Unsupervised Learning of Visual Features by Contrasting Cluster Assignments

Mathilde Caron[1,2] **Ishan Misra**[2] **Julien Mairal**[1]

Priya Goyal[2] **Piotr Bojanowski**[2] **Armand Joulin**[2]

[1] Inria* [2] Facebook AI Research

Abstract

Unsupervised image representations have significantly reduced the gap with supervised pretraining, notably with the recent achievements of contrastive learning methods. These contrastive methods typically work online and rely on a large number of explicit pairwise feature comparisons, which is computationally challenging. In this paper, we propose an online algorithm, SwAV, that takes advantage of contrastive methods without requiring to compute pairwise comparisons. Specifically, our method simultaneously clusters the data while enforcing consistency between cluster assignments produced for different augmentations (or "views") of the same image, instead of comparing features directly as in contrastive learning. Simply put, we use a "swapped" prediction mechanism where we predict the code of a view from the representation of another view. Our method can be trained with large and small batches and can scale to unlimited amounts of data. Compared to previous contrastive methods, our method is more memory efficient since it does not require a large memory bank or a special momentum network. In addition, we also propose a new data augmentation strategy, `multi-crop`, that uses a mix of views with different resolutions in place of two full-resolution views, without increasing the memory or compute requirements. We validate our findings by achieving 75.3% top-1 accuracy on ImageNet with ResNet-50, as well as surpassing supervised pretraining on all the considered transfer tasks.

1 Introduction

Unsupervised visual representation learning, or self-supervised learning, aims at obtaining features without using manual annotations and is rapidly closing the performance gap with supervised pretraining in computer vision [10, 24, 44]. Many recent state-of-the-art methods build upon the instance discrimination task that considers each image of the dataset (or "instance") and its transformations as a separate class [16]. This task yields representations that are able to discriminate between different images, while achieving some invariance to image transformations. Recent self-supervised methods that use instance discrimination rely on a combination of two elements: (i) a contrastive loss [23] and (ii) a set of image transformations. The contrastive loss removes the notion of instance classes by directly comparing image features while the image transformations define the invariances encoded in the features. Both elements are essential to the quality of the resulting networks [10, 44] and our work improves upon both the objective function and the transformations.

* Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France
Correspondence to `mathilde@fb.com`
Code: `https://github.com/facebookresearch/swav`

The contrastive loss explicitly compares pairs of image representations to push away representations from different images while pulling together those from transformations, or views, of the same image. Since computing all the pairwise comparisons on a large dataset is not practical, most implementations approximate the loss by reducing the number of comparisons to random subsets of images during training [10, 24, 58]. An alternative to approximate the loss is to approximate the task—that is to relax the instance discrimination problem. For example, clustering-based methods discriminate between groups of images with similar features instead of individual images [7]. The objective in clustering is tractable, but it does not scale well with the dataset as it requires a pass over the entire dataset to form image "codes" (*i.e.*, cluster assignments) that are used as targets during training. In this work, we use a different paradigm and propose to compute the codes online while enforcing consistency between codes obtained from views of the same image. Comparing cluster assignments allows to contrast different image views while not relying on explicit pairwise feature comparisons. Specifically, we propose a simple "swapped" prediction problem where we predict the code of a view from the representation of another view. We learn features by **Sw**apping **A**ssignments between multiple **V**iews of the same image (**SwAV**). The features and the codes are learned online, allowing our method to scale to potentially unlimited amounts of data. In addition, SwAV works with small and large batch sizes and does not need a large memory bank [58] or a momentum encoder [24].

Besides our online clustering-based method, we also propose an improvement to the image transformations. Most contrastive methods compare one pair of transformations per image, even though there is evidence that comparing more views during training improves the resulting model [44]. In this work, we propose `multi-crop` that uses smaller-sized images to increase the number of views while not increasing the memory or computational requirements during training. We also observe that mapping small parts of a scene to more global views significantly boosts the performance. Directly working with downsized images introduces a bias in the features [53], which can be avoided by using a mix of different sizes. Our strategy is simple, yet effective, and can be applied to many self-supervised methods with consistent gain in performance.

We validate our contributions by evaluating our method on several standard self-supervised benchmarks. In particular, on the ImageNet linear evaluation protocol, we reach 75.3% top-1 accuracy with a standard ResNet-50, and 78.5% with a wider model. We also show that our `multi-crop` strategy is general, and improves the performance of different self-supervised methods, namely SimCLR [10], DeepCluster [7], and SeLa [2], between 2% and 4% top-1 accuracy on ImageNet. Overall, we make the following contributions:

- We propose a scalable online clustering loss that improves performance by $+2\%$ on ImageNet and works in both large and small batch settings without a large memory bank or a momentum encoder.
- We introduce the `multi-crop` strategy to increase the number of views of an image with no computational or memory overhead. We observe a consistent improvement of between 2% and 4% on ImageNet with this strategy on several self-supervised methods.
- Combining both technical contributions into a single model, we improve the performance of self-supervised by $+4.2\%$ on ImageNet with a standard ResNet and outperforms supervised ImageNet pretraining on multiple downstream tasks. This is the first method to do so without finetuning the features, *i.e.*, only with a linear classifier on top of frozen features.

2 Related Work

Instance and contrastive learning. Instance-level classification considers each image in a dataset as its own class [5, 16, 58]. Dosovitskiy *et al.* [16] assign a class explicitly to each image and learn a linear classifier with as many classes as images in the dataset. As this approach becomes quickly intractable, Wu *et al.* [58] mitigate this issue by replacing the classifier with a memory bank that stores previously-computed representations. They rely on noise contrastive estimation [22] to compare instances, which is a special form of contrastive learning [29, 47]. He *et al.* [24] improve the training of contrastive methods by storing representations from a momentum encoder instead of the trained network. More recently, Chen *et al.* [10] show that the memory bank can be entirely replaced with the elements from the same batch if the batch is large enough. In contrast to this line of works, we avoid comparing every pair of images by mapping the image features to a set of trainable prototype vectors.

Clustering for deep representation learning. Our work is also related to clustering-based methods [2, 4, 7, 8, 19, 30, 59, 62, 63, 68]. Caron *et al.* [7] show that k-means assignments can be used as pseudo-labels to learn visual representations. This method scales to large uncurated dataset and can be used for pre-training of supervised networks [8]. However, their formulation is not principled and recently, Asano *et al.* [2] show how to cast the pseudo-label assignment problem as an instance of the optimal transport problem. We consider a similar formulation to map representations to prototype vectors, but unlike [2] we keep the soft assignment produced by the Sinkhorn-Knopp algorithm [13] instead of approximating it into a hard assignment. Besides, unlike Caron *et al.* [7, 8] and Asano *et al.* [2], we obtain online assignments which allows our method to scale gracefully to any dataset size.

Handcrafted pretext tasks. Many self-supervised methods manipulate the input data to extract a supervised signal in the form of a pretext task [1, 14, 31, 34, 36, 42, 45, 48, 49, 55, 56, 66]. We refer the reader to Jing *et al.* [32] for an exhaustive and detailed review of this literature. Of particular interest, Misra and van der Maaten [44] propose to encode the jigsaw puzzle task [46] as an invariant for contrastive learning. Jigsaw tiles are non-overlapping crops with small resolution that cover only part ($\sim$20%) of the entire image area. In contrast, our `multi-crop` strategy consists in simply sampling multiple random crops with two different sizes: a standard size and a smaller one.

3 Method

Our goal is to learn visual features in an online fashion without supervision. To that effect, we propose an online clustering-based self-supervised method. Typical clustering-based methods [2, 7] are offline in the sense that they alternate between a cluster assignment step where image features of the entire dataset are clustered, and a training step where the cluster assignments, *i.e.*, "codes" are predicted for different image views. Unfortunately, these methods are not suitable for online learning as they require multiple passes over the dataset to compute the image features necessary for clustering. In this section, we describe an alternative where we enforce consistency between codes from different augmentations of the same image. This solution is inspired by contrastive instance learning [58] as we do not consider the codes as a target, but only enforce consistent mapping between views of the same image. Our method can be interpreted as a way of contrasting between multiple image views by comparing their cluster assignments instead of their features.

More precisely, we compute a code from an augmented version of the image and predict this code from other augmented versions of the same image. Given two image features $\mathbf{z}_t$ and $\mathbf{z}_s$ from two different augmentations of the same image, we compute their codes $\mathbf{q}_t$ and $\mathbf{q}_s$ by matching these features to a set of K prototypes $\{\mathbf{c}_1, \dots, \mathbf{c}_K\}$. We then setup a "swapped" prediction problem with the following loss function:

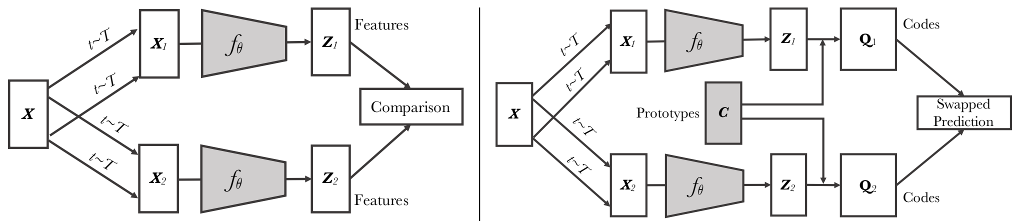
$$L(\mathbf{z}_t, \mathbf{z}_s) \quad = \quad \ell(\mathbf{z}_t, \mathbf{q}_s) + \ell(\mathbf{z}_s, \mathbf{q}_t), \tag{1}$$

where the function $\ell(\mathbf{z}, \mathbf{q})$ measures the fit between features $\mathbf{z}$ and a code $\mathbf{q}$, as detailed later. Intuitively, our method compares the features $\mathbf{z}_t$ and $\mathbf{z}_s$ using the intermediate codes $\mathbf{q}_t$ and $\mathbf{q}_s$. If these two features capture the same information, it should be possible to predict the code from the other feature. A similar comparison appears in contrastive learning where features are compared directly [58]. In Fig. 1, we illustrate the relation between contrastive learning and our method.

3.1 Online clustering

Each image $\mathbf{x}_n$ is transformed into an augmented view $\mathbf{x}_{nt}$ by applying a transformation t sampled from the set $\mathcal{T}$ of image transformations. The augmented view is mapped to a vector representation by applying a non-linear mapping f_θ to $\mathbf{x}_{nt}$. The feature is then projected to the unit sphere, *i.e.*, $\mathbf{z}_{nt} = f_\theta(\mathbf{x}_{nt}) / \|f_\theta(\mathbf{x}_{nt})\|_2$. We then compute a code $\mathbf{q}_{nt}$ from this feature by mapping $\mathbf{z}_{nt}$ to a set of K trainable prototypes vectors, $\{\mathbf{c}_1, \dots, \mathbf{c}_K\}$. We denote by $\mathbf{C}$ the matrix whose columns are the $\mathbf{c}_1, \dots, \mathbf{c}_k$. We now describe how to compute these codes and update the prototypes online.

Swapped prediction problem. The loss function in Eq. (1) has two terms that setup the "swapped" prediction problem of predicting the code $\mathbf{q}_t$ from the feature $\mathbf{z}_s$, and $\mathbf{q}_s$ from $\mathbf{z}_t$. Each term represents the cross entropy loss between the code and the probability obtained by taking a softmax of the dot



Contrastive instance learning Swapping Assignments between Views (Ours)

Figure 1: **Contrastive instance learning (left) *vs.* SwAV (right).** In contrastive learning methods applied to instance classification, the features from different transformations of the same images are compared directly to each other. In SwAV, we first obtain "codes" by assigning features to prototype vectors. We then solve a "swapped" prediction problem wherein the codes obtained from one data augmented view are predicted using the other view. Thus, SwAV does not directly compare image features. Prototype vectors are learned along with the ConvNet parameters by backpropragation.

products of $\mathbf{z}_i$ and all prototypes in $\mathbf{C}$, *i.e.*,

$$\ell(\mathbf{z}_t, \mathbf{q}_s) = -\sum_k \mathbf{q}_s^{(k)} \log \mathbf{p}_t^{(k)}, \quad \text{where} \quad \mathbf{p}_t^{(k)} = \frac{\exp\left(\frac{1}{\tau}\mathbf{z}_t^\top \mathbf{c}_k\right)}{\sum_{k'} \exp\left(\frac{1}{\tau}\mathbf{z}_t^\top \mathbf{c}_{k'}\right)}. \tag{2}$$

where τ is a temperature parameter [58]. Taking this loss over all the images and pairs of data augmentations leads to the following loss function for the swapped prediction problem:

$$-\frac{1}{N}\sum_{n=1}^{N}\sum_{s,t\sim\mathcal{T}}\left[\frac{1}{\tau}\mathbf{z}_{nt}^\top \mathbf{C}\mathbf{q}_{ns} + \frac{1}{\tau}\mathbf{z}_{ns}^\top \mathbf{C}\mathbf{q}_{nt} - \log\sum_{k=1}^{K}\exp\left(\frac{\mathbf{z}_{nt}^\top \mathbf{c}_k}{\tau}\right) - \log\sum_{k=1}^{K}\exp\left(\frac{\mathbf{z}_{ns}^\top \mathbf{c}_k}{\tau}\right)\right].$$

This loss function is jointly minimized with respect to the prototypes $\mathbf{C}$ and the parameters θ of the image encoder f_θ used to produce the features $(\mathbf{z}_{nt})_{n,t}$.

Computing codes online. In order to make our method online, we compute the codes using only the image features within a batch. Intuitively, as the prototypes $\mathbf{C}$ are used across different batches, SwAV clusters multiple instances to the prototypes. We compute codes using the prototypes $\mathbf{C}$ such that all the examples in a batch are equally partitioned by the prototypes. This equipartition constraint ensures that the codes for different images in a batch are distinct, thus preventing the trivial solution where every image has the same code. Given B feature vectors $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_B]$, we are interested in mapping them to the prototypes $\mathbf{C} = [\mathbf{c}_1, \dots, \mathbf{c}_K]$. We denote this mapping or codes by $\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_B]$, and optimize $\mathbf{Q}$ to maximize the similarity between the features and the prototypes , *i.e.*,

$$\max_{\mathbf{Q}\in\mathcal{Q}} \operatorname{Tr}\left(\mathbf{Q}^\top \mathbf{C}^\top \mathbf{Z}\right) + \varepsilon H(\mathbf{Q}), \tag{3}$$

where H is the entropy function, $H(\mathbf{Q}) = -\sum_{ij} \mathbf{Q}_{ij} \log \mathbf{Q}_{ij}$ and ε is a parameter that controls the smoothness of the mapping. We observe that a strong entropy regularization (*i.e.* using a high ε) generally leads to a trivial solution where all samples collapse into an unique representation and are all assigned uniformely to all prototypes. Hence, in practice we keep ε low. Asano *et al.* [2] enforce an equal partition by constraining the matrix $\mathbf{Q}$ to belong to the transportation polytope. They work on the full dataset, and we propose to adapt their solution to work on minibatches by restricting the transportation polytope to the minibatch:

$$\mathcal{Q} = \left\{\mathbf{Q} \in \mathbb{R}_+^{K\times B} \mid \mathbf{Q}\mathbf{1}_B = \frac{1}{K}\mathbf{1}_K, \mathbf{Q}^\top \mathbf{1}_K = \frac{1}{B}\mathbf{1}_B\right\}, \tag{4}$$

where $\mathbf{1}_K$ denotes the vector of ones in dimension K. These constraints enforce that on average each prototype is selected at least $\frac{B}{K}$ times in the batch.

Once a continuous solution $\mathbf{Q}^*$ to Prob. (3) is found, a discrete code can be obtained by using a rounding procedure [2]. Empirically, we found that discrete codes work well when computing codes in an offline manner on the full dataset as in Asano *et al.* [2]. However, in the online setting where

Method	Arch.	Param.	Top1
Supervised	R50	24	76.5
Colorization [65]	R50	24	39.6
Jigsaw [46]	R50	24	45.7
NPID [58]	R50	24	54.0
BigBiGAN [15]	R50	24	56.6
LA [68]	R50	24	58.8
NPID++ [44]	R50	24	59.0
MoCo [24]	R50	24	60.6
SeLa [2]	R50	24	61.5
PIRL [44]	R50	24	63.6
CPC v2 [28]	R50	24	63.8
PCL [37]	R50	24	65.9
SimCLR [10]	R50	24	70.0
MoCov2 [11]	R50	24	71.1
SwAV	R50	24	**75.3**

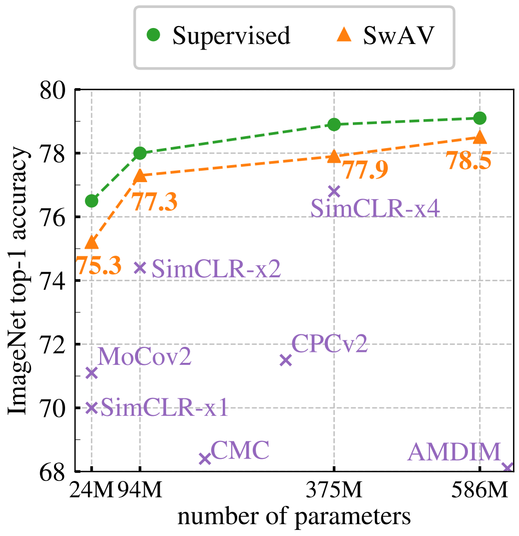


Figure 2: **Linear classification on ImageNet.** Top-1 accuracy for linear models trained on frozen features from different self-supervised methods. **(left)** Performance with a standard ResNet-50. **(right)** Performance as we multiply the width of a ResNet-50 by a factor $\times 2$, $\times 4$, and $\times 5$.

we use only minibatches, using the discrete codes performs worse than using the continuous codes. An explanation is that the rounding needed to obtain discrete codes is a more aggressive optimization step than gradient updates. While it makes the model converge rapidly, it leads to a worse solution. We thus preserve the soft code $\mathbf{Q}^*$ instead of rounding it. These soft codes $\mathbf{Q}^*$ are the solution of Prob. (3) over the set $\mathcal{Q}$ and takes the form of a normalized exponential matrix [13]:

$$\mathbf{Q}^* = \text{Diag}(\mathbf{u}) \exp\left(\frac{\mathbf{C}^\top \mathbf{Z}}{\varepsilon}\right) \text{Diag}(\mathbf{v}), \tag{5}$$

where $\mathbf{u}$ and $\mathbf{v}$ are renormalization vectors in $\mathbb{R}^K$ and $\mathbb{R}^B$ respectively. The renormalization vectors are computed using a small number of matrix multiplications using the iterative Sinkhorn-Knopp algorithm [13]. In practice, we observe that using only 3 iterations is fast and sufficient to obtain good performance. Indeed, this algorithm can be efficiently implemented on GPU, and the alignment of 4K features to 3K codes takes 35ms in our experiments, see § 4.

Working with small batches. When the number B of batch features is too small compared to the number of prototypes K, it is impossible to equally partition the batch into the K prototypes. Therefore, when working with small batches, we use features from the previous batches to augment the size of $\mathbf{Z}$ in Prob. (3). Then, we only use the codes of the batch features in our training loss. In practice, we store around 3K features, *i.e.*, in the same range as the number of code vectors. This means that we only keep features from the last 15 batches with a batch size of 256, while contrastive methods typically need to store the last 65K instances obtained from the last 250 batches [24].

3.2 Multi-crop: Augmenting views with smaller images

As noted in prior works [10, 44], comparing random crops of an image plays a central role by capturing information in terms of relations between parts of a scene or an object. Unfortunately, increasing the number of crops or "views" quadratically increases the memory and compute requirements. We propose a `multi-crop` strategy where we use two standard resolution crops and sample V additional low resolution crops that cover only small parts of the image. Using low resolution images ensures only a small increase in the compute cost. Specifically, we generalize the loss of Eq (1):

$$L(\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \dots, \mathbf{z}_{t_{V+2}}) = \sum_{i \in \{1,2\}} \sum_{v=1}^{V+2} \mathbf{1}_{v \neq i} \ell(\mathbf{z}_{t_v}, \mathbf{q}_{t_i}). \tag{6}$$

Note that we compute codes using only the full resolution crops. Indeed, computing codes for all crops increases the computational time and we observe in practice that it also alters the transfer performance

Table 1: **Semi-supervised learning on ImageNet with a ResNet-50.** We finetune the model with 1% and 10% labels and report top-1 and top-5 accuracies. *: *uses RandAugment [12].*

		1% labels		10% labels	
	Method	Top-1	Top-5	Top-1	Top-5
	Supervised	25.4	48.4	56.4	80.4
Methods using label-propagation	UDA [60]	-	-	68.8*	88.5*
	FixMatch [51]	-	-	**71.5***	89.1*
Methods using self-supervision only	PIRL [44]	30.7	57.2	60.4	83.8
	PCL [37]	-	75.6	-	86.2
	SimCLR [10]	48.3	75.5	65.6	87.8
	SwAV	**53.9**	**78.5**	70.2	**89.9**

of the resulting network. An explanation is that using only partial information (small crops cover only small area of images) degrades the assignment quality. Figure 3 shows that `multi-crop` improves the performance of several self-supervised methods and is a promising augmentation strategy.

4 Main Results

We analyze the features learned by SwAV by transfer learning on multiple datasets. We implement in SwAV the improvements used in SimCLR, *i.e.*, LARS [64], cosine learning rate [40, 44] and the MLP projection head [10]. We provide the full details and hyperparameters for pretraining and transfer learning in the Appendix.

4.1 Evaluating the unsupervised features on ImageNet

We evaluate the features of a ResNet-50 [27] trained with SwAV on ImageNet by two experiments: linear classification on frozen features and semi-supervised learning by finetuning with few labels. When using frozen features (Fig. 2 left), SwAV outperforms the state of the art by $+4.2\%$ top-1 accuracy and is only 1.2% below the performance of a fully supervised model. Note that we train SwAV during 800 epochs with large batches (4096). We refer to Fig. 3 for results with shorter trainings and to Table 3 for experiments with small batches. On semi-supervised learning (Table 1), SwAV outperforms other self-supervised methods and is on par with state-of-the-art semi-supervised models [51], despite the fact that SwAV is not specifically designed for semi-supervised learning.

Variants of ResNet-50. Figure 2 (right) shows the performance of multiple variants of ResNet-50 with different widths [35]. The performance of our model increases with the width of the model, and follows a similar trend to the one obtained with supervised learning. When compared with concurrent work like SimCLR, we see that SwAV reduces the difference with supervised models even further. Indeed, for large architectures, our method shrinks the gap with supervised training to 0.6%.

Table 2: **Transfer learning on downstream tasks.** Comparison between features from ResNet-50 trained on ImageNet with SwAV or supervised learning. We consider two settings. (1) Linear classification on top of frozen features. We report top-1 accuracy on all datasets except VOC07 where we report mAP. (2) Object detection with finetuned features on VOC07+12 `trainval` using Faster R-CNN [50] and on COCO [38] using Mask R-CNN [26] or DETR [6]. We report the most standard detection metrics for these datasets: AP_{50} on VOC07+12 and AP on COCO.

	Linear Classification			Object Detection		
	Places205	VOC07	iNat18	VOC07+12 (Faster R-CNN R50-C4)	COCO (Mask R-CNN R50-FPN)	COCO (DETR)
Supervised	53.2	87.5	46.7	81.3	39.7	40.8
SwAV	**56.7**	**88.9**	**48.6**	**82.6**	**41.6**	**42.1**

Table 3: **Training in small batch setting.** Top-1 accuracy on ImageNet with a linear classifier trained on top of frozen features from a ResNet-50. All methods are trained with a batch size of 256. We also report the number of stored features, the type of cropping used and the number of epochs.

Method	Mom. Encoder	Stored Features	`multi-crop`	epoch	batch	Top-1
SimCLR		0	2×224	200	256	61.9
MoCov2	✓	65,536	2×224	200	256	67.5
MoCov2	✓	65,536	2×224	800	256	71.1
SwAV		3,840	$2\times160+4\times96$	200	256	72.0
SwAV		3,840	$2\times224+6\times96$	200	256	72.7
SwAV		3,840	$2\times224+6\times96$	400	256	**74.3**

4.2 Transferring unsupervised features to downstream tasks

We test the generalization of ResNet-50 features trained with SwAV on ImageNet (without labels) by transferring to several downstream vision tasks. In Table 2, we compare the performance of SwAV features with ImageNet supervised pretraining. First, we report the linear classification performance on the Places205 [67], VOC07 [17], and iNaturalist2018 [54] datasets. Our method outperforms supervised features on all three datasets. Note that SwAV is the first self-supervised method to surpass ImageNet supervised features on these datasets. Second, we report network finetuning on object detection on VOC07+12 using Faster R-CNN [50] with a R50-C4 backbone and on COCO [38] using Mask R-CNN [26] with a R50-FPN backbone and finally using DETR detector [6]. DETR is a recent object detection framework that reaches competitive performance with Faster R-CNN while being conceptually simpler and trainable end-to-end. Interestingly, unlike Faster R-CNN [50], using a pretrained backbone for DETR is crucial to obtain good results compared to training from scratch [6]. In Table 2, we show that SwAV outperforms the supervised pretrained model on both VOC07+12 and COCO datasets. Note that this is line with previous works that also show that self-supervision can outperform supervised pretraining on object detection [19, 24, 44]. We report more detection evaluation metrics and results from other self-supervised methods in the Appendix. Overall, our SwAV ResNet-50 model surpasses supervised ImageNet pretraining on all the considered transfer tasks and datasets. We have released this model so other researchers might also benefit by replacing the ImageNet supervised network with our model.

4.3 Training with small batches

We train SwAV with small batches of 256 images on 4 GPUs and compare with MoCov2 and SimCLR trained in the same setup. In Table 3, we see that SwAV maintains state-of-the-art performance even when trained in the small batch setting. Note that SwAV only stores a queue of 3,840 features. In comparison, to obtain good performance, MoCov2 needs to store 65,536 features while keeping an additional momentum encoder network. When SwAV is trained using $2\times160+4\times96$ crops, SwAV has a running time $1.2\times$ higher than SimCLR with 2×224 crops and is around $1.4\times$ slower than MoCov2 due to the additional back-propagation [11]. Hence, one epoch of MoCov2 or SimCLR is faster in wall clock time than one of SwAV, but these methods need more epochs for good downstream performance. Indeed, as shown in Table 3, SwAV learns much faster and reaches higher performance in $4\times$ fewer epochs: 72% after 200 epochs (102 hours) while MoCov2 needs 800 epochs to achieve 71.1%. Increasing the resolution and the number of epochs, SwAV reaches 74.3% with a small batch size, a small number of stored features and no momentum encoder. Finally, note that SwAV could be combined with a momentum mechanism and a large queue [24]; we leave these explorations to future work.

5 Ablation Study

5.1 Clustering-based self-supervised learning

Improving prior clustering-based approaches. In this section, we re-implement and improve previously published clustering-based models in order to assess if they can compete with recent

	Top-1		Δ
Method	2x224	2x160+4x96	
Supervised	76.5	76.0	−0.5
Contrastive-instance approaches			
SimCLR	68.2	70.6	+2.4
Clustering-based approaches			
SeLa-v2	67.2	71.8	+4.6
DeepCluster-v2	70.2	74.3	+4.1
SwAV	70.1	74.1	+4.0

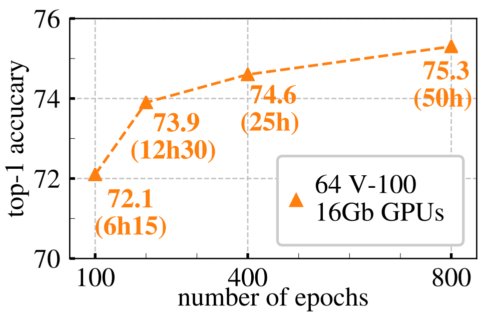


Figure 3: Top-1 accuracy on ImageNet with a linear classifier trained on top of frozen features from a ResNet-50. **(left) Comparison between clustering-based and contrastive instance methods and impact of multi-crop.** Self-supervised methods are trained for 400 epochs and supervised models for 200 epochs. **(right) Performance as a function of epochs.** We compare SwAV models trained with different number of epochs and report their running time based on our implementation.

contrastive methods such as SimCLR. In particular, we consider two clustering-based models: DeepCluster-v2 and SeLa-v2, which are obtained by applying various training improvements introduced in other self-supervised learning papers to DeepCluster [7] and SeLa [2]. Among these improvements are the use of stronger data augmentation [10], MLP projection head [10], cosine learning rate schedule [44], temperature parameter [58], memory bank [58], multi-clustering [2], etc. Full implementation details can be found in the Appendix. Besides, we also improve DeepCluster model by introducing explicit comparisons to k-means centroids, which increase stability and performance. Indeed, a main issue in DeepCluster is that there is no correspondance between two consecutive cluster assignments. Hence, the final classification layer learned for an assignment becomes irrelevant for the following one and thus needs to be re-initialized from scratch at each epoch. This considerably disrupts the convnet training. In DeepCluster-v2, instead of learning a classification layer predicting the cluster assignments, we perform explicit comparison between features and centroids.

Comparing clustering with contrastive instance learning. In Fig. 3 (left), we make a best effort fair comparison between clustering-based and contrastive instance (SimCLR) methods by implementating these methods with the same data augmentation, number of epochs, batch-sizes, etc. In this setting, we observe that SwAV and DeepCluster-v2 outperform SimCLR by 2% without `multi-crop` and by 3.5% with `multi-crop`. This suggests the learning potential of clustering-based methods over instance classification.

Advantage of SwAV compared to DeepCluster-v2. In Fig. 3 (left), we observe that SwAV performs on par with DeepCluster-v2. In addition, we train DeepCluster-v2 in SwAV best setting (800 epochs - 8 crops) and obtain 75.2% top-1 accuracy on ImageNet (versus 75.3% for SwAV). However, unlike SwAV, DeepCluster-v2 is not online which makes it impractical for extremely large datasets (§ 5.4). For billion scale trainings for example, a single pass on the dataset is usually performed [24]. DeepCluster-v2 cannot be trained for only one epoch since it works by performing several passes on the dataset to regularly update centroids and cluster assignments for each image.

As a matter of fact, DeepCluster-v2 can be interpreted as a special case of our proposed swapping mechanism: swapping is done across epochs rather than within a batch. Given a crop of an image DeepCluster-v2 predicts the assignment of another crop, which was obtained at the previous epoch. SwAV swaps assignments directly at the batch level and can thus work online.

5.2 Applying the multi-crop strategy to different methods

In Fig. 3 (left), we report the impact of applying our `multi-crop` strategy on the performance of a selection of other methods. Details of how we apply `multi-crop` to SimCLR loss can be found in the Appendix. We see that the `multi-crop` strategy consistently improves the performance for all the considered methods by a significant margin of $2-4\%$ top-1 accuracy. Interestingly, `multi-crop` seems to benefit more clustering-based methods than contrastive methods. We note that `multi-crop` does not improve the supervised model.

Method	Frozen	Finetuned
Random	15.0	76.5
MoCo	-	77.3*
SimCLR	60.4	77.2
SwAV	**66.5**	**77.8**

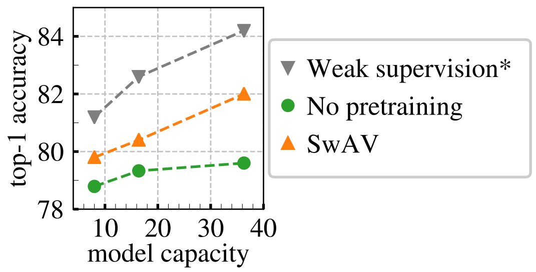


Figure 4: **Pretraining on uncurated data.** Top-1 accuracy on ImageNet for pretrained models on an uncurated set of 1B random Instagram images. **(left)** We compare ResNet-50 pretrained with either SimCLR or SwAV on two downstream tasks: linear classification on frozen features or finetuned features. **(right)** Performance of finetuned models as we increase the capacity of a ResNext following [41]. The capacity is provided in billions of Mult-Add operations.
*: *pretrained on a curated set of* 1*B Instagram images filtered with* 1.5*k hashtags similar to ImageNet classes.*

5.3 Impact of longer training

In Fig. 3 (right), we show the impact of the number of training epochs on performance for SwAV with `multi-crop`. We train separate models for 100, 200, 400 and 800 epochs and report the top-1 accuracy on ImageNet using the linear classification evaluation. We train each ResNet-50 on 64 V100 16GB GPUs and a batch size of 4096. While SwAV benefits from longer training, it already achieves strong performance after 100 epochs, *i.e.*, 72.1% in 6h15.

5.4 Unsupervised pretraining on a large uncurated dataset

We evaluate SwAV on random, uncurated images that have different properties from ImageNet which allows us to test if our online clustering scheme and multi-crop augmentation work out of the box. In particular, we pretrain SwAV on an uncurated dataset of 1 billion random public non-EU images from Instagram. We test if SwAV can serve as a pretraining method for supervised learning. In Fig. 4 (left), we measure the performance of ResNet-50 models when transferring to ImageNet with frozen or finetuned features. We report the results from He *et al*. [24] but note that their setting is different. They use a curated set of Instagram images, filtered by hashtags similar to ImageNet labels [41]. We compare SwAV with a randomly initialized network and with a network pretrained on the same data using SimCLR. We observe that SwAV maintains a similar gain of 6% over SimCLR as when pretrained on ImageNet (Fig. 2), showing that our improvements do not depend on the data distribution. We also see that pretraining with SwAV on random images significantly improves over training from scratch on ImageNet (+1.3%) [8, 24]. This result is in line with Caron *et al*. [8] and He *et al*. [24]. In Fig. 4 (right), we explore the limits of pretraining as we increase the model capacity. We consider the variants of the ResNeXt architecture [61] as in Mahajan *et al*. [41]. We compare SwAV with supervised models trained from scratch on ImageNet. For all models, SwAV outperforms training from scratch by a significant margin showing that it can take advantage of the increased model capacity. For reference, we also include the results from Mahajan *et al*. [41] obtained with a weakly-supervised model pretrained by predicting hashtags filtered to be similar to ImageNet classes. Interestingly, SwAV performance is strong when compared to this topline despite not using any form of supervision or filtering of the data.

6 Discussion

Self-supervised learning is rapidly progressing compared to supervised learning, even surpassing it on transfer learning, even though the current experimental settings are designed for supervised learning. In particular, architectures have been designed for supervised tasks, and it is not clear if the same models would emerge from exploring architectures with no supervision. Several recent works have shown that exploring architectures with search [39] or pruning [9] is possible without supervision, and we plan to evaluate the ability of our method to guide model explorations.

Acknowledgement. We thank Nicolas Carion, Kaiming He, Herve Jegou, Benjamin Lefaudeux, Thomas Lucas, Francisco Massa, Sergey Zagoruyko, and the rest of Thoth and FAIR teams for their help and fruitful discussions. Julien Mairal was funded by the ERC grant number 714381 (SOLARIS project) and by ANR 3IA MIAI@Grenoble Alpes (ANR-19-P3IA-0003).

References

[1] Agrawal, P., Carreira, J., Malik, J.: Learning to see by moving. In: Proceedings of the International Conference on Computer Vision (ICCV) (2015) 3

[2] Asano, Y.M., Rupprecht, C., Vedaldi, A.: Self-labelling via simultaneous clustering and representation learning. International Conference on Learning Representations (ICLR) (2020) 2, 3, 4, 5, 8, 21, 22, 23

[3] Bachman, P., Hjelm, R.D., Buchwalter, W.: Learning representations by maximizing mutual information across views. In: Advances in Neural Information Processing Systems (NeurIPS) (2019) 18

[4] Bautista, M.A., Sanakoyeu, A., Tikhoncheva, E., Ommer, B.: Cliquecnn: Deep unsupervised exemplar learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2016) 3

[5] Bojanowski, P., Joulin, A.: Unsupervised learning by predicting noise. In: Proceedings of the International Conference on Machine Learning (ICML) (2017) 2, 20

[6] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. arXiv preprint arXiv:2005.12872 (2020) 6, 7, 16, 18, 19, 20

[7] Caron, M., Bojanowski, P., Joulin, A., Douze, M.: Deep clustering for unsupervised learning of visual features. In: Proceedings of the European Conference on Computer Vision (ECCV) (2018) 2, 3, 8, 20, 21, 22

[8] Caron, M., Bojanowski, P., Mairal, J., Joulin, A.: Unsupervised pre-training of image features on non-curated data. In: Proceedings of the International Conference on Computer Vision (ICCV) (2019) 3, 9

[9] Caron, M., Morcos, A., Bojanowski, P., Mairal, J., Joulin, A.: Pruning convolutional neural networks with self-supervision. arXiv preprint arXiv:2001.03554 (2020) 9

[10] Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. arXiv preprint arXiv:2002.05709 (2020) 1, 2, 5, 6, 8, 14, 15, 18, 19

[11] Chen, X., Fan, H., Girshick, R., He, K.: Improved baselines with momentum contrastive learning. arXiv preprint arXiv:2003.04297 (2020) 5, 7, 15, 16, 19, 20

[12] Cubuk, E.D., Zoph, B., Shlens, J., Le, Q.V.: Randaugment: Practical data augmentation with no separate search. arXiv preprint arXiv:1909.13719 (2019) 6

[13] Cuturi, M.: Sinkhorn distances: Lightspeed computation of optimal transport. In: Advances in Neural Information Processing Systems (NeurIPS) (2013) 3, 5, 21

[14] Doersch, C., Gupta, A., Efros, A.A.: Unsupervised visual representation learning by context prediction. In: Proceedings of the International Conference on Computer Vision (ICCV) (2015) 3

[15] Donahue, J., Simonyan, K.: Large scale adversarial representation learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2019) 5, 18

[16] Dosovitskiy, A., Fischer, P., Springenberg, J.T., Riedmiller, M., Brox, T.: Discriminative unsupervised feature learning with exemplar convolutional neural networks. IEEE transactions on pattern analysis and machine intelligence **38**(9), 1734–1747 (2016) 1, 2

[17] Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International journal of computer vision **88**(2), 303–338 (2010) 7

[18] Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. Journal of machine learning research (2008) 16

[19] Gidaris, S., Bursuc, A., Komodakis, N., Pérez, P., Cord, M.: Learning representations by predicting bags of visual words. arXiv preprint arXiv:2002.12247 (2020) 3, 7, 19, 20

[20] Gidaris, S., Singh, P., Komodakis, N.: Unsupervised representation learning by predicting image rotations. In: International Conference on Learning Representations (ICLR) (2018) 18

[21] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., He, K.: Accurate, large minibatch sgd: Training imagenet in 1 hour. arXiv preprint arXiv:1706.02677 (2017) 15

[22] Gutmann, M., Hyvärinen, A.: Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (2010) 2

[23] Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2006) 1

[24] He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. arXiv preprint arXiv:1911.05722 (2019) 1, 2, 5, 7, 8, 9, 16, 18, 19, 20

[25] He, K., Girshick, R., Dollár, P.: Rethinking imagenet pre-training. In: Proceedings of the International Conference on Computer Vision (ICCV) (2019) 16

[26] He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Proceedings of the International Conference on Computer Vision (ICCV) (2017) 6, 7, 18, 19

[27] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 6

[28] Hénaff, O.J., Razavi, A., Doersch, C., Eslami, S., Oord, A.v.d.: Data-efficient image recognition with contrastive predictive coding. arXiv preprint arXiv:1905.09272 (2019) 5, 18

[29] Hjelm, R.D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., Bengio, Y.: Learning deep representations by mutual information estimation and maximization. International Conference on Learning Representations (ICLR) (2019) 2

[30] Huang, J., Dong, Q., Gong, S.: Unsupervised deep learning by neighbourhood discovery. In: Proceedings of the International Conference on Machine Learning (ICML) (2019) 3

[31] Jenni, S., Favaro, P.: Self-supervised feature learning by learning to spot artifacts. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2018) 3

[32] Jing, L., Tian, Y.: Self-supervised visual feature learning with deep neural networks: A survey. arXiv preprint arXiv:1902.06162 (2019) 3

[33] Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., Krishnan, D.: Supervised contrastive learning. arXiv preprint arXiv:2004.11362 (2020) 17

[34] Kim, D., Cho, D., Yoo, D., Kweon, I.S.: Learning image representations by completing damaged jigsaw puzzles. In: Winter Conference on Applications of Computer Vision (WACV) (2018) 3

[35] Kolesnikov, A., Zhai, X., Beyer, L.: Revisiting self-supervised visual representation learning. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2019) 6

[36] Larsson, G., Maire, M., Shakhnarovich, G.: Learning representations for automatic colorization. In: Proceedings of the European Conference on Computer Vision (ECCV) (2016) 3

[37] Li, J., Zhou, P., Xiong, C., Socher, R., Hoi, S.C.: Prototypical contrastive learning of unsupervised representations. arXiv preprint arXiv:2005.04966 (2020) 5, 6, 15, 16, 19, 21

[38] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Proceedings of the European Conference on Computer Vision (ECCV) (2014) 6, 7, 16, 19

[39] Liu, C., Dollár, P., He, K., Girshick, R., Yuille, A., Xie, S.: Are labels necessary for neural architecture search? arXiv preprint arXiv:2003.12056 (2020) 9

[40] Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016) 6, 15, 16

[41] Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., van der Maaten, L.: Exploring the limits of weakly supervised pretraining. In: Proceedings of the European Conference on Computer Vision (ECCV) (2018) 9

[42] Mahendran, A., Thewlis, J., Vedaldi, A.: Cross pixel optical flow similarity for self-supervised learning. arXiv preprint arXiv:1807.05636 (2018) 3

[43] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al.: Mixed precision training. arXiv preprint arXiv:1710.03740 (2017) 15

[44] Misra, I., van der Maaten, L.: Self-supervised learning of pretext-invariant representations. arXiv preprint arXiv:1912.01991 (2019) 1, 2, 3, 5, 6, 7, 8, 15, 16, 19, 20

[45] Misra, I., Zitnick, C.L., Hebert, M.: Shuffle and learn: unsupervised learning using temporal order verification. In: Proceedings of the European Conference on Computer Vision (ECCV) (2016) 3

[46] Noroozi, M., Favaro, P.: Unsupervised learning of visual representations by solving jigsaw puzzles. In: Proceedings of the European Conference on Computer Vision (ECCV) (2016) 3, 5

[47] Oord, A.v.d., Li, Y., Vinyals, O.: Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748 (2018) 2

[48] Pathak, D., Girshick, R., Dollár, P., Darrell, T., Hariharan, B.: Learning features by watching objects move. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 3

[49] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A.: Context encoders: Feature learning by inpainting. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 3

[50] Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems (NeurIPS) (2015) 6, 7, 16, 18, 19, 20

[51] Sohn, K., Berthelot, D., Li, C.L., Zhang, Z., Carlini, N., Cubuk, E.D., Kurakin, A., Zhang, H., Raffel, C.: Fixmatch: Simplifying semi-supervised learning with consistency and confidence. arXiv preprint arXiv:2001.07685 (2020) 6

[52] Tian, Y., Krishnan, D., Isola, P.: Contrastive multiview coding. arXiv preprint arXiv:1906.05849 (2019) 18

[53] Touvron, H., Vedaldi, A., Douze, M., Jégou, H.: Fixing the train-test resolution discrepancy. In: Advances in Neural Information Processing Systems (NeurIPS) (2019) 2

[54] Van Horn, G., Mac Aodha, O., Song, Y., Cui, Y., Sun, C., Shepard, A., Adam, H., Perona, P., Belongie, S.: The inaturalist species classification and detection dataset. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2018) 7

[55] Wang, X., Gupta, A.: Unsupervised learning of visual representations using videos. In: Proceedings of the International Conference on Computer Vision (ICCV) (2015) 3

[56] Wang, X., He, K., Gupta, A.: Transitive invariance for self-supervised visual representation learning. In: Proceedings of the International Conference on Computer Vision (ICCV) (2017) 3

[57] Wu, Y., Kirillov, A., Massa, F., Lo, W.Y., Girshick, R.: Detectron2. `https://github.com/facebookresearch/detectron2` (2019) 16

[58] Wu, Z., Xiong, Y., Yu, S.X., Lin, D.: Unsupervised feature learning via non-parametric instance discrimination. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2018) 2, 3, 4, 5, 8, 19, 21, 22

[59] Xie, J., Girshick, R., Farhadi, A.: Unsupervised deep embedding for clustering analysis. In: Proceedings of the International Conference on Machine Learning (ICML) (2016) 3

[60] Xie, Q., Dai, Z.D., Hovy, E., Luong, M.T., Le, Q.V.: Unsupervised data augmentation for consistency training. arXiv preprint arXiv:1904.12848 (2020) 6

[61] Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 9

[62] Yan, X., Misra, Ishan, I., Gupta, A., Ghadiyaram, D., Mahajan, D.: ClusterFit: Improving generalization of visual representations. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2020) 3

[63] Yang, J., Parikh, D., Batra, D.: Joint unsupervised learning of deep representations and image clusters. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 3

[64] You, Y., Gitman, I., Ginsburg, B.: Large batch training of convolutional networks. arXiv preprint arXiv:1708.03888 (2017) 6, 15, 16

[65] Zhang, R., Isola, P., Efros, A.A.: Colorful image colorization. In: Proceedings of the European Conference on Computer Vision (ECCV) (2016) 5

[66] Zhang, R., Isola, P., Efros, A.A.: Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 3

[67] Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., Oliva, A.: Learning deep features for scene recognition using places database. In: Advances in Neural Information Processing Systems (NeurIPS) (2014) 7

[68] Zhuang, C., Zhai, A.L., Yamins, D.: Local aggregation for unsupervised learning of visual embeddings. In: Proceedings of the International Conference on Computer Vision (ICCV) (2019) 3, 5, 19, 21

A Implementation Details

In this section, we provide the details and hyperparameters for SwAV pretraining and transfer learning. Our code is publicly available at https://github.com/facebookresearch/swav.

A.1 Implementation details of SwAV training

First, we provide a pseudo-code for SwAV training loop using two crops in Pytorch style:

```
# C: prototypes (DxK)
# model:  convnet + projection head
# temp:  temperature

for x in loader: # load a batch x with B samples
   x_t = t(x) # t is a random augmentation
   x_s = s(x) # s is a another random augmentation

   z = model(cat(x_t, x_s)) # embeddings:  2BxD

   scores = mm(z, C) # prototype scores:  2BxK
   scores_t = scores[:B]
   scores_s = scores[B:]

   # compute assignments
   with torch.no_grad():
      q_t = sinkhorn(scores_t)
      q_s = sinkhorn(scores_s)

   # convert scores to probabilities
   p_t = Softmax(scores_t / temp)
   p_s = Softmax(scores_s / temp)

   # swap prediction problem
   loss = - 0.5 * mean(q_t * log(p_s) + q_s * log(p_t))

   # SGD update:  network and prototypes
   loss.backward()
   update(model.params)
   update(C)

   # normalize prototypes
   with torch.no_grad():
      C = normalize(C, dim=0, p=2)

# Sinkhorn-Knopp
def sinkhorn(scores, eps=0.05, niters=3):
   Q = exp(scores / eps).T
   Q /= sum(Q)
   K, B = Q.shape
   u, r, c = zeros(K), ones(K) / K, ones(B) / B
   for _ in range(niters):
      u = sum(Q, dim=1)
      Q *= (r / u).unsqueeze(1)
      Q *= (c / sum(Q, dim=0)).unsqueeze(0)
   return (Q / sum(Q, dim=0, keepdim=True)).T
```

Most of our training hyperparameters are directly taken from SimCLR work [10]. We train SwAV with stochastic gradient descent using large batches of 4096 different instances. We distribute the batches over 64 V100 16Gb GPUs, resulting in each GPU treating 64 instances. The temperature parameter τ is set to 0.1 and the Sinkhorn regularization parameter ε is set to 0.05 for all runs. We use a weight decay of 10^{-6}, LARS optimizer [64] and a learning rate of 4.8 which is linearly ramped up during the first 10 epochs. After warmup, we use the cosine learning rate decay [40, 44] with a final value of 0.0048. To help the very beginning of the optimization, we freeze the prototypes during the first epoch of training. We synchronize batch-normalization layers across GPUs using the optimized implementation with kernels through CUDA/C-v2 extension from `apex`[-2]. We also use `apex` library for training with mixed precision [43]. Overall, thanks to these training optimizations (mixed precision, kernel batch-normalization and use of large batches [21]), 100 epochs of training for our best SwAV model take approximately 6 hours (see Table 4). Similarly to previous works [10, 11, 37], we use a projection head on top of the convnet features that consists in a 2-layer multi-layer perceptron (MLP) that projects the convnet output to a 128-D space.

Note that SwAV is more suitable for a multi-node distributed implementation compared to contrastive approaches SimCLR or MoCo. The latter methods require sharing the feature matrix across all GPUs at every batch which might become a bottleneck when distributing across many GPUs. On the contrary, SwAV requires sharing only matrix normalization statistics (sum of rows and columns) during the Sinkhorn algorithm.

A.2 Data augmentation used in SwAV

We obtain two different views from an image by performing crops of random sizes and aspect ratios. Specifically we use the `RandomResizedCrop` method from `torchvision.transforms` module of PyTorch with the following scaling parameters: `s=(0.14, 1)`. Note that we sample crops in a narrower range of scale compared to the default `RandomResizedCrop` parameters. Then, we resize both full resolution views to 224×224 pixels, unless specified otherwise (we use 160×160 resolutions in some of our experiments). Besides, we obtain V additional views by cropping small parts in the image. To do so, we use the following `RandomResizedCrop` parameters: `s=(0.05, 0.14)`. We resize the resulting crops to 96×96 resolution. Note that we always deal with resolutions that are divisible by 32 to avoid roundings in the ResNet-50 pooling layers. Finally, we apply random horizontal flips, color distortion and Gaussian blur to each resulting crop, exactly following the SimCLR implementation [10]. An illustration of our `multi-crop` augmentation strategy can be viewed in Fig. 5.

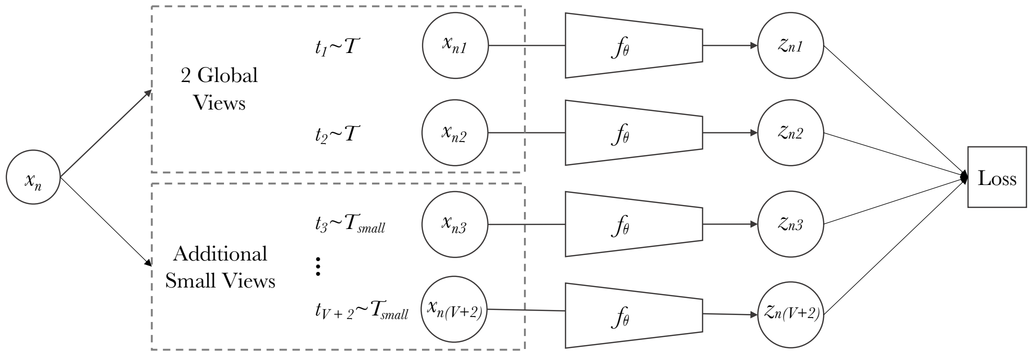


Figure 5: **Multi-crop**: the image x_n is transformed into $V + 2$ views: two global views and V small resolution zoomed views.

A.3 Implementation details of linear classification on ImageNet with ResNet-50

We obtain 75.3 top-1 accuracy on ImageNet by training a linear classifier on top of frozen final representations (2048-D) of a ResNet-50 trained with SwAV. This linear layer is trained during 100 epochs, with a learning rate of 0.3 and a weight decay of 10^{-6}. We use cosine learning rate decay and a batch size of 256. We use standard data augmentations, i.e., cropping of random sizes and aspect ratios (default parameters of `RandomResizedCrop`) and random horizontal flips.

[-2] `github.com/NVIDIA/apex`

A.4 Implementation details of semi-supervised learning (finetuning with 1% or 10% labels)

We finetune with either 1% or 10% of ImageNet labeled images a ResNet-50 pretrained with SwAV. We use the 1% and 10% splits specified in the official code release of SimCLR. We mostly follow hyperparameters from PCL [37]: we train during 20 epochs with a batch size of 256, we use distinct learning rates for the convnet weights and the final linear layer, and we decay the learning rates by a factor 0.2 at epochs 12 and 16. We do not apply any weight decay during finetuning. For 1% finetuning, we use a learning rate of 0.02 for the trunk and 5 for the final layer. For 10% finetuning, we use a learning rate of 0.01 for the trunk and 0.2 for the final layer.

A.5 Implementation details of transfer learning on downstream tasks

Linear classifiers. We mostly follow PIRL [44] for training linear models on top of representations given by a ResNet-50 pretrained with SwAV. On VOC07, all images are resized to 256 pixels along the shorter side, before taking a 224×224 center crop. Then, we train a linear SVM with LIBLINEAR [18] on top of corresponding global average pooled final representations (2048-D). For linear evaluation on other datasets (Places205 and iNat18), we train linear models with stochastic gradient descent using a batch size of 256, a learning rate of 0.01 reduced by a factor of 10 three times (equally spaced intervals), weight decay of 0.0001 and momentum of 0.9. On Places205, we train the linear models for 28 epochs and on iNat18 for 84 epochs. We report the top-1 accuracy computed using the 224×224 center crop on the validation set.

Object Detection on VOC07+12. We use a Faster R-CNN [50] model as implemented in Detectron2 [57] and follow the finetuning protocol from He *et al.* [24] making the following changes to the hyperparameters – our initial learning rate is 0.1 which is warmed with a slope (`WARMUP_FACTOR` flag in Detectron2) of 0.333 for 1000 iterations. Other training hyperparamters are kept exactly the same as in He *et al.* [24], *i.e.*, batchsize of 16 across 8 GPUs, training for 24K iterations on VOC07+12 `trainval` with the learning rate reduced by a factor of 10 after 18K and 22K iterations, using SyncBatchNorm to finetune BatchNorm parameters, and adding an extra BatchNorm layer after the `res5` layer (`Res5ROIHeadsExtraNorm` head in Detectron2). We report results on VOC07 test set averaged over 5 independant runs.

Object Detection on COCO. We test the generalization of our ResNet-50 features trained on ImageNet with SwAV by transferring them to object detection on COCO dataset [38] with DETR framework [6]. DETR is a recent object detection framework that relies on a transformer encoder-decoder architecture. It reaches competitive performance with Faster R-CNN while being conceptually simpler and trainable end-to-end. Interestingly, unlike other frameworks [25], current results with DETR have shown that using a pretrained backbone is crucial to obtain good results compared to training from scratch. Therefore, we investigate if we can boost DETR performance by using features pretrained on ImageNet with SwAV instead of standard supervised features. We also evaluate features from MoCov2 [11] pretraining. We train DETR during 300 epochs with AdamW, we use a learning rate of 10^{-4} for the transformer and apply a weight decay of 10^{-4}. We select for each method the best learning rate for the backbone among the following three values: 10^{-5}, 5×10^{-5} and 10^{-4}. We decay the learning rates by a factor 0.1 after 200 epochs.

A.6 Implementation details of training with small batches of 256 images

We start using a queue composed of the feature representations from previous batches after 15 epochs of training. Indeed, we find that using the queue before 15 epochs disturbs the convergence of the model since the network is changing a lot from an iteration to another during the first epochs. We simulate large batches of size 4096 by storing the last 15 batches, that is $3,840$ vectors of dimension 128. We use a weight decay of 10^{-6}, LARS optimizer [64] and a learning rate of 0.6. We use the cosine learning rate decay [40] with a final value of 0.0006.

A.7 Implementation details of ablation studies

In our ablation studies (results in Table 5 of the main paper for example), we choose to follow closely the data augmentation used in concurrent work SimCLR. This allows a fair comparison and importantly, isolates the effect of our contributions. In practice, it means that we use the default parameters of the random crop method (`RandomResizedCrop`), `s=(0.08, 1)` instead of `s=(0.14, 1)`, when sampling the two large resolution views.

A.8 SimCLR loss with multi-crop augmentation

When implementing SimCLR with `multi-crop` augmentation, we have to deal with several positive pairs formed by an anchor features and the different crops coming from the same instance. We denote by B the total number of unique dataset instances in the batch and by M the number of crops per instance. For example, in the case of 2x160+4x96 crops, we have $M = 6$ crops per instance. We call $N = B \times M$ the effective total number of crops in the batch. Overall, we minimize the following loss

$$\mathcal{L} = -\frac{1}{N}\frac{1}{M-1}\sum_{i=1}^{N}\sum_{v^+ \in \{v_i^+\}} \log \frac{\exp z_i^T v^+ / \tau}{\exp z_i^T v^+ / \tau + \sum_{v^- \in \{v_i^-\}} \exp z_i^T v^- / \tau}. \tag{7}$$

For a feature representation z_i, the corresponding set of positive examples $\{v_i^+\}$ is formed by the representations of the other crops from the same instance. The set of negatives $\{v_i^-\}$ is formed by the representations of all crops in the same batch except ones coming from the same instance as x_i. Note that this extension of SimCLR loss with several pairs of positive is similar to the one used in Khosla *et al.* [33].

B Additional Results

B.1 Running times

In Table 4, we report compute and GPU memory requirements based on our implementation for different settings. As described in § A.1, we train each method on 64 V100 16GB GPUs, with a batch size of 4096, using mixed precision and `apex` optimized version of synchronized batch-normalization layers. We report results with ResNet-50 for all methods. In Fig. 6, we report SwAV performance for different training lengths measured in hours based on our implementation. We observe that after only 6 hours of training, SwAV outperforms SimCLR trained for 1000 epochs (40 hours based on our implementation) by a large margin. If we train SwAV for longer, we see that the performance gap between the two methods increases even more.

Table 4: **Computational cost.** We report time and GPU memory requirements based on our implementation for different models trained during 100 epochs.

Method	`multi-crop`	time / 100 epochs	peak memory / GPU
SimCLR	2×224	4h00	8.6G
SwAV	2×224	4h09	8.6G
SwAV	$2 \times 160 + 4 \times 96$	4h50	8.5G
SwAV	$2 \times 224 + 6 \times 96$	6h15	12.8G

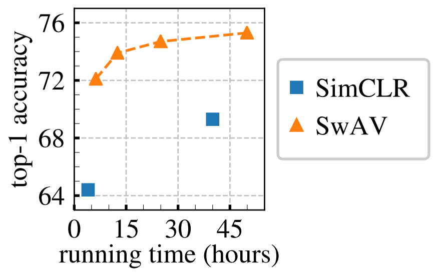


Figure 6: **Influence of longer training.** Top-1 ImageNet accuracy for linear models trained on frozen features. We report SwAV performance for different training lengths measured in hours based on our implementation. We train each ResNet-50 models on 64 V100 16GB GPUs with a batch size of 4096 (see § A.1 for implementation details).

B.2 Larger architectures

In Table 5, we show results when training SwAV on large architectures. We observe that SwAV benefits from training on large architectures and plan to explore in this direction to furthermore boost self-supervised methods.

Implementation details for SwAV R50-w2. The model is trained for 400 epochs on 128 GPUS (batch size 4096). We train the model with 2x224+4x96 (total of 6 crops). All other hyperparameters are the same as the ones described in appendix A.1.

Table 5: **Large architectures.** Top-1 accuracy for linear models trained on frozen features from different self-supervised methods on large architectures.

Method	Arch.	Param.	Top1
Supervised	EffNet-B7	66	84.4
Rotation [20]	RevNet50-4w	86	55.4
BigBiGAN [15]	RevNet50-4w	86	61.3
AMDIM [3]	Custom-RN	626	68.1
CMC [52]	R50-w2	188	68.4
MoCo [24]	R50-w4	375	68.6
CPC v2 [28]	R161	305	71.5
SimCLR [10]	R50-w4	375	76.8
SwAV	R50-w2	188	**77.3**
SwAV	R50-w4	375	**77.9**
SwAV	R50-w5	586	**78.5**

Implementation details for SwAV R50-w4. The model is trained for 400 epochs on 64 GPUS (batch size 2560) with a queue of 2560 samples starting from the beginning of training. We train the model with 2x224+4x96 (total of 6 crops). All other hyperparameters are the same as the ones described in appendix A.1.

Implementation details for SwAV R50-w5. The model is trained for 400 epochs on 128 GPUS (batch size 1536) with a queue of 1536 samples starting from the beginning of training. We train the model with 2x224+4x96 (total of 6 crops). All other hyperparameters are the same as the ones described in appendix A.1.

B.3 Transferring unsupervised features to downstream tasks

In Table 6, we expand results from the main paper by providing numbers from previously and concurrently published self-supervised methods. In the left panel of Table 6, we show performance after training a linear classifier on top of frozen representations on different datasets while on the right panel we evaluate the features by finetuning a ResNet-50 on object detection with Faster R-CNN [50] and DETR [6]. Overall, we observe on Table 6 that SwAV is the first self-supervised method to outperform ImageNet supervised backbone on all the considered transfer tasks and datasets. Other self-supervised learners are capable of surpassing the supervised counterpart but only for one type of transfer (object detection with finetuning for MoCo/PIRL for example). We will release this model so other researchers might also benefit by replacing the ImageNet supervised network with our model.

B.4 More detection metrics for object detection

In Table 7, Table 8 and Table 9, we evaluate the features by finetuning a ResNet-50 with different detectors: Mask R-CNN [26], Faster R-CNN [50] and DETR [6]. We report more detection metrics compared to the table in the main paper Table 6. We observe in Table 7, Table 8 and in Table 9 that SwAV outperforms the ImageNet supervised pretrained model on all the detection evaluation metrics.

Note that MoCov2 backbone performs particularly well on the object detection benchmark, and even outperform SwAV features for some detection metrics. However, as shown in Table 6, this backbone is not competitive with the supervised features when evaluating on classification tasks without finetuning.

B.5 Low-Shot learning on ImageNet for SwAV pretrained on Instagram data

We now test whether SwAV pretrained on Instagram data can serve as a pretraining method for low-shot learning on ImageNet. We report in Table 9 results when finetuning Instagram SwAV features with only few labels per ImageNet category. We observe that using pretrained features from Instagram improves considerably the performance compared to training from scratch.

Table 6: **Transfer learning on downstream tasks.** Comparison between features from ResNet-50 trained on ImageNet with SwAV or supervised learning. We also report numbers from other self-supervised methods († for numbers from other methods run by us). We consider two settings. (1) Linear classification on top of frozen features. We report top-1 accuracy on all datasets except VOC07 where we report mAP. (2) Object detection with finetuned features on VOC07+12 `trainval` using Faster R-CNN [50] and on COCO [38] using Mask R-CNN [26] or DETR [6]. We report the most standard detection metrics for these datasets: AP_{50} on VOC07+12 and AP on COCO.

	Linear Classification			Object Detection		
	Places205	VOC07	iNat18	VOC07+12 (Faster R-CNN R50-C4)	COCO (Mask R-CNN R50-FPN)	COCO (DETR)
Supervised	53.2	87.5	46.7	81.3	39.7	40.8
RotNet [19]	45.0	64.6	-	-	-	
NPID++ [44]	46.4	76.6	32.4	79.1	-	
MoCo [24]	46.9†	79.8†	31.5†	81.5	-	
PIRL [44]	49.8	81.1	34.1	80.7	-	
PCL [37]	49.8	84.0	-	-	-	
BoWNet [19]	51.1	79.3	-	81.3	-	
SimCLR [10]	53.3†	86.4†	36.2†	-	-	
MoCov2 [24]	52.9†	87.1†	38.9†	82.5	39.8	42.0†
SwAV	**56.7**	**88.9**	**48.6**	**82.6**	**41.6**	**42.1**

Table 7: **Object detection and instance segmentation finetuned on COCO.** Following the setup of [24], we use Mask R-CNN detector with ResNet50-FPN trained on `train2017` with default $1\times$ schedule and evaluated on `val2017`. The metrics include bounding box AP (AP^b) and mask AP (AP^m).

Method	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
Supervised	39.7	59.5	43.3	35.9	56.6	38.6
MoCo-v2 [11]	39.8	59.8	43.6	36.1	56.9	38.7
SwAV	**41.6**	**62.3**	**45.5**	**37.8**	**59.6**	**40.5**

B.6 Image classification with KNN classifiers on ImageNet

Following previous work protocols [58, 68], we evaluate the quality of our unsupervised features with K-nearest neighbor (KNN) classifiers on ImageNet. We get features from the computed network outputs for center crops of training and test images. We report results with 20 and 200 NN in Table 11. We outperform the current state-of-the-art of this evaluation. Interestingly we also observe that using fewer NN actually boosts the performance of our model.

C Ablation Studies on Clustering

C.1 Number of prototypes

In Table 12, we evaluate the influence of the number of prototypes used in SwAV. We train ResNet-50 with SwAV for 400 epochs with $2 \times 160 + 4 \times 96$ crops (ablation study setting) and evaluate the performance by training a linear classifier on top of frozen final representations. We observe in Table 12 that varying the number of prototypes by an order of magnitude (3k-100k) does not affect much the performance (at most 0.3 on ImageNet). This suggests that the number of prototypes has little influence as long as there are "enough". Throughout the paper, we train SwAV with 3000 prototypes. We find that using more prototypes increases the computational time both in the Sinkhorn algorithm and during back-propagation for an overall negligible gain in performance.

Table 8: **More detection metrics for object detection on VOC07+12 with finetuned features using Faster R-CNN [50].**

Method	AP^{all}	AP^{50}	AP^{75}
Supervised	53.5	81.3	58.8
Random	28.1	52.5	26.2
NPID++ [44]	52.3	79.1	56.9
PIRL [44]	54.0	80.7	59.7
BoWNet [19]	55.8	81.3	61.1
MoCov1 [24]	55.9	81.5	62.6
MoCov2 [11]	**57.4**	82.5	**64.0**
SwAV	56.1	**82.6**	62.7

Table 9: **More detection metrics for object detection on COCO with finetuned features using DETR [6].**

Method	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
ImageNet labels	40.8	61.2	42.9	20.1	44.5	60.3
MoCo-v2	42.0	62.7	44.4	**20.8**	45.6	**60.9**
SwAV	**42.1**	**63.1**	**44.5**	19.7	**46.3**	**60.9**

C.2 Learning the prototypes

We investigate the impact of learning the prototypes compared to using fixed random prototypes. Assigning features to fixed random targets has been explored in NAT [5]. However, unlike SwAV, NAT uses a target per instance in the dataset, the assignment is hard and performed with Hungarian algorithm. In Table 13 (left), we observe that learning prototypes improves SwAV from 73.1 to 73.9 which shows the effect of adapting the prototypes to the dataset distribution.

Overall, these results suggest that our framework learns from a different signal from "offline" approaches that attribute a pseudo-label to each instance while considering the full dataset and then predict these labels (like DeepCluster [7] for example). Indeed, the prototypes in SwAV are not strongly encouraged to be categorical and random fixed prototypes work almost as well. Rather, they help contrasting different image views without relying on pairwise comparison with many negatives samples. This might explain why the number of prototypes does not impact the performance significantly.

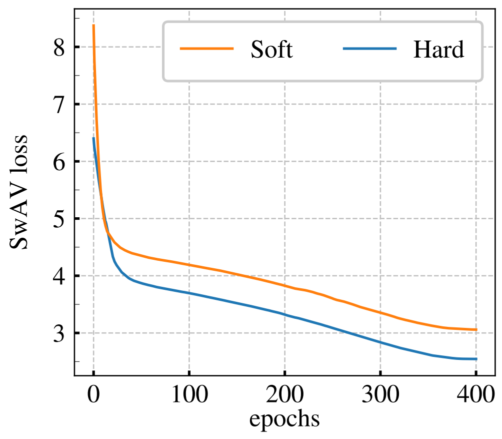


Figure 7: **Hard versus soft assignments.** We report the training loss for SwAV models trained with either soft or hard assignments. The models are trained during 400 epochs with $2 \times 160 + 4 \times 96$ crops.

Table 10: **Low-shot learning on ImageNet.** Top-1 and top-5 accuracies when training with 13 or 128 examples per category.

# examples per class	13		128	
	top1	top5	top1	top5
No pretraining	25.4	48.4	56.4	80.4
SwAV IG-1B	**38.2**	**67.1**	**64.7**	**87.2**

Table 11: **KNN classifiers on ImageNet.** We report top-1 accuracy with 20 and 200 nearest neighbors.

Method	20-NN	200-NN
NPID [58]	-	46.5
LA [68]	-	49.4
PCL [37]	54.5	-
SwAV	**65.7**	**62.7**

C.3 Hard versus soft assignments

In Table 13 (right), we evaluate the impact of using hard assignment instead of the default soft assignment in SwAV. We train the models during 400 epochs with $2 \times 160 + 4 \times 96$ crops (ablation study setting) and evaluate the performance by training a linear classifier on top of frozen final representations. We also report the training losses in Fig. 7. We observe that using the hard assignments performs worse than using the soft assignments. An explanation is that the rounding needed to obtain discrete codes is a more aggressive optimization step than gradient updates. While it makes the model converge rapidly (see Fig. 7), it leads to a worse solution.

C.4 Impact of the number of iterations in Sinkhorn algorithm

In Table 14, we investigate the impact of the number of normalization steps performed during Sinkhorn-Knopp algorithm [13] on the performance of SwAV. We observe that using only 3 iterations is enough for the model to converge. When performing less iterations, the loss fails to converge. We observe that using more iterations slightly alters the transfer performance of the model. We conjecture that it is for the same reason that rounding codes to discrete values deteriorate the quality of our model by converging too rapidly.

D Details on Clustering-Based methods: DeepCluster-v2 and SeLa-v2

In this section, we provide details on our improved implementation of clustering-based approaches DeepCluster-v2 and SeLa-v2 compared to their corresponding original publications [2, 7]. These two methods follow the same pipeline: they alternate between pseudo-labels generation ("assignment phase") and training the network with a classification loss supervised by these pseudo-labels ("training phase").

D.1 Training phase

During the training phase, both methods minimize the multinomial logistic loss of the pseudo-labels $\mathbf{q}$ classification problem:

$$\ell(\mathbf{z}, \mathbf{c}, \mathbf{q}) = -\sum_k \mathbf{q}^{(k)} \log \mathbf{p}^{(k)}, \quad \text{where} \quad \mathbf{p}^{(k)} = \frac{\exp\left(\frac{1}{\tau}\mathbf{z}^\top \mathbf{c}_k\right)}{\sum_{k'} \exp\left(\frac{1}{\tau}\mathbf{z}^\top \mathbf{c}_{k'}\right)}. \tag{8}$$

The pseudo-labels are kept fixed during training and updated for the entire dataset once per epoch during the assignment phase.

Training phase in DeepCluster-v2. In the original DeepCluster work, both the classification head $\mathbf{c}$ and the convnet weights are trained to classify the images into their corresponding pseudo-label

Table 12: **Impact of number of prototypes.** Top-1 ImageNet accuracy for linear models trained on frozen features.

Number of prototypes	300	1000	3000	10000	30000	100000
Top-1	72.8	73.6	73.9	74.1	73.8	73.8

Table 13: **Ablation studies on clustering.** Top-1 ImageNet accuracy for linear models trained on frozen features. **(left)** Impact of learning the prototypes. **(right)** Hard versus soft assignments.

Prototypes	Learned	Fixed
Top-1	73.9	73.1

Assignment	Soft	Hard
Top-1	73.9	73.3

between two assignments. Intuitively, this classification head is optimized to represent prototypes for the different pseudo-classes. However, since there is no mapping between two consecutive assignments: the classification head learned during an assignment becomes irrelevant for the following one. Thus, this classification head needs to be re-set at each new assignment which considerably disrupts the convnet training. For this reason, we propose to simply use for classification head $\mathbf{c}$ the centroids given by k-means clustering (Eq. 11). Overall, during training, DeepCluster-v2 optimizes the following problem with mini-batch SGD:

$$\min_{\mathbf{z}} \ell(\mathbf{z}, \mathbf{c}, \mathbf{q}). \tag{9}$$

Training phase in SeLa-v2. In SeLa work, the prototypes $\mathbf{c}$ are learned with stochastic gradient descend during the training phase. Overall, during training, SeLa-v2 optimizes the following problem:

$$\min_{\mathbf{z}, \mathbf{c}} \ell(\mathbf{z}, \mathbf{c}, \mathbf{q}). \tag{10}$$

D.2 Assignment phase

The purpose of the assignment phase is to provide assignments $\mathbf{q}$ for each instance of the dataset. For both methods, this implies having access to feature representations $\mathbf{z}$ for the entire dataset. Both original works [2, 7] perform regularly a pass forward on the whole dataset to get these features. Using the original implementation, if assignments are updated at each epoch, then the assignment phase represents one third of the total training time. Therefore, in order to speed up training, we choose to use the features computed during the previous epoch instead of dedicating pass forwards to the assignments. This is similar to the memory bank introduced by Wu *et al.* [58], without momentum.

Assignment phase in DeepCluster-v2. DeepCluster-v2 uses spherical k-means to get pseudo-labels. In particular, pseudo-labels $\mathbf{q}$ are obtained by minimizing the following problem:

$$\min_{\mathbf{C} \in \mathbb{R}^{d \times K}} \frac{1}{N} \sum_{n=1}^{N} \min_{\mathbf{q}} -\mathbf{z}_n^\top \mathbf{C} \mathbf{q}, \tag{11}$$

where $\mathbf{z}_n$ and the columns of $\mathbf{C}$ are normalized. The original work DeepCluster uses tricks such as cluster re-assignments and balanced batch sampling to avoid trivial solutions but we found these unnecessary, and did not observe collapsing during our trainings. As noted by Asano *et al.*, this is due to the fact that assignment and training are well separated phases.

Table 14: **Impact of the number of iterations in Sinkhorn algorithm.** Top-1 ImageNet accuracy for linear models trained on frozen features.

Sinkhorn iterations	1	3	10	30
Top-1	*fail*	73.9	73.8	73.7

Assignment phase in SeLa-v2. Unlike DeepCluster, SeLa uses the same loss during training and assignment phases. In particular, we use Sinkhorn-Knopp algorithm to optimize the following assignment problem (see details and derivations in the original SeLa paper [2]):

$$\min_{\mathbf{q}} \ell(\mathbf{z}, \mathbf{c}, \mathbf{q}). \tag{12}$$

Implementation details We use the same hyperparameters as SwAV to train SeLa-v2 and DeepCluster-v2: these are described in § A. Asano *et al*. [2] have shown that multi-clustering boosts performance of clustering-based approaches, and so we use 3 sets of 3000 prototypes $\mathbf{c}$ when training SeLa-v2 and DeepCluster-v2. Note that unlike online methods (like SwAV, SimCLR and MoCo), the clustering approaches SeLa-v2 and DeepCluster-v2 can be implemented with only a single crop per image per batch. The major limitation of SeLa-v2 and DeepCluster-v2 is that these methods are not online and therefore scaling them to very large scale dataset is not posible without major adjustments.