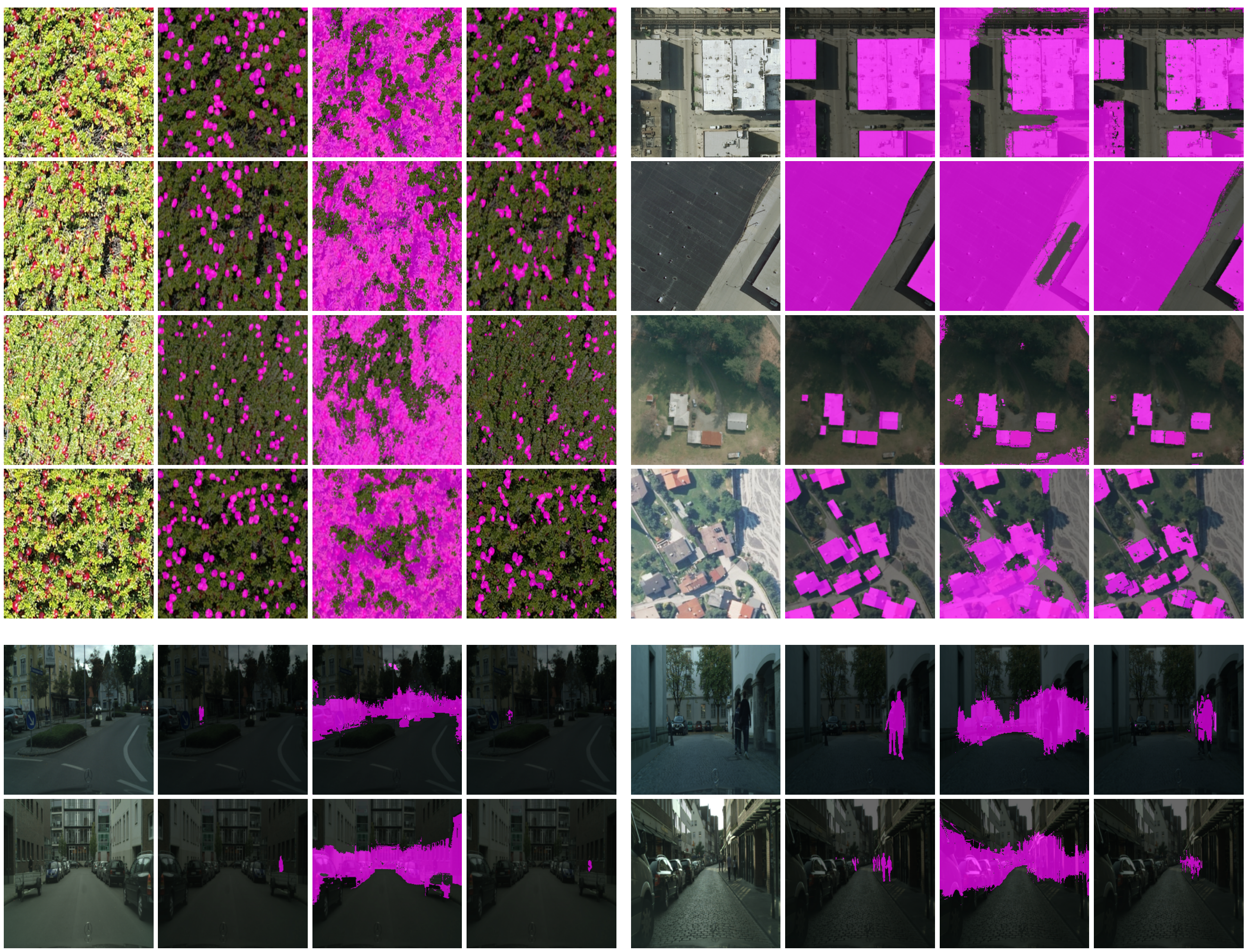

Image | GT | Araslanov *et al*. [3] | Ours | Image | GT | Araslanov *et al*. [3] | Ours

Figure 3: Additional qualitative results of our method on CRAID [2] (top left), IAD [13] (top right), and CityPersons [22] (bottom). It can be seen that our method provides superior results for all real-world datasets. Best viewed in color and zoomed. Dark gray pixels represent background class.

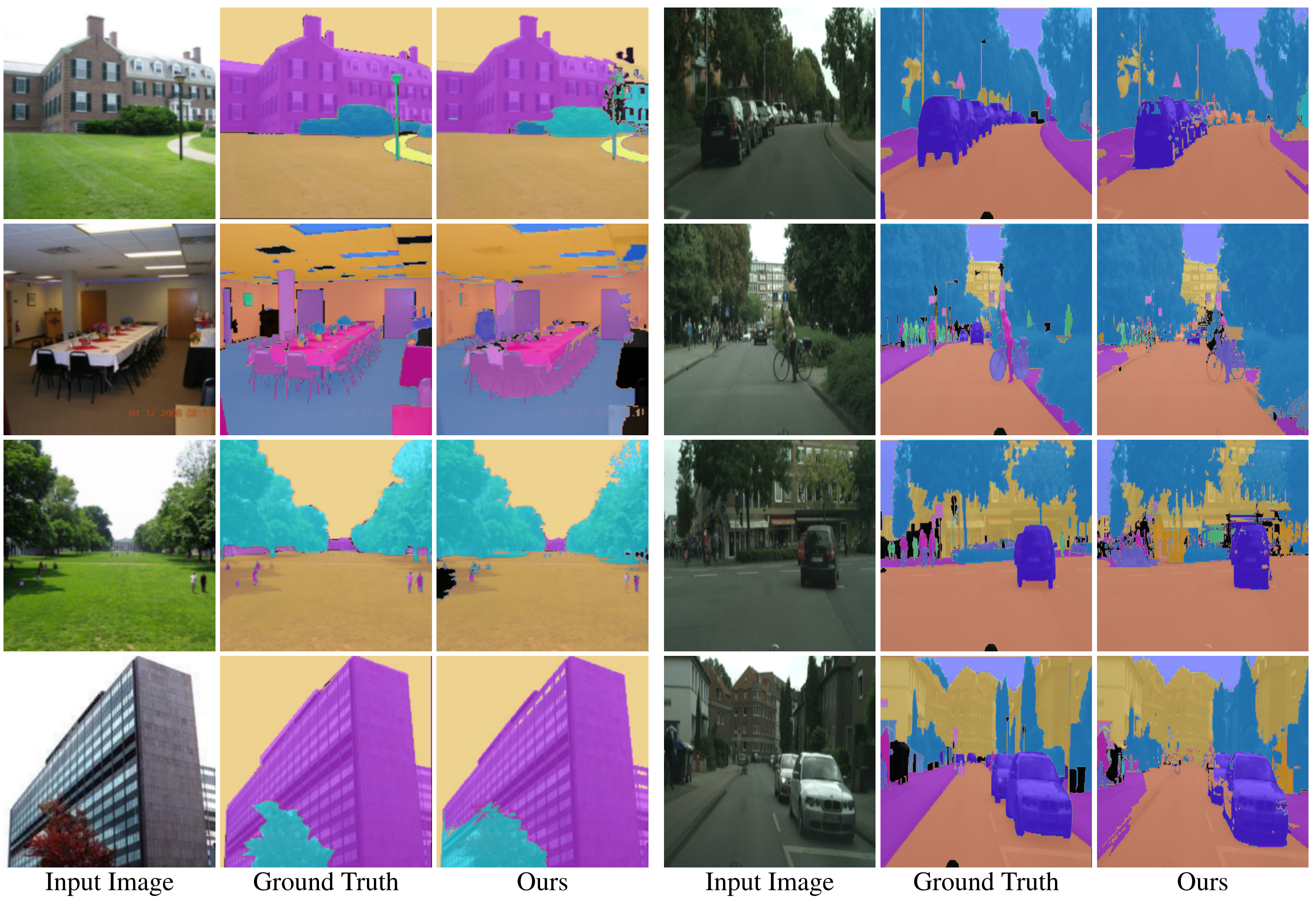

Figure 4: Additional qualitative results of our method on ADE20K [24] (left), and CityScapes [6] (right). Best viewed in color and zoomed. Dark gray pixels represent background class.

Dataset	Pascal VOC 2012 [9]			
Method	Sup.	# of stages	val	test
Single Stage, Full Supervision				
WideResNet38 [21]	$\mathcal{F}$	1	80.8	82.5
DeepLab v3 [5]	$\mathcal{F}$	1	-	87.8
Multi Stage				
Bearman *et al*. [4]	$\mathcal{S}, \mathcal{P}$	3	46.0	43.6
BoxSup [7]	$\mathcal{S}, \mathcal{B}$	3	62.0	64.6
AffinityNet [1]	$\mathcal{I}$	4	61.7	63.7
SEAM [19]	$\mathcal{I}$	4	64.5	65.7
SMPL [10]	$\mathcal{I}$	9	69.5	71.6
SMPL [10]	$\mathcal{B}$	5	73.5	74.7
Single Stage				
EM [14]	$\mathcal{I}$	1	38.2	39.6
MIL-LSE [15]	$\mathcal{I}$	1	42.0	40.6
CRF-RNN [23]	$\mathcal{I}$	1	52.8	53.7
Araslanov *et al*. [3]	$\mathcal{I}$	1	59.7	60.5
Ours	$\mathcal{P}$	1	60.7	60.8

Table 4: mIoU (%) accuracy (higher is better) on Pascal VOC 2012 validation and test sets [9]. $\mathcal{F}$, $\mathcal{I}$, $\mathcal{B}$, $\mathcal{S}$, and $\mathcal{P}$ represent full, image, box, saliency, and point level annotations respectively. Our method achieves SOTA performance compared to our single-stage weakly supervised baselines, even with from-scratch training. Class-wise performance is available in supplementary material.

Method	bkg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbk	person	plant	sheep	sofa	train	tv	mIoU
Multi Stage																						
FickleNet [11]	89.8	78.3	34.1	73.4	41.2	67.2	81.0	77.3	81.2	29.1	72.4	47.2	76.8	76.5	76.1	72.9	56.5	82.9	43.6	48.7	64.7	65.3
AffinityNet [1]	89.1	70.6	31.6	77.2	42.2	68.9	79.1	66.5	74.9	29.6	68.7	56.1	82.1	64.8	78.6	73.5	50.8	70.7	47.7	63.9	51.1	63.7
SSDD [18]	89.5	71.8	31.4	79.3	47.3	64.2	79.9	74.6	84.9	30.8	73.5	58.2	82.7	73.4	76.4	69.9	37.4	80.5	54.5	65.7	50.3	65.5
Single Stage																						
MIL+LSE [15]	78.7	48.0	21.2	31.1	28.4	35.1	51.4	55.5	52.8	7.8	56.2	19.9	53.8	50.3	40.0	38.6	27.8	51.8	24.7	33.3	46.3	40.6
Araslanov *et al*. [3]	87.4	63.6	34.7	59.9	40.1	63.3	70.2	56.5	71.4	29.0	71.0	38.3	76.7	73.2	70.5	71.6	55.0	66.3	47.0	63.5	60.3	60.5
Ours	88.5	70.0	22.7	57.2	51.3	58.6	58.9	57.3	77.2	30.9	77.5	60.2	73.6	70.6	54.9	58.8	52.4	76.6	56.2	67.9	55.3	60.8
Single Stage + CRF																						
Araslanov *et al*. + CRF [3]	89.2	73.4	37.3	68.3	45.8	68.0	72.7	64.1	74.1	32.9	74.9	39.2	81.3	74.6	72.6	75.4	58.1	71.0	48.7	67.7	60.1	64.3
Ours + CRF	89.1	72.2	25.1	62.9	56.2	61.7	60.8	61.5	79.5	33.3	78.8	59.8	76.8	74.5	59.1	62.8	58.2	77.0	58.9	70.9	59.8	63.8

Table 5: Class-wise Mean Intersection over Union (%) accuracy (higher is better) on Pascal VOC 2012 test set [9].

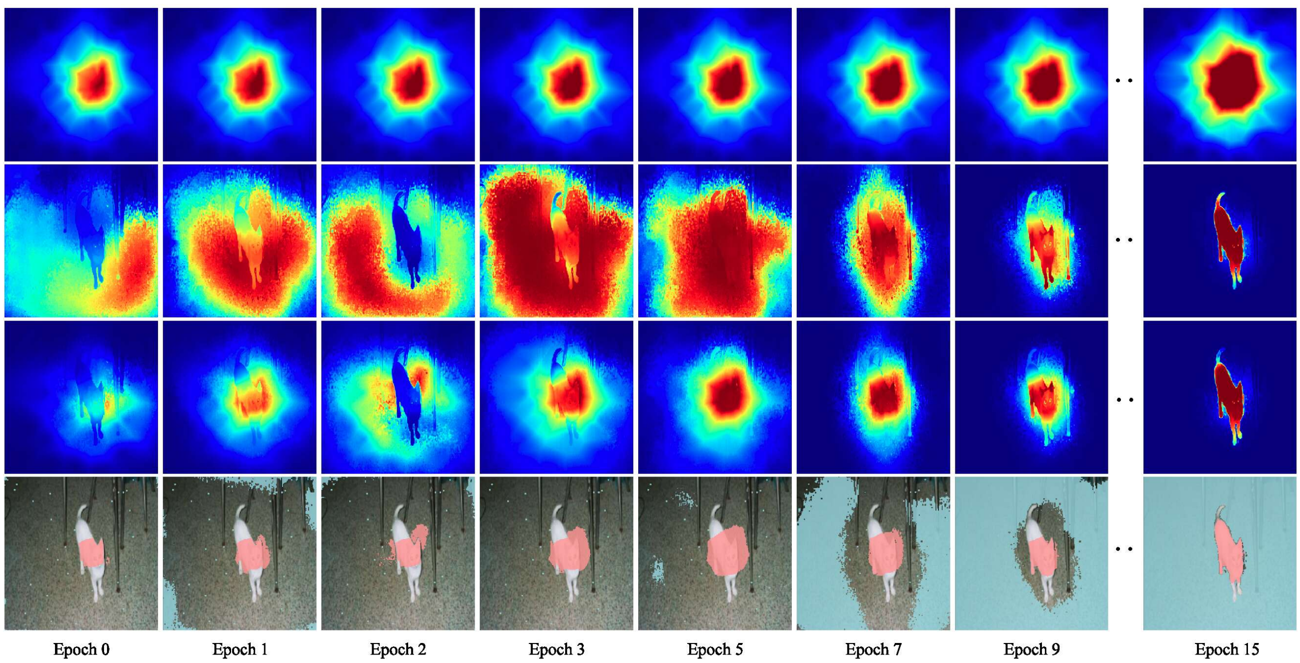

Figure 5: From top to bottom rows: Expanding Distance Fields, refined features, filtered refined features, and final pseudo-mask. Epoch-by-epoch progressions of generated pseudo-mask with corresponding expanding distance field. It can be seen that in early epochs, when features are not well defined, the expanding distance fields prevents inclusion of "bad" features in the intermediate pseudo-mask. As features improve, the expanding distance field allows more features in the final output. The initial pink pseudo-mask is the point blot. Light blue pixels represent background class.

References

[1] Jiwoon Ahn and Suha Kwak. Learning pixel-level semantic affinity with image-level supervision for weakly supervised semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4981–4990, 2018.

[2] Peri Akiva, Kristin Dana, Peter Oudemans, and Michael Mars. Finding berries: Segmentation and counting of cranberries using point supervision and shape priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 50–51, 2020.

[3] Nikita Araslanov and Stefan Roth. Single-stage semantic segmentation from image labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4253–4262, 2020.

[4] Amy Bearman, Olga Russakovsky, Vittorio Ferrari, and Li Fei-Fei. What's the point: Semantic segmentation with point supervision. In *European conference on computer vision*, pages 549–565. Springer, 2016.

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[6] 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 *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] Jifeng Dai, Kaiming He, and Jian Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1635–1643, 2015.

[8] Michael B Dillencourt, Hanan Samet, and Markku Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM (JACM)*, 39(2):253–280, 1992.

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[10] Tsung-Wei Ke, Jyh-Jing Hwang, and Stella X Yu. Universal weakly supervised segmentation by pixel-to-segment contrastive learning. *arXiv preprint arXiv:2105.00957*, 2021.

[11] Jungbeom Lee, Eunji Kim, Sungmin Lee, Jangho Lee, and Sungroh Yoon. Ficklenet: Weakly and semi-supervised semantic image segmentation using stochastic inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5267–5276, 2019.

[12] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[13] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3226–3229. IEEE, 2017.

[14] George Papandreou, Liang-Chieh Chen, Kevin P Murphy, and Alan L Yuille. Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1742–1750, 2015.

[15] Pedro O Pinheiro and Ronan Collobert. From image-level to pixel-level labeling with convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1713–1721, 2015.

[16] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

[17] Anirban Roy and Sinisa Todorovic. Combining bottom-up, top-down, and smoothness cues for weakly supervised image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3529–3538, 2017.

[18] Wataru Shimoda and Keiji Yanai. Self-supervised difference detection for weakly-supervised semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5208–5217, 2019.

[19] Yude Wang, Jie Zhang, Meina Kan, Shiguang Shan, and Xilin Chen. Self-supervised equivariant attention mechanism for weakly supervised semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12275–12284, 2020.

[20] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[21] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90:119–133, 2019.

[22] Shanshan Zhang, Rodrigo Benenson, and Bernt Schiele. Citypersons: A diverse dataset for pedestrian detection. In *CVPR*, 2017.

[23] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr. Conditional random fields as recurrent neural networks. In *International Conference on Computer Vision (ICCV)*, 2015.

[24] 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.