DetCo: Unsupervised Contrastive Learning for Object Detection

Enze Xie[1*], Jian Ding[3*], Wenhai Wang[4], Xiaohang Zhan[5],
Hang Xu[2], Peize Sun[1], Zhenguo Li[2], Ping Luo[1]
[1]The University of Hong Kong [2]Huawei Noah's Ark Lab
[3]Wuhan University [4]Nanjing University [5]Chinese University of Hong Kong

A. Appendix

In appendix, we first show that the results of DetCo on Semi-Supervised Object Detection in Section A.1 and more downstream tasks in Section A.2. Second, in Section A.3, we show the visualization results of DetCo and MoCo v2. Third, we show more implementation details in Section A.4. Finally, we analysis more ablation studies of DetCo in Section A.5.

A.1. Semi-Supervised Object Detection

To verify the effectiveness of self-supervised learning on small scale dataset, we randomly sample 1%, 2%, 5%, 10% data to fine-tune the Mask RCNN C4 / FPN. For all the settings, we fine-tune the detectors with 12k iterations to avoid overfitting. Other settings are the same as COCO 1× and 2× schedule. The results for Mask RCNN with 1% and 2% data are shown in Table I. The results for Mask RCNN with 5% and 10% data are shown in Table II. From Table I, we find that with only 1% and 2% data, all other unsupervised methods have lower results than supervised counterparts. However, DetCo performs better than all supervised / unsupervised methods. For example, DetCo is **2.5** AP and 4.8 AP_{50} higher than MoCo v2 with 1% data. Moreover, for 5% and 10% training data, DetCo also outperforms all the counterparts with a large margin, *e.g.* 1.9 AP higher than MoCo v2 with 10% data. These results shows that the feature representation pre-trained from self-supervised learning approach is beneficial for semi-supervised object detection.

A.2. More Experiment Results

COCO with 2× schedule. Table III shows the results of Mask RCNN R50 C4 / FPN on COCO with 2× schedule. DetCo achieves state-of-the-art performance on both object detection and instance segmentation. For example, for Mask RCNN-C4, DetCo is *1.9* better than supervised method on AP^{bb}_{75}, *0.6* better than MoCo v2 on AP^{bb}_{50}.

*equal contribution

LVIS Instance Segmentation. We use LVIS v1.0 for training and evaluation. MoCo [3] adopted LVIS v0.5, but it is outdated and can not be downloaded from the official website. So we fine-tune and compare all the methods using LVIS v1.0 dataset. The training schedule of LVIS is 180k iterations, the same as MoCo. Other settings also keep the same with MoCo. The results are shown in Table IV. DetCo also outperforms MoCo v2 and supervised methods in both detection and instance segmentation.

A.3. More Visualized Results

A.3.1 Visualization of Attention Map

Implementation Details. We visualize the attention map of `Res5` on the ImageNet dataset, which is 1/32 resolution of the input image size. To get relatively clear attention map, we enlarge the input size from $224\times224\times3$ to $448\times448\times3$. The shape of output tensor `Res5` is $14\times14\times2048$. We calculate the mean of tensor `Res5` in the channel dimension and normalize the value to 0-1. Then we get the attention map, which shape is $14\times14\times1$. We further upsample the attention map to input image's size using bilinear interpolation and project the attention map on to the image to get the visualized results.

Visualization Results. As shown in Figure I, it is surprising to see that both MoCo v2 and DetCo can generate relatively high-quality attention map that focuses on the foreground objects. It demonstrates that contrastive learning-based self-supervised representation methods can potentially solve saliency object detection or object localization in an unsupervised manner. *Moreover*, the attention map of DetCo is much better than MoCo v2 mainly in two aspects: (1) more accurate boundary localization. (2) more object discovery. We analyze that it is mainly due to introducing the global-to-local contrasts into DetCo, forcing each local patch aware of instance discrimination. To optimize global-to-local contrastive loss, each local patch needs to distinguish the foreground feature; that is why DetCo can output a more accurate attention map. However, MoCo v2 uses the whole image to extract features, so it only needs to

Method	Mask R-CNN R50-FPN COCO 1% Data						Mask R-CNN R50-FPN COCO 2% Data					
	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Rand Init	2.5	5.8	1.7	2.3	4.9	1.8	4.5	10.3	3.2	4.3	9.3	3.5
Supervised	10.0	19.9	9.2	9.7	18.3	9.2	13.7	26.6	12.8	13.0	24.2	12.6
MoCo[3]	9.1(-0.9)	17.3(-2.6)	8.6(-0.6)	8.6(-1.1)	16.1(-2.2)	8.3(-0.9)	13.0(-0.7)	24.1(-2.5)	12.6(-0.2)	12.3(-0.7)	22.4(-1.8)	12.2(-0.4)
MoCo v2[2]	9.9(-0.1)	18.7(-1.2)	9.5(+0.3)	9.5(-0.2)	17.2(-1.1)	9.2(0.0)	13.8(+0.1)	25.3(-1.3)	13.4(+0.6)	12.9(-0.1)	23.3(-0.9)	12.7(+0.1)
DetCo	**12.4**(+2.4)	**23.5**(+3.6)	**11.8**(+2.6)	**12.1**(+2.4)	**21.9**(+3.6)	**12.0**(+2.8)	**16.0**(+2.3)	**29.6**(+3.0)	**15.6**(+2.8)	**15.3**(+2.3)	**27.4**(+2.2)	**15.1**(+2.5)

Table I. **Semi-Supervised two-stage Detection fine-tuned on COCO 1% and 2% data**. All methods are pretrained 200 epochs on ImageNet. **Green** means increase and **gray** means decrease. DetCo is better than supervised / unsupervised counterparts in all metrics.

Method	Mask R-CNN R50-FPN COCO 5% Data						Mask R-CNN R50-FPN COCO 10% Data					
	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Rand Init	9.2	18.6	7.9	8.8	16.9	8.1	10.1	20.2	9.1	9.8	18.6	9.2
Supervised	19.9	37.0	19.3	18.6	33.7	18.4	23.8	42.8	23.9	22.2	39.6	22.3
MoCo[3]	19.6(-0.3)	35.1(-1.9)	20.0(+0.7)	18.3(-0.3)	32.3(-1.4)	18.6(+0.2)	23.3(-0.5)	40.7(-2.1)	23.9(0.0)	21.9(-0.3)	38.0(-1.6)	22.4(+0.1)
MoCo v2[2]	20.6(+0.7)	36.6(-0.4)	21.0(+1.7)	19.1(+0.5)	33.7(0.0)	19.2(+0.8)	24.1(+0.3)	42.0(-0.8)	24.8(+0.9)	22.5(+0.3)	39.1(-0.5)	23.3(+1.0)
DetCo	**21.9**(+2.0)	**39.1**(+2.1)	**22.2**(+2.9)	**20.4**(+1.8)	**36.1**(+2.4)	**20.6**(+2.2)	**26.0**(+2.2)	**45.2**(+2.4)	**27.0**(+3.1)	**24.3**(+2.1)	**42.0**(+2.4)	**25.0**(+2.7)

Table II. **Semi-Supervised two-stage Detection fine-tuned on COCO 5% and 10% data**. All methods are pretrained 200 epochs on ImageNet. DetCo is better than supervised / unsupervised counterparts in all metrics.

Method	Mask R-CNN R50-C4 COCO 180k						Mask R-CNN R50-FPN COCO 180k					
	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Rand Init	35.6	54.6	38.2	31.4	51.5	33.5	36.7	56.7	40.0	33.7	53.8	35.9
Supervised	40.0	59.9	43.1	34.7	56.5	36.9	40.6	61.3	44.4	36.8	58.1	39.5
MoCo[3]	40.7(+0.7)	60.5(+0.6)	44.1(+1.0)	35.4(+0.7)	57.3(+0.8)	37.6(+0.7)	40.8(+0.2)	61.6(+0.3)	44.7(+0.3)	36.9(+0.1)	58.4(+0.3)	39.7(+0.2)
MoCov2[2]	41.0(+1.0)	60.6(+0.7)	44.5(+1.4)	35.6(+0.9)	57.2(+0.7)	38.0(+1.1)	40.9(+0.3)	61.5(+0.2)	44.7(+0.3)	37.0(+0.2)	58.7(+0.6)	39.8(+0.3)
DetCo	**41.3**(+1.3)	**61.2**(+1.3)	**45.0**(+1.9)	**35.8**(+1.1)	**57.9**(+1.4)	**38.2**(+1.3)	**41.5**(+0.9)	**62.5**(+1.2)	**45.6**(+1.2)	**37.7**(+0.9)	**59.5**(+1.4)	**40.5**(+1.0)

Table III. **Object detection and instance segmentation fine-tuned on COCO**. All methods are pretrained 200 epochs on ImageNet. DetCo outperforms all supervised and unsupervised counterparts.

Method	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Rand Init	19.6	31.8	20.9	19.0	29.8	20.1
Supervised	22.7	36.8	24.1	22.2	34.6	23.5
MoCo[3]	23.1	37.4	24.5	22.5	35.1	23.8
MoCo v2[2]	23.2	37.4	24.7	22.8	35.1	24.4
DetCo	**23.5**	**37.7**	**24.8**	**23.0**	**35.5**	**24.5**

Table IV. **DetCo *vs.* supervised and other unsupervised methods on LVIS v1.0 dataset.** All methods are pretrained 200 epochs on ImageNet. We evaluate object detection and instance segmentation tasks.

activate the most discriminative area.

A.3.2 Visualization of Image Retrieval

Implementation Details. We visualize the image retrieval results on the ImageNet validation dataset. First, we extract the final-layer feature of all the images using the features learned by DetCo. Then we use global average pooling(GAP) on the extracted feature, followed by a $L2$ normalization. The shape of each feature vector is $1 \times 1 \times 2048$. For retrieval, we randomly select several images as query images, then directly find K nearest images in the feature space. K is set to 9 in this paper.

Visualization Results. Figure II shows the nearest-neighbor retrieval results. We find that DetCo can successfully group images according to their categories in most cases in an unsupervised learning manner.

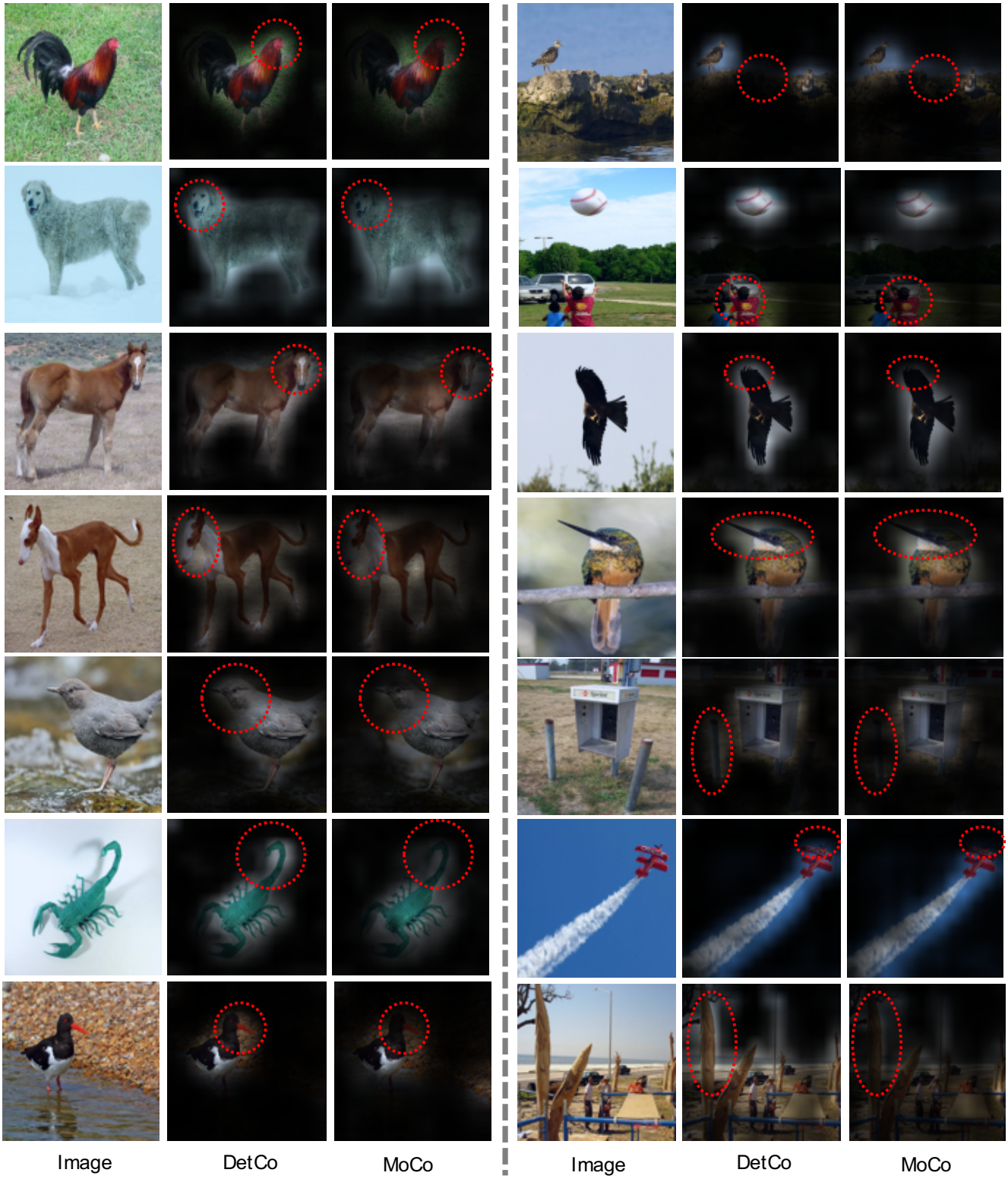

Figure I. **Attention maps generated by DetCo and MoCov2.** DetCo can activate more object regions in the heatmap than MoCov2, and the attention map of DetCo is more accurate than MoCo v2 in object boundary. *Zoom in for better visualized results.*

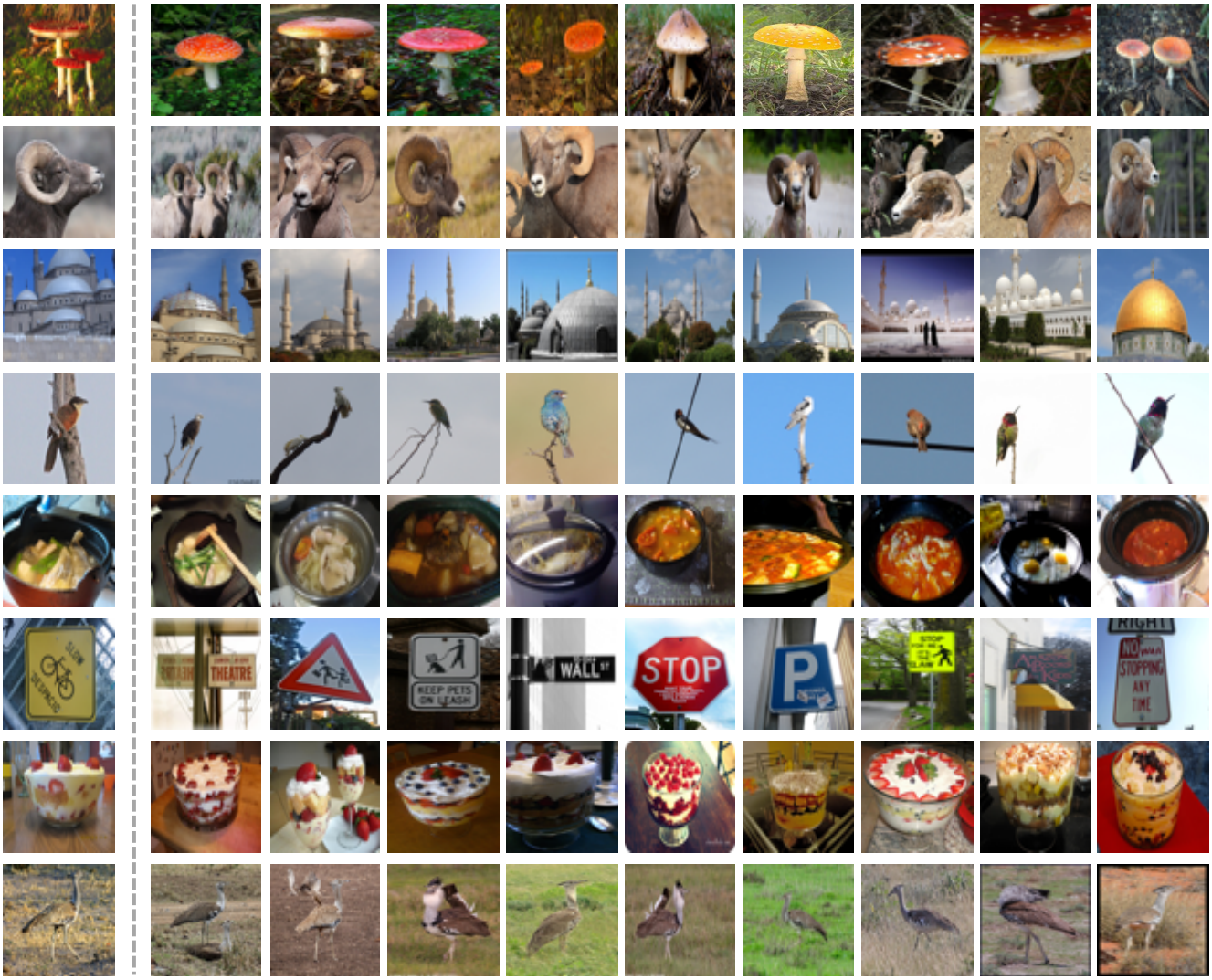

Figure II. **Retrieval results of DetCo on ImageNet.** The left column are queries from the validation set, while the right columns show 9 nearest neighbors retrieved from the validation set.

A.4. More Implementation Details

First, we provide a pseudo code for the DetCo training loop in Pytorch style, as shown in Algorithm 1. We use Apex [1] for mixed-precision training to speed up the training process. Most of our training hyperparameters are directly taken from MoCo [3]. The loss weight for intermediate contrastive loss is 1,0.7,0.4,0.1 for `Res5`, `Res4`, `Res3`, `Res2` as default. The learning rate for pre-training is 0.06 with the cosine decay schedule. For each intermediate layer and global-to-local contrast, we use a 2-layer multi-layer perceptron(MLP) head that projects the feature to a 128-D space. The design of MLP is the same as MoCo v2, except for the input channel. We also build an individual memory bank for each head to store the negative samples. In summary, there are 12 heads and 12 memory banks in DetCo. For $\mathcal{L}_{g\leftrightarrow g}$, $\mathcal{L}_{g\leftrightarrow l}$ and $\mathcal{L}_{l\leftrightarrow l}$, the temperature is 0.2, 0.5 0.2, respectively.

Downstream tasks. *(1) DensePose*. In the main paper, we report the Densepose results. For the Densepose task, MoCo used Detectron2 to evaluate Densepose. However, we find the Detectron2 updated recently, and the performance is higher than MoCo. So we re-finetuned all the methods use the latest Detectron2 code. We fine-tune Densepose RCNN with 26k iterations for all methods and report the results of "densepose_gps". *(2) RetinaNet*. We use ResNet50 as the backbone. We follow the setting of MoCo on Mask RCNN, adding extra normalization layers in both backbone and fpn. *(3) other method.* For SwAV [1], we download the pre-trained weights from the official code [2]. For a fair comparison, we choose the SwAV with batch size 256, and the training epochs are 200. The corresponding ImageNet linear classification is 72.7% in Top-1 accuracy. Then we fine-tuned the weights on PASCAL VOC, increasing the learning rate from 0.01 to 0.1, and set the `warmup_factor`=0.333 for 1000 iterations. The above settings the same with the SwAV [1] paper.

A.5. More Ablation Studies

Weight of Hierarchical Intermediate Loss. We find that the transfer detection performance is highly sensitive to the loss weight hyper-parameter for hierarchical intermediate loss, so we set different weights for `Res2`, `Res3`, `Res4`, `Res5`. The result is shown in Table V. As shown in Table V (a), if we set the loss weight equals (0,0,0,1), our method degenerates to MoCo. In Table V (b)(c), we find that directly adding hierarchical intermediate loss with inappropriate weights leads to negative results. We find that the shallow layers and the deep layers are in a competitive relationship. In other words, if we set loss weight (1,1,1,1) equally, the discriminative ability of deep layers are large negatively influenced by shallow layers. Here we revisit PSPNet [4], which also use the shallow feature as the auxiliary loss. In PSPNet, the loss weights of the shallow and deep feature are (0.4,1). In DetCo, we set the loss weight to (0.1,0.4,0.7,1.0), as shown in Table V (d)(e), and we find this loss weight improves the transfer detection performance. We argue that making the shallow layer's weight equal to deep layers is too aggressive to optimize. Moreover, if we normalize the weight, making the sum of loss weight equals to 1, the accuracy also drops.

Memory Banks for Hierarchical Intermediate Loss. Original MoCo only utilizes the final feature to calculate contrastive loss, so MoCo only uses one memory bank (queue) to store negative samples. However, here we utilize `Res2`, `Res3`, `Res4`, `Res5` to calculate the multi-level contrastive loss. An intuitive idea is we also use one memory bank to store negative samples from four stages. In this way, one memory bank is shared for both high-level and low-level features. However, we find sharing memory bank leads to performance drop. So we build an individual memory bank for the feature from each level. Under this setting, the performance improves. The results are shown in

Algorithm 1 Pseudocode of Intermediate Contrastive Loss.

```
# net_q: encoder for query
# net_q = {backbone_q, head_q_list=[head 2-5]},

# net_k: same structure as net_q, encoder for key
# queue_list: a list of queues 2-5 of K keys (CxK)
# m: momentum
# t: temperature
# x: input image
# w: loss weight for 2-5
# loss_nce: contrastive loss

net_q.params = net_k.params # initialize
x_q = aug(x) # a randomly augmented version
x_k = aug(x) # another randomly augmented version

list_q = backbone_q.forward(x_q) # list of queries
list_k = backbone_k.forward(x_k) # list of keys

total_loss = 0 # weight-sum of loss from 4 stages
for i in range(4): # loop through 4 stages
   # feature, queue and weight of current stage
   q = list_q[i] # queries: NxC
   k = list_k[i] # keys: NxC
   queue = queue_list[i] # dictionary of K keys (CxK)
   weight = w[i]

   # forward mlp head
   q = head_q_list[i].forward(q)
   k = head_k_list[i].forward(k).detach()

   # calculate loss
   loss = loss_nce(q, k, queue, t)
   total_loss = total_loss+loss*weight

   # update the queue
   dequeue_and_enqueue(queue, k)

# SGD update: query network
total_loss.backward()
update(net_q.params)

# momentum update: key network
net_k.params = m*net_k.params+(1-m)*net_q.params
```

2-5: 4 different stages of backone, termed res2, res3, res4, res5.

[1] https://github.com/NVIDIA/apex

[2] https://github.com/facebookresearch/swav

	weight	AP	AP_{50}	AP_{75}
(a)	(0,0,0,1)	56.3	81.8	62.1
(b)	(1,1,1,1)+Norm	55.1	80.4	60.4
(c)	(1,1,1,1)	55.8	81.6	62.2
(d)	(0.1,0.4,0.7,1)+Norm	56.5	82.2	62.7
(e)	(0.1,0.4,0.7,1)	57.0	82.2	63.1

Table V. Ablation study of intermediate loss weight, under 100 epoch pre-training. "Norm" means normalized the loss weights, making the sum of weights equals 1.0.

	share queue	AP	AP_{50}	AP_{75}
(a)	✓	56.2	81.8	62.3
(b)	×	57.0	82.2	63.1

Table VI. Ablation study of share or not share queue, under 100 epoch pre-training.

	$+\mathcal{L}_{g\leftrightarrow g}$	$+\mathcal{L}_{g\leftrightarrow l}$	$+\mathcal{L}_{l\leftrightarrow l}$	AP
(a)	✓	×	×	56.0
(b)	✓	✓	×	56.2
(c)	✓	✓	✓	56.5

Table VII. Ablation study of cross contrastive loss, under 50 epoch pre-training.

Table VI. We consider if sharing memory bank cross levels, the positive samples of each stage need to discriminate negative samples from all levels, which is challenging to optimize. Suppose each layer owns an individual memory bank. In that case, each stage's positive samples only need to discriminate negative samples from its corresponding layer, making the network converge easy.

Global and Local Contrastive Losses. As shown in Table VII, adding loss $\mathcal{L}_{g\leftrightarrow l}$ can improve the transfer detection performance. Moreover, adding loss $\mathcal{L}_{l\leftrightarrow l}$ can further improve the results. The results match the main paper's analysis that adding global-and-local contrastive losses can (1) actually improve the lower bound of mutual information. (2) improve the instance discrimination of local patches, which is beneficial to object detection.

References

[1] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *arXiv preprint arXiv:2006.09882*, 2020. 5

[2] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020. 2

[3] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020. 1, 2, 5

[4] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017. 5