REPRESENTATION LEARNING VIA CONSISTENT ASSIGNMENT OF VIEWS OVER RANDOM PARTITIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Consistent Assignment of views over Random Partitions (CARP), a self-supervised clustering method for representation learning of visual features. CARP learns prototypes in an end-to-end online fashion using gradient descent without additional non-differentiable modules to solve the cluster assignment problem. We present a new pretext task based on random partitions of prototypes by enforcing consistency between views' assignments over these random subsets. We use a fast (student) and a slow (teacher) learners to provide stable targets for the assignment task. We present an extensive ablation study and show that our proposed random partition pretext task (1) improves the quality of the learned representations by devising multiple random classification tasks and (2) improves training stability and prevents collapsed solutions in joint-embedding training. CARP achieves top-1 linear accuracy of 71.7 % and k-NN performance of 64.8 % on the ImageNet-1M, surpassing contemporary work under limited training conditions. When trained for longer epochs, CARP outperforms state-of-the-art methods in the k-NN evaluation and performs comparably in other benchmarks.

1 INTRODUCTION

Learning from unlabeled data has been one of the main challenges in computer vision. Recent approaches based on self-supervised learning (SSL) have significantly reduced the gap between supervised and unsupervised pre-trained representations. Nowadays, self-supervised pre-training on vast quantities of unlabeled data, prior to learning a downstream supervised task of interest, can be more effective than supervised pre-training for many tasks (Caron et al., 2020; Gidaris et al., 2020; Grill et al., 2020).

We divide current SSL methods for representation learning into two classes: (1) self-supervised embedding prediction (Chen et al., 2020a; He et al., 2020; Tian et al., 2020) and (2) clustering (Asano et al., 2020; Caron et al., 2018; 2019; Li et al., 2021). As the name suggests, the self-supervised methods based on embedding prediction work directly in the embedding space. They are trained with either contrastive or non-contrastive loss functions. Instead of reconstructing the input signal, their loss function operates in the representation space by approximating embeddings of the same view and optionally pushing representations from different views apart. On the other hand, clustering methods discretize the representation space by learning a finite set of prototypes. These prototypes aggregate augmented versions of the same image and cluster representations from different images that are similar enough to be assigned to the same prototype. Nevertheless, recent self-supervised methods for representation learning are built on top of the same ideas: (1) synthetic view generation, (2) joint embedding architecture, and (3) a similarity-based loss function.

The most challenging and significant difference among these methods is how they avoid trivial solutions when training joint-embedding architectures. One strategy is to incorporate a contrastive term in the loss function. Contrastive methods not only approximate embeddings from views of the same image but explicitly push representations of different views apart. Without the contrastive force, the model trivially optimizes the loss function. Non-contrastive methods employ a "stop-gradient" operation in one of the branches of the joint-embedding architecture so that gradients only flow from one branch at a time. This "hack" avoids the mode-collapse of joint-embedding architectures even if we only approximate representations of the same view without a contrastive term (Chen & He, 2021).

Among self-supervised clustering methods, recent work proposed avoiding trivial solutions by using separate modules to solve the cluster assignment problem. Caron et al. (2020) and Asano et al. (2020) proposed to use the Sinkhorn-Knopp (Cuturi, 2013) algorithm to solve the cluster assignment problem and provide suitable targets for self-supervised optimization. Work from Li et al. (2021) and Van Gansbeke et al. (2020b) avoid trivial solutions by using classic machine learning methods such as k-Means clustering or k-Nearest Neighbor as intermediate steps in the learning process.

Consistent assignments (Silva & Ramírez Rivera, 2021) were recently proposed as a way to learn prototypes to improve the representations. Similarly, our loss function imposes two constraints (1) consistent assignment of views over learnable prototypes and (2) a uniform distribution for the average predictions within a batch. However, we show that such a strategy does not scale well to enormous datasets containing millions of classes. In such situations, we need to model a large number of prototypes while enforcing consistent assignment between views and uniform distribution over the average predictions. We show that a naive implementation of this strategy makes the learning problem challenging from a training stability perspective, where the model quickly settles for a trivial solution by assigning all views' embeddings to the same prototype.

To overcome these issues, we propose a novel self-supervised approach based on the consistent assignment of views over random partition sets (CARP). We train CARP to minimize a consistency loss which encourages the model to assign different views of the same unlabeled example to the same prototype. We solve the dimensionality problem by enforcing smaller classification problems through the introduction of random partitions that enforce consistency and regularize the model. The energy between the view's representations and the trainable prototypes (within random partitions) allows us to automatically bootstrap predictions and targets to our consistency loss. Our contributions are three-fold:

1. A novel and entirely online joint-embedding learning strategy based on self-supervised clustering, see Fig. 1. We propose a divide and conquer pretext task based on randomly generated partitions of learnable prototypes. Our loss function allows stable training of joint-embedding architectures in a self-supervised context.
2. A framework that simplifies self-supervised training and does not require normalization tecniques (Caron et al., 2020; Chen et al., 2020a) or the necessity of mining negatives for contrastive training (Chen et al., 2020a; Misra & Maaten, 2020; Oord et al., 2018).
3. A differentiable assigner module that generates soft pseudo-labels by comparing the representations of image views to prototypes within random partition subsets. To avoid trivial solutions, we enforce the average predictions over a batch to be non-informative over the set of prototypes within a random subset.

2 Related Work

Self-supervised embedding prediction methods operate directly in the embedding space work by learning a metric such that embeddings from views of the same image are closer to one another while embeddings from views of different images are far away in the feature space. These methods can be trained using contrastive or non-contrastive loss functions. Methods that minimize a loss function with a contrastive term date back to 1990s (Bromley et al., 1993; Chopra et al., 2005; Goldberger et al., 2004). They must explicitly find representations from non-correlated images to use as negatives. Recent contrastive methods include InstDisc (Wu et al., 2018), CPC (Oord et al., 2018), SimCLR (Chen et al., 2020a) and MoCo (Chen et al., 2020c; He et al., 2020). These methods learn unsupervised representations by minimizing nearly the same contrastive loss function, i.e., the InfoNCE (Oord et al., 2018). The InfoNCE loss is used to optimize a pretext task called instance discrimination, in which the network is challenged to correctly identify the pair of positive embeddings among a set of negative pairs. CARP does not operate in the embedding space, nor is it a contrastive method. Instead of directly optimizing views' embeddings, we learn a set of general prototypes using a random partition strategy that stabilizes the learning process and avoids trivial solutions commonly found when training joint-embedding architectures.

On the other hand, non-contrastive methods work by approximating embeddings of the same view in the feature space. The main advantage is not requiring explicit opposing representations in the

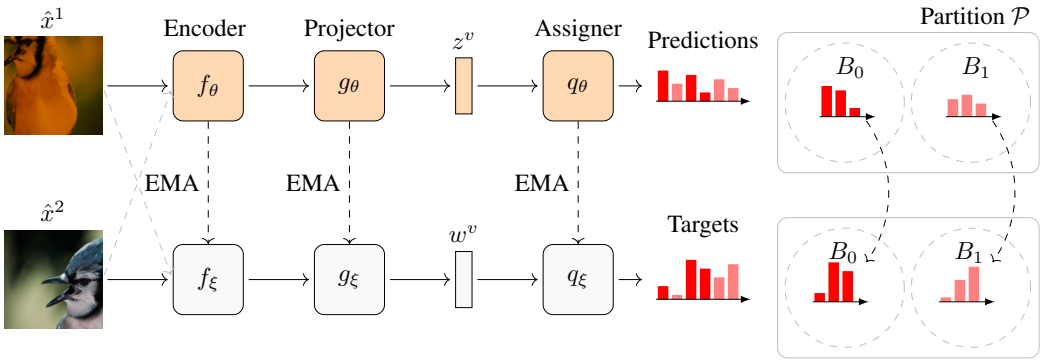

Figure 1: CARP's training architecture. From two synthetic views generated from a given image, we train an encoder network $f_{(\cdot)}$, followed by an MLP projection head $g_{(\cdot)}$ to produce representation vectors z^v and w^v, respective to the parameters θ and ξ for each branch, for each view indexed by v. The representations are fed to an assigner function $q_{(\cdot)}$ that produces normalized distributions of views w.r.t. the learnable prototypes. Note that θ are the trainable weights and ξ are an exponential moving average of θ. We create random partitions by randomly arranging the prototypes into a predefined number of blocks. Then, we enforce consistent assignment of views over prototypes within the blocks. To avoid trivial solutions, we force the average predictions over a batch to be uniform across each partition block.

loss formulation. To avoid trivial solutions, a common approach is to implement a "stop-gradient" operation that prevents the gradient signal from flowing to the two branches of the join-embedding architecture simultaneously. BYOL (Grill et al., 2020), and SimSiam (Chen & He, 2021), are examples of such methods. These non-contrastive methods optimize a loss function that takes a pair of representations from views of the same image as input and learns a metric by making these representations similar, usually using a mean squared error loss. CARP takes advantage of non-contrastive training since it does not require mining negatives for optimization. Also, different from Grill et al.'s (2020) work, CARP does not require a momentum encoder, though using it significantly improves the learned representations. CARP trains a joint-embedding architecture and uses the "stop-gradient" operation in conjunction with a regularized pretext task based on random partitions of prototypes to avoid mode collapse.

Self-supervised clustering methods do not work directly on the views' embeddings. Instead, they learn a set of prototypes that are used to solve subsequent pretext tasks in different ways. Caron et al. (2018), for instance, use k-Means clustering at every epoch to cluster the representations from the entire dataset and produce pseudo-labels for the training images. Then, a classifier head is trained to predict the pseudo-labels. Following, Caron et al. (2019) proposed a method to combine the rotation prediction pretext task (Gidaris et al., 2018) with clustering. Li et al. (2021) presented a method based on expectation-maximization (EM) that merges clustering with the contrastive learning framework from (He et al., 2020). Recent work by Caron et al. (2020) and Asano et al. (2020) combines self-supervised learning with clustering. They utilize the non-differentiable Sinkhorn-Knopp algorithm to solve the cluster assignment problem without falling into collapsed solutions. More recently, Caron et al. (2021) proposed a teacher-student self-supervised clustering method to learn unsupervised representations using vision transformers (Vaswani et al., 2017). The system keeps a moving average of previous predictions, and uses it as a mean subtraction normalizer, to center the current targets from the teacher network and avoids collapsed solutions.

Contrast from previous approaches. Instead of solving the cluster assignment problem using a non-differentiable algorithm such as the Sinkhorn- Knopp, our model is trained end-to-end with gradient descent. Different from Caron et al.'s (2021) work, our model does not require extra momentum encoders or data structures to store previous predictions to avoid trivial solutions. CARP avoids trivial solutions by posing the optimization problem at the level of random partitions of prototypes. We penalize the learning algorithm if the average predictions over a minibatch of views are not distributed uniformly across prototypes. Unlike Caron et al.'s (2019) work, our method does not

require clustering the entire dataset every epoch to generate pseudo-labels. Instead, CARP generates soft pseudo-labels in an online fashion from examples in a single minibatch.

3 CONSISTENT ASSIGNMENT OF VIEWS

From an image x_i, we create two views, denoted as $\hat{x}_i^1 = T(x_i)$ and $\hat{x}_i^2 = T(x_i)$, using a stochastic function T that applies a set of random image transformations to x_i (cf. Appendix B.1). We extend Silva & Ramírez Rivera's (2021) work by introducing another pair of streams and contrasting between a slow and fast learned versions of the embeddings. CARP is a joint-embedding architecture with two modules: a differentiable (student) and a non-differentiable (teacher) branch. Each module has its own set of weights and the same architectural design. Both contain an encoder $f_{(\cdot)}$ and a projection head $g_{(\cdot)}$. The differentiable student module receives a view and produces an embedding vector $z_i^v = g_\theta(f_\theta(\hat{x}_i^v))$, for $v \in \{1, 2\}$. Similarly, the non-differentiable teacher receives a view and produces a target embedding $w_i^v = g_\xi(f_\xi(\hat{x}_i^v))$. The weights ξ_j are updated at every training step j, following an exponential moving average (EMA), $\xi_{j+1} = \eta\xi_j + (1 - \eta)\theta_j$, where $\eta = [0, 1]$ is a decay rate scalar. We omit the training step subscripts, j, for brevity, in this paper.

The assigner function $q(h, \Phi) = \text{softmax}\left(h \cdot \Phi^T\right)$ linearly combines the views representation vectors, h, with a set of prototypes, Φ. The objective is to learn a set of prototype vectors C to discretize the embedding space. Note that these prototypes are not meant to represent the true classes of the data. Instead, these general prototypes can be viewed as anchors to attract views of a given image to a commonplace in the embedding space. The function $q(\cdot, \cdot)$ receives the views' representations, z_i^v and $w_i^v \in \mathbb{R}^{1\times d}$, as input and outputs normalized probability vectors relating the views' embeddings with the prototypes such that $s_i^v = q(z_i^v, C)$ and $t_i^v = q(w_i^v, C)$, where s_i^v and $t_i^v \in \mathbb{R}^{1\times K}$ are the normalized probabilities of a view, $\hat{x}_i^v$, w.r.t. the prototypes $C \in \mathbb{R}^{K\times d}$, where d is the dimensionality of the embedding vector and K is the number of prototypes.

To avoid trivial solutions in the joint-embedding training, we need a loss function that avoids the assignment of all representation vectors z_i to a unique prototype. Related work by Asano et al. (2020) and Caron et al. (2020) utilize a non-differentiable Sinkhorn-Knopp algorithm to solve the cluster assignment problem. Others (Li et al., 2021; Van Gansbeke et al., 2020a), use classic machine learning algorithms such as k-Means clustering and k-Nearest Neighbors. Unlike previous work, we seek a method that solves the cluster assignment problem in an online fashion using gradient descent.

We propose a loss function composed of two terms: consistency and entropy. The consistency term learns the relations between embedding vectors and prototypes. In other words, the consistency loss enforces different views of the same image to be assigned to the same prototype with high confidence. Given the normalized probability vectors a and b, we define their consistency term as $\mathcal{L}_c(a, b) = -\log \langle a, b\rangle$, where $\langle \cdot, \cdot \rangle$ is a dot product.

The consistency loss is optimized when the two views $\hat{x}_i^1$ and $\hat{x}_i^2$ get assigned to the same prototype with maximal confidence. Thus, optimization is achieved when the probability distributions of the two views s_i^1 and s_i^2 resemble equal one-hot vectors. Unlike prior work, we propose to contrast probability distributions across views and streams (i.e., between student and teacher) to obtain comparisons anchored at different solutions in the loss-landscape. See the final loss (2) for the formalization of the previous statement.

If we only optimize the consistency loss, $\mathcal{L}_c$, training would collapse to a state where all views are assigned to the same prototype. To ensure that all the prototypes get roughly the same number of assignments over a batch of views, let us define the function

$$\text{avg}\left(\{(a_i, b_i)\}_{i=1}^L\right) = \frac{1}{L}\sum_{i=1}^{L} a_i + b_i \tag{1}$$

as the average probability across the representations within a batch of size L. For our distributions, we define $\bar{p} = \text{avg}(\{(s_i^v, t_i^v)\}_{i=1}^N)$. If we maximize the entropy of the mean probabilities of a batch, $H(\bar{p})$, we will encourage the average predictions to be closer to a uniform distribution. In other words, this term prevents the system from collapsing by ensuring that, for a batch of size N, on average, each prototype roughly receives the same number of assignments. Since we want to maximize the entropy over assignments averaged within a batch, we can view the entropy term as

enforcing a uniform distribution relating embeddings to prototypes. Previous work has used this entropy term in various scenarios, ranging from discriminative unsupervised clustering to semi-supervised learning (Assran et al., 2021; Joulin & Bach, 2012; Van Gansbeke et al., 2020b). Thus, the final proposed objective to minimize is

$$\mathcal{L} = \frac{1}{N}\sum_{i}^{N} \mathcal{L}_c(s_i^1, t_i^2) + \mathcal{L}_c(s_i^2, t_i^1) - \lambda_e H(\bar{p}), \tag{2}$$

where $\lambda_e > 0$ trades off consistency at the view level with the average uniform assignment at the batch level. Note that the contribution of the entropy term decays as training progresses.

Training an unsupervised system by minimizing the loss (2) is challenging. The main limitation is how to avoid trivial solutions in unsupervised training of joint-embedding architectures. For instance, if the value of λ_e is too small, the consistency term wins the arms race, and the average distribution over the batch, $H(\bar{p})$ becomes one-hot alike, i.e., all views end up assigned to the same prototype. If the value of λ_e is too large, the entropy term gets the upper hand, and collapse is avoided (Silva & Ramírez Rivera, 2021). However, the process of view assignment is neglected over the policy of distributing views uniformly, which results in poor performance of the learned representations.

For a small number of general prototypes, training is more stable, and the model avoids collapse with a simple tuning of the λ_e parameter (Silva & Ramírez Rivera, 2021). However, for a larger number of general prototypes, stability becomes an issue. The main problem lies with the entropy term. For more interesting cases, when the distribution is larger, i.e., $K \gg N$, regular batch sizes, such as $N = 64$ or $N = 128$, become too small to properly model the distribution, i.e., the signal is too weak for most prototypes. Consequently, to avoid collapse, we need to increase the contribution of the entropy term or increase the batch size.

To address such limitation, we propose to decouple the loss function (2) into smaller sub-problems. Instead of enforcing both consistency and uniform assignments over all the general prototypes, we propose a pretext task over subsets or blocks of a random partition of the general prototype set C.

4 Assignment based on Random Partitions

Given the set of K trainable prototypes $C = \{c_1, c_2, ..., c_K\}$, we define a partition of C as $\mathcal{P} = \{B_i \subset C\}_{i=1}^{N_\mathcal{P}}$, such that $\emptyset \notin \mathcal{P}$, $\bigcup_i B_i = \mathcal{P}$ where $B_i \in \mathcal{P}$, and $B_i \cap B_j = \emptyset$ for all $B_i, B_j \in \mathcal{P}$, and $i \neq j$. We refer to B_i as a block or subset of the partition. We are interested in a partition set $\mathcal{P} = \{B_i\}_{i=1}^{N_\mathcal{P}}$ of size $N_\mathcal{P}$, i.e., $|\mathcal{P}| = N_\mathcal{P}$.

Using the concept of a partition of a set, we can define a framework of pretext tasks over partition blocks that satisfies the learning problem defined in Section 3. If the size of a partition block, B_i, equals the number of prototypes, $N_B = K$ then the partition $\mathcal{P}$ is trivial, i.e., $\mathcal{P} = \{B_1\} = \{C\}$. If the size of the partition blocks equals $N_B = 1$, then we have K blocks in $\mathcal{P}$, and each block has a unique prototype. Here, the learning task is equivalent to multiple binary classification problems, where each output score, if normalized, expresses the likelihood of a data point x_i to independently belong to each prototype.

However, if the block size $1 < N_B < K$, and N_B divides K, then the partition $\mathcal{P}$ will be composed of $N_\mathcal{P} = \lfloor K/N_P \rfloor$ blocks. We define $\mathcal{P}$ by randomly assigning N_B prototypes c_j, for $j = 0, 1, \ldots, N_B$, to each block $B_i = \{c_j\}_j$, where $i = 0, 1, \ldots, N_\mathcal{P}$.

Instead of mapping a single representation z_i^v as a linear combination of all prototypes in C, we compare the view's representations z_i^v and w_i^v against all the prototypes in the j-th block. That is, $s_{i,j}^v = q(z_i^v, B_j)$ and $t_{i,j}^v = q(w_i^v, B_j)$, for every block in the partition $\mathcal{P}$, to obtain the normalized probability distribution relating a view from image i with the prototypes of the j-th block of the random partition, where $s_{i,j}^v$ and $t_{i,j}^v \in \mathbb{R}^{1\times1\times N_P}$.

To ensure that views are consistent among the blocks, we optimize the views' distributions $s_{i,j}^v$ and $t_{i,j}^v$ over the prototypes of a block indexed by j, so that the two distributions are consistent with one another. Thus, the consistency term of our partition loss is $\mathcal{L}_c(s_{i,j}^1, t_{i,j}^2)$, where for each block B_j, the loss is minimized when the pair of views, $\hat{x}_i^1$ and $\hat{x}_i^2$, gets assigned to the same prototypes across

modules. In other words, we look for the agreement between student and teacher assignments' probabilities across views of a given sample.

Following a similar reasoning, the block-wise entropy term is defined as $H(\bar{p}_j)$, where $\bar{p}_j = \text{avg}(\{(s^v_{i,j}, t^v_{i,j})\}_{i=1}^N)$ is the average prediction over each block B_j for a batch of size N. Thus, the final objective for consistent assignment of random partition is,

$$\mathcal{L} = \frac{1}{NN_{\mathcal{P}}} \sum_{i}^{N} \sum_{j}^{N_{\mathcal{P}}} \mathcal{L}_c(s^1_{i,j}, t^2_{i,j}) + \mathcal{L}_c(s^2_{i,j}, t^1_{i,j}) - H(\bar{p}_j). \tag{3}$$

Note that to fully use the pair of views at each iteration, we symmetrically use the $\mathcal{L}_c$ consistency function. The probability vectors $t^v_{i,j}$ come from the momentum encoder and are used as target distributions.

One of the benefits of using the random partition strategy is that we no longer require the hyper-parameter λ_e to avoid trivial solutions. The stochastic nature of the random partition pretext task, blended with the multiple prediction tasks over small blocks of prototypes, provides a regularization effect that improves the learned representations and training stability.

Other clustering based methods (Caron et al., 2020; 2021) rely on sharpening the distributions to improve their self-supervised signals used as targets in a cross-entropy loss. On the contrary, our formulation does not require the temperature parameter for sharpening the predictions and guide the learning of the student. We can think of the consistency loss as implicitly learning the temperature parameter to make the predictions sharper at each iteration. This is an important advantage of our consistency loss setup in contrast to previous methods.

5 Results and Experiments

We evaluate the features learned by CARP using three main protocols: (1) downstream linear evaluation (He et al., 2020) on the ImageNet-1M dataset using linear models and k-NN classifiers, (2) transfer learning on VOC07 (Everingham et al., 2010) and (3) semi-supervised fine-tuning using 1 % and 10 % of ImageNet labels. See Appendix B.1 for the experimental setup.

5.1 Unsupervised Feature Evaluation

Table 1 reports linear evaluation performance of various self-supervised methods. See Appendix C.1 for details. CARP achieves top-1 accuracy of 71.7 % on the ImageNet validation set and surpasses contemporary methods pre-trained for 200 epochs. When trained for 400 epochs, CARP also achieves top-1 accuracy and beats the leading methods. In fact, our method trained for 400 epochs outperforms some competitors trained for at least double the number of epochs, highlighting our online-training strategy's efficiency. When trained for 800 epochs, CARP outperforms competitors in the k-NN evaluation, reaching 66.8 %. Note that SimCLR, BYOL, and SwAV use a large batch size of 4096; InfoMin aug. uses RandAugment (Cubuk et al., 2020) to generate synthetic views, and SwAV uses multi-view augmentation and extra queue containing 3840 embeddings. For fairness, we also report the performance of SwAV using two views. Table 1 also reports the transfer learning results on the PASCAL VOC07 dataset. Following Li et al. (2021), we train a support vector machine (SVM) on top of the frozen representations from the ResNet50 encoder and report mean average precision (mAP) for object classification. CARP achieves mAP of 85.6 %, surpassing competitors among methods trained for 200 epochs.

5.2 k-NN Evaluation

In Table 1, we evaluate CARP's representations using a weighted k-Nearest Neighbor (k-NN) classifier on ImageNet-1M. CARP's outperform current methods in all epoch configurations. It even surpasses methods trained for 1000 epochs using two and four times larger batch sizes. For more details on the k-NN evaluation protocol, refer to Appendix C.3.

Table 1: Evaluation using ResNet50 encoders on ImageNet-1M grouped by training epochs. "Epochs" and "Batch" columns specify the number of epochs and the respective batch size used for unsupervised pre-training. We report top-1 and top-5 accuracies using the linear evaluation protocol, k-NN performance with $K_{\text{near}} = 20$, and mAP on PASCAL VOC07. All methods employ 2×224^2 views. † Results computed by us using the officially released pre-trained models.

			ImageNet			VOC07
Method	Epochs	Batch	Top-1	Top-5	k-NN	mAP
Supervised	100	256	76.5	–	–	87.5
MoCo v2 (He et al., 2020)	200	256	67.7	–	55.6	84.0
PCL v2 (Li et al., 2021)	200	256	67.6	–	58.1	85.4
SimSiam (Chen & He, 2021)	200	256	70.0	–	–	–
CARL (Silva & Ramírez Rivera, 2021)	200	256	65.3	–	–	–
CARP (ours)	200	1024	**71.7**	**90.1**	**64.8**	**85.6**
SimCLR (Chen et al., 2020a)	200	4096	61.9	–	60.7	–
InfoMin aug. (Tian et al., 2020)	200	4096	70.1	89.4	59.3†	81.2†
BYOL (Grill et al., 2020)	200	4096	70.6	–	–	–
SwAV (two-views) (Chen & He, 2021)	200	4096	69.1	–	–	–
SimSiam (Chen & He, 2021)	400	256	70.8	–	–	–
CARP (ours)	400	1024	**72.1**	90.3	**66.1**	85.9
SELA v2 (Caron et al., 2020)	400	4096	67.2	–	58.0†	85.3†
DeepCluster v2 (Caron et al., 2020)	400	4096	70.2	–	62.4†	86.6†
SwAV (two-views) (Caron et al., 2020)	400	4096	70.1	–	61.3†	**86.7**†
MoCo v2 (He et al., 2020)	800	256	71.1	90.1	60.7	–
SimSiam (Chen & He, 2021)	800	256	71.3	–	–	–
CARP (ours)	800	1024	73.3	**91.1**	**66.8**	86.4
InfoMin aug. (Tian et al., 2020)	800	4096	73.0	**91.1**	65.3†	83.5†
SimCLR v2 (Chen et al., 2020b)	800	4096	71.7	–	–	–
SwAV (multi-views) (Caron et al., 2020)	800	4096	**75.3**	–	66.3	**88.9**
SwAV (two-views) (Chen & He, 2021)	800	4096	71.8	–	–	–
BarlowTwins (Zbontar et al., 2021)	1000	2048	73.2	91.0	66.1†	86.3†
SimCLR (Chen et al., 2020a)	1000	4096	69.3	89.0	60.7	80.5
BYOL (Grill et al., 2020)	1000	4096	74.3	–	66.6	–

5.3 Semi-Supervised Feature Evaluation

We evaluate CARP's representations on semi-supervised protocols using 1 % and 10 % of labeled data using the same split as SimCLR (Chen et al., 2020a), following the protocol proposed by Misra & Maaten's (2020). Refer to Appendix C.2 for details. In Table 2, CARP achieves the best overall top-5 accuracy of 75.4 % and 87.1 % across models pre-trained for 200 epochs for the respective semi-supervised data regimes.

6 Ablations

In this section, we answer whether a consistent assignment of views over random partitions benefits the learned representations and training stability. We ablate CARP's main hyperparameters to establish a good baseline for running experiments on the ImageNet-1M dataset. 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 B.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.

Table 2: Top-5 accuracy on ImageNet validation set of self-supervised models that are fine-tuned on 1 % and 10 % of labeled data. We report the literature's accuracy from the corresponding papers.

Method	Arch.	Epochs	Top-5 accuracy 1 %	Top-5 accuracy 10 %
Supervised (Zhai et al., 2019)	RN50	100	48.4	80.4
SimCLR (Chen et al., 2020a)	RN50	200	56.5	82.7
MoCo v2 (He et al., 2020)	RN50	200	66.3	84.4
PCL v2 (Li et al., 2021)	RN50	200	73.9	85.0
CARP (ours)	RN50	200	**75.4**	**87.1**
CARP (ours)	RN50	400	76.4	87.5
CARP (ours)	RN50	800	77.5	87.9
PIRL (Misra & Maaten, 2020)	RN50	800	57.2	83.8
SwAV (multi-view) (Caron et al., 2020)	RN50	800	78.5	**89.9**
SimCLR (Chen et al., 2020a)	RN50	1000	75.5	87.8
BYOL (Grill et al., 2020)	RN50	1000	78.4	89.0
Barlow Twins (Zbontar et al., 2021)	RN50	1000	**79.2**	89.3

Table 3: 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**

Table 4: CARP learns better representations when larger batch sizes are employed.

bs	128	256	512	1024
k-NN	46.56	51.32	54.23	**56.63**

6.1 Does the number of learnable prototypes affect the learned representations?

Table 3 examines the effect of training CARP with different configurations of prototypes K. Similar to other clustering-based self-supervised learning methods (Caron et al., 2020; 2021; Li et al., 2021), we notice that 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.

6.2 Training CARP with different batch sizes

Most self-supervised methods (Caron et al., 2020; 2021; Grill et al., 2020; Zbontar et al., 2021) report their best results when using substantially large batch sizes. In Table 4, we observe a similar pattern when training CARP. Using our default configuration of 1024 observations yields a k-NN top-1 performance 10% higher than a batch size of 128. Table 4 confirms that training with large batch sizes benefits the learned representations. However, training with smaller batch sizes requires further tuning of other hyperparameters, such as the block size N_B. Specifically, we observed that reducing the block size N_B improves the learned representations when training with small batch sizes, which makes CARP robust to low resource training.

6.3 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 5 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 Appedix A.1. We also present an ablation on the effect of the momentum update in Table 6.

Table 5: 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 6: 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**

Table 7: In 'constant,' partition blocks are created once, sequentially, and kept fixed during training. In 'random,' blocks are recreated at every training step.

Epochs	25	50	75	100
Constant	45.15	49.89	52.81	53.32
Random	**48.68**	**53.98**	**55.93**	**56.38**

We observe that using the divide and conquer approach of devising random partitions from the learnable prototypes avoids collapsed solutions. 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.

For the case where we set the block size $N_B = 64$, CARP reaches a k-NN accuracy of 50.75% in 50 epochs. On the other hand, if we set a single partition $N_B = 65536$, the training collapses, and the learned representations are useless. 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 prediction task becomes easier with fewer prototypes to ensure consistency between views. 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 $N_B = 512$.

6.4 Exploring different strategies to build partitions

Table 7 explores different ways of creating random partition blocks from the learnable prototypes. CARP default strategy recreates the random partitions at every training step. In other words, at every training iteration, we assign N_B randomly chosen prototypes to the $N_{\mathcal{P}}$ partition blocks. Table 7 contrasts CARP's default strategy with one in which the partition blocks are created only once, in a sequential manner, and kept fixed throughout training. We observe that training CARP with fixed partition blocks does yield useful representations. However, as measured by k-NN performance, randomly recreating the partition blocks at each iteration further benefits the learned representations. Since the partition blocks are randomly recreated at every iteration of gradient descent, the classification subproblems are always different. In practice, this variance allows for many unique pretext tasks at each iteration, which provides a positive regularization effect on CARP.

7 Conclusions

We presented consistent assignment of views over random partitions (CARP), a self-supervised clustering method for representation learning of unlabeled images. CARP learns prototypes in an online fashion end-to-end using gradient descent by minimizing a cost function that optimizes consistency between views' assignments and uniform distribution across prototypes within a random partition. Our experiments demonstrate that posing the optimization problem at the level of random partitions of learnable prototypes not only stabilizes training by avoiding trivial solutions in joint-embeddings architectures but also increases the performance of the learned representation. Moreover, by contrasting the probability assignments of the teacher and the student streams, we can create more stable contrast pairs.

REFERENCES

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.

Mahmoud Assran, Mathilde Caron, Ishan Misra, Piotr Bojanowski, Armand Joulin, Nicolas Ballas, and Michael Rabbat. Semi-supervised learning of visual features by non-parametrically predicting view assignments with support samples. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 8443–8452, 2021.

Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, pp. 669–688. World Scientific, 1993.

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

Mathilde Caron, Piotr Bojanowski, Julien Mairal, and Armand Joulin. Unsupervised pre-training of image features on non-curated data. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 2959–2968, 2019.

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.

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)*, pp. 9650–9660, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020a.

Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. *Adv. Neural Inf. Process. Sys. (NeurIPS)*, 33:22243–22255, 2020b.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 15750–15758, 2021.

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

Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, volume 1, pp. 539–546. IEEE, 2005.

Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Adv. Neural Inf. Process. Sys. (NeurIPS)*, volume 33, pp. 18613–18624. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/d85b63ef0ccb114d0a3bb7b7d808028f-Paper.pdf.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Adv. Neural Inf. Process. Sys. (NeurIPS)*, 26:2292–2300, 2013.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Inter. J. Comput. Vis.*, 88(2):303–338, 2010.

Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *Inter. Conf. Learn. Represent. (ICLR)*, 2018.

Spyros Gidaris, Andrei Bursuc, Gilles Puy, Nikos Komodakis, Matthieu Cord, and Patrick Pérez. Online bag-of-visual-words generation for unsupervised representation learning. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2020.

Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, volume 17, 2004.

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. *arXiv preprint arXiv:2006.07733*, 2020.

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)*, pp. 770–778, 2016.

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)*, pp. 9729–9738, 2020.

Armand Joulin and Francis Bach. A convex relaxation for weakly supervised classifiers. *arXiv preprint arXiv:1206.6413*, 2012.

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

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

Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 6707–6717, 2020.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. In *Wksp. Adv. Neural Inf. Process. Sys. (NeurIPSW)*, 2018.

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

Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Adv. Neural Inf. Process. Sys. (NeurIPS)*, volume 33, pp. 6827–6839. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/4c2e5eaae9152079b9e95845750bb9ab-Paper.pdf.

Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Learning to classify images without labels. *arXiv preprint arXiv:2005.12320*, 2020a.

Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Scan: Learning to classify images without labels. In *European Conf. Comput. Vis. (ECCV)*, pp. 268–285. Springer, 2020b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, volume 30, 2017.

Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 3733–3742, 2018.

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

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.

Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *IEEE/CVF Inter. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 1476–1485, 2019.

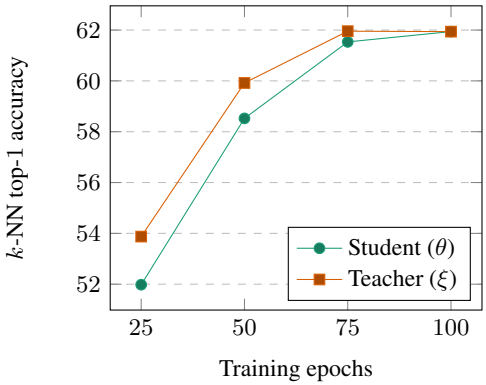

Figure A.1: During training, the representations extracted from the teacher outperform the representations from the student network.

A Additional Experiments

A.1 The importance of the momentum encoder

Table 5 contrasts CARP's joint-embedding architectures with and without a momentum encoder. This is equivalent to setting $\eta = 0$ in the momentum encoder update equation. Different from other self-supervised methods (Caron et al., 2021; Grill et al., 2020), CARP works with both setups. However, we observe that using a momentum encoder significantly boosts the performance of the learned representations. Table 5 shows that regardless of block sizes, representations learned using a momentum encoder-based architecture consistently outperform the siamese counterpart.

A.2 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 student throughout the learning process. The addition of the momentum encoder raises the question of which module produces the best representations. To answer this question, Fig. A.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.

B Implementation Details

B.1 Experimental Setup

To ensure fair comparison to existing self-supervised methods, we follow the same protocols of He et al. (2020) for training and evaluating CARP. We train CARP on the ImageNet-1M unlabeled dataset using a ResNet50 (He et al., 2016) encoder. We take the output representation of the last global average pooling layer of the ResNet50 encoder (a 2048-dim vector) and projects it to a 256-dim vector. Following Caron et al.'s (2021) work, our MLP projection head contains 3 dense layers with batch normalization and the GeLU activation functions. 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. (2020) to generate synthetic views. CARP is trained with SGD, end to end, with weight decay of 0.000001, using the LARS (You et al., 2017) optimizer with learning starting from 0.6 and decaying to 0.06 with a cosine scheduling (Loshchilov & Hutter, 2016) without warmups. We train the system with a global batch size of 1024 observations evenly split across 4 NVIDIA RTX A6000.

B.2 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 branches
    z1 = enc(x1) # [N, K]
    z2 = enc(x2)
    # teacher branches
    w1 = mom_enc(x1) # [N, K]
    w2 = mom_enc(x2)

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

    # sample cluster indices with no replacement
    rand_cluster_indices = multinomial(ones(K), K, False)

    split_cluster_ids = stack(split(rand_cluster_indices, NB))

    preds_list = []
    targets_list = []

    for s_log, t_log in zip(s_logits, t_logits):
        preds_group = get_logits_group(s_log, split_cluster_ids)
        targets_group = get_logits_group(t_log, split_cluster_ids)

        preds_list.append(preds_group)
        targets_list.append(targets_group)

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


def loss_fn(s_list, t_list):
    consistency = consistency_loss(s_list[0], t_list[1]
    consistency += 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) # [4*N, NP, NB]

    entropy = kl_div(mean(probs, dim=0))
    return consistency + entropy

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_group(logits, split_cluster_ids):
    logits_group = logits[:, split_cluster_ids.flatten()]
    logits_group = logits_group.split(NB, dim=1)
    logits_group = stack(logits_group, dim=0)
    return softmax(logits_group, dim=-1) # [NP, N, NB]
```

C EVALUATION PROTOCOLS

C.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 (2020) protocol. We minimize the cross-entropy loss with the SGD optimizer, a learning rate of 0.3, and a batch size of 256 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 a size of 224×224.

C.2 SEMI-SUPERVISED EVALUATION

We append a classification layer on top of the pre-trained CARP ResNet50 encoder. The pre-trained encoder is finetuned using a learning rate of 0.002, and the classification layer uses a learning rate of 0.5. The learning rate is multiplied by a factor of 0.2 after the 12th and 16th epochs. We use gradient descent (SGD) optimizer for 20 epochs and a batch size of 256 observations.

C.3 k-NN EVALUATION

To perform the k-NN evaluation, we keep the weights of the pre-trained ResNet50 encoder fixed to compute and store the representations from the ImageNet-1M training data. Following Caron et al.'s (2021) setup, the representation vector for a test image is compared against all representations from the training data, 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 $K_{\text{near}} \in \{10, 20, 100, 200\}$ and discover that $K_{\text{near}} = 20$ is consistently the best setup across all methods.