Appendix

A. Further Illustration of Position-aware Dynamic Convs

We have described the bounding box and centerness-based positional encoding in the paper. Here, we provide the pseudo-code of dynamic convs with the two kinds of PE in Algorithm 1.

**Algorithm 1 Position-aware Dynamic Convs (PyTorch)**

```python
# q: proposal features, (N, c)
# r: roi features, (N, 7 * 7, c)
# b: proposal boxes(x, y, w, h), (N, 4)
# m: centerness, (7 * 7)
# pe_i: PE of image coordinates, (N, 7 * 7, c)
def PositionAwareDynamicConvs(q, r, b, pe_i, m):
    # kernels of two 1x1 conv layers
    params = dynamic_layer(q)
    k = params[:, :c*d].view(N, c, d)
    v = params[:, c*d:].view(N, d, c)
    # encode box center (x, y) into PE vector
    center, w, h = b[:, :2], b[:, 2], b[:, 3]
    qc = mlp_c(q)  # center transformation
    qs = mlp_s(q)  # w_ref, h_ref
    pe = sinusoidal(center)
    pe = pe * qc
    pe[:, :c/2] = qs[:, 0] / w * pe[:, :c/2]
    pe[:, c/2:] = qs[:, 1] / h * pe[:, c/2:]  # modulate r and PE by centerness
    pe_i = pe_i * m.flatten()[None, :, None]
    pe = pe.view(N, 1, c) * m.flatten()[None, :, None]
    # concate (r, pe_img) and (k, pe)
    r = torch.cat([r, pe_i], dim=-1)
    k = k.unsqueeze(1).repeat(1, 7*7, 1, 1)
    pe = pe.unsqueeze(-1).repeat(1, 1, 1, d)
    k = torch.cat([k, pe], dim=2)
    # interaction between r and q
    r = relu(norm(torch.matmul(r.unsqueeze(-2), k).squeeze(-2)))
    r = relu(norm(bmm(r, v)))
    # reduce spatial dimension to obtain object
    features_o = out_layer(r)
    return o
```

Two conv kernels are generated from the proposal feature first. Then we map the proposal feature to two vectors in geometry space. One of them is $q_c \in \mathbb{R}^c$, the another is $q_s \in \mathbb{R}^2$, consisting of two scaler $w_{ref}$ and $h_{ref}$. Then they work together to modulate the sinusoidal embedding from box center $(x, y)$ as Eq.(5).

The centerness-based PE is built on the bounding box PE to vary local positions within a proposal box. We also list it in Algorithm 1. The single channel mask is generated as Eq.(6), and it is shown in Fig. 1(a). It is the same for all proposal boxes. We try two strategies to enhance it. First, we make the centerness-based mask trainable, initialized with Fig. 1(a). Second, as different objects have different shapes and semantic centers, we predict the semantic center coordinate. The largest value of centerness mask is at the predicted semantic center as Fig. 1(b). However, the static one in Fig. 1(a) gives the best results.

B. Effect of Number of Stages

We provide the effectiveness of the number of stages on Sparse R-CNN and RecursiveDet in Fig.4 in the paper. It shows our model gains better results on any number of stages. We further present the comparisons on AdaMixer and DiffusionDet in Fig. 2 and Fig. 3. Except for the first stage in DiffusionDet, RecursiveDet behaves better than its counterpart.

Fig. 4 gives the visualization of predicted boxes from cascade structure Sparse R-CNN and recursive structure RecursiveDet. It shows that both methods detect objects progressively, and the recursive structure even gets a better final result than Sparse R-CNN. Similar phenomenon is illustrated in Fig. 5 and Fig. 6.

C. Results on CrowdHuman

CrowdHuman [7] dataset is a highly crowd pedestrian benchmark with only one class. There are about 23 persons per image on average and many overlaps. We only use
the full-body bounding box to train models. The standard Caltech[2] evaluation metric, Average Precision (AP) and Recall are reported. We train the model for 50 epochs, and dividing the learning rate by 10 at 40th epoch. The results are listed in Tab. 1. It shows that Sparse R-CNN is already better than other well established detectors, like RetinaNet and AdaptiveNMS. It also behaves better than DETR-series. Our enhanced version achieves 91.9 AP, gains 1.1 AP from Sparse R-CNN. The other two metric mMR and Recall are also better than it.

<table>
<thead>
<tr>
<th>Method</th>
<th>Queries</th>
<th>NMS</th>
<th>AP</th>
<th>mMR</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster R-CNN [6]</td>
<td>-</td>
<td>✓</td>
<td>85.0</td>
<td>50.4</td>
<td>90.2</td>
</tr>
<tr>
<td>RetinaNet [4]</td>
<td>-</td>
<td>✓</td>
<td>81.7</td>
<td>57.6</td>
<td>88.6</td>
</tr>
<tr>
<td>FCOS [9]</td>
<td>-</td>
<td>✓</td>
<td>86.1</td>
<td>55.2</td>
<td>94.3</td>
</tr>
<tr>
<td>AdaptiveNMS [5]</td>
<td>-</td>
<td>✓</td>
<td>84.7</td>
<td>49.7</td>
<td>91.3</td>
</tr>
<tr>
<td>DETR [1]</td>
<td>100</td>
<td>○</td>
<td>66.1</td>
<td>80.6</td>
<td></td>
</tr>
<tr>
<td>Deformable DETR [10]</td>
<td>400</td>
<td>○</td>
<td>86.7</td>
<td>54.0</td>
<td>92.5</td>
</tr>
<tr>
<td>PETR [3]</td>
<td>1000</td>
<td>○</td>
<td>89.5</td>
<td>45.6</td>
<td>94.0</td>
</tr>
<tr>
<td>Sparse R-CNN [8]</td>
<td>500</td>
<td>○</td>
<td>90.7</td>
<td>44.7</td>
<td>81.4</td>
</tr>
<tr>
<td>RecursiveDet</td>
<td>500</td>
<td>○</td>
<td>91.8</td>
<td>43.3</td>
<td>96.7</td>
</tr>
</tbody>
</table>

Table 1. Performance comparisons with other detectors on CrowdHuman. Our RecursiveDet is built on Sparse R-CNN.

Figure 3. Effect of the number of stages in DiffusionDet and RecursiveDet.

References


Figure 4. Predictions of each stage in Sparse R-CNN and RecursiveDet. Boxes with classification score above 0.5 are showed.
Figure 5. Predictions of each stage in Sparse R-CNN and RecursiveDet. Boxes with classification score above 0.5 are showed.
Figure 6. Predictions of each stage in Sparse R-CNN and RecursiveDet. Boxes with classification score above 0.5 are showed.