Bootstrap Your Own Latent: A new approach to self-supervised learning

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

Jean-Bastien Grill[*,1] **Florian Strub**[*,1] **Florent Altché**[*,1] **Corentin Tallec**[*,1] **Pierre H. Richemond**[*,1,2]
Elena Buchatskaya[1] **Carl Doersch**[1] **Bernardo Avila Pires**[1] **Zhaohan Daniel Guo**[1]
Mohammad Gheshlaghi Azar[1] **Bilal Piot**[1] **Koray Kavukcuoglu**[1] **Rémi Munos**[1] **Michal Valko**[1]

[1]DeepMind [2]Imperial College

[jbgrill,fstrub,altche,corentint,richemond]@google.com

Abstract

We introduce **B**ootstrap **Y**our **O**wn **L**atent (BYOL), a new approach to self-supervised image representation learning. BYOL relies on two neural networks, referred to as *online* and *target* networks, that interact and learn from each other. From an augmented view of an image, we train the online network to predict the target network representation of the same image under a different augmented view. At the same time, we update the target network with a slow-moving average of the online network. While state-of-the art methods intrinsically rely on negative pairs, BYOL achieves a new state of the art *without them*. BYOL reaches 74.3% top-1 classification accuracy on ImageNet using the standard linear evaluation protocol with a ResNet-50 architecture and 79.6% with a larger ResNet. We show that BYOL performs on par or better than the current state of the art on both transfer and semi-supervised benchmarks.

1 Introduction

Learning good image representations is a key challenge in computer vision [1, 2, 3] as it allows for efficient training on downstream tasks [4, 5, 6, 7]. Many different training approaches have been proposed to learn such representations, usually relying on visual pretext tasks. Among them, state-of-the-art contrastive methods [8, 9, 10, 11, 12] are trained by reducing the distance between representations of different augmented views of the same image ('positive pairs'), and increasing the distance between representations of augmented views from different images ('negative pairs'). These methods require careful treatment of negative pairs [13] by either relying on large batch sizes [8, 12], memory banks [9] or customized mining strategies [14, 15] to retrieve the negative pairs. In addition, their performance critically depends on the choice of image augmentations [8, 12].

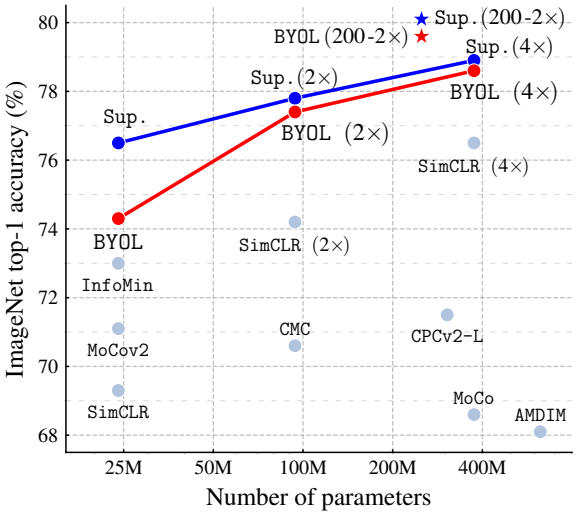


Figure 1: Performance of BYOL on ImageNet (linear evaluation) using ResNet-50 and our best architecture ResNet-200 ($2\times$), compared to other unsupervised and supervised (Sup.) baselines [8].

In this paper, we introduce **B**ootstrap **Y**our **O**wn **L**atent (BYOL), a new algorithm for self-supervised learning of image representations. BYOL achieves higher performance than state-of-the-art contrastive methods

[*]Equal contribution; the order of first authors was randomly selected.

without using negative pairs. It iteratively bootstraps[2] the outputs of a network to serve as targets for an enhanced representation. Moreover, BYOL is more robust to the choice of image augmentations than contrastive methods; we suspect that not relying on negative pairs is one of the leading reasons for its improved robustness. While previous methods based on bootstrapping have used pseudo-labels or cluster indices [16, 17] as targets, we propose to *directly bootstrap the representations*. In particular, BYOL uses two neural networks, referred to as online and target networks, that interact and learn from each other. Starting from an augmented view of an image, BYOL trains its online network to predict the target network's representation of another augmented view of the same image. Even if this problem admits trivial solutions, e.g., outputting zero for all images, we empirically show that using a slow-moving average of the online network as the target network suffices to avoid such collapse to uninformative solutions.

We evaluate the representation learned by BYOL on ImageNet [18] and other vision benchmarks using ResNet architectures [19]. Under the linear evaluation protocol on ImageNet, consisting in training a linear classifier on top of the frozen representation, BYOL reaches 74.3% top-1 accuracy with a standard ResNet-50 and 79.6% top-1 accuracy with a larger ResNet (Figure 1). In the semi-supervised and transfer settings on ImageNet, we obtain results on par or superior to the current state of the art. Our contributions are: (*i*) We introduce BYOL, a self-supervised representation learning method (Section 3) which achieves state-of-the-art results under the linear evaluation protocol on ImageNet without using negative pairs. (*ii*) We show that our learned representation outperforms the state of the art on semi-supervised and transfer benchmarks (Section 4). (*iii*) We show that BYOL is more resilient to changes in the batch size and in the set of image augmentations compared to its contrastive counterparts (Section 5). In particular, BYOL suffers a much smaller performance drop than SimCLR, a strong contrastive baseline, when only using random crops as image augmentations.

2 Related work

Most unsupervised methods for representation learning can be categorized as either generative or discriminative [20, 8]. Generative approaches to representation learning build a distribution over data and latent embedding and use the learned embeddings as image representations. Many of these approaches rely either on auto-encoding of images [21, 22, 23] or on adversarial learning [24], jointly modelling data and representation [25, 26, 27, 28]. Generative methods typically operate directly in pixel space. This however is computationally expensive, and the high level of detail required for image generation may not be necessary for representation learning.

Among discriminative methods, contrastive methods [9, 10, 29, 30, 31, 11, 32, 33] currently achieve state-of-the-art performance in self-supervised learning [34, 8, 12]. Contrastive approaches avoid a costly generation step in pixel space by bringing representation of different views of the same image closer ('positive pairs'), and spreading representations of views from different images ('negative pairs') apart [35, 36]. Contrastive methods often require comparing each example with many other examples to work well [9, 8] prompting the question of whether using negative pairs is necessary.

DeepCluster [17] partially answers this question. It uses bootstrapping on previous versions of its representation to produce targets for the next representation; it clusters data points using the prior representation, and uses the cluster index of each sample as a classification target for the new representation. While avoiding the use of negative pairs, this requires a costly clustering phase and specific precautions to avoid collapsing to trivial solutions.

Some self-supervised methods are not contrastive but rely on using auxiliary handcrafted prediction tasks to learn their representation. In particular, relative patch prediction [20, 36], colorizing gray-scale images [37, 38], image inpainting [39], image jigsaw puzzle [40], image super-resolution [41], and geometric transformations [42, 43] have been shown to be useful. Yet, even with suitable architectures [44], these methods are being outperformed by contrastive methods [34, 8, 12].

Our approach has some similarities with *Predictions of Bootstrapped Latents* (PBL, [45]), a self-supervised representation learning technique for reinforcement learning. PBL jointly trains the agent's history representation and an encoding of future observations. The observation encoding is used as a target to train the agent's representation, and the agent's representation as a target to train the observation encoding. Unlike PBL, BYOL uses a slow-moving average of its representation to provide its targets, and does not require a second network.

In self-supervised learning, MoCo [9] uses a moving average network (*momentum encoder*) to maintain consistent representations of negative pairs drawn from a memory bank. Instead, BYOL uses a moving average network to produce prediction targets as a means of stabilizing the bootstrap step. We show in Section 5 that this mere stabilizing effect can also improve existing contrastive methods.

[2]Throughout this paper, the term *bootstrap* is used in its idiomatic sense rather than the statistical sense.

3 Method

We start by motivating our method before explaining its details in Section 3.1. Many successful self-supervised learning approaches build upon the cross-view prediction framework introduced in [46]. Typically, these approaches learn representations by predicting different views (e.g., different random crops) of the same image from one another. Many such approaches cast the prediction problem directly in representation space: the representation of an augmented view of an image should be predictive of the representation of another augmented view of the same image. However, predicting directly in representation space can lead to collapsed representations: for instance, a representation that is constant across views is always fully predictive of itself. Contrastive methods circumvent this problem by reformulating the prediction problem into one of discrimination: from the representation of an augmented view, they learn to discriminate between the representation of another augmented view of the same image, and the representations of augmented views of different images. In the vast majority of cases, this prevents the training from finding collapsed representations. Yet, this discriminative approach typically requires comparing each representation of an augmented view with many negative examples, to find ones sufficiently close to make the discrimination task challenging. In this work, we thus tasked ourselves to find out whether these negative examples are indispensable to prevent collapsing while preserving high performance.

To prevent collapse, a straightforward solution is to use a fixed randomly initialized network to produce the targets for our predictions. While avoiding collapse, it empirically does not result in very good representations. Nonetheless, it is interesting to note that the representation obtained using this procedure can already be much better than the initial fixed representation. In our ablation study (Section 5), we apply this procedure by predicting a fixed randomly initialized network and achieve 18.8% top-1 accuracy (Table 5a) on the linear evaluation protocol on ImageNet, whereas the randomly initialized network only achieves 1.4% by itself. This experimental finding is the core motivation for BYOL: from a given representation, referred to as *target*, we can train a new, potentially enhanced representation, referred to as *online*, by predicting the target representation. From there, we can expect to build a sequence of representations of increasing quality by iterating this procedure, using subsequent online networks as new target networks for further training. In practice, BYOL generalizes this bootstrapping procedure by iteratively refining its representation, but using a slowly moving exponential average of the online network as the target network instead of fixed checkpoints.[3]

3.1 Description of BYOL

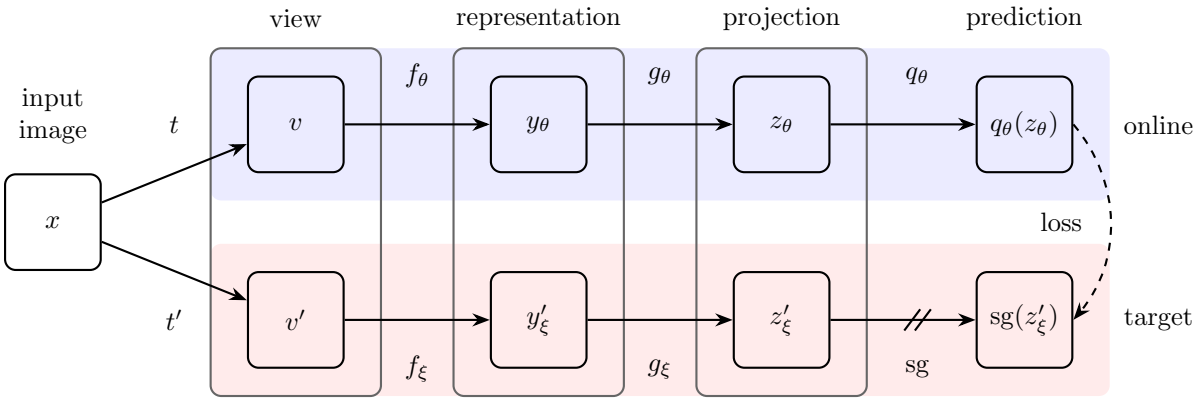


Figure 2: BYOL's architecture. BYOL minimizes a similarity loss between $q_\theta(z)$ and $\mathrm{sg}(z')$, where θ are the trained weights, ξ are an exponential moving average of θ and sg means stop-gradient. At the end of training, everything but f_θ is discarded and y is used as the image representation.

BYOL's goal is to learn a representation y which can then be used for downstream tasks. As described previously, BYOL uses two neural networks to learn: the *online* and *target* networks. The online network is defined by a set of weights θ and is comprised of three stages: an *encoder* f_θ, a *projector* g_θ and a *predictor* q_θ, as shown in Figure 2.

[3]Target networks have been introduced in [47] and are commonly used [48, 49, 50] in deep reinforcement learning (RL). In deep RL, target networks stabilize the bootstrapping updates provided by the Bellman equation, making them appealing to stabilize the bootstrap mechanism in BYOL. While fixed target networks are more common in deep RL, BYOL uses a weighted moving average of previous networks, similar to [51], in order to provide smoother changes in the target representation.

The target network has the same architecture as the online network, but uses a different set of weights ξ. The target network provides the regression targets to train the online network, and its parameters ξ are an exponential moving average of the online parameters θ [51]. More precisely, given a target decay rate $\tau \in [0, 1]$, after each training step we perform the following update,

$$\xi \leftarrow \tau\xi + (1 - \tau)\theta. \tag{1}$$

Given a set of images $\mathcal{D}$, an image $x \sim \mathcal{D}$ sampled uniformly from $\mathcal{D}$, and two distributions of image augmentations $\mathcal{T}$ and $\mathcal{T}'$, BYOL produces two augmented views $v \triangleq t(x)$ and $v' \triangleq t'(x)$ from x by applying respectively image augmentations $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}'$. From the first augmented view v, the online network outputs a *representation* $y \triangleq f_\theta(v)$ and a projection $z_\theta \triangleq g_\theta(y)$. The target network outputs $y' \triangleq f_\xi(v')$ and the *target projection* $z'_\xi \triangleq g_\xi(y')$ from the second augmented view v'. We then output a *prediction* $q_\theta(z_\theta)$ of z'_ξ and ℓ_2-normalize both $q_\theta(z_\theta)$ and z'_ξ to $\overline{q_\theta}(z_\theta) \triangleq q_\theta(z_\theta)/\|q_\theta(z_\theta)\|_2$ and $\overline{z}'_\xi \triangleq z'_\xi/\|z'_\xi\|_2$. Finally we define the following mean squared error between the normalized predictions and target projections,[4]

$$\mathcal{L}^{\texttt{BYOL}}_\theta \triangleq \left\|\overline{q_\theta}(z_\theta) - \overline{z}'_\xi\right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi\rangle}{\left\|q_\theta(z_\theta)\right\|_2 \cdot \left\|z'_\xi\right\|_2}. \tag{2}$$

We symmetrize the loss $\mathcal{L}^{\texttt{BYOL}}_\theta$ in Eq. 2 by separately feeding v' to the online network and v to the target network to compute $\widetilde{\mathcal{L}}^{\texttt{BYOL}}_\theta$. At each training step, we perform a stochastic optimization step to minimize $\mathcal{L}^{\texttt{BYOL}}_\theta + \widetilde{\mathcal{L}}^{\texttt{BYOL}}_\theta$ with respect to θ only, but *not* ξ, as depicted by the stop gradient in Figure 2.

At the end of training, we only keep the encoder f_θ; as in [9]. When comparing to other methods we consider the number of inference-time weights only in the final representation f_θ. The full training procedure is summarized in Appendix A, and `python` pseudo-code based on the libraries `JAX` [52] and `Haiku` [53] is provided in Appendix G.

3.2 Implementation details

Image augmentations BYOL uses the same set of image augmentations as in SimCLR [8]. First, a random patch of the image is selected and resized to 224×224 with a random horizontal flip, followed by a color distortion, consisting of a random sequence of brightness, contrast, saturation, hue adjustments, and an optional grayscale conversion. Finally Gaussian blur and solarization are applied to the patches. Additional details on the image augmentations are in Appendix B.

Architecture We use a convolutional residual network [19] with 50 layers and post-activation (ResNet-50($1\times$) v1) as our base parametric encoders f_θ and f_ξ. We also use deeper (50, 101, 152 and 200 layers) and wider (from $1\times$ to $4\times$) ResNets, as in [54, 44, 8]. Specifically, the representation y corresponds to the output of the final average pooling layer, which has a feature dimension of 2048 (for a width multiplier of $1\times$). As in SimCLR [8], the representation y is projected to a smaller space by a *multi-layer perceptron* (MLP) g_θ, and similarly for the target projection g_ξ. This MLP consists in a linear layer with output size 4096 followed by batch normalization [55], rectified linear units (ReLU) [56], and a final linear layer with output dimension 256. Contrary to SimCLR, the output of this MLP is not batch normalized. The predictor q_θ uses the same architecture as g_θ.

Optimization We use the LARS optimizer [57] with a cosine decay learning rate schedule [58], without restarts, over 1000 epochs, with a warm-up period of 10 epochs. We set the base learning rate to 0.2, scaled linearly [59] with the batch size (LearningRate $= 0.2 \times$ BatchSize$/256$). In addition, we use a global weight decay parameter of $1.5 \cdot 10^{-6}$ while excluding the biases and batch normalization parameters from both LARS adaptation and weight decay. For the target network, the exponential moving average parameter τ starts from $\tau_{\text{base}} = 0.996$ and is increased to one during training. Specifically, we set $\tau \triangleq 1 - (1 - \tau_{\text{base}}) \cdot (\cos(\pi k/K) + 1)/2$ with k the current training step and K the maximum number of training steps. We use a batch size of 4096 split over 512 Cloud TPU v3 cores. With this setup, training takes approximately 8 hours for a ResNet-50($\times 1$). All hyperparameters are summarized in Appendix G.1.

[4] While we could directly predict the representation y and not a projection z, previous work [8] have empirically shown that using this projection improves performance.

4 Experimental evaluation

We assess the performance of BYOL's representation[5] after self-supervised pretraining on the training set of the ImageNet ILSVRC-2012 dataset [18]. We first evaluate it on ImageNet (IN) in both linear evaluation and semi-supervised setups. We then measure its transfer capabilities on other datasets and tasks, including classification, segmentation, object detection and depth estimation. For comparison, we also report scores for a representation trained using labels from the `train` ImageNet subset, referred to as Supervised-IN. In Appendix E, we assess the generality of BYOL by pretraining a representation on the Places365-Standard dataset [60] before reproducing this evaluation protocol.

Linear evaluation on ImageNet We first evaluate BYOL's representation by training a linear classifier on top of the frozen representation, following the procedure described in [44, 61, 37, 10, 8], and appendix C.1; we report top-1 and top-5 accuracies in % on the `test` set in Table 1. With a standard ResNet-50 ($\times 1$) BYOL obtains 74.3% top-1 accuracy (91.6% top-5 accuracy), which is a 1.3% (resp. 0.5%) improvement over the previous self-supervised state of the art [12]. This tightens the gap with respect to the supervised baseline of [8], 76.5%, but is still significantly below the stronger supervised baseline of [62], 78.9%. With deeper and wider architectures, BYOL consistently outperforms the previous state of the art (Appendix C.2), and obtains a best performance of 79.6% top-1 accuracy, ranking higher than previous self-supervised approaches. On a ResNet-50 ($4\times$) BYOL achieves 78.6%, similar to the 78.9% of the best supervised baseline in [8] for the same architecture.

Method	Top-1	Top-5
Local Agg.	60.2	-
PIRL [32]	63.6	-
CPC v2 [29]	63.8	85.3
CMC [11]	66.2	87.0
SimCLR [8]	69.3	89.0
MoCo v2 [34]	71.1	-
InfoMin Aug. [12]	73.0	91.1
BYOL (ours)	**74.3**	**91.6**

(a) ResNet-50 encoder.

Method	Architecture	Param.	Top-1	Top-5
SimCLR [8]	ResNet-50 ($2\times$)	94M	74.2	92.0
CMC [11]	ResNet-50 ($2\times$)	94M	70.6	89.7
BYOL (ours)	ResNet-50 ($2\times$)	94M	77.4	93.6
CPC v2 [29]	ResNet-161	305M	71.5	90.1
MoCo [9]	ResNet-50 ($4\times$)	375M	68.6	-
SimCLR [8]	ResNet-50 ($4\times$)	375M	76.5	93.2
BYOL (ours)	ResNet-50 ($4\times$)	375M	78.6	94.2
BYOL (ours)	ResNet-200 ($2\times$)	250M	**79.6**	**94.8**

(b) Other ResNet encoder architectures.

Table 1: Top-1 and top-5 accuracies (in %) under linear evaluation on ImageNet.

Semi-supervised training on ImageNet Next, we evaluate the performance obtained when fine-tuning BYOL's representation on a classification task with a small subset of ImageNet's `train` set, this time using label information. We follow the semi-supervised protocol of [61, 63, 8, 29] detailed in Appendix C.1, and use the same fixed splits of respectively 1% and 10% of ImageNet labeled training data as in [8]. We report both top-1 and top-5 accuracies on the `test` set in Table 2. BYOL consistently outperforms previous approaches across a wide range of architectures. Additionally, as detailed in Appendix C.1, BYOL reaches 77.7% top-1 accuracy with ResNet-50 when fine-tuning over 100% of ImageNet labels.

Method	Top-1		Top-5	
	1%	10%	1%	10%
Supervised [64]	25.4	56.4	48.4	80.4
InstDisc	-	-	39.2	77.4
PIRL [32]	-	-	57.2	83.8
SimCLR [8]	48.3	65.6	75.5	87.8
BYOL (ours)	**53.2**	**68.8**	**78.4**	**89.0**

(a) ResNet-50 encoder.

Method	Architecture	Param.	Top-1		Top-5	
			1%	10%	1%	10%
CPC v2 [29]	ResNet-161	305M	-	-	77.9	91.2
SimCLR [8]	ResNet-50 ($2\times$)	94M	58.5	71.7	83.0	91.2
BYOL (ours)	ResNet-50 ($2\times$)	94M	62.2	73.5	84.1	91.7
SimCLR [8]	ResNet-50 ($4\times$)	375M	63.0	74.4	85.8	92.6
BYOL (ours)	ResNet-50 ($4\times$)	375M	69.1	75.7	87.9	92.5
BYOL (ours)	ResNet-200 ($2\times$)	250M	**71.2**	**77.7**	**89.5**	**93.7**

(b) Other ResNet encoder architectures.

Table 2: Semi-supervised training with a fraction of ImageNet labels.

[5] Our training code and pre-trained models will be publicly released at a later date.

Method	Food101	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	VOC2007	DTD	Pets	Caltech-101	Flowers
Linear evaluation:												
BYOL (ours)	**75.3**	91.3	**78.4**	**57.2**	**62.2**	**67.8**	60.6	82.5	75.5	90.4	94.2	**96.1**
SimCLR (repro)	72.8	90.5	74.4	42.4	60.6	49.3	49.8	81.4	**75.7**	84.6	89.3	92.6
SimCLR [8]	68.4	90.6	71.6	37.4	58.8	50.3	50.3	80.5	74.5	83.6	90.3	91.2
Supervised-IN [8]	72.3	**93.6**	78.3	53.7	61.9	66.7	**61.0**	**82.8**	74.9	**91.5**	**94.5**	94.7
Fine-tuned:												
BYOL (ours)	**88.5**	**97.8**	86.1	**76.3**	63.7	91.6	**88.1**	**85.4**	**76.2**	91.7	**93.8**	97.0
SimCLR (repro)	87.5	97.4	85.3	75.0	63.9	91.4	87.6	84.5	75.4	89.4	91.7	96.6
SimCLR [8]	88.2	97.7	85.9	75.9	63.5	91.3	88.1	84.1	73.2	89.2	92.1	97.0
Supervised-IN [8]	88.3	97.5	**86.4**	75.8	**64.3**	**92.1**	86.0	85.0	74.6	**92.1**	93.3	**97.6**
Random init [8]	86.9	95.9	80.2	76.1	53.6	91.4	85.9	67.3	64.8	81.5	72.6	92.0

Table 3: Transfer learning results from ImageNet (IN) with the standard ResNet-50 architecture.

Transfer to other classification tasks We evaluate our representation on other classification datasets to assess whether the features learned on ImageNet (IN) are generic and thus useful across image domains, or if they are ImageNet-specific. We perform linear evaluation and fine-tuning on the same set of classification tasks used in [8, 61], and carefully follow their evaluation protocol, as detailed in Appendix D. Performance is reported using standard metrics for each benchmark, and results are provided on a held-out `test` set after hyperparameter selection on a validation set. We report results in Table 3, both for linear evaluation and fine-tuning. BYOL outperforms SimCLR on all benchmarks and the Supervised-IN baseline on 7 of the 12 benchmarks, providing only slightly worse performance on the 5 remaining benchmarks. BYOL's representation can be transferred over to small images, e.g., CIFAR [65], landscapes, e.g., SUN397 [66] or VOC2007 [67], and textures, e.g., DTD [68].

Transfer to other vision tasks We evaluate our representation on different tasks relevant to computer vision practitioners, namely semantic segmentation, object detection and depth estimation. With this evaluation, we assess whether BYOL's representation generalizes beyond classification tasks.

We first evaluate BYOL on the VOC2012 semantic segmentation task as detailed in Appendix D.4, where the goal is to classify each pixel in the image [7]. We report the results in Table 4a. BYOL outperforms both the Supervised-IN baseline (+1.9 mIoU) and SimCLR (+1.1 mIoU).

Similarly, we evaluate on object detection by reproducing the setup in [9] using a Faster R-CNN architecture [69], as detailed in Appendix D.5. We fine-tune on `trainval2007` and report results on `test2007` using the standard AP_{50} metric; BYOL is significantly better than the Supervised-IN baseline (+3.1 AP_{50}) and SimCLR (+2.3 AP_{50}).

Finally, we evaluate on depth estimation on the NYU v2 dataset, where the depth map of a scene is estimated given a single RGB image. Depth prediction measures how well a network represents geometry, and how well that information can be localized to pixel accuracy [36]. The setup is based on [70] and detailed in Appendix D.6. We evaluate on the commonly used `test` subset of 654 images and report results using several common metrics in Table 4b: relative (rel) error, root mean squared (rms) error, and the percent of pixels (pct) where the error, $\max(d_{gt}/d_p, d_p/d_{gt})$, is below 1.25^n thresholds where d_p is the predicted depth and d_{gt} is the ground truth depth [36]. BYOL is better or on par with other methods for each metric. For instance, the challenging pct.$<$1.25 measure is respectively improved by +3.5 points and +1.3 points compared to supervised and SimCLR baselines.

Method	AP_{50}	mIoU
Supervised-IN [9]	74.4	74.4
MoCo [9]	74.9	72.5
SimCLR (repro)	75.2	75.2
BYOL (ours)	**77.5**	**76.3**

(a) Transfer results in semantic segmentation and object detection.

Method	Higher better			Lower better	
	pct.<1.25	pct.$<1.25^2$	pct.$<1.25^3$	rms	rel
Supervised-IN [70]	81.1	95.3	98.8	0.573	**0.127**
SimCLR (repro)	83.3	96.5	99.1	0.557	0.134
BYOL (ours)	**84.6**	**96.7**	**99.1**	**0.541**	0.129

(b) Transfer results on NYU v2 depth estimation.

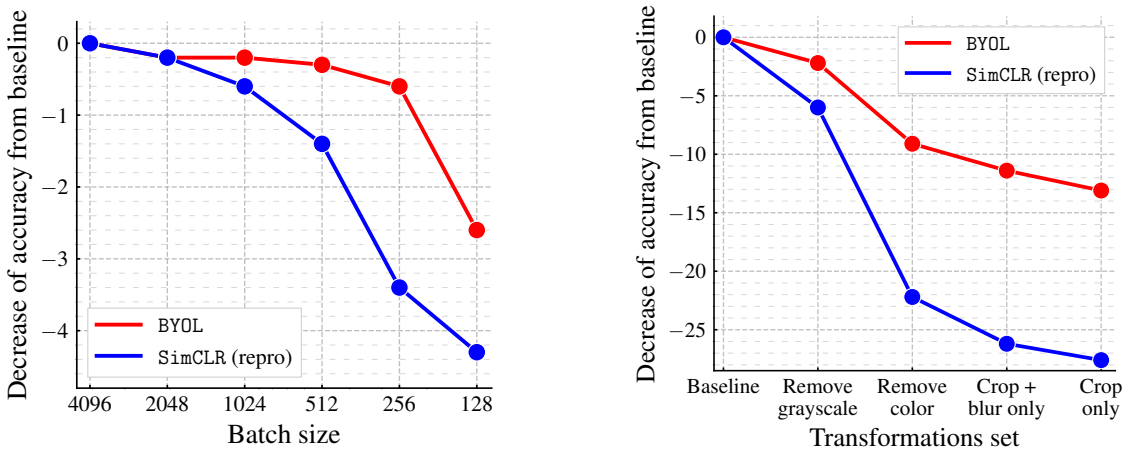
Table 4: Results on transferring BYOL's representation to other vision tasks.

5 Building intuitions with ablations

We present ablations on BYOL to give an intuition of its behavior and performance. For reproducibility, we run each configuration of parameters over three seeds, and report the average performance. We also report the half difference between the best and worst runs when it is larger than 0.25. Although previous works perform ablations at 100 epochs [8, 12], we notice that relative improvements at 100 epochs do not always hold over longer training. For this reason, we run ablations over 300 epochs on 64 TPU v3 cores, which yields consistent results compared to our baseline training of 1000 epochs. For all the experiments in this section, we set the initial learning rate to 0.3 with batch size 4096, the weight decay to 10^{-6} as in SimCLR [8] and the base target decay rate τ_{base} to 0.99. In this section we report results in top-1 accuracy on ImageNet under the linear evaluation protocol as in Appendix C.1.

Batch size Among contrastive methods, the ones that draw negative examples from the minibatch suffer performance drops when their batch size is reduced. BYOL does not use negative examples and we expect it to be more robust to smaller batch sizes. To empirically verify this hypothesis, we train both BYOL and SimCLR using different batch sizes from 128 to 4096. To avoid re-tuning other hyperparameters, we average gradients over N consecutive steps before updating the online network when reducing the batch size by a factor N. The target network is updated once every N steps, after the update of the online network; we accumulate the N-steps in parallel in our runs.

As shown in Figure 3a, the performance of SimCLR rapidly deteriorates with batch size, likely due to the decrease in the number of negative examples. In contrast, the performance of BYOL remains stable over a wide range of batch sizes from 256 to 4096, and only drops for smaller values due to batch normalization layers in the encoder.[6]



(a) Impact of batch size

(b) Impact of progressively removing transformations

Figure 3: Decrease in top-1 accuracy (in % points) of BYOL and our own reproduction of SimCLR at 300 epochs, under linear evaluation on ImageNet.

Image augmentations Contrastive methods are sensitive to the choice of image augmentations. For instance, SimCLR does not work well when removing color distortion from its image augmentations. As an explanation, SimCLR shows that crops of the same image mostly share their color histograms. At the same time, color histograms vary across images. Therefore, when a contrastive task only relies on random crops as image augmentations, it can be mostly solved by focusing on color histograms alone. As a result the representation is not incentivized to retain information beyond color histograms. To prevent that, SimCLR adds color distortion to its set of image augmentations. Instead, BYOL is incentivized to keep any information captured by the target representation into its online network, to improve its predictions. Therefore, even if augmented views of a same image share the same color histogram, BYOL is still incentivized to retain additional features in its representation. For that reason, we believe that BYOL is more robust to the choice of image augmentations than contrastive methods.

Results presented in Figure 3b support this hypothesis: the performance of BYOL is much less affected than the performance of SimCLR when removing color distortions from the set of image augmentations (-9.1 accuracy points for BYOL, -22.2 accuracy points for SimCLR). When image augmentations are reduced to mere random crops, BYOL still displays good performance (59.4%, *i.e.* -13.1 points from 72.5%), while SimCLR loses more than a third of its performance (40.3%, *i.e.* -27.6 points from 67.9%). We report additional ablations in Appendix F.3.

[6]The only dependency on batch size in our training pipeline sits within the batch normalization layers.

Bootstrapping BYOL uses the projected representation of a target network, whose weights are an exponential moving average of the weights of the online network, as target for its predictions. This way, the weights of the target network represent a delayed and more stable version of the weights of the online network. When the target decay rate is 1, the target network is never updated, and remains at a constant value corresponding to its initialization. When the target decay rate is 0, the target network is instantaneously updated to the online network at each step. There is a trade-off between updating the targets too often and updating them too slowly, as illustrated in Table 5a. Instantaneously updating the target network ($\tau = 0$) destabilizes training, yielding very poor performance while never updating the target ($\tau = 1$) makes the training stable but prevents iterative improvement, ending with low-quality final representation. All values of the decay rate between 0.9 and 0.999 yield performance above 68.4% top-1 accuracy at 300 epochs.

Target	τ_{base}	Top-1
Constant random network	1	18.8±0.7
Moving average of online	0.999	69.8
Moving average of online	0.99	**72.5**
Moving average of online	0.9	68.4
Stop gradient of online†	0	0.3

(a) Results for different target modes. †In the *stop gradient of online*, $\tau = \tau_{\text{base}} = 0$ is kept constant throughout training.

Method	Predictor	Target network	β	Top-1
BYOL	✓	✓	0	**72.5**
	✓	✓	1	70.9
		✓	1	70.7
SimCLR			1	69.4
	✓		1	69.1
	✓		0	0.3
		✓	0	0.2
			0	0.1

(b) Intermediate variants between BYOL and SimCLR.

Table 5: Ablations with top-1 accuracy (in %) at 300 epochs under linear evaluation on ImageNet.

Ablation to contrastive methods In this subsection, we recast SimCLR and BYOL using the same formalism to better understand where the improvement of BYOL over SimCLR comes from. Let us consider the following objective that extends the InfoNCE objective [10],

$$\text{InfoNCE}_\theta \triangleq \frac{2}{B}\sum_{i=1}^{B} S_\theta(v_i, v'_i) - \beta \cdot \frac{2\alpha}{B}\sum_{i=1}^{B} \ln\left(\sum_{j\neq i} \exp\frac{S_\theta(v_i, v_j)}{\alpha} + \sum_j \exp\frac{S_\theta(v_i, v'_j)}{\alpha}\right), \tag{3}$$

where $\alpha > 0$ is a fixed temperature, $\beta \in [0, 1]$ a weighting coefficient, B the batch size, v and v' are batches of augmented views where for any batch index i, v_i and v'_i are augmented views from the same image; the real-valued function S_θ quantifies pairwise similarity between augmented views. For any augmented view u we denote $z_\theta(u) \triangleq f_\theta(g_\theta(u))$ and $z_\xi(u) \triangleq f_\xi(g_\xi(u))$. For given ϕ and ψ, we consider the normalized dot product

$$S_\theta(u_1, u_2) \triangleq \frac{\langle \phi(u_1), \psi(u_2)\rangle}{\|\phi(u_1)\|_2 \cdot \|\psi(u_2)\|_2}. \tag{4}$$

Up to minor details (cf. Appendix F.5), we recover the SimCLR loss with $\phi(u_1) = z_\theta(u_1)$ (no predictor), $\psi(u_2) = z_\theta(u_2)$ (no target network) and $\beta = 1$. We recover the BYOL loss when using a predictor and a target network, *i.e.,* $\phi(u_1) = p_\theta(z_\theta(u_1))$ and $\psi(u_2) = z_\xi(u_2)$ with $\beta = 0$. To evaluate the influence of the target network, the predictor and the coefficient β, we perform an ablation over them. Results are presented in Table 5b and more details are given in Appendix F.4.

The only variant that performs well without negative examples (i.e., with $\beta = 0$) is BYOL, using *both* a bootstrap target network *and* a predictor. Adding the negative pairs to BYOL's loss without re-tuning the temperature parameter hurts its performance. In Appendix F.4, we show that we can add back negative pairs and still match the performance of BYOL with proper tuning of the temperature.

Simply adding a target network to SimCLR already improves performance (+1.6 points). This sheds new light on the use of the target network in MoCo [9], where the target network is used to provide more negative examples. Here, we show that by mere stabilization effect, even when using the same number of negative examples, using a target network is beneficial. Finally, we observe that modifying the architecture of S_θ to include a predictor only mildly affects the performance of SimCLR.

6 Conclusion

We introduced BYOL, a new algorithm for self-supervised learning of image representations. BYOL learns its representation by predicting previous versions of its outputs, without using negative pairs. We show that BYOL achieves state-of-the-art results on various benchmarks. In particular, under the linear evaluation protocol on ImageNet with a ResNet-50 ($1\times$), BYOL achieves a new state of the art and bridges most of the remaining gap between self-supervised methods and the supervised learning baseline of [8]. Using a ResNet-200 ($2\times$), BYOL reaches a top-1 accuracy of 79.6% which improves over the previous state of the art (76.8%) while using 30% fewer parameters.

Nevertheless, BYOL remains dependent on existing sets of augmentations that are specific to vision applications. To generalize BYOL to other modalities (e.g., audio, video, text, ...) it is necessary to obtain similarly suitable augmentations for each of them. Designing such augmentations may require significant effort and expertise. Therefore, automating the search for these augmentations would be an important next step to generalize BYOL to other modalities.

Broader impact

The presented research should be categorized as research in the field of unsupervised learning. This work may inspire new algorithms, theoretical, and experimental investigation. The algorithm presented here can be used for many different vision applications and a particular use may have both positive or negative impacts, which is known as the dual use problem. Besides, as vision datasets could be biased, the representation learned by BYOL could be susceptible to replicate these biases.

Acknowledgements

The authors would like to thank the following people for their help throughout the process of writing this paper, in alphabetical order: Aaron van den Oord, Andrew Brock, Jason Ramapuram, Jeffrey De Fauw, Karen Simonyan, Katrina McKinney, Nathalie Beauguerlange, Olivier Henaff, Oriol Vinyals, Pauline Luc, Razvan Pascanu, Sander Dieleman, and the DeepMind team.

References

[1] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[2] Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4), 2002.

[3] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[4] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition*, 2014.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2014.

[7] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Computer Vision and Pattern Recognition*, 2015.

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

[9] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[10] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[11] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849v4*, 2019.

[12] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning. *arXiv preprint arXiv:2005.10243*, 2020.

[13] Nikunj Saunshi, Orestis Plevrakis, Sanjeev Arora, Mikhail Khodak, and Hrishikesh Khandeparkar. A theoretical analysis of contrastive unsupervised representation learning. In *International Conference on Machine Learning*, 2019.

[14] R. Manmatha, Chao-Yuan Wu, Alexander J. Smola, and Philipp Krähenbühl. Sampling matters in deep embedding learning. In *International Conference on Computer Vision*, 2017.

[15] Ben Harwood, Vijay B. G. Kumar, Gustavo Carneiro, Ian Reid, and Tom Drummond. Smart mining for deep metric learning. In *International Conference on Computer Vision*, 2017.

[16] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *International Conference on Machine Learning*, 2013.

[17] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *European Conference on Computer Vision*, 2018.

[18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, 2016.

[20] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Computer Vision and Pattern Recognition*, 2015.

[21] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, 2008.

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[23] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and variational inference in deep latent gaussian models. *arXiv preprint arXiv:1401.4082*, 2014.

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Neural Information Processing Systems*, 2014.

[25] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[26] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martín Arjovsky, Olivier Mastropietro, and Aaron C. Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2017.

[27] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. In *Neural Information Processing Systems*, 2019.

[28] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[29] Olivier J. Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, S. M. Ali Eslami, and Aäron van den Oord. Data-efficient image recognition with contrastive predictive coding. In *International Conference on Machine Learning*, 2019.

[30] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2019.

[31] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *Neural Information Processing Systems*, 2019.

[32] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. *arXiv preprint arXiv:1912.01991*, 2019.

[33] Junnan Li, Pan Zhou, Caiming Xiong, Richard Socher, and Steven CH Hoi. Prototypical contrastive learning of unsupervised representations. *arXiv preprint arXiv:2005.04966*, 2020.

[34] Rishabh Jain, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[35] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Computer Vision and Pattern Recognition*, 2018.

[36] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *International Conference on Computer Vision*, 2017.

[37] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In *European Conference on Computer Vision*, 2016.

[38] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. In *European Conference on Computer Vision*, 2016.

[39] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. In *Computer Vision and Pattern Recognition*, 2016.

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

[41] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Computer Vision and Pattern Recognition*, 2017.

[42] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Neural Information Processing Systems*, 2014.

[43] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[44] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Computer Vision and Pattern Recognition*, 2019.

[45] Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-Bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In *International Conference on Machine Learning*, 2020.

[46] Suzanna Becker and Geoffrey E. Hinton. Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163, 1992.

[47] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen. King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[48] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[49] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2018.

[50] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *Deep Reinforcement Learning Workshop NeurIPS*, 2018.

[51] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[52] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018.

[53] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020.

[54] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[55] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[56] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.

[57] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017.

[58] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[59] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[60] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[61] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better ImageNet models transfer better? In *Computer Cision and Pattern Recognition*, 2019.

[62] Chengyue Gong, Tongzheng Ren, Mao Ye, and Qiang Liu. Maxup: A simple way to improve generalization of neural network training. *arXiv preprint arXiv:2002.09024*, 2020.

[63] Xiaohua Zhai, Joan Puigcerver, Alexander I Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, André Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, Lucas Beyer, Olivier Bachem, Michael Tschannen, Marcin Michalski, Olivier Bousquet, Sylvain Gelly, and Neil Houlsby. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv: Computer Vision and Pattern Recognition*, 2019.

[64] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4L: Self-supervised semi-supervised learning. In *International Conference on Computer Vision*, 2019.

[65] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[66] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer Vision and Pattern Recognition*, 2010.

[67] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The Pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[68] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Computer Vision and Pattern Recognition*, 2014.

[69] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems*, 2015.

[70] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *International Conference on 3D Vision*, 2016.

[71] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, 2013.

[72] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. *arXiv preprint arXiv:1909.13719*, 2019.

[73] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[74] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

[75] Thomas Berg, Jiongxin Liu, Seung Woo Lee, Michelle L. Alexander, David W. Jacobs, and Peter N. Belhumeur. Birdsnap: Large-scale fine-grained visual categorization of birds. In *Computer Vision and Pattern Recognition*, 2014.

[76] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *Workshop on 3D Representation and Recognition*, Sydney, Australia, 2013.

[77] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew B. Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[78] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *Computer Vision and Pattern Recognition*, 2012.

[79] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Pattern Recognition Workshop*, 2004.

[80] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

[81] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, 2014.

[82] Art B Owen. A robust hybrid of lasso and ridge regression. *Contemporary Mathematics*, 443(7):59–72, 2007.

[83] Chengxu Zhuang, Alex Lin Zhai, and Daniel Yamins. Local aggregation for unsupervised learning of visual embeddings. In *International Conference on Computer Vision*, 2019.

[84] Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. Learning features by watching objects move. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[85] Yuxin Wu and Kaiming He. Group normalization. In *European Conference on Computer Vision*, 2018.

A Algorithm

Algorithm 1: BYOL: **B**ootstrap **Y**our **O**wn **L**atent

Inputs :

$\mathcal{D}$, $\mathcal{T}$, and $\mathcal{T}'$	set of images and distributions of transformations
θ, f_θ, g_θ, and q_θ	initial online parameters, encoder, projector, and predictor
ξ, f_ξ, g_ξ	initial target parameters, target encoder, and target projector
optimizer	optimizer, updates online parameters using the loss gradient
K and N	total number of optimization steps and batch size
$\{\tau_k\}_{k=1}^K$ and $\{\eta_k\}_{k=1}^K$	target network update schedule and learning rate schedule

for $k = 1$ **to** K **do**

  $\mathcal{B} \leftarrow \{x_i \sim \mathcal{D}\}_{i=1}^N$ // sample a batch of N images

  for $x_i \in \mathcal{B}$ **do**

    $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}'$ // sample image transformations

    $z_1 \leftarrow g_\theta(f_\theta(t(x_i)))$ and $z_2 \leftarrow g_\theta(f_\theta(t'(x_i)))$ // compute projections

    $z'_1 \leftarrow g_\xi(f_\xi(t'(x_i)))$ and $z'_2 \leftarrow g_\xi(f_\xi(t(x_i)))$ // compute target projections

    $l_i \leftarrow -2 \cdot \left(\frac{\langle q_\theta(z_1), z'_1 \rangle}{\|q_\theta(z_1)\|_2 \cdot \|z'_1\|_2} + \frac{\langle q_\theta(z_2), z'_2 \rangle}{\|q_\theta(z_2)\|_2 \cdot \|z'_2\|_2} \right)$ // compute the loss for x_i

  end

  $\delta\theta \leftarrow \frac{1}{N} \sum_{i=1}^N \partial_\theta l_i$ // compute the total loss gradient w.r.t. θ

  $\theta \leftarrow \text{optimizer}(\theta, \delta\theta, \eta_k)$ // update online parameters

  $\xi \leftarrow \tau_k \xi + (1 - \tau_k)\theta$ // update target parameters

end

Output : encoder f_θ

B Image augmentations

During self-supervised training, BYOL uses the following image augmentations (which are a subset of the ones presented in [8]):

- random cropping: a random patch of the image is selected, with an area uniformly sampled between 8% and 100% of that of the original image, and an aspect ratio logarithmically sampled between $3/4$ and $4/3$. This patch is then resized to the target size of 224×224 using bicubic interpolation;
- optional left-right flip;
- color jittering: the brightness, contrast, saturation and hue of the image are shifted by a uniformly random offset applied on all the pixels of the same image. The order in which these shifts are performed is randomly selected for each patch;
- color dropping: an optional conversion to grayscale. When applied, output intensity for a pixel (r, g, b) corresponds to its luma component, computed as $0.2989r + 0.5870g + 0.1140b$;
- Gaussian blurring: for a 224×224 image, a square Gaussian kernel of size 23×23 is used, with a standard deviation uniformly sampled over $[0.1, 2.0]$;
- solarization: an optional color transformation $x \mapsto x \cdot \mathbf{1}_{\{x<0.5\}} + (1 - x) \cdot \mathbf{1}_{\{x \geq 0.5\}}$ for pixels with values in $[0, 1]$.

Augmentations from the sets $\mathcal{T}$ and $\mathcal{T}'$ (introduced in Section 3) are compositions of the above image augmentations in the listed order, each applied with a predetermined probability. The image augmentations parameters are listed in Table 6.

During evaluation, we use a center crop similar to [8]: images are resized to 256 pixels along the shorter side using bicubic resampling, after which a 224×224 center crop is applied. In both training and evaluation, we normalize color channels by subtracting the average color and dividing by the standard deviation, computed on ImageNet, after applying the augmentations.

Parameter	$\mathcal{T}$	$\mathcal{T}'$
Random crop probability	1.0	1.0
Flip probability	0.5	0.5
Color jittering probability	0.8	0.8
Brightness adjustment max intensity	0.4	0.4
Contrast adjustment max intensity	0.4	0.4
Saturation adjustment max intensity	0.2	0.2
Hue adjustment max intensity	0.1	0.1
Color dropping probability	0.2	0.2
Gaussian blurring probability	1.0	0.1
Solarization probability	0.0	0.2

Table 6: Parameters used to generate image augmentations.

C Evaluation on ImageNet training

C.1 Self-supervised learning evaluation on ImageNet

Linear evaluation protocol on ImageNet As in [44, 61, 8, 34], we use the standard linear evaluation protocol on ImageNet, which consists in training a linear classifier on top of the frozen representation, *i.e.*, without updating the network parameters nor the batch statistics. At training time, we apply spatial augmentations, i.e., random crops with resize to 224×224 pixels, and random flips. At test time, images are resized to 256 pixels along the shorter side using bicubic resampling, after which a 224×224 center crop is applied. In both cases, we normalize the color channels by subtracting the average color and dividing by the standard deviation (computed on ImageNet), after applying the augmentations. We optimize the cross-entropy loss using `SGD` with Nesterov momentum over 80 epochs, using a batch size of 4096 and a momentum of 0.9. We do not use any regularization methods such as weight decay, gradient clipping [71], tclip [31], or logits regularization. We finally sweep over 5 learning rates $\{0.4, 0.3, 0.2, 0.1, 0.05\}$ on a local validation set (10009 images from ImageNet `train` set), and report the accuracy of the best validation hyperparameter on the `test` set (which is the public validation set of the original ILSVRC2012 ImageNet dataset).

Variant on linear evaluation on ImageNet In this paragraph only, we deviate from the protocol of [8, 34] and propose another way of performing linear evaluation on top of a frozen representation. This method achieves better performance both in top-1 and top-5 accuracy.

- We replace the spatial augmentations (random crops with resize to 224×224 pixels and random flips) with the pre-train augmentations of Appendix B. This method was already used in [29] with a different subset of pre-train augmentations.
- We regularize the linear classifier as in [31][7] by clipping the logits using a hyperbolic tangent function

$$\text{tclip}(x) \triangleq \alpha \cdot \tanh(x/\alpha),$$

  where α is a positive scalar, and by adding a logit-regularization penalty term in the loss

$$\text{Loss}(x, y) \triangleq \text{cross\_entropy}(\text{tclip}(x), y) + \beta \cdot \text{average}(\text{tclip}(x)^2),$$

  where x are the logits, y are the target labels, and β is the regularization parameter. We set $\alpha = 20$ and $\beta = 1e{-}2$.

We report in Table 7 the top-1 and top-5 accuracy on ImageNet using this modified protocol. These modifications in the evaluation protocol increase the `BYOL`'s top-1 accuracy from 74.3% to 74.8% with a ResNet-50 ($1\times$).

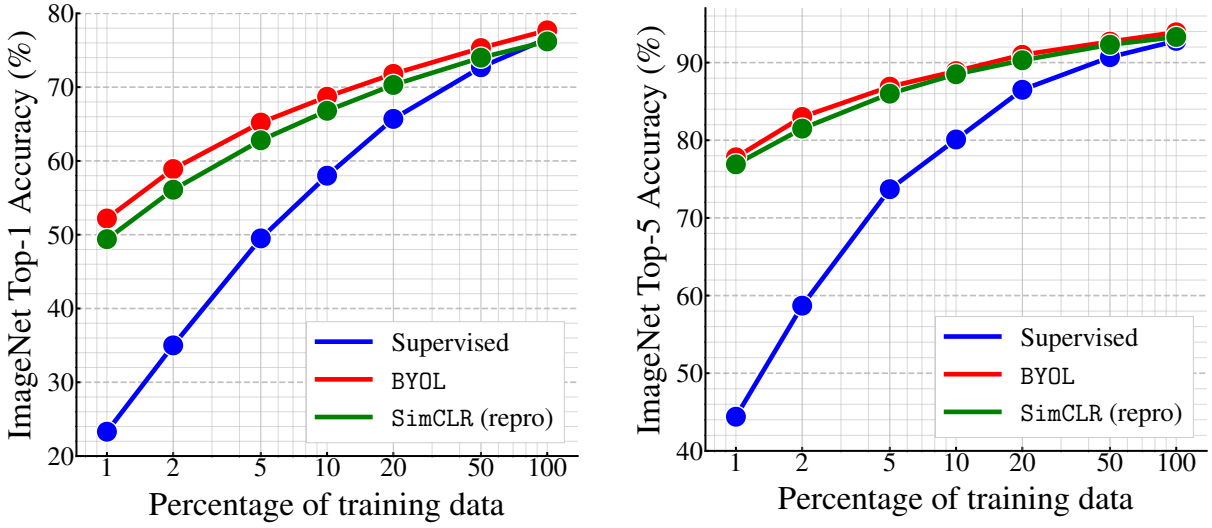
Semi-supervised learning on ImageNet We follow the semi-supervised learning protocol of [8, 64]. We first initialize the network with the parameters of the pretrained representation, and fine-tune it with a subset of ImageNet labels. At training time, we apply spatial augmentations, i.e., random crops with resize to 224×224 pixels and random flips. At test time, images are resized to 256 pixels along the shorter side using bicubic resampling, after which a 224×224 center crop is applied. In both cases, we normalize the color channels by subtracting the average color and dividing by the standard deviation (computed on ImageNet), after applying the augmentations. We optimize the cross-entropy loss using `SGD` with Nesterov momentum. We used a batch size of 4096, a momentum of

[7] https://github.com/Philip-Bachman/amdim-public/blob/master/costs.py

Architecture	Pre-train augmentations	Logits regularization	Top-1	Top-5
ResNet-50 (1×)			74.3	91.6
	✓		74.4	91.8
		✓	74.7	91.8
	✓	✓	**74.8**	**91.8**
ResNet-50 (4×)			78.6	94.2
	✓		78.6	94.3
		✓	78.9	94.3
	✓	✓	**79.0**	**94.5**
ResNet-200 (2×)			79.6	94.8
	✓		79.6	94.8
		✓	79.8	95.0
	✓	✓	**80.0**	**95.0**

Table 7: Different linear evaluation protocols on ResNet architectures by either replacing the spatial augmentations with pre-train augmentations, or regularizing the linear classifier. No pre-train augmentations and no logits regularization correspond to the evaluation protocol of the main paper, which is the same as in [8, 34].

0.9. We do not use any regularization methods such as weight decay, gradient clipping [71], tclip [31], or logits rescaling. We sweep over the learning rate $\{0.01, 0.02, 0.05, 0.1, 0.005\}$ and the number of epochs $\{30, 50\}$ and select the hyperparameters achieving the best performance on our local validation set to report test performance.



(a) Top-1 accuracy

(b) Top-5 accuracy

Figure 4: Semi-supervised training with a fraction of ImageNet labels on a ResNet-50 (×1).

In Table 2 presented in the main text, we fine-tune the representation over the 1% and 10% ImageNet splits from [8] with various ResNet architectures.

In Figure 4, we fine-tune the representation over 1%, 2%, 5%, 10%, 20%, 50%, and 100% of the ImageNet dataset as in [29] with a ResNet-50 (1×) architecture. In this case and contrary to Table 2 we don't reuse the splits from SimCLR but we create our own via a balanced selection.

Finally, we fine-tune the representation over the full ImageNet dataset. We report the results in Table 8 along with supervised baselines trained on ImageNet. We observe that fine-tuning the SimCLR checkpoint does not yield better results (in our reproduction, which matches the results reported in the original paper [8]) than using a random initialization (76.5 top-1). Instead, BYOL's initialization checkpoint leads to a high final score (77.7 top-1), higher than the vanilla supervised baseline of [8], matching the strong supervised baseline of AutoAugment[72] but still 1.2 points below the stronger supervised baseline [62], which uses advanced supervised learning techniques.

Supervised: Method	Top-1	Top-5	*Semi-supervised* (100%): Method	Top-1	Top-5
Supervised[8]	76.5	–	SimCLR [8]	76.0	93.1
AutoAugment [72]	77.6	93.8	SimCLR (repro)	76.5	93.5
MaxUp [62]	**78.9**	**94.2**	BYOL	**77.7**	**93.9**

Table 8: Semi-supervised training with the full ImageNet on a ResNet-50 ($\times 1$). We also report other fully supervised methods for extensive comparisons.

C.2 Linear evaluation on larger architectures and supervised baselines

			BYOL		Supervised (ours)		Supervised [8]
Architecture	Multiplier	Weights	Top-1	Top-5	Top-1	Top-5	Top-1
ResNet-50	$1\times$	24M	74.3	91.6	76.4	92.9	76.5
ResNet-101	$1\times$	43M	76.4	93.0	78.0	94.0	-
ResNet-152	$1\times$	58M	77.3	93.7	79.1	94.5	-
ResNet-200	$1\times$	63M	77.8	93.9	79.3	94.6	-
ResNet-50	$2\times$	94M	77.4	93.6	79.9	95.0	77.8
ResNet-101	$2\times$	170M	78.7	94.3	80.3	95.0	-
ResNet-50	$3\times$	211M	78.2	93.9	80.2	95.0	-
ResNet-152	$2\times$	232M	79.0	94.6	80.6	95.3	-
ResNet-200	$2\times$	250M	**79.6**	**94.9**	80.1	95.2	-
ResNet-50	$4\times$	375M	78.6	94.2	80.7	**95.3**	78.9
ResNet-101	$3\times$	382M	78.4	94.2	80.7	**95.3**	-
ResNet-152	$3\times$	522M	79.5	94.6	**80.9**	95.2	-

Table 9: Linear evaluation of BYOL on ImageNet using larger encoders. Top-1 and top-5 accuracies are reported in %.

Here we investigate the performance of BYOL with deeper and wider ResNet architectures. We compare ourselves to the best supervised baselines from [8] when available (rightmost column in table 9), which are also presented in Figure 1. Importantly, we close in on those baselines using the ResNet-50 ($2\times$) and the ResNet-50 ($4\times$) architectures, where we are within 0.4 accuracy points of the supervised performance. To the best of our knowledge, this is the first time that the gap to supervised has been closed to such an extent using a self-supervised method under the linear evaluation protocol. Therefore, in order to ensure fair comparison, and suspecting that the supervised baselines' performance in [8] could be even further improved with appropriate data augmentations, we also report on our own reproduction of strong supervised baselines. We use RandAugment [72] data augmentation for all large ResNet architectures (which are all version 1, as per [19]). We train our supervised baselines for up to 200 epochs, using SGD with a Nesterov momentum value of 0.9, a cosine-annealed learning rate after a 5 epochs linear warmup period, weight decay with a value of $1e-4$, and a label smoothing [73] value of 0.1. Results are presented in Figure 5.

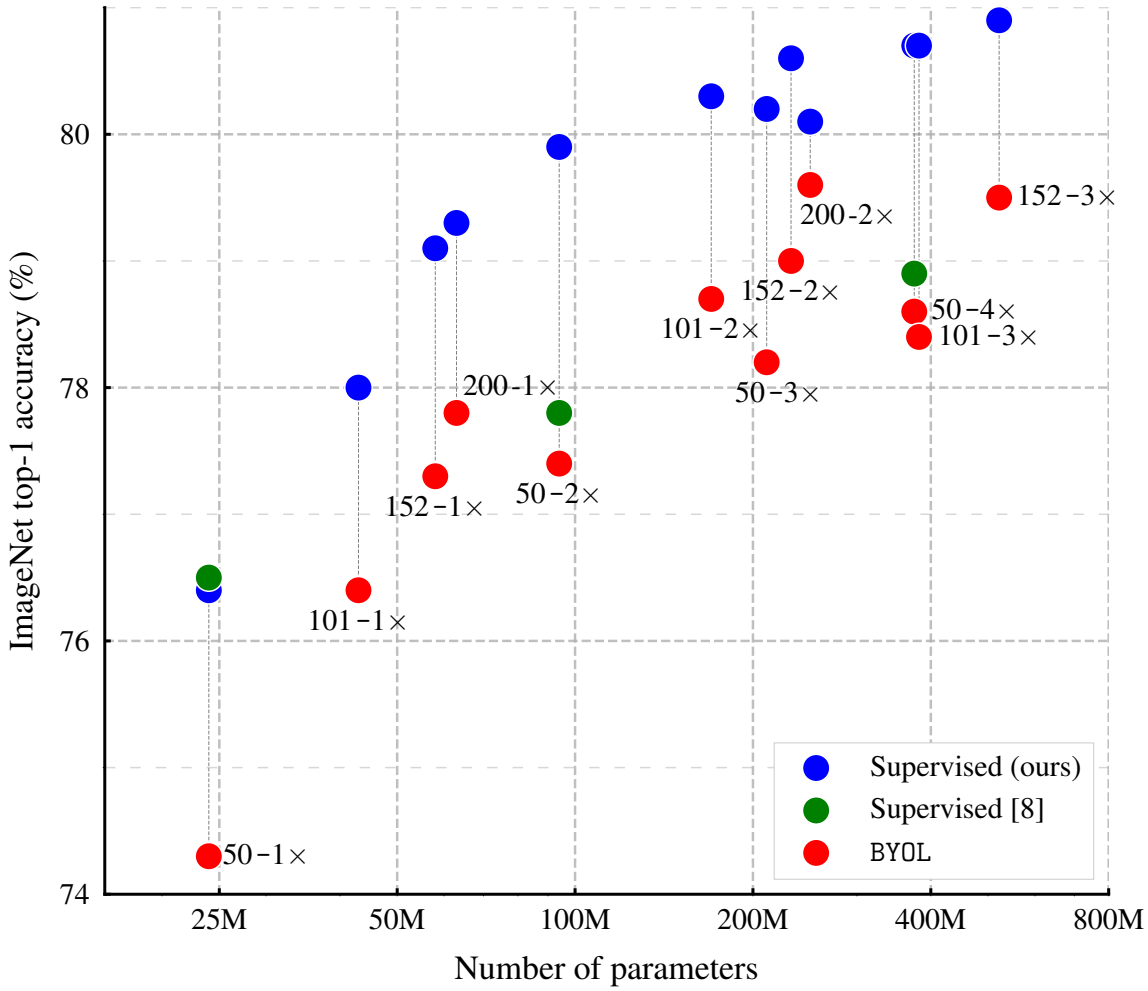


Figure 5: Results for linear evaluation of BYOL compared to fully supervised baselines with various ResNet architectures.

D Transfer to other datasets

D.1 Datasets

Dataset	Classes	Original train examples	Train examples	Valid. examples	Test examples	Accuracy measure	Test provided
ImageNet [18]	1000	1281167	1271158	10009	50000	Top-1 accuracy	-
Food101 [74]	101	75750	68175	7575	25250	Top-1 accuracy	-
CIFAR-10 [65]	10	50000	45000	5000	10000	Top-1 accuracy	-
CIFAR-100 [65]	100	50000	44933	5067	10000	Top-1 accuracy	-
Birdsnap [75]	500	47386	42405	4981	2443	Top-1 accuracy	-
Sun397 (split 1) [66]	397	19850	15880	3970	19850	Top-1 accuracy	-
Cars [76]	196	8144	6494	1650	8041	Top-1 accuracy	-
Aircraft [77]	100	3334	3334	3333	3333	Mean per-class accuracy	Yes
PASCAL-VOC2007 [67]	20	5011	2501	2510	4952	11-point mAP / AP50	-
PASCAL-VOC2012 [67]	21	10582	–	2119	1449	Mean IoU	-
DTD (split 1) [68]	47	1880	1880	1880	1880	Top-1 accuracy	Yes
Pets [78]	37	3680	2940	740	3669	Mean per-class accuracy	-
Caltech-101 [79]	101	3060	2550	510	6084	Mean per-class accuracy	-
Places365 [60]	365	1803460	1803460	–	36500	Top-1 accuracy	-
Flowers [80]	102	1020	1020	1020	6149	Mean per-class accuracy	Yes

Table 10: Characteristics of image datasets used in transfer learning. When an official test split with labels is not publicly available, we use the official validation split as test set, and create a held-out validation set from the training examples.

We perform transfer via linear classification and fine-tuning on the same set of datasets as in [8], namely Food-101 dataset [74], CIFAR-10 [65] and CIFAR-100 [65], Birdsnap [75], the SUN397 scene dataset [66], Stanford Cars [76], FGVC Aircraft [77], the PASCAL VOC 2007 classification task [67], the Describable Textures Dataset (DTD) [68], Oxford-IIIT Pets [78], Caltech-101 [79], and Oxford 102 Flowers [80]. As in [8], we used the validation sets specified by the dataset creators to select hyperparameters for FGVC Aircraft, PASCAL VOC 2007, DTD, and Oxford 102 Flowers. On other datasets, we use the validation examples as test set, and hold out a subset of the training examples that we use as validation set. We use standard metrics for each datasets:

- *Top*-1: We compute the proportion of correctly classified examples.
- *Mean per class*: We compute the top-1 accuracy for each class separately and then compute the empirical mean over the classes.
- *Point 11-mAP*: We compute the empirical mean *average precision* as defined in [67].
- *Mean IoU:* We compute the empirical mean Intersection-Over-Union as defined in [67].
- *AP50:* We compute the Average Precision as defined in [67].

We detail the validation procedures for some specific datasets:

- For Sun397 [66], the original dataset specifies 10 train/test splits, all of which contain 50 examples/images of 397 different classes. We use the first train/test split. The original dataset specifies no validation split and therefore, the training images have been further subdivided into 40 images per class for the train split and 10 images per class for the valid split.
- For Birdsnap [75], we use a random selection of valid images with the same number of images per category as the test split.
- For DTD [68], the original dataset specifies 10 train/validation/test splits, we only use the first split.
- For Caltech-101 [79], the original does not dataset specifies any train/test splits. We have followed the approach used in [81]: This file defines datasets for 5 random splits of 25 training images per category, with 5 validation images per category and the remaining images used for testing.
- For ImageNet, we took the last 10009 last images of the official tensorflow ImageNet split.
- For Oxford-IIIT Pets, the valid set consists of 20 randomly selected images per class.

Information about the dataset are summarized in Table 10.

D.2 Transfer via linear classification

We follow the linear evaluation protocol of [44, 61, 8] that we detail next for completeness. We train a regularized multinomial logistic regression classifier on top of the frozen representation, i.e., with frozen pretrained parameters

and without re-computing batch-normalization statistics. In training and testing, we do not perform any image augmentations; images are resized to 224 pixels along the shorter side using bicubic resampling and then normalized with ImageNet statistics. Finally, we minimize the cross-entropy objective using `LBFGS` with ℓ_2-regularization, where we select the regularization parameters from a range of 45 logarithmically-spaced values between 10^{-6} and 10^5. After choosing the best-performing hyperparameters on the validation set, the model is retrained on combined training and validation images together, using the chosen parameters. The final accuracy is reported on the test set.

D.3 Transfer via fine-tuning

We follow the same fine-tuning protocol as in [29, 44, 63, 8] that we also detail for completeness. Specifically, we initialize the network with the parameters of the pretrained representation. At training time, we apply spatial transformation, i.e., random crops with resize to 224×224 pixels and random flips. At test time, images are resized to 256 pixels along the shorter side using bicubic resampling, after which a 224×224 center crop is extracted. In both cases, we normalize the color channels by subtracting the average color and dividing by the standard deviation (computed on ImageNet), after applying the augmentations. We optimize the loss using `SGD` with Nesterov momentum for 20000 steps with a batch size of 256 and with a momentum of 0.9. We set the momentum parameter for the batch normalization statistics to $\max(1 - 10/s, 0.9)$ where s is the number of steps per epoch. The learning rate and weight decay are selected respectively with a grid of seven logarithmically spaced learning rates between 0.0001 and 0.1, and 7 logarithmically-spaced values of weight decay between 10^{-6} and 10^{-3}, as well as no weight decay. These values of weight decay are divided by the learning rate. After choosing the best-performing hyperparameters on the validation set, the model is retrained on combined training and validation images together, using the chosen parameters. The final accuracy is reported on the test set.

D.4 Implementation details for semantic segmentation

We use the same fully-convolutional network (FCN)-based [7] architecture as [9]. The backbone consists of the convolutional layers in ResNet-50. The 3×3 convolutions in the conv5 blocks use dilation 2 and stride 1. This is followed by two extra 3×3 convolutions with 256 channels, each followed by batch normalization and ReLU activations, and a 1×1 convolution for per-pixel classification. The dilation is set to 6 in the two extra 3×3 convolutions. The total stride is 16 (FCN-16s [7]).

We train on the `train_aug2012` set and report results on `val2012`. Hyperparameters are selected on a 2119 images held-out validation set. We use a standard per-pixel softmax cross-entropy loss to train the FCN. Training is done with random scaling (by a ratio in $[0.5, 2.0]$), cropping, and horizontal flipping. The crop size is 513. Inference is performed on the $[513, 513]$ central crop. For training we use a batch size of 16 and weight decay of 0.0001. We select the base learning rate by sweeping across 5 logarithmically spaced values between 10^{-3} and 10^{-1}. The learning rate is multiplied by 0.1 at the 70-th and 90-th percentile of training. We train for 30000 iterations, and average the results on 5 seeds.

D.5 Implementation details for object detection

For object detection, we follow prior work on Pascal detection transfer [36, 20] wherever possible. We use a Faster R-CNN [69] detector with a R50-C4 backbone with a frozen representation. The R50-C4 backbone ends with the conv4 stage of a ResNet-50, and the box prediction head consists of the conv5 stage (including global pooling). We preprocess the images by applying multi-scale augmentation (rescaling the image so its longest edge is between 480 and 1024 pixels) but no other augmentation. We use an asynchronous `SGD` optimizer with 9 workers and train for 1.5M steps. We used an initial learning rate of 10^{-3}, which is reduced to 10^{-4} at 1M steps and to 10^{-5} at 1.2M steps.

D.6 Implementation details for depth estimation

For depth estimation, we follow the same protocol as in [70], and report its core components for completeness. We use a standard ResNet-50 backbone and feed the conv5 features into 4 fast up-projection blocks with respective filter sizes 512, 256, 128, and 64. We use a reverse Huber loss function for training [70, 82].

The original NYU Depth v2 frames of size $[640, 480]$ are down-sampled by a factor 0.5 and center-cropped to $[304, 228]$ pixels. Input images are randomly horizontally flipped and the following color transformations are applied:

- *Grayscale* with an application probability of 0.3.

- *Brightness* with a maximum brightness difference of 0.1255.
- *Saturation* with a saturation factor randomly picked in the interval $[0.5, 1.5]$.
- *Hue* with a hue adjustment factor randomly picked in the interval $[-0.2, 0.2]$.

We train for 7500 steps with batch size 256, weight decay 0.0005, and learning rate 0.16 (scaled linearly from the setup of [70] to account for the bigger batch size).

D.7 Further comparisons on PASCAL and NYU v2 Depth

For completeness, Table 11 and 12 extends Table 4 with other published baselines which use comparable networks. We see that in almost all settings, BYOL outperforms these baselines, even when those baselines use more data or deeper models. One notable exception is RMS error for NYU Depth prediction, which is a metric that's sensitive to outliers. The reason for this is unclear, but one possibility is that the network is producing higher-variance predictions due to being more confident about a test-set scene's similarities with those in the training set.

Method	AP_{50}	mIoU
Supervised-IN [9]	74.4	74.4
RelPos [20], by [36]*	66.8	-
Multi-task [36]*	70.5	-
LocalAgg [83]	69.1	-
MoCo [9]	74.9	72.5
MoCo + IG-1B [9]	75.6	73.6
CPC[29]**	76.6	-
SimCLR (repro)	75.2	75.2
BYOL (ours)	**77.5**	**76.3**

Table 11: Transfer results in semantic segmentation and object detection.

* uses a larger model (ResNet-101). ** uses an even larger model (ResNet-161).

Method	Higher better pct.<1.25	pct.$<1.25^2$	pct.$<1.25^3$	Lower better rms	rel
Supervised-IN [70]	81.1	95.3	98.8	0.573	**0.127**
RelPos [20], by [36]*	80.6	94.7	98.3	**0.399**	0.146
Color [37], by [36]*	76.8	93.5	97.7	0.444	0.164
Exemplar [42, 36]*	71.3	90.6	96.5	0.513	0.191
Mot. Seg. [84], by [36]*	74.2	92.4	97.4	0.473	0.177
Multi-task [36]*	79.3	94.2	98.1	0.422	0.152
SimCLR (repro)	83.3	96.5	99.1	0.557	0.134
BYOL (ours)	**84.6**	**96.7**	**99.1**	0.541	0.129

Table 12: Transfer results on NYU v2 depth estimation.

E Pretraining on Places 365

To ascertain that BYOL learns good representations on other datasets, we applied our representation learning protocol on the scene recognition dataset Places365-Standard [60] before performing linear evaluation. This dataset contains 1.80 million training images and 36500 validation images with labels, making it roughly similar to ImageNet in scale. We reuse the *exact* same parameters as in Section 4 and train the representation for 1000 epochs, using BYOL and our SimCLR reproduction. Results for the linear evaluation setup (using the protocol of Appendix C.1 for ImageNet and Places365, and that of Appendix D on other datasets) are reported in Table 13.

Interestingly, the representation trained by using BYOL on Places365 (BYOL-PL) consistently outperforms that of SimCLR on the same dataset, but underperforms the BYOL representation trained on ImageNet (BYOL-IN) on all tasks except Places365 and SUN397 [66], another scene understanding dataset. Interestingly, all three unsupervised representation learning methods achieve a relatively high performance on the Places365 task; for comparison,

reference [60] (in its linked repository) reports a top-1 accuracy of 55.2% for a ResNet-50v2 trained from scratch using labels on this dataset.

Method	Places365	ImageNet	Food101	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	DTD	Pets	Caltech-101	Flowers
BYOL-IN	51.0	74.3	75.3	91.3	78.4	57.3	62.6	67.2	60.6	76.5	90.4	94.3	96.1
BYOL-PL	**53.2**	**58.5**	**64.7**	**84.5**	**66.1**	**28.8**	**64.2**	**55.6**	**55.9**	**68.5**	**66.1**	**84.3**	**90.0**
SimCLR-PL	53.0	56.5	61.7	80.8	61.1	21.2	62.5	40.1	44.3	64.3	59.4	77.1	85.9

Table 13: Transfer learning results (linear evaluation, ResNet-50) from Places365 (PL). For comparison purposes, we also report the results from BYOL trained on ImageNet (BYOL-IN).

F Additional ablation results

To extend on the above results, we provide additional ablations obtained using the same experimental setup as in Section 5, *i.e.,* 300 epochs, averaged over 3 seeds with the initial learning rate set to 0.3, the batch size to 4096, the weight decay to 10^{-6} and the base target decay rate τ_{base} to 0.99 unless specified otherwise. Confidence intervals correspond to the half-difference between the maximum and minimum score of these seeds; we omit them for half-differences lower than 0.25 accuracy points.

F.1 Architecture settings

Table 14 shows the influence of projector and predictor architecture on BYOL. We examine the effect of different depths for both the projector and predictor, as well as the effect of the projection size. We do not apply a ReLU activation nor a batch normalization on the final linear layer of our MLPs such that a depth of 1 corresponds to a linear layer. Using the default projector and predictor of depth 2 yields the best performance.

Proj. g_θ depth	Pred. q_θ depth	Top-1	Top-5
1	1	61.9	86.0
	2	65.0	86.8
	3	65.7	86.8
2	1	71.5	90.7
	2	**72.5**	**90.8**
	3	71.4	90.4
3	1	71.4	90.4
	2	72.1	90.5
	3	72.1	90.5

(a) Projector and predictor depth (i.e. the number of Linear layers).

Projector g_θ output dim	Top-1	Top-5
16	69.9±0.3	89.9
32	71.3	90.6
64	72.2	90.9
128	72.5	91.0
256	72.5	90.8
512	**72.6**	**91.0**

(b) Projection dimension.

Table 14: Effect of architectural settings where top-1 and top-5 accuracies are reported in %.

Table 15a shows the influence of the initial learning rate on BYOL. Note that the optimal value depends on the number of training epochs. Table 15b displays the influence of the weight decay on BYOL.

F.2 Batch size

We run a sweep over the batch size for both BYOL and our reproduction of SimCLR. As explained in Section 5, when reducing the batch size by a factor N, we average gradients over N consecutive steps and update the target network once every N steps. We report in Table 16, the performance of both our reproduction of SimCLR and BYOL for batch sizes between 4096 (BYOL and SimCLR default) down to 64. We observe that the performance of SimCLR deteriorates faster than the one of BYOL which stays mostly constant for batch sizes larger than 256. We believe that the performance at batch size 256 could match the performance of the large 4096 batch size with proper parameter tuning when accumulating the gradient. We think that the drop in performance at batch size 64 in table 16 is mainly related to the ill behaviour of batch normalization at low batch sizes [85].

Learning rate	Top-1	Top-5
0.01	34.8±3.0	60.8±3.2
0.1	65.0	87.0
0.2	71.7	90.6
0.3	**72.5**	**90.8**
0.4	72.3	90.6
0.5	71.5	90.1
1	69.4	89.2

(a) Base learning rate.

Weight decay coefficient	Top-1	Top-5
$1 \cdot 10^{-7}$	72.1	90.4
$5 \cdot 10^{-7}$	**72.6**	**91.0**
$1 \cdot 10^{-6}$	72.5	90.8
$5 \cdot 10^{-6}$	71.0±0.3	90.0
$1 \cdot 10^{-5}$	69.6±0.4	89.3

(b) Weight decay.

Table 15: Effect of learning rate and weight decay. We note that BYOL's performance is quite robust within a range of hyperparameters.

Batch size	Top-1		Top-5	
	BYOL (ours)	SimCLR (repro)	BYOL (ours)	SimCLR (repro)
4096	**72.5**	67.9	**90.8**	88.5
2048	72.4	67.8	90.7	88.5
1024	72.2	67.4	90.7	88.1
512	72.2	66.5	90.8	87.6
256	71.8	64.3±2.1	90.7	86.3±1.0
128	69.6±0.5	63.6	89.6	85.9
64	59.7±1.5	59.2±2.9	83.2±1.2	83.0±1.9

Table 16: Influence of the batch size.

F.3 Image augmentations

Table 17 compares the impact of individual image transformations on BYOL and SimCLR. BYOL is more resilient to changes of image augmentations across the board.

Image augmentation	Top-1		Top-5	
	BYOL (ours)	SimCLR (repro)	BYOL (ours)	SimCLR (repro)
Baseline	**72.5**	67.9	**90.8**	88.5
Remove flip	71.9	67.3	90.6	88.2
Remove blur	71.2	65.2	90.3	86.6
Remove color (jittering and grayscale)	63.4±0.7	45.7	85.3±0.5	70.6
Remove color jittering	71.8	63.7	90.7	85.9
Remove grayscale	70.3	61.9	89.8	84.1
Remove blur in $\mathcal{T}'$	72.4	67.5	90.8	88.4
Remove solarize in $\mathcal{T}'$	72.3	67.7	90.8	88.2
Remove blur and solarize in $\mathcal{T}'$	72.2	67.4	90.8	88.1
Crop only	59.4±0.3	40.3±0.3	82.4	64.8±0.4
Crop and flip only	60.1±0.3	40.2	83.0±0.3	64.8
Crop and color only	70.7	64.2	90.0	86.2
Crop and blur only	61.1±0.3	41.7	83.9	66.4

Table 17: Ablation on image transformations.

F.4 Details on the relation to contrastive methods

As mentioned in Section 5, the BYOL loss Eq. 2 can be derived from the InfoNCE loss

$$\text{InfoNCE}_\theta \triangleq \frac{2}{B}\sum_{i=1}^{B} S_\theta(v_i, v'_i) - \frac{2\alpha\cdot\beta}{B}\sum_{i=1}^{B}\ln\left(\sum_{j\neq i}\exp\frac{S_\theta(v_i,v_j)}{\alpha} + \sum_j \exp\frac{S_\theta(v_i,v'_j)}{\alpha}\right), \tag{5}$$

Loss weight β	Temperature α	Top-1	Top-5
0	0.1	72.5	90.8
0.1	0.01	72.2	90.7
	0.1	72.4	90.9
	0.3	**72.7**	91.0
	1	72.6	90.9
	3	72.5	90.9
	10	72.5	90.9
0.5	0.01	70.9	90.2
	0.1	72.0	90.8
	0.3	**72.7**	**91.2**
	1	72.7	91.1
	3	72.6	91.1
	10	72.5	91.0
1	0.01	$53.9_{\pm 0.5}$	$77.5_{\pm 0.5}$
	0.1	70.9	90.3
	0.3	**72.7**	91.1
	1	**72.7**	91.1
	3	72.6	91.0
	10	72.6	91.1

Table 18: Top-1 accuracy in % under linear evaluation protocol at 300 epochs of sweep over the temperature α and the dispersion term weight β when using a predictor and a target network.

with

$$S_\theta(u_1, u_2) \triangleq \frac{\langle \phi(u_1), \psi(u_2) \rangle}{\|\phi(u_1)\|_2 \cdot \|\psi(u_2)\|_2}. \tag{6}$$

In our ablation in Table 5b, we set the temperature α to its best value in the SimCLR setting (i.e., $\alpha = 0.1$). With this value, setting β to 1 (which adds negative examples), in the BYOL setting (i.e., with both a predictor and a target network) hurts the performances. In Table 18, we report results of a sweep over both the temperature α and the weight parameter β with a predictor and a target network where BYOL corresponds to $\beta = 0$. No run significantly outperforms BYOL and some values of α and β hurt the performance. While the best temperature for SimCLR (without the target network and a predictor) is 0.1, after adding a predictor and a target network the best temperature α is higher than 0.3.

Using a target network in the loss has two effects: stopping the gradient through the prediction targets and stabilizing the targets with averaging. Stopping the gradient through the target change the objective while averaging makes the target stable and stale. In Table 5b we only shows results of the ablation when either using the online network as the prediction target (and flowing the gradient through it) or with a target network (both stopping the gradient into the prediction targets and computing the prediction targets with a moving average of the online network). We shown in Table 5b that using a target network is beneficial but it has two distinct effects we would like to understand from which effect the improvement comes from. We report in Table 19 the results already in Table 5b but also when the prediction target is computed with a stop gradient of the online network (the gradient does not flow into the prediction targets). This shows that making the prediction targets stable and stale is the main cause of the improvement rather than the change in the objective due to the stop gradient.

F.5 SimCLR baseline of Section 5

The SimCLR baseline in Section 5 ($\beta = 1$, without predictor nor target network) is slightly different from the original one in [8]. First we multiply the original loss by 2α. For comparaison here is the original SimCLR loss,

$$\text{InfoNCE}_\theta \triangleq \frac{1}{B} \sum_{i=1}^{B} \frac{S_\theta(v_i, v'_i)}{\alpha} - \frac{1}{B} \sum_{i=1}^{B} \ln \left(\sum_{j \neq i} \exp \frac{S_\theta(v_i, v_j)}{\alpha} + \sum_j \exp \frac{S_\theta(v_i, v'_j)}{\alpha} \right). \tag{7}$$

Note that this multiplication by 2α matters as the LARS optimizer is not completely invariant with respect to the scale of the loss. Indeed, LARS applies a preconditioning to gradient updates on all weights, except for biases and batch normalization parameters. Updates on preconditioned weights are invariant by multiplicative scaling of the

Method	Predictor	Target parameters	β	Top-1
BYOL	✓	ξ	0	**72.5**
	✓	ξ	1	70.9
		ξ	1	70.7
	✓	$\text{sg}(\theta)$	1	70.2
SimCLR		θ	1	69.4
	✓	$\text{sg}(\theta)$	1	70.1
		$\text{sg}(\theta)$	1	69.2
	✓	θ	1	69.0
	✓	$\text{sg}(\theta)$	0	5.5
	✓	θ	0	0.3
		ξ	0	0.2
		$\text{sg}(\theta)$	0	0.1
		θ	0	0.1

Table 19: Top-1 accuracy in %, under linear evaluation protocol at 300 epochs, of intermediate variants between BYOL and SimCLR (with caveats discussed in Appendix F.5). sg means stop gradient.

loss. However, the bias and batch normalization parameter updates remain sensitive to multiplicative scaling of the loss.

We also increase the original SimCLR hidden and output size of the projector to respectively 4096 and 256. In our reproduction of SimCLR, these three combined changes improves the top-1 accuracy at 300 epochs from 67.9% (without the changes) to 69.2% (with the changes).

F.6 Ablation on the normalization in the loss function

Normalization	Top-1	Top-5
ℓ_2-norm	**72.5**	**90.8**
LayerNorm	$72.5_{\pm 0.4}$	90.1
No normalization	67.4	87.1
BatchNorm	65.3	85.3

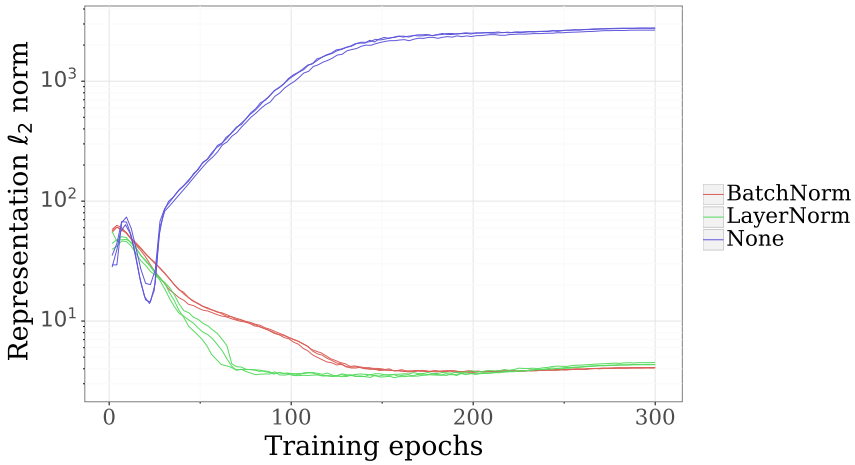
Table 20: Top-1 accuracy in % under linear evaluation protocol at 300 epochs for different normalizations in the loss.

BYOL minimizes a squared error between the ℓ_2-normalized prediction and target. We report results of BYOL at 300 epochs using different normalization function and no normalization at all. More precisely, given batch of prediction and targets in $\mathbb{R}^d$, $(p_i, t_i)_{i\leq B}$ with B the batch size, BYOL uses the loss function $\frac{1}{B}\sum_{i=1}^{B} \|n_{\ell_2}(p_i) - n_{\ell_2}(z_i)\|_2^2$ with $n_{\ell_2} : x \to x/\|x\|_2$. We run BYOL with other normalization functions: non-trainable batch-normalization and layer-normalization and no normalization. We divide the batch normalization and layer normalization by $\sqrt{d}$ to have a consistent scale with the ℓ_2-normalization. We report results in Table 20 where ℓ_2, LayerNorm, no normalization and BatchNorm respectively denote using n_{ℓ_2}, n_{BN}, n_{LN} and n_{ID} with

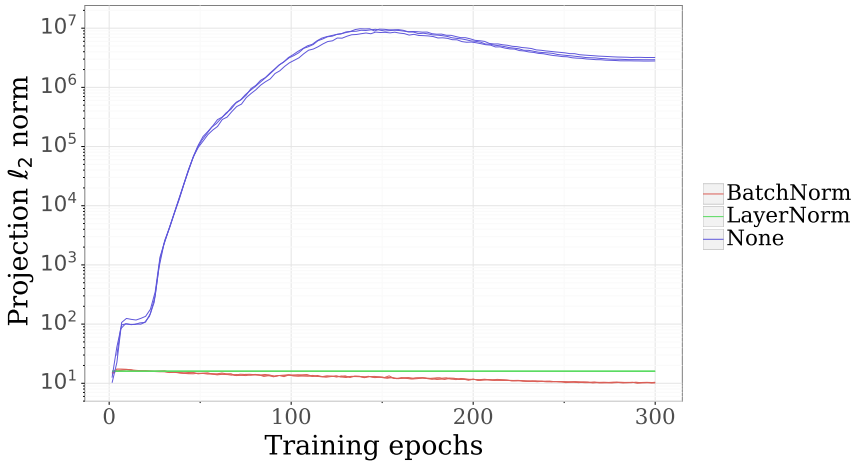
$$n^j_{\text{BN}\,i} : x \to \frac{x^j_i - \mu^j_{\text{BN}}(x)}{\sigma^j_{\text{BN}}(x)\cdot\sqrt{d}}, \quad n^j_{\text{LN}\,i} : x \to \frac{x^j_i - \mu_{\text{LN}\,i}(x)}{\sigma_{\text{LN}\,i}(x)\cdot\sqrt{d}}, \quad n_{\text{ID}} : x \to x,$$

$$\mu^j_{\text{BN}} : x \to \frac{1}{B}\sum_{i=1}^{B} x^j_i, \quad \sigma^j_{\text{BN}} : x \to \sqrt{\frac{1}{B}\sum_{i=1}^{B}\left(x^j_i\right)^2 - \mu^j_{\text{BN}}(x)^2},$$

$$\mu_{\text{LN}\,i} : x \to \frac{1}{d}\sum_{j=1}^{d} x^j_i, \quad \sigma_{\text{LN}\,i} : x \to \frac{\|x_i - \mu_{\text{LN}\,i}(x)\|_2}{\sqrt{d}}$$



(a) Representation ℓ_2-norm



(b) Projection ℓ_2-norm

Figure 6: Effect of normalization on the ℓ_2 norm of network outputs.

When using no normalization at all, the projection ℓ_2 norm rapidly increases during the first 100 epochs and stabilizes at around $3 \cdot 10^6$ as shown in Figure 6. Despite this behaviour, using no normalization still performs reasonably well (67.4%). The ℓ_2 normalization performs the best.

G BYOL pseudo-code in JAX

G.1 Hyper-parameters

```
HPS = dict(
    max_steps=int(1000. * 1281167 / 4096),  # 1000 epochs
    batch_size=4096,
    mlp_hidden_size=4096,
    projection_size=256,
    base_target_ema=4e-3,
    optimizer_config=dict(
        optimizer_name='lars',
        beta=0.9,
        trust_coef=1e-3,
        weight_decay=1.5e-6,
        # As in SimCLR and official implementation of LARS, we exclude bias
        # and batchnorm weight from the Lars adaptation and weightdecay.
        exclude_bias_from_adaption=True),
    learning_rate_schedule=dict(
        # The learning rate is linearly increase up to
        # its base value * batchisze / 256 after warmup_steps
        # global steps and then anneal with a cosine schedule.
        base_learning_rate=0.2,
        warmup_steps=int(10. * 1281167 / 4096),
        anneal_schedule='cosine'),
    batchnorm_kwargs=dict(
        decay_rate=0.9,
        eps=1e-5),
    seed=1337,
    )
```

G.2 Network definition

```
def network(inputs):
  """Build the encoder, projector and predictor."""
  embedding = ResNet(name='encoder', configuration='ResNetV1_50x1')(inputs)
  proj_out = MLP(name='projector')(embedding)
  pred_out = MLP(name='predictor')(proj_out)
  return dict(projection=proj_out, prediction=pred_out)


class MLP(hk.Module):
  """Multi Layer Perceptron, with normalization."""

  def __init(self, name):
    super().__init__(name=name)

  def __call__(self, inputs):
    out = hk.Linear(output_size=HPS['mlp_hidden_size'])(inputs)
    out = hk.BatchNorm(**HPS['batchnorm_kwargs'])(out)
    out = jax.nn.relu(out)
    out = hk.Linear(output_size=HPS['projection_size'])(out)
    return out

# For simplicity, we omit BatchNorm related states.
# In the actual code, we use hk.transform_with_state. The corresponding
# net_init function outputs both a params and a state variable,
# with state containing the moving averages computed by BatchNorm
net_init, net_apply = hk.transform(network)
```

G.3 Loss function

```
def loss_fn(online_params, target_params, image_1, image_2):
  """Compute BYOL's loss function.

  Args:
    online_params: parameters of the online network (the loss is later
      differentiated with respect to the online parameters).
    target_params: parameters of the target network.
    image_1: first transformation of the input image.
    image_2: second transformation of the input image.

  Returns:
    BYOL's loss function.
  """

  online_network_out_1 = net_apply(params=online_params, inputs=image_1)
  online_network_out_2 = net_apply(params=online_params, inputs=image_2)
  target_network_out_1 = net_apply(params=target_params, inputs=image_1)
  target_network_out_2 = net_apply(params=target_params, inputs=image_2)

  def regression_loss(x, y):
    norm_x, norm_y = jnp.linalg.norm(x), jnp.linalg.norm(y)
    return -2. * jnp.sum(x * y, axis=-1) / (norm_x * norm_y)

  # The stop_gradient is not necessary as we explicitly take the gradient with
  # respect to online parameters only. We leave it to indicate that gradients
  # are not backpropagated through the target network.
  loss = regression_loss(online_network_out_1['prediction'],
                         jax.lax.stop_gradient(target_network_out_2['projection']))
  loss += regression_loss(online_network_out_2['prediction'],
                          jax.lax.stop_gradient(target_network_out_1['projection']))
  return loss
```

G.4 Training loop

```
def main(dataset):
  """Main training loop."""

  rng = jax.random.PRNGKey(HPS['seed'])
  rng, rng_init = jax.random.split(rng, num=2)
  dataset = dataset.batch(HPS['batch_size'])
  dummy_input = dataset.next()
  byol_state = init(rng_init, dummy_input)

  for global_step in range(HPS['max_steps']):
    inputs = dataset.next()

    rng, rng1, rng2 = jax.random.split(rng, num=3)
    image_1 = simclr_augmentations(inputs, rng1, image_number=1)
    image_2 = simclr_augmentations(inputs, rng2, image_number=2)
    byol_state = update_fn(
        **byol_state,
        global_step=global_step,
        image_1=image_1,
        image_2=image_2)

  return byol_state['online_params']
```

G.5 Update function

```
optimizer = Optimizer(**HPS['optimizer_config'])


def update_fn(online_params, target_params, opt_state, global_step, image_1,
              image_2):
  """Update online and target parameters.

  Args:
    online_params: parameters of the online network (the loss is  differentiated
      with respect to the online parameters only).
    target_params: parameters of the target network.
    opt_state: state of the optimizer.
    global_step: current training step.
    image_1: first transformation of the input image.
    image_2: second transformation of the input image.

  Returns:
    Dict containing updated online parameters, target parameters and
    optimization state.
  """
  # update online network
  grad_fn = jax.grad(loss_fn, argnums=0)
  grads = grad_fn(online_params, target_params, image_1, image_2)
  lr = learning_rate(global_step, **HPS['learning_rate_schedule'])
  updates, opt_state = optimizer(lr).apply(grads, opt_state, online_params)
  online_params = optix.apply_updates(online_params, updates)

  # update target network
  tau = target_ema(global_step, base_ema=HPS['base_target_ema'])
  target_params = jax.tree_multimap(lambda x, y: x + (1 - tau) * (y - x),
                                    target_params, online_params)
  return dict(
      online_params=online_params,
      target_params=target_params,
      opt_state=opt_state)


def init(rng, dummy_input):
  """BYOL's state initialization.

  Args:
    rng: random number generator used to initialize parameters.
    dummy_input: a dummy image, used to compute intermediate outputs shapes.

  Returns:
    Dict containing initial online parameters, target parameters and
    optimization state.
  """
  online_params = net_init(rng, dummy_input)
  target_params = net_init(rng, dummy_input)
  opt_state = optimizer(0).init(online_params)
  return dict(
      online_params=online_params,
      target_params=target_params,
      opt_state=opt_state)
```