Revitalizing CNN Attention via Transformers in Self-Supervised Visual Representation Learning <Supplementary Material>

Anonymous Author(s)
Affiliation
Address
email

In this supplementary material, we show the pseudo code and more numerical comparisons to the state-of-the-art approaches. The Python source code is provided as well.

Algorithm 1: Pseudocode of CARE

Inputs :

$\mathcal{D}, \mathcal{T}, \mathcal{T}'$	set of images and distributions of transformations
$\theta, C_\theta, T_\theta$	net parameters in the first branch, C-stream and T-stream in the first branch
ξ, C_ξ, T_ξ	net parameters in the second branch, C-stream and T-stream in the second branch
$t_0, \tau_{\text{base}}, \eta_{\text{base}}$	warmup steps, base network updating coefficient, base learning rate
T, N	total optimization steps, batch size

for $t = 1$ **to** T **do**
  $\mathcal{I} \leftarrow \{x^{(i)} \sim \mathcal{D}\}_{i=1}^{N}$ // sample a batch of N images
  for $x^{(i)} \in \mathcal{I}$ **do**
    $x_1 = \text{aug}(x^{(i)})$, where aug $\sim \mathcal{T}$ // random transformation
    $x_2 = \text{aug}'(x^{(i)})$, where aug$' \sim \mathcal{T}'$ // random transformation
    $f_1 \leftarrow C_\theta(x_1)$, ${f_1}' \leftarrow C_\theta(x_2)$ // output C-stream vectors in the first branch
    $f_2 \leftarrow C_\xi(x_2)$, ${f_2}' \leftarrow C_\xi(x_1)$ // output C-stream vectors in the second branch
    $f_3 \leftarrow T_\theta(x_1)$, ${f_3}' \leftarrow T_\theta(x_2)$ // output T-stream vectors in the first branch
    $f_4 \leftarrow T_\xi(x_2)$, ${f_4}' \leftarrow T_\theta(x_1)$ // output T-stream vectors in the second branch
    $\mathcal{L}_c := -\langle f_1, f_2 \rangle - \langle {f_1}', {f_2}' \rangle$ // compute SSL loss for the C-stream
    $\mathcal{L}_t := -\langle f_3, f_4 \rangle - \langle {f_3}', {f_4}' \rangle$ // compute SSL loss for the T-stream
    $\mathcal{L}_{\text{att}} := \|f_1 - f_3\|_2 + \|{f_1}' - {f_3}'\|_2$ // compute attention supervision loss
    $\mathcal{L}_{\text{total}}^{(i)} := \mathcal{L}_c + \mathcal{L}_t + \lambda \mathcal{L}_{\text{att}}$ // sum up the total loss for $x^{(i)}$
  end
  $\nabla_\theta \leftarrow \frac{1}{N} \sum_{i=1}^{N} \partial_\theta \mathcal{L}_{\text{total}}^{(i)}$ // compute the total loss gradient w.r.t. θ
  if $t < t_0$ **then**
    $\eta \leftarrow \eta_{\text{base}} \cdot t / t_0$ // warmup learning rate
  else
    $\eta \leftarrow \eta_{\text{base}} \cdot (\cos(\pi(t - t_0)/(T - t_0)) + 1)/2$ // cosine decay learning rate
  end
  $\theta \leftarrow \text{optimizer}(\theta, \nabla_\theta, \eta)$ // update parameters in the first branch
  $\tau \leftarrow 1 - (1 - \tau_{\text{base}}) \cdot (\cos(\pi t / T) + 1)/2$ // update momentum
  $\xi \leftarrow \tau\xi + (1 - \tau)\theta$ // update parameters in the second branch
end

Output : encoder C_θ

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

1 Revisiting the algorithms

In Algorithms 1, we first present the overall training procedure of our proposed CARE, which details the training process and configurations of C-stream and T-steam. We also illustrate how to set the learning rate and network update momentum during training. To make the training process clearer, we also provide the pseudo-code of CARE in a PyTorch-like style in Algorithms 2.

Algorithm 2: Pseudocode of CARE in PyTorch-like style.

```
# C_theta, C_xi: the first and second branches on C-stream
# T_theta, T_xi: the first and second branches on T-stream
# aug1, aug2: two random augmentations

# initialize
C_xi.params = C_theta.params
T_xi.params = T_theta.params

for x in loader: # load a minibatch x with N samples
    # generate two randomly augmented views of x
    x_1, x_2 = aug1(x), aug2(x)

    # compute the output the first branches of of C- and T-stream
    c_q1, c_q2 = C_theta(x_1), C_theta(x_2)
    t_q1, t_q2 = T_theta(x_1), T_theta(x_2)

    # stop gradient for the second branches of C- and T-stream
    with torch.no_grad():
        c_k1, c_k2 = C_xi(x_2), C_xi(x_1)
        t_k1, t_k2 = T_xi(x_2), T_xi(x_1)

    # compute the loss for C-stream
    l_c = (4 - 2 * ((c_q1 * c_k1).sum(dim=1) + (c_q2 * c_k2).sum(dim=1))).mean()
    # compute the loss for T-stream
    l_t = (4 - 2 * ((t_q1 * t_k1).sum(dim=1) + (t_q2 * t_k2).sum(dim=1))).mean()
    # compute the supervision loss of C- and T-stream
    l_att = l2_loss(c_q1 - t_q1) + l2_loss(t_q1 - t_q2)
    L = l_c + l_t + lambda * l_att # sum up the loss

    L.backward() # back-propagation

    # get the learning rate from the learning rate scheduler
    lr = lr_scheduler()

    # SGD update on the first branches of C- and T-stream
    optimizer(C_theta, lr)
    optimizer(T_theta, lr)

    # get the momentum from the momentum scheduler
    m = momentum_scheduler()

    # momentum update of the second branches of C- and T-stream
    C_xi.params = m * C_xi.params + (1-m) * C_theta.params
    T_xi.params = m * T_xi.params + (1-m) * T_theta.params

# l2 loss function
def l2_loss(x):
    return x.square().sum(dim=1).mean()
```

2 Numerical evaluations

In this section, we provide more results compared to the state-of-the-art approaches under different downstream recognition scenarios including self-supervised and semi-supervised image classifications, object detection, and semantic segmentation.

Self-supervised learning on image classifications. We first evaluate the image classifications based on the standard linear classification protocol as introduced in Section 4.2. Table 1 shows more results. Our CARE method consistently outperforms other methods under different training epochs. Specifically, our method helps ResNet-50 encoder achieve 74.7% top-1 accuracy under 400 training epochs, which is even higher than the encoder trained with BYOL under 800 epochs. We also notice that CARE performs consistently well on the stronger backbone (e.g., ResNet-101). For example, we achieve 75.4% top-1 accuracy with ResNet-101 encoder trained with 200 epochs. This indicates that CARE is generally beneficial to the performance boost of CNN encoders on image classification tasks.

Table 1: Classification accuracy via different CNN and Transformer encoders.

Method	Arch.	Param.	Epoch	GFlops	Top-1
BYOL [7]	ResNet-50	25M	800	4.1	74.2
SimSiam [4]	ResNet-50	25M	400	4.1	69.6
SimCLR [2]	ResNet-50	25M	1000	4.1	70.8
Jigsaw [12]	ResNet-50(2×)	69M	-	11.4	44.6
RelativePosition [6]	ResNet-50(2×)	69M	-	11.4	51.4
MoCo [8]	ResNet-50(2×)	69M	-	11.4	65.4
BYOL [7]	ResNet-50(2×)	69M	100	11.4	71.9
CMC [13]	ResNet-50(2×)	69M	-	11.4	70.6
SimCLR [2]	ResNet-50(2×)	69M	1000	11.4	74.2
BYOL [7]	ResNet-101	45M	100	7.8	72.3
CPC v2 [9]	ResNet-101	45M	800	7.8	48.7
BYOL [7]	ResNet-152	60M	100	11.6	73.3
BYOL [7]	ViT-S	22M	300	4.6	71.0
BYOL [7]	ViT-B	86M	300	17.7	73.9
MoCo v3 [5]	ViT-S	22M	300	4.6	72.5
CARE (ours)	ResNet-50	25M	200	4.1	**73.8**
CARE (ours)	ResNet-50	25M	400	4.1	**74.7**
CARE (ours)	ResNet-50(2×)	69M	100	11.4	**73.5**
CARE (ours)	ResNet-50(2×)	69M	200	11.4	**75.0**
CARE (ours)	ResNet-101	45M	100	7.8	**73.5**
CARE (ours)	ResNet-152	60M	100	11.6	**74.9**
CARE (ours)	ResNet-101	69M	200	7.8	**75.4**

Semi-supervised learning on image classifications. We also conduct more experiments by using a semi-supervised training configuration on the ImageNet dataset. The semi-supervised protocol is introduced in Section 4.2. The evaluation results of CNN encoders that are trained by using more epochs are provided in Table 2. The results show that our CARE method achieves higher top-1 and top-5 accuracy than the other SSL methods under different configurations (e.g., backbones and training epochs). We also notice that our CARE method takes consistent advantages (at least +1.5%) over other SSL methods under different training epochs.

Transfer learning to object detection and semantic segmentation. Here, we provide more experimental results using the same setting in Section 4.2 in the main text and report the results in Table 3. Besides, we further evaluate CARE's representation on the COCO dataset using a more powerful detector, the feature pyramid network [10], and report the results in Table 4. We follow the same evaluation protocol as in Section 4.2 in our main text. Specifically, we extract the parameters of the ResNet-50 encoder from the pretrained model and use them to initialize the corresponding parameters in Mask R-CNN R50-FPN detector from Detectron2 [14]. We train the detector using 1× schedule (90k iterations) on the COCO training set and evaluate its performance on the COCO test set.

Again, CARE trained for 200/400 epochs outperforms other state-of-the-art SSL methods trained for 800/1000 epochs on object detection and semantic segmentation on the COCO dataset, which suggests that the CNN encoder in CARE is empowered by the attention mechanism of the transformer which supervises the CNN encoder in the pretraining (note that we use no transformers (or multi-head self-attention modules) in our transferred detectors).

Table 2: Classification accuracy by using other CNN encoders.

Method	Arch.	Epoch	Top-1		Top-5	
			1%	10%	1%	10%
BYOL [7]	ResNet-50	1000	53.2	68.8	78.4	89.0
MoCov2 [3]	ResNet-50	800	42.3	63.8	70.1	86.2
SWAV [1]	ResNet-50	1000	53.9	70.2	78.5	89.9
Barlow Twins [15]	ResNet-50	1000	55.0	69.7	79.2	89.3
BYOL [7]	ResNet-50(2×)	100	55.6	66.7	77.5	87.7
BYOL [7]	ResNet-50(2×)	200	59.5	68.0	82.4	88.3
BYOL [7]	ResNet-101	100	55.8	65.8	79.5	87.4
BYOL [7]	ResNet-101	200	60.3	69.1	82.7	89.3
BYOL [7]	ResNet-152	100	56.8	67.2	79.3	88.1
CARE (ours)	ResNet-50	400	60.0	69.6	81.3	89.3
CARE (ours)	ResNet-50(2×)	100	57.4	67.5	79.8	88.3
CARE (ours)	ResNet-50(2×)	200	61.2	69.6	82.3	89.5
CARE (ours)	ResNet-101	100	57.1	67.1	80.8	88.2
CARE (ours)	ResNet-101	200	62.2	70.4	85.0	89.8
CARE (ours)	ResNet-152	100	59.4	69.0	82.3	89.0

Table 3: Transfer learning to object detection and instance segmentation (some of the results are also shown in Table 3 in the main text). The best two results in each column are in bold.

		COCO det.			COCO instance seg.			VOC07+12 det.		
Method	Epoch	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}	AP	AP_{50}	AP_{75}
Rand Init	-	26.4	44.0	27.8	29.3	46.9	30.8	33.8	60.2	33.1
Supervised	90	38.2	58.2	41.2	33.3	54.7	35.2	53.5	81.3	58.8
PIRL[11]	200	37.4	56.5	40.2	32.7	53.4	34.7	55.5	81.0	61.3
MoCo[8]	200	38.5	58.3	41.6	33.6	54.8	35.6	55.9	81.5	62.6
MoCo-v2[3]	200	38.9	58.4	42.0	34.2	55.2	36.5	57.0	82.4	63.6
MoCo-v2[3]	800	39.3	58.9	42.5	34.4	55.8	36.5	57.4	82.5	64.0
SwAV[1]	800	38.4	58.6	41.3	33.8	55.2	35.9	56.1	82.6	62.7
BYOL [7]	800	39.0	58.6	42.1	34.1	55.3	36.4	57.3	82.5	64.0
Barlow Twins[15]	1000	39.2	59.0	42.5	34.3	56.0	36.5	56.8	82.6	63.4
CARE (Ours)	200	**39.4**	**59.2**	**42.6**	**34.6**	**56.1**	**36.8**	**57.7**	**83.0**	**64.5**
CARE (Ours)	400	**39.6**	**59.4**	**42.9**	**34.7**	**56.1**	**36.9**	**57.9**	**83.0**	**64.7**

Table 4: Transfer learning to object detection and instance segmentation with Mask R-CNN R50-FPN detector. The best two results in each column are in bold. Our method achieves favorable detection and segmentation performance by using limited training epochs.

		COCO det.			COCO instance seg.		
Method	Epoch	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Rand Init	-	31.0	49.5	33.2	28.5	46.8	30.4
Supervised	90	38.9	59.6	42.7	35.4	56.5	38.1
PIRL[11]	200	37.5	57.6	41.0	34.0	54.6	36.2
MoCo[8]	200	38.5	58.9	42.0	35.1	55.9	37.7
MoCo-v2[3]	200	38.9	59.4	42.4	35.5	56.5	38.1
MoCo-v2[3]	800	39.4	59.9	43.0	35.8	56.9	38.4
SwAV[1]	200	38.5	**60.4**	41.4	35.4	57.0	37.7
BYOL [7]	200	39.1	59.5	42.7	35.6	56.5	38.2
BYOL [7]	400	39.2	59.6	42.9	35.6	56.7	38.2
BYOL [7]	800	39.4	59.9	43.0	35.8	56.8	**38.5**
Barlow Twins[15]	1000	36.9	58.5	39.7	34.3	55.4	36.5
CARE (Ours)	200	**39.5**	60.2	**43.1**	**35.9**	**57.2**	**38.5**
CARE (Ours)	400	**39.8**	**60.5**	**43.5**	**36.2**	**57.4**	**38.8**

References

[1] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems*, 2020.

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 2020.

[3] Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *CoRR*, abs/2003.04297, 2020.

[4] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[5] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised visual transformers. *arXiv preprint arXiv:2104.02057*, 2021.

[6] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *IEEE/CVF International Conference on Computer Vision*, 2015.

[7] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*, 2020.

[8] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[9] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *International Conference on Machine Learning*, 2020.

[10] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[11] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6707–6717, 2020.

[12] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, 2016.

[13] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *European Conference on Computer Vision*, 2020.

[14] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[15] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, 2021.