Representation Learning via Consistent Assignment of Views over Random Partitions

Appendix

Anonymous Author(s)
Affiliation
Address
email

A Extended results

In this section, we expand the evaluation experiments from the main text. We use the officially released pre-trained methods across all benchmarks to represent each SSL method. We did not retrain/reimplement any competing method. In total, we pre-trained 4 instances of CARP. To evaluate CARP's performance in a low training regime, we trained two 200-epoch models, one with multi-crop (mc) and the other without it. Similarly, to evaluate longer training performance, we trained two 400-epoch models, one with multi-crop and the other without it.

A.1 Transfer learning evaluation

In Table A.1, we report detailed results for the transfer learning k-NN experiments. We evaluate SSL methods for values of $k \in \{10, 20, 100, 200\}$, including all instances of CARP. Cf. Appendix D.3 for the evaluation protocol.

CARP achieved either top-1 or top-2 performance in seven out of 8 datasets. STL-10 is the only dataset where CARP is neither the top-1 nor top-2. **Moreover, in the FGVCAircraft, Stanford Cars, and GTSRB datasets, CARP archived top-1 and top-2 performances with large margins.** The average k-NN performance per value of k across all datasets is reported in Table 1 in the main text.

A.2 Image retrieval and copy detection

Tables A.2 and A.3 show additional results for image retrieval and copy detection evaluations. For both tasks, we compare CARP's performance against nine SSL methods. We report mAP on the Medium and Hard splits of the revisited Oxford and Paris datasets for image retrieval. We report mAP on the "strong" set of the Copydays dataset for copy detection.

A.3 Few-shot evaluation

Table A.4 shows additional few-shot evaluation results on the Pascal VOC07 and INat-2018 datasets. We report mAP and top-1 accuracy @k, averaged over 5 independent runs, for VOC07 and INat-2018, respectively. We include CARP's 200 and 400 multi-crop models. Cf. Appendix D.2 for details on the evaluation protocol.

A.4 Dense prediction evaluation

We evaluated our CARP's multi-crop model on the object detection downstream tasks using the Pascal VOC07 dataset. We followed the guidelines from He et al. [11]. We fine-tuned for 24k iterations on PASCAL VOC trainval07+12 and evaluated on test2007. CARP performs at AP=44.2,

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Table A.1: **Transfer learning evaluation**. We compare CARP's performance against nine SSL methods on eight datasets. We report results for $k \in \{10, 20, 100, 200\}$. Top methods in **bold**, top-2 underlined. Methods with (mc) use multi-crop.

		Oxford-IIIT Pet				Oxford Flowers-102				FGVCAircraft				Stanford Cars			
Method	ep	10	20	100	200	10	20	100	200	10	20	100	200	10	20	100	200
oBoW (mc)	200	55.8	57.3	57.2	57.5	63.5	61.9	58.9	59.4	19.1	18.1	15.8	14.9	11.9	11.5	10.9	10.5
SeLa-v2	400	66.7	66.8	65.8	66.0	59.5	58.6	56.8	57.2	21.3	20.7	18.1	16.4	13.3	13.3	13.5	13.4
InfoMin	800	78.0	77.8	76.6	76.2	63.6	61.9	60.1	60.8	18.9	18.2	15.8	13.4	14.7	14.4	13.2	12.9
SWAV (mc)	800	77.2	77.0	74.9	74.5	76.4	75.2	73.7	74.6	29.6	29.0	27.5	25.1	22.7	22.7	21.8	21.0
DINO (mc)	800	81.5	80.9	79.0	78.9	**82.3**	**81.6**	**80.8**	**81.2**	36.1	35.3	33.5	31.1	30.0	30.1	28.9	27.5
DeepC v2 (mc)	800	79.0	78.3	76.3	75.4	78.3	76.3	75.3	76.0	32.5	32.0	28.9	26.5	25.2	25.0	23.4	22.1
Triplet	980	83.3	83.5	82.4	82.4	78.5	77.7	76.9	77.3	33.3	33.4	31.7	29.6	24.4	25.2	25.5	25.0
MoCo v3	1000	86.6	86.4	85.8	85.8	79.8	79.0	78.3	78.6	37.7	36.9	33.5	32.1	28.6	29.3	28.4	27.2
BarlowT	1000	82.5	82.9	82.2	82.3	79.8	78.8	77.9	78.1	32.9	32.7	30.6	29.2	25.9	26.3	26.1	25.2
CARP	200	**86.8**	**86.8**	**86.2**	85.9	79.0	78.2	77.4	77.7	40.5	38.9	36.0	34.4	29.4	29.8	29.6	29.6
	400	86.4	**86.8**	**86.2**	**86.0**	81.0	80.0	79.3	79.6	**42.5**	**42.1**	**39.3**	**38.6**	**32.6**	**33.5**	**32.6**	**31.5**
CARP (mc)	200	78.7	78.7	77.1	76.8	80.6	79.7	78.7	78.8	35.7	35.0	32.4	30.6	26.4	26.6	25.4	24.3
	400	83.9	83.9	83.6	83.2	81.4	80.3	79.0	79.6	35.2	34.8	32.4	30.7	26.6	27.1	26.1	24.9

		Country-211				Food-101				STL-10				GTSRB			
Method	ep	10	20	100	200	10	20	100	200	10	20	100	200	10	20	100	200
oBoW (mc)	200	11.7	12.0	11.8	11.4	45.8	47.4	47.3	46.0	96.6	96.6	**96.3**	95.7	50.1	50.6	49.9	48.2
SeLa-v2	400	10.1	10.5	11.0	11.0	45.7	46.8	46.6	45.5	94.0	94.0	93.5	93.3	58.1	59.0	58.7	57.9
InfoMin	800	11.2	11.6	11.9	11.9	51.5	52.4	51.4	49.5	96.5	96.4	96.2	**96.0**	54.9	54.8	53.5	52.3
SWAV (mc)	800	13.6	13.8	13.1	12.7	57.9	59.1	58.2	57.0	95.5	95.2	94.0	93.0	62.9	63.2	61.6	60.0
DINO (mc)	800	14.1	14.4	14.2	13.6	60.9	62.0	61.4	60.0	95.9	95.6	94.7	93.8	62.7	62.9	62.6	61.8
DeepC v2 (mc)	800	13.3	13.6	12.5	12.1	61.2	62.3	61.4	60.1	95.7	95.6	94.5	93.3	62.9	63.4	62.4	61.3
Triplet	980	13.7	14.1	**14.3**	**14.2**	60.1	61.5	61.0	60.0	95.7	95.6	94.9	94.5	63.4	63.5	62.9	62.0
MOCO v3	1000	12.4	12.4	13.2	13.2	59.0	60.0	59.1	57.7	**96.9**	**96.7**	96.2	95.9	72.4	72.8	72.6	71.7
BarlowT	1000	12.8	13.3	13.7	13.6	60.3	61.4	60.7	59.4	94.8	94.8	94.2	93.8	65.3	65.6	65.6	64.4
CARP	200	11.9	12.2	12.7	12.8	57.4	58.4	57.7	56.3	95.5	95.5	94.6	93.9	73.1	73.7	73.5	72.6
	400	11.9	12.3	12.8	12.8	57.6	58.4	57.6	56.3	96.1	95.9	95.0	94.3	**74.7**	**75.3**	**75.2**	**74.4**
CARP (mc)	200	**14.2**	**14.5**	**14.3**	13.9	60.5	61.8	60.7	59.3	95.8	95.5	94.1	93.4	64.6	64.7	64.2	63.0
	400	14.1	14.2	**14.3**	13.9	**61.7**	**62.9**	**62.1**	**60.7**	95.9	95.5	94.3	93.4	62.2	62.8	62.4	61.4

AP50=79.4, and AP75=47.6; results are averaged over 5 trials. Compared to other SSL methods, such as MoCo-v2 Chen et al. [6] (AP=57.4 AP50=82.5 AP75=64.0), CARP underperforms significantly.

B Ablations

Due to a limited execution budget, the ablations and the main experiments differ slightly in some hyperparameters. Here, we describe the configurations used for the ablations—for the main experiments, see Appendix C.1. For ablations, we trained CARP using the full ImageNet-1M dataset for 50 epochs. The projection head learns a latent representation of 128-dim. The batch size is set to 256 observations, and the projection head hidden layers contain 2048 neurons. We set the number of learnable prototypes $K = 65\,536$, and the number of random partition blocks $N_{\mathcal{P}} = 128$. Hence, each block contains $N_B = 512$ prototypes. We report results for single runs.

B.1 Does the number of learnable prototypes affect the learned representations?

Table B.1 examines the effect of training CARP with different configurations of prototypes K. Similar to other clustering-based SSL methods [3, 4, 13], CARP also benefits from over-clustering. As the number of trainable prototypes K grows, the k-NN performance of the learned representations increases. In addition, note that if the number of prototypes K is smaller than the number of actual classes in the dataset, the k-NN performance of the learned representations degrades. Based on these experiments, we set the default number of prototypes $K = 65536$ for the ImageNet-1M dataset.

Table A.2: **Image retrieval evaluation.** We report mAP performance of various self-supervised methods for the image retrieval downstream task on the revisited Oxford and Paris datasets. All SLL methods were pre-trained on ImageNet-1M. We used the officially released pre-trained models from respective methods for evaluation. Top-1 performers in **bold**, top-2 underlined.

		$\mathcal{R}\mathcal{O}$x		$\mathcal{R}$par	
Method	Ep	Medium	Hard	Medium	Hard
Supervised	100	49.8	18.5	74	52.1
Scratch		1.6	0.7	4.1	2.5
oBoW (mc)	200	20.4	4.4	40.6	16.2
SeLa-v2 (mc)	400	20.1	4.9	37.1	13.6
InfoMin	800	24.4	5.7	44.6	18.8
DeepC-v2 (mc)	800	32.6	10.9	50.0	20.2
SwAV (mc)	800	31.1	10.1	48.9	20.6
DINO (mc)	800	35.4	11.1	55.9	27.5
Triplet	980	35.3	12.0	58.2	28.7
VICReg	1000	32.7	8.5	57.5	29.0
MoCo-v3	1000	33.1	10.9	**59.1**	**31.3**
CARP	200	38.8	**15.5**	58.8	30.4
	400	**38.9**	15.1	58.5	30.2
CARP (mc)	200	32.8	10.4	53.6	24.9
	400	33.7	11.6	54.0	26.5

Table A.3: **Copy detection evaluation**. We report mAP on the "strong" subset of the Copydays dataset and compare CARP's performance against seven SSL methods. Top-1 performers in **bold**, top-2 underlined.

Method	Ep	mAP
Scratch		25.7
oBoW (mc)	200	61.5
SeLa-v2 (mc)	400	76.6
InfoMin	800	67.5
DeepC-v2 (mc)	800	76.0
SwAV (mc)	800	76.1
DINO (mc)	800	78.8
Triplet	980	81.7
VICReg	1000	83.7
MoCo-v3	1000	80.6
CARP	200	82.3
	400	82.6
CARP (mc)	200	80.8
	400	**84.0**

B.2 Does the number of partition blocks matter?

To better understand the effect of the hyperparameters $N_{\mathcal{P}}$ and N_B on the learned representations and in the training stability, the first row of Table B.2 demonstrates the performance of CARP using different configurations for the number of partition blocks $N_{\mathcal{P}}$ and their sizes N_B. For completeness, we analyze the effect of removing the momentum encoder in Appendix B.3. We also present an ablation on the effect of the momentum update in Table B.3.

Specifically, as the partition sizes grow and the number of partition blocks $N_{\mathcal{P}}$ decreases, the quality of the learned representations tends to decline and eventually collapse. Note that setting a partition size $N_B = 65536$ produces a single partition block $N_{\mathcal{P}}$ containing all prototypes. Precisely, the setup in the last row and last column of Table B.2 is equivalent to CARL [16]. It shows that a naive implementation leads to a collapsed solution, and the divide-and-conquer approach of devising random partitions from the learnable prototypes avoids such trivialities.

Table A.4: **Few-shot classification on VOC07 and INat-2018**. We report mAP at n on VOC07 and top-1 accuracy for INat-2018 across 5 independent runs, where n denotes the number of training examples. Standard deviations rounded to the first decimal place.

		Pascal VOC07					
Method	Ep	n=1	n=2	n=4	n=8	n=16	full
PCL v2 [13]	200	**47.9 ± 4.1**	59.6 ± 2.7	66.2 ± 2.2	74.5 ± 0.5	78.3 ± 0.4	85.4
SeLa-v2 (mc) [1]	400	42.0 ± 2.2	54.5 ± 3.2	62.2 ± 1.5	71.4 ± 0.5	76.9 ± 0.4	85.3
DeepC v2 (mc) [2]	800	46.5 ± 2.4	58.4 ± 2.9	66.5 ± 1.6	74.5 ± 0.9	79.5 ± 0.4	87.6
SwAV (mc) [3]	800	42.9 ± 2.1	54.9 ± 4.4	64.0 ± 2.1	72.9 ± 1.1	78.7 ± 0.6	88.1
DINO (mc) [4]	800	45.6 ± 2.4	58.4 ± 3.2	66.6 ± 2.1	74.8 ± 0.8	79.6 ± 0.6	88.2
Triplet [17]	980	43.6 ± 3.3	56.2 ± 3.5	64.6 ± 1.8	73.8 ± 0.1	79.6 ± 0.7	**88.3**
MoCo v3 [7]	1000	46.6 ± 3.7	59.6 ± 2.9	67.0 ± 2.4	75.4 ± 0.7	**80.2 ± 0.6**	87.4
BarlowT [19]	1000	42.6 ± 3.7	55.5 ± 3.2	63.5 ± 1.8	72.6 ± 0.1	77.6 ± 0.5	86.3
CARP (mc)	200	46.0 ± 3.2	58.3 ± 3.3	66.5 ± 2.4	75.5 ± 0.1	79.5 ± 0.6	88.0
	400	47.1 ± 3.2	**59.8 ± 3.2**	**67.3 ± 2.2**	**75.8 ± 1.1**	80.0 ± 0.7	88.2
		INat-2018					
Method	Ep	n=1	n=2	n=4	n=8	n=16	full
PCL [13]	200	1.4 ± 0.1	1.6 ± 0.1	2.3 ± 0.2	2.9 ± 0.1	4.8 ± 0.1	2.1
SeLa-v2 (mc) [1]	400	2.9 ± 0.2	4.2 ± 0.1	6.3 ± 0.1	10.0 ± 0.1	13.5 ± 0.1	8.2
DeepC v2 (mc) [2]	800	7.6 ± 0.2	13.0 ± 0.8	20.9 ± 0.5	29.6 ± 0.4	36.4 ± 0.2	32.8
SwAV (mc) [3]	800	5.3 ± 0.1	9.2 ± 0.5	15.6 ± 0.1	23.1 ± 0.2	29.4 ± 0.2	24.2
DINO [4]	800	6.5 ± 0.1	12.0 ± 0.5	20.4 ± 0.5	29.6 ± 0.3	35.9 ± 0.3	30.4
Triplet [17]	980	11.4 ± 0.2	19.1 ± 0.7	28.9 ± 0.8	37.6 ± 0.3	44.0 ± 0.1	41.4
MoCo v3 [7]	1000	8.1 ± 0.1	12.2 ± 0.3	18.5 ± 0.3	27.2 ± 0.3	33.5 ± 0.1	28.0
BarlowT [19]	1000	8.8 ± 0.1	12.2 ± 0.5	17.2 ± 0.2	24.6 ± 0.1	30.8 ± 0.1	25.3
CARP (mc)	200	8.6 ± 0.2	14.4 ± 0.1	23.6 ± 0.3	32.7 ± 0.3	38.2 ± 0.2	33.9
	400	**11.5 ± 0.1**	**19.6 ± 0.1**	**29.6 ± 0.9**	**39.1 ± 0.3**	**45.1 ± 0.2**	**42.6**

Table B.1: CARP benefits from over-clustering. Setting a small number of prototypes may hurt the learned representations.

K	1024	2048	4096	16384	65536	262144
k-NN	48.81	49.98	50.69	50.81	51.2	**51.31**

Note that as smaller the block size N_B, more stable the algorithm will be. However, the quality of the learned representation might decrease since the pseudo-classification tasks, posed by the random partitions, becomes easier with fewer prototypes. On the other hand, a larger block size N_B poses a more challenging consistency task at the expense of contributing to mode collapse.

For most cases, however, for block sizes ranging from $N_B = 128$ to $N_B = 4096$, CARP learns useful representations and shows robustness to this hyperparameter. By default we set the partition block size $N_B = 512$.

B.3 The importance of the momentum encoder

Table B.2 contrasts CARP's joint-embedding architectures with and without a momentum encoder, which is equivalent to setting $\eta = 0$ in the momentum encoder update equation. Different from other SSL methods [4, 9], CARP works with both setups. However, we observe that using a momentum encoder significantly boosts the performance of the learned representations. Table B.2 shows that regardless of block sizes, representations learned using a momentum encoder-based architecture consistently outperform the siamese counterpart.

B.4 Who provides the best features for downstream evaluation?

One way to understand CARP's joint-embedding architecture with a momentum encoder is through the teacher-student framework, where the momentum encoder is the teacher that guides the learning

Table B.2: CARP with and without a momentum encoder. Without the random partition strategy (last column), training collapses regardless of using a momentum encoder or a pure siamese architecture.

N_B	32	64	128	256	512	1024	2048	4096	16384	65536
w/ mom. enc.	49.56	50.75	51.19	51.20	51.32	51.06	51.31	51.08	49.67	0.11
w/o mom. enc.	48.95	49.28	48.81	47.37	46.16	44.68	44.29	44.39	47.25	0.11

Table B.3: The effect of the hyperparameter η on the momentum encoder updates. In the last column, η starts as $\eta = 0.99$ and it is annealed to $\eta = 1.0$ following a cosine schedule.

η	0	0.5	0.9	0.99	0.999	$0.99 \rightarrow 1.0$
k-NN	51.0	50.2	50.3	51.1	50.1	**51.3**

student. The addition of the momentum encoder raises the question of which module produces the best representations. To answer this question, Figure B.1 explores the k-NN performance when extracting features from the momentum encoder (teacher) versus the student. We observe that teachers' representations constantly outperform the students' during training. However, by the end of the training, the student catches up with the teacher.

C Implementation Details

C.1 Experimental Setup

We train CARP on the ImageNet-1M unlabeled dataset using ResNet50 [10] encoders. We take the output representation of the last global average pooling layer (a 2048-dim vector) and project it to a 256-dim vector. Following Caron et al.'s [4] work, our MLP projection head contains 3 dense layers with batch normalization and the GELU activations. The hidden units of the projection head contain 2048 neurons. The 256-dim representation vector is fed to an assigner MLP that outputs unnormalized probabilities w.r.t. the learnable prototypes. By default, the assigner function is implemented as a linear layer and trains $K = 65\,536$ prototypes. To generate the random partitions, we set the number of partitions $N_{\mathcal{P}} = 128$, which creates subsets containing $N_B = 512$ randomly chosen prototypes. We use the same data augmentations proposed by Grill et al. [9] to generate synthetic views. The protocol creates three data augmentation pipelines, the first two to generate global views and the last to generate multi-crops. CARP is pre-trained with the LARS [18] optimizer, end to end, with weight decay of 1×10^{-6}. For models training up to 200 epochs, the learning rate starts from 0.6 and decays to 0.006 with a cosine scheduling [14] without warmups. For models pre-trained for more than 400 epochs, the learning rate starts at 0.3 and decays to 0.003 using the same cosine scheduler. We train the system with a global batch size of 4096 observations. For all experiments, we used 4 A100 40GB

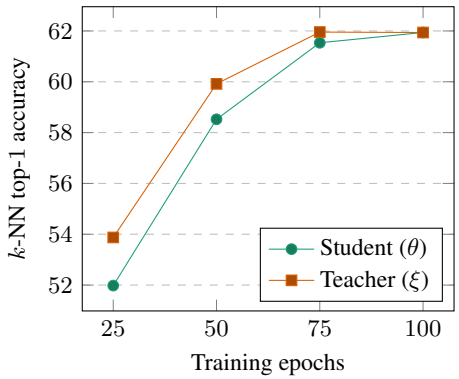

Figure B.1: During training, the representations extracted from the teacher outperform the representations from the student network.

GPUs and gradient accumulation to simulate large batch sizes. Cf. to Appendix E for a PyTorch style pseudo-code.

D Evaluation Protocols

D.1 Linear evaluation

For ImageNet evaluation, we trained a linear classifier on top of the frozen representations extracted from the last average pooling layer of the ResNet50 encoder, for 100 epochs, following He et al.'s [11] protocol. We minimize the cross-entropy loss with the LARS optimizer, a learning rate of 1.0, and a batch size of 4096 observations. For each input image, we take a random crop followed by a resize to 224×224, and an optional horizontal flipping. For testing, images are resized to 256×256 and center-cropped to 224×224.

D.2 Few-shot evaluation

We measure the few-shot learning capabilities of SSL methods on the Pascal VOC07 and iNaturalist 2018 datasets. For VOC07, we are interested in the multi-label classification performance. We closely follow Li et al.'s [13] protocol and train Linear SVNs on fixed 2048-dim representations from many SSL ResNet-50 encoders.

For INat-2018, we expand the few-shot evaluation challenge to a complex scenario containing more than 8k classes. We train linear classifiers for 20 epochs on fixed 2048-dim representations. We use the SGD optimizer. The learning rate starts at 0.03 and decays by a factor of 10 in epochs 12 and 18, respectively.

For both datasets, we vary the number n of labeled examples per class and report the average results across 5 independent runs.

D.3 k-NN evaluation

To perform the k-NN evaluation, we use pre-trained SSL ResNet50 encoders as feature extractors to compute and store the representations from many vision datasets. Following Caron et al.'s [4] setup, the representation vector for a test image is compared against all representations from the training split and a prediction is made via weighted voting. If one of the closest neighbors has the same class as the test image, it contributes to the final voting as $\alpha_i = \exp\left(\frac{M_i z}{\tau}\right)$ where M is a memory bank containing representations from the training data, z is the representation from the test data, and τ is the temperature hyper-parameter. For all experiments, we run k-NN with configurations of $K_{\text{near}} \in \{10, 20, 100, 200\}$. For most experiments, a value of $k = 20$ is consistently the best setup across all methods.

D.4 k-means evaluation

Similar to k-NN evaluation, we take self-supervised pre-trained encoders and use them to extract 2048-dim feature vectors from the training set of datasets like CIFAR-10/100 and ImageNet-1M. For ImageNet-1M, we use only 10% of the training data following the 10% subset from Chen et al. [5]. We fit k-means classifiers on the learned representations of the training set and use the validation split to assess the quality of the learned prototypes. We report three metrics to assess clustering performance: Normalized Mutual Information (NMI), Adjusted Mutual Information (AMI), and Adjusted Rand Index (ARI). The number of prototypes k is set to be the number of true classes of each dataset. We use the faiss library [12] for fast k-means. For each experiment, we run k-means for 100 iterations, 20 redos, and spherical normalization. To measure the clustering performance of CARP, we observed that a 400 epoch model with a learning rate of 0.3 slightly outperformed the other instances; therefore, we use this model to report results in Table 2. This instance of CARP uses two views and is only used for clustering evaluation.

D.5 Image retrieval evaluation

We strictly follow the evaluation script `eval_image_retrieval.py` provided by Caron et al. [4], for the image retrieval evaluation task. The script uses the revisited Oxford and Paris image retrieval

datasets [15]. The dataset contains three protocols of varying difficulty levels. We take the ImageNet pre-trained ResNet-50 encoder from CARP, freeze the weights, and apply k-NN evaluation directly to the frozen 2048-d features for retrieval, conditioned on a query image.

D.6 Copy detection evaluation

We strictly follow the evaluation script `eval_copy_detection.py` provided by Caron et al. [4] for copy detection evaluation. The evaluation is performed on the INRIA Copydays dataset [8]. The dataset contains holiday pictures in the format query/database. Each image has suffered three kinds of artificial attacks: JPEG, cropping, and "strong." We report performance evaluation on the "strong" subset. Images in the "strong" subset were intentionally distorted by blur, insertions, print, and scan. The task is to recognize these images despite distortion. We take the frozen CARP ResNet-50 encoder and extract 2048-dim vectors from query and database images at resolution 320^2. Then, we perform copy detection with cosine similarity between query and database features. We report mean average precision (mAP) as a performance metric. Unlike the benchmark described by Caron et al. [4], we do not utilize additional distractors, nor do we centralize the data using statistics learned in a different set on images.

E Pseudocode of CARP in a PyTorch-like Style

```
# NB: number of random prototypes within a block
# K: number of prototypes
# NP: number of blocks in the partition, i.e. K // NB
# N: batch size
for x1, x2 in loader:
    # student and teacher branches
    z1, w1 = enc(x1), mom_enc(x1) # [N, K]
    z2, w2 = enc(x2), mom_enc(x2) # [N, K]

    s_logits, t_logits = [z1, z2], [w1, w2]

    # sample cluster indices with no replacement
    rand_proto_ids = multinomial(ones(K), K, False)
    split_proto_ids = stack(split(rand_proto_ids, NB))
    preds_list, targets_list = [], []

    for s_log, t_log in zip(s_logits, t_logits):
        preds = get_logits_gr(s_log, split_proto_ids)
        targets = get_logits_gr(t_log, split_proto_ids)

        preds_list.append(preds)
        targets_list.append(targets)

    loss = loss_fn(preds_list, targets_list)
    # perform gradient descent steps


def loss_fn(s_list, t_list):
    c_loss = consistency_loss(s_list[0], t_list[1]
    c_loss += consistency_loss(s_list[1], t_list[0]

    s = cat(s_list, dim=1)
    t = cat(t_list, dim=1)
    probs = cat([s, t], dim=1).transpose(0, 1)

    e_loss = kl_div(mean(probs, dim=0))
    return c_loss + e_loss

def consistency_loss(s, t):
    loss = einsum("knc,knc->kn", [s, t])
    return -log(loss).mean()
```

```
def kl_div(p):
    return mean(log(K) + sum(p * log(p), dim=-1))

def get_logits_gr(logits, proto_ids):
    logits_gr = logits[:, proto_ids.flatten()]
    logits_gr = logits_gr.split(NB, dim=1)
    logits_gr = stack(logits_gr, dim=0)
    return softmax(logits_gr, dim=-1) # [NP, N, NB]
```

References

[1] Y.M. Asano, C. Rupprecht, and A. Vedaldi. Self-labelling via simultaneous clustering and representation learning. In *Inter. Conf. Learn. Represent. (ICLR)*, 2020. URL https://openreview.net/forum?id=Hyx-jyBFPr.

[2] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *European Conf. Comput. Vis. (ECCV)*, pages 132–149, 2018.

[3] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, 2020.

[4] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pages 9650–9660, 2021.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Inter. Conf. Mach. Learn. (ICML)*, 2020.

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

[7] Xinlei Chen*, Saining Xie*, and Kaiming He. An empirical study of training self-supervised vision transformers. In *IEEE Inter. Conf. Comput. Vis. (ICCV)*, 2021.

[8] Matthijs Douze, Hervé Jégou, Harsimrat Sandhawalia, Laurent Amsaleg, and Cordelia Schmid. Evaluation of gist descriptors for web-scale image search. In *ACM Inter. Conf. Image Video Ret. (CIVR)*, pages 1–8, 2009.

[9] 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 *Adv. Neural Inf. Process. Sys. (NeurIPS)*, 2020.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pages 770–778, 2016.

[11] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pages 9729–9738, 2020.

[12] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[13] Junnan Li, Pan Zhou, Caiming Xiong, and Steven C.H. Hoi. Prototypical contrastive learning of unsupervised representations. In *Inter. Conf. Learn. Represent. (ICLR)*, 2021.

[14] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Inter. Conf. Learn. Represent. (ICLR)*, 2016.

[15] Filip Radenović, Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondřej Chum. Revisiting oxford and paris: Large-scale image retrieval benchmarking. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pages 5706–5715, 2018.

[16] Thalles Santos Silva and Adín Ramírez Rivera. Consistent assignment for representation learning. In *Inter. Conf. Learn. Represent. Wksps. (ICLRW)*, 2021.

[17] Guangrun Wang, Keze Wang, Guangcong Wang, Philip HS Torr, and Liang Lin. Solving inefficiency of self-supervised representation learning. In *IEEE Inter. Conf. Comput. Vis. (ICCV)*, pages 9505–9515, 2021.

[18] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[19] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *Inter. Conf. Learn. Represent. Wksps. (ICLRW)*, 2021.