A. Data Augmentation Details

In our default pretraining setting (which is used to train our best models), we utilize random crop (with resize and random flip), random color distortion, and random Gaussian blur as the data augmentations. The details of these three augmentations are provided below.

Random crop and resize to 224x224 We use standard Inception-style random cropping (Szegedy et al., 2015). The crop of random size (uniform from 0.08 to 1.0 in area) of the original size and a random aspect ratio (default: of 3/4 to 4/3) of the original aspect ratio is made. This crop is finally resized to the original size. This has been implemented in Tensorflow as "slim.preprocessing.inception_preprocessing.distorted_bounding_box_crop", or in Pytorch as "torchvision.transforms.RandomResizedCrop". Additionally, the random crop (with resize) is always followed by a random horizontal/left-to-right flip with 50% probability. This is helpful but not essential. By removing this from our default augmentation policy, the top-1 linear evaluation drops from 64.5% to 63.4% for our ResNet-50 model trained in 100 epochs.

Color distortion Color distortion is composed by color jittering and color dropping. We find stronger color jittering usually helps, so we set a strength parameter.

A pseudo-code for color distortion using TensorFlow is as follows.

```
import tensorflow as tf
def color_distortion(image, s=1.0):
    # image is a tensor with value range in [0, 1].
    # s is the strength of color distortion.

    def color_jitter(x):
        # one can also shuffle the order of following augmentations
        # each time they are applied.
        x = tf.image.random_brightness(x, max_delta=0.8*s)
        x = tf.image.random_contrast(x, lower=1-0.8*s, upper=1+0.8*s)
        x = tf.image.random_saturation(x, lower=1-0.8*s, upper=1+0.8*s)
        x = tf.image.random_hue(x, max_delta=0.2*s)
        x = tf.clip_by_value(x, 0, 1)
        return x

    def color_drop(x):
        image = tf.image.rgb_to_grayscale(image)
        image = tf.tile(image, [1, 1, 3])

    # randomly apply transformation with probability p.
    image = random_apply(color_jitter, image, p=0.8)
    image = random_apply(color_drop, image, p=0.2)
    return image
```

A pseudo-code for color distortion using Pytorch is as follows [12].

```
from torchvision import transforms
def get_color_distortion(s=1.0):
    # s is the strength of color distortion.
    color_jitter = transforms.ColorJitter(0.8*s, 0.8*s, 0.8*s, 0.2*s)
    rnd_color_jitter = transforms.RandomApply([color_jitter], p=0.8)
    rnd_gray = transforms.RandomGrayscale(p=0.2)
    color_distort = transforms.Compose([
        rnd_color_jitter,
        rnd_gray])
```

[12] Our code and results are based on Tensorflow, the Pytorch code here is a reference.

```
    return color_distort
```

Gaussian blur This augmentation is in our default policy. We find it helpful, as it improves our ResNet-50 trained for 100 epochs from 63.2% to 64.5%. We blur the image 50% of the time using a Gaussian kernel. We randomly sample $\sigma \in [0.1, 2.0]$, and the kernel size is set to be 10% of the image height/width.

B. Additional Experimental Results

B.1. Batch Size and Training Steps

Figure B.1 shows the top-5 accuracy on linear evaluation when trained with different batch sizes and training epochs. The conclusion is very similar to top-1 accuracy shown before, except that the differences between different batch sizes and training steps seems slightly smaller here.

In both Figure 9 and Figure B.1, we use a linear scaling of learning rate similar to (Goyal et al., 2017) when training with different batch sizes. Although linear learning rate scaling is popular with SGD/Momentum optimizer, we find a square root learning rate scaling is more desirable with LARS optimizer. With square root learning rate scaling, we have $\text{LearningRate} = 0.075 \times \sqrt{\text{BatchSize}}$, instead of $\text{LearningRate} = 0.3 \times \text{BatchSize}/256$ in the linear scaling case, but the learning rate is the same under both scaling methods when batch size of 4096 (our default batch size). A comparison is presented in Table B.1, where we observe that square root learning rate scaling improves the performance for models trained with small batch sizes and in smaller number of epochs.

Batch size \ Epochs	100	200	400	800
256	57.5 / **62.8**	61.9 / **64.3**	64.7 / **65.7**	66.6 / 66.5
512	60.7 / **63.8**	64.0 / **65.6**	66.2 / 66.7	67.8 / 67.4
1024	62.8 / **64.3**	65.3 / **66.1**	67.2 / 67.2	68.5 / 68.3
2048	64.0 / **64.7**	66.1 / **66.8**	68.1 / 67.9	68.9 / 68.8
4096	64.6 / 64.5	66.5 / 66.8	68.2 / 68.0	68.9 / 69.1
8192	64.8 / 64.8	66.6 / 67.0	67.8 / 68.3	69.0 / 69.1

Table B.1. Linear evaluation (top-1) under different batch sizes and training epochs. On the left side of slash sign are models trained with linear LR scaling, and on the right are models trained with square root LR scaling. The result is bolded if it is more than 0.5% better. Square root LR scaling works better for smaller batch size trained in fewer epochs (with LARS optimizer).

We also train with larger batch size (up to 32K) and longer (up to 3200 epochs), with the square root learning rate scaling. A shown in Figure B.2, *the performance seems to saturate with a batch size of 8192, while training longer can still significantly improve the performance*.

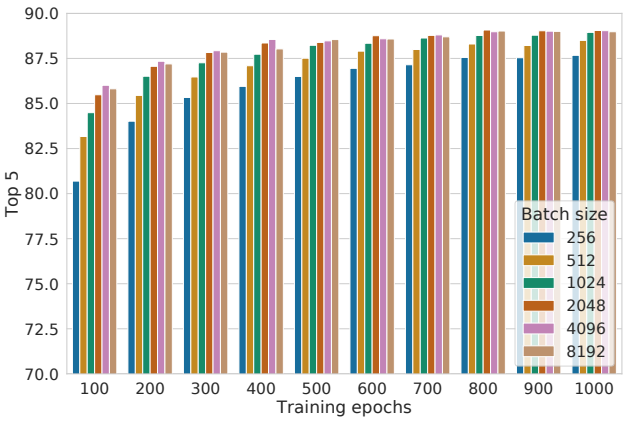


Figure B.1. Linear evaluation (top-5) of ResNet-50 trained with different batch sizes and epochs. Each bar is a single run from scratch. See Figure 9 for top-1 accuracy.

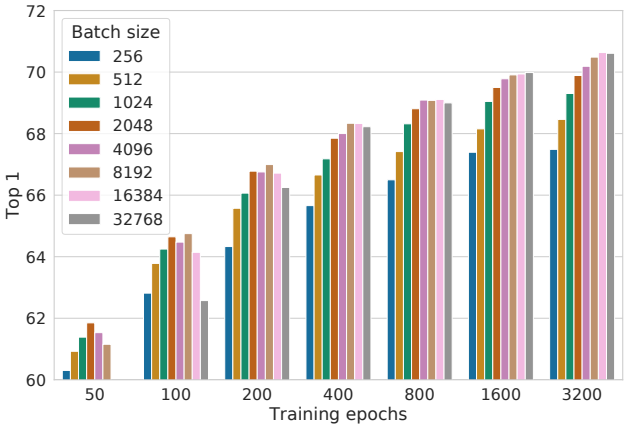


Figure B.2. Linear evaluation (top-1) of ResNet-50 trained with different batch sizes and *longer* epochs. Here a *square root* learning rate, instead of a linear one, is utilized.

B.2. Broader composition of data augmentations further improves performance

Our best results in the main text (Table 6 and 7) can be further improved when expanding the default augmentation policy to include the following: (1) Sobel filtering, (2) additional color distortion (equalize, solarize), and (3) motion blur. For linear evaluation protocol, the ResNet-50 models ($1\times, 2\times, 4\times$) trained with broader data augmentations achieve 70.0 (+0.7), 74.4 (+0.2), 76.8 (+0.3), respectively.

Table B.2 shows ImageNet accuracy obtained by fine-tuning the SimCLR model (see Appendix B.5 for the details of fine-tuning procedure). Interestingly, when fine-tuned on full (100%) ImageNet training set, our ResNet ($4\times$) model achieves 80.4% top-1 / 95.4% top-5 [13], which is significantly better than that (78.4% top-1 / 94.2% top-5) of training from scratch using the same set of augmentations (i.e. random crop and horizontal flip). For ResNet-50 ($2\times$), fine-tuning our pre-trained ResNet-50 ($2\times$) is also better than training from scratch (77.8% top-1 / 93.9% top-5). There is no improvement from fine-tuning for ResNet-50.

Architecture	Label fraction					
	1%		10%		100%	
	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
ResNet-50	49.4	76.6	66.1	88.1	76.0	93.1
ResNet-50 ($2\times$)	59.4	83.7	71.8	91.2	79.1	94.8
ResNet-50 ($4\times$)	64.1	86.6	74.8	92.8	80.4	95.4

Table B.2. Classification accuracy obtained by fine-tuning the SimCLR (which is pretrained with broader data augmentations) on 1%, 10% and full of ImageNet. As a reference, our ResNet-50 ($4\times$) trained from scratch on 100% labels achieves 78.4% top-1 / 94.2% top-5.

B.3. Effects of Longer Training for Supervised Models

Here we perform experiments to see how training steps and stronger data augmentation affect supervised training. We test ResNet-50 and ResNet-50 ($4\times$) under the same set of data augmentations (random crops, color distortion, 50% Gaussian blur) as used in our unsupervised models. Figure B.3 shows the top-1 accuracy. We observe that there is no significant benefit from training supervised models longer on ImageNet. Stronger data augmentation slightly improves the accuracy of ResNet-50 ($4\times$) but does not help on ResNet-50. When stronger data augmentation is applied, ResNet-50 generally requires longer training (e.g. 500 epochs [14]) to obtain the optimal result, while ResNet-50 ($4\times$) does not benefit from longer training.

Model	Training epochs	Top 1		
		Crop	+Color	+Color+Blur
ResNet-50	90	76.5	75.6	75.3
	500	76.2	76.5	76.7
	1000	75.8	75.2	76.4
ResNet-50 ($4\times$)	90	78.4	78.9	78.7
	500	78.3	78.4	78.5
	1000	77.9	78.2	78.3

Table B.3. Top-1 accuracy of supervised models trained longer under various data augmentation procedures (from the same set of data augmentations for contrastive learning).

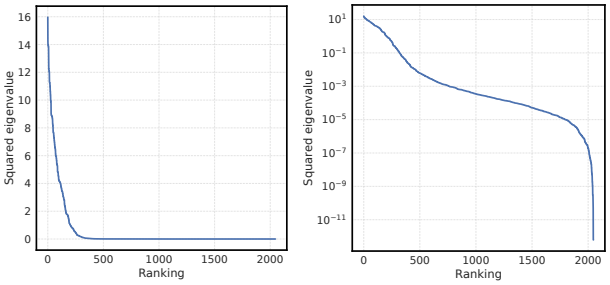
B.4. Understanding The Non-Linear Projection Head

Figure B.3 shows the eigenvalue distribution of linear projection matrix $W \in R^{2048\times2048}$ used to compute $\boldsymbol{z} = W\boldsymbol{h}$. This matrix has relatively few large eigenvalues, indicating that it is approximately low-rank.

Figure B.4 shows t-SNE (Maaten & Hinton, 2008) visualizations of $\boldsymbol{h}$ and $\boldsymbol{z} = g(\boldsymbol{h})$ for randomly selected 10 classes by our best ResNet-50 (top-1 linear evaluation 69.3%). Classes represented by $\boldsymbol{h}$ are better separated compared to $\boldsymbol{z}$.

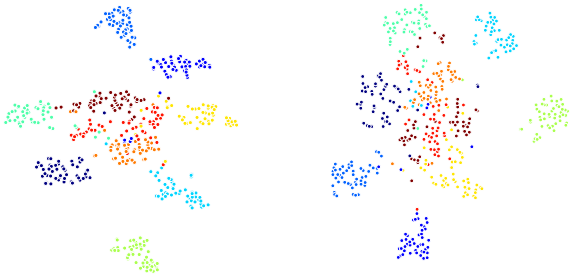
[13] It is 80.1% top-1 / 95.2% top-5 without broader augmentations for pretraining SimCLR.

[14] With AutoAugment (Cubuk et al., 2019), optimal test accuracy can be achieved between 900 and 500 epochs.



(a) Y-axis in uniform scale. (b) Y-axis in log scale.

Figure B.3. Squared real eigenvalue distribution of linear projection matrix $W \in R^{2048\times2048}$ used to compute $g(\boldsymbol{h}) = W\boldsymbol{h}$.



(a) $\boldsymbol{h}$ (b) $\boldsymbol{z} = g(\boldsymbol{h})$

Figure B.4. t-SNE visualizations of hidden vectors of images from a randomly selected 10 classes in the validation set.

B.5. Semi-supervised Learning via Fine-Tuning

Fine-tuning Procedure We fine-tune using the Nesterov momentum optimizer with a batch size of 4096, momentum of 0.9, and a learning rate of 0.8 (following $\text{LearningRate} = 0.05 \times \text{BatchSize}/256$) without warmup. Only random cropping (with random left-to-right flipping and resizing to 224x224) is used for preprocessing. We do not use any regularization (including weight decay). For 1% labeled data we fine-tune for 60 epochs, and for 10% labeled data we fine-tune for 30 epochs. For the inference, we resize the given image to 256x256, and take a single center crop of 224x224.

Table B.4 shows the comparisons of top-1 accuracy for different methods for semi-supervised learning. Our models significantly improve state-of-the-art.

Method	Architecture	Label fraction (Top 1) 1%	10%
Supervised baseline	ResNet-50	25.4	56.4
Methods using label-propagation:			
UDA (w. RandAug)	ResNet-50	-	68.8
FixMatch (w. RandAug)	ResNet-50	-	71.5
S4L (Rot+VAT+Ent. Min.)	ResNet-50 (4×)	-	73.2
Methods using self-supervised representation learning only:			
CPC v2	ResNet-161(∗)	52.7	73.1
SimCLR (ours)	ResNet-50	48.3	65.6
SimCLR (ours)	ResNet-50 (2×)	58.5	71.7
SimCLR (ours)	ResNet-50 (4×)	**63.0**	**74.4**

Table B.4. ImageNet top-1 accuracy of models trained with few labels. See Table 7 for top-5 accuracy.

B.6. Linear Evaluation

For linear evaluation, we follow similar procedure as fine-tuning (described in Appendix B.5), except that a larger learning rate of 1.6 (following $\text{LearningRate} = 0.1 \times \text{BatchSize}/256$) and longer training of 90 epochs. Alternatively, using LARS optimizer with the pretraining hyper-parameters also yield similar results. Furthermore, we find that attaching the linear classifier on top of the base encoder (with a stop_gradient on the input to linear classifier to prevent the label information from influencing the encoder) and train them simultaneously during the pretraining achieves similar performance.

B.7. Correlation Between Linear Evaluation and Fine-Tuning

Here we study the correlation between linear evaluation and fine-tuning under different settings of training step and network architecture.

Figure B.5 shows linear evaluation versus fine-tuning when training epochs of a ResNet-50 (using batch size of 4096) are varied from 50 to 3200 as in Figure B.2. While they are almost linearly correlated, it seems fine-tuning on a small fraction

of labels benefits more from training longer.

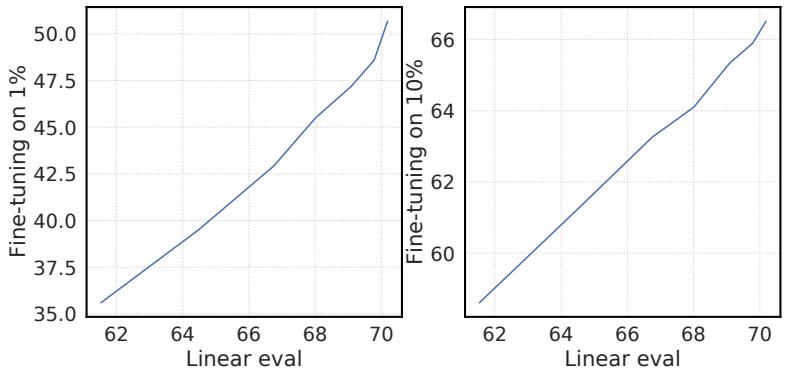


Figure B.5. Top-1 accuracy of models trained in different epochs (from Figure B.2), under linear evaluation and fine-tuning.

Figure B.6 shows shows linear evaluation versus fine-tuning for different architectures of choice.

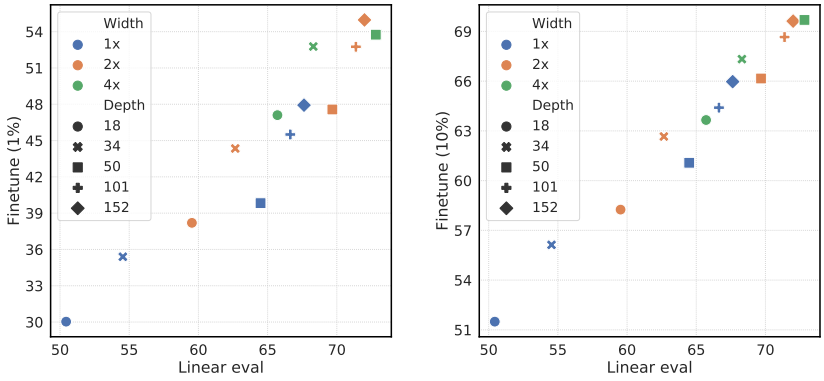


Figure B.6. Top-1 accuracy of different architectures under linear evaluation and fine-tuning.

B.8. Transfer Learning

We evaluated the performance of our self-supervised representation for transfer learning in two settings: linear evaluation, where a logistic regression classifier is trained to classify a new dataset based on the self-supervised representation learned on ImageNet, and fine-tuning, where we allow all weights to vary during training. In both cases, we follow the approach described by Kornblith et al. (2019), although our preprocessing differs slightly.

B.8.1. Methods

Datasets We investigated transfer learning performance on the Food-101 dataset (Bossard et al., 2014), CIFAR-10 and CIFAR-100 (Krizhevsky & Hinton, 2009), Birdsnap (Berg et al., 2014), the SUN397 scene dataset (Xiao et al., 2010), Stanford Cars (Krause et al., 2013), FGVC Aircraft (Maji et al., 2013), the PASCAL VOC 2007 classification task (Everingham et al., 2010), the Describable Textures Dataset (DTD) (Cimpoi et al., 2014), Oxford-IIIT Pets (Parkhi et al., 2012), Caltech-101 (Fei-Fei et al., 2004), and Oxford 102 Flowers (Nilsback & Zisserman, 2008). We follow the evaluation protocols in the papers introducing these datasets, *i.e.*, we report top-1 accuracy for Food-101, CIFAR-10, CIFAR-100, Birdsnap, SUN397, Stanford Cars, and DTD; mean per-class accuracy for FGVC Aircraft, Oxford-IIIT Pets, Caltech-101, and Oxford 102 Flowers; and the 11-point mAP metric as defined in Everingham et al. (2010) for PASCAL VOC 2007. For DTD and SUN397, the dataset creators defined multiple train/test splits; we report results only for the first split. Caltech-101 defines no train/test split, so we randomly chose 30 images per class and test on the remainder, for fair comparison with previous work (Donahue et al., 2014; Simonyan & Zisserman, 2014).

We used the validation sets specified by the dataset creators to select hyperparameters for FGVC Aircraft, PASCAL VOC

2007, DTD, and Oxford 102 Flowers. For other datasets, we held out a subset of the training set for validation while performing hyperparameter tuning. After selecting the optimal hyperparameters on the validation set, we retrained the model using the selected parameters using all training and validation images. We report accuracy on the test set.

Transfer Learning via a Linear Classifier We trained an ℓ_2-regularized multinomial logistic regression classifier on features extracted from the frozen pretrained network. We used L-BFGS to optimize the softmax cross-entropy objective and we did not apply data augmentation. As preprocessing, all images were resized to 224 pixels along the shorter side using bicubic resampling, after which we took a 224×224 center crop. We selected the ℓ_2 regularization parameter from a range of 45 logarithmically spaced values between 10^{-6} and 10^{5}.

Transfer Learning via Fine-Tuning We fine-tuned the entire network using the weights of the pretrained network as initialization. We trained for 20,000 steps at a batch size of 256 using SGD with Nesterov momentum with a momentum parameter 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. As data augmentation during fine-tuning, we performed only random crops with resize and flips; in contrast to pretraining, we did not perform color augmentation or blurring. At test time, we resized images to 256 pixels along the shorter side and took a 224×224 center crop. (Additional accuracy improvements may be possible with further optimization of data augmentation, particularly on the CIFAR-10 and CIFAR-100 datasets.) We selected the learning rate and weight decay, with a grid of 7 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. We divide these values of weight decay by the learning rate.

Training from Random Initialization We trained the network from random initialization using the same procedure as for fine-tuning, but for longer, and with an altered hyperparameter grid. We chose hyperparameters from a grid of 7 logarithmically spaced learning rates between 0.001 and 1.0 and 8 logarithmically spaced values of weight decay between 10^{-5} and $10^{-1.5}$. Importantly, our random initialization baselines are trained for 40,000 steps, which is sufficiently long to achieve near-maximal accuracy, as demonstrated in Figure 8 of Kornblith et al. (2019).

On Birdsnap, there are no statistically significant differences among methods, and on Food-101, Stanford Cars, and FGVC Aircraft datasets, fine-tuning provides only a small advantage over training from random initialization. However, on the remaining 8 datasets, pretraining has clear advantages.

Supervised Baselines We compare against architecturally identical ResNet models trained on ImageNet with standard cross-entropy loss. These models are trained with the same data augmentation as our self-supervised models (crops, strong color augmentation, and blur) and are also trained for 1000 epochs. We found that, although stronger data augmentation and longer training time do not benefit accuracy on ImageNet, these models performed significantly better than a supervised baseline trained for 90 epochs and ordinary data augmentation for linear evaluation on a subset of transfer datasets. The supervised ResNet-50 baseline achieves 76.3% top-1 accuracy on ImageNet, vs. 69.3% for the self-supervised counterpart, while the ResNet-50 ($4\times$) baseline achieves 78.3%, vs. 76.5% for the self-supervised model.

Statistical Significance Testing We test for the significance of differences between model with a permutation test. Given predictions of two models, we generate 100,000 samples from the null distribution by randomly exchanging predictions for each example and computing the difference in accuracy after performing this randomization. We then compute the percentage of samples from the null distribution that are more extreme than the observed difference in predictions. For top-1 accuracy, this procedure yields the same result as the exact McNemar test. The assumption of exchangeability under the null hypothesis is also valid for mean per-class accuracy, but not when computing average precision curves. Thus, we perform significance testing for a difference in accuracy on VOC 2007 rather than a difference in mAP. A caveat of this procedure is that it does not consider run-to-run variability when training the models, only variability arising from using a finite sample of images for evaluation.

B.8.2. Results with Standard ResNet

The ResNet-50 ($4\times$) results shown in Table 8 of the text show no clear advantage to the supervised or self-supervised models. With the narrower ResNet-50 architecture, however, supervised learning maintains a clear advantage over self-supervised learning. The supervised ResNet-50 model outperforms the self-supervised model on all datasets with linear evaluation, and most (10 of 12) datasets with fine-tuning. The weaker performance of the ResNet model compared to the ResNet ($4\times$)

	Food	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	VOC2007	DTD	Pets	Caltech-101	Flowers
Linear evaluation:												
SimCLR (ours)	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	**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:												
SimCLR (ours)	**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	**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	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 B.5. Comparison of transfer learning performance of our self-supervised approach with supervised baselines across 12 natural image datasets, using ImageNet-pretrained ResNet models. See also Figure 8 for results with the ResNet $(4\times)$ architecture.

model may relate to the accuracy gap between the supervised and self-supervised models on ImageNet. The self-supervised ResNet gets 69.3% top-1 accuracy, 6.8% worse than the supervised model in absolute terms, whereas the self-supervised ResNet $(4\times)$ model gets 76.5%, which is only 1.8% worse than the supervised model.

B.9. CIFAR-10

While we focus on using ImageNet as the main dataset for pretraining our unsupervised model, our method also works with other datasets. We demonstrate it by testing on CIFAR-10 as follows.

Setup As our goal is not to optimize CIFAR-10 performance, but rather to provide further confirmation of our observations on ImageNet, we use the same architecture (ResNet-50) for CIFAR-10 experiments. Because CIFAR-10 images are much smaller than ImageNet images, we replace the first 7x7 Conv of stride 2 with 3x3 Conv of stride 1, and also remove the first max pooling operation. For data augmentation, we use the same Inception crop (flip and resize to 32x32) as ImageNet,[15] and color distortion (strength=0.5), leaving out Gaussian blur. We pretrain with learning rate in $\{0.5, 1.0, 1.5\}$, temperature in $\{0.1, 0.5, 1.0\}$, and batch size in $\{256, 512, 1024, 2048, 4096\}$. The rest of the settings (including optimizer, weight decay, etc.) are the same as our ImageNet training.

Our best model trained with batch size 1024 can achieve a linear evaluation accuracy of 94.0%, compared to 95.1% from the supervised baseline using the same architecture and batch size. The best self-supervised model that reports linear evaluation result on CIFAR-10 is AMDIM (Bachman et al., 2019), which achieves 91.2% with a model $25\times$ larger than ours. We note that our model can be improved by incorporating extra data augmentations as well as using a more suitable base network.

Performance under different batch sizes and training steps Figure B.7 shows the linear evaluation performance under different batch sizes and training steps. The results are consistent with our observations on ImageNet, although the largest batch size of 4096 seems to cause a small degradation in performance on CIFAR-10.

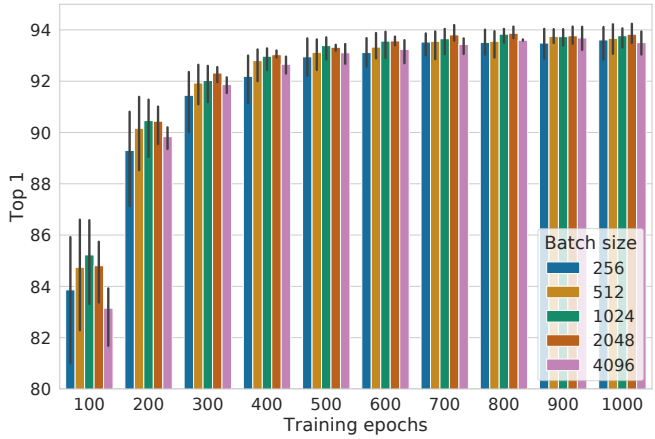
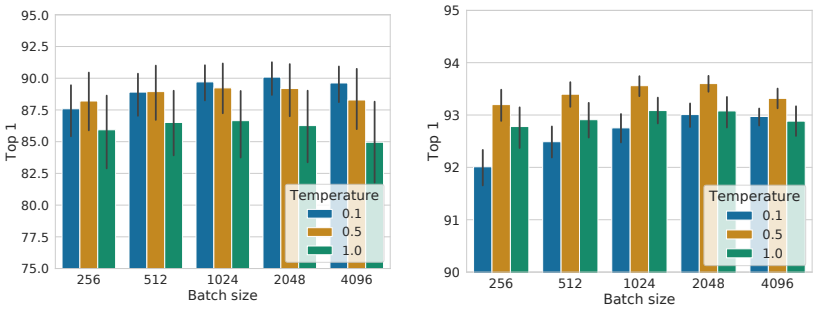


Figure B.7. Linear evaluation of ResNet-50 (with adjusted stem) trained with different batch size and epochs on CIFAR-10 dataset. Each bar is averaged over 3 runs with different learning rates (0.5, 1.0, 1.5) and temperature $\tau = 0.5$. Error bar denotes standard deviation.

[15] It is worth noting that, although CIFAR-10 images are much smaller than ImageNet images and image size does not differ among examples, cropping with resizing is still a very effective augmentation for contrastive learning.

Optimal temperature under different batch sizes Figure B.8 shows the linear evaluation of model trained with three different temperatures under various batch sizes. We find that when training to convergence (e.g. training epochs > 300), the optimal temperature in $\{0.1, 0.5, 1.0\}$ is 0.5 and seems consistent regardless of the batch sizes. However, the performance with $\tau = 0.1$ improves as batch size increases, which may suggest a small shift of optimal temperature towards 0.1.



(a) Training epochs ≤ 300 (b) Training epochs > 300

Figure B.8. Linear evaluation of the model (ResNet-50) trained with three temperatures on different batch sizes on CIFAR-10. Each bar is averaged over multiple runs with different learning rates and total train epochs. Error bar denotes standard deviation.

B.10. Tuning For Other Loss Functions

The learning rate that works best for NT-Xent loss may not be a good learning rate for other loss functions. To ensure a fair comparison, we also tune hyperparameters for both margin loss and logistic loss. Specifically, we tune learning rate in $\{0.01, 0.1, 0.3, 0.5, 1.0\}$ for both loss functions. We further tune the margin in $\{0, 0.4, 0.8, 1.6\}$ for margin loss, the temperature in $\{0.1, 0.2, 0.5, 1.0\}$ for logistic loss. For simplicity, we only consider the negatives from one augmentation view (instead of both sides), which slightly impairs performance but ensures fair comparison.

C. Further Comparison to Related Methods

As we have noted in the main text, most individual components of SimCLR have appeared in previous work, and the improved performance is a result of a combination of these design choices. Table C.1 provides a high-level comparison of the design choices of our method with those of previous methods. Compared with previous work, our design choices are generally simpler.

Model	Data Augmentation	Base Encoder	Projection Head	Loss	Batch Size	Train Epochs
CPC v2	Custom	ResNet-161 (modified)	PixelCNN	Xent	$512^{\#}$	$\sim$200
AMDIM	Fast AutoAug.	Custom ResNet	Non-linear MLP	Xent w/ clip,reg	$1008^{\#}$	150
CMC	Fast AutoAug.	ResNet-50 (2×, L+ab)	Linear layer	Xent w/ ℓ_2, τ	156^{*}	280
MoCo	Crop+color	ResNet-50 (4×)	Linear layer	Xent w/ ℓ_2, τ	256^{*}	200
PIRL	Crop+color	ResNet-50 (2×)	Linear layer	Xent w/ ℓ_2, τ	1024^{*}	800
SimCLR	Crop+color+blur	ResNet-50 (4×)	Non-linear MLP	Xent w/ ℓ_2, τ	4096	1000

Table C.1. A high-level comparison of design choices and training setup (for best result on ImageNet) for each method. Note that descriptions provided here are general; even when they match for two methods, formulations and implementations may differ (e.g. for color augmentation). Refer to the original papers for more details. $^{\#}$Examples are split into multiple patches, which enlarges the effective batch size. *A memory bank is employed.

In below, we provide an in-depth comparison of our method to the recently proposed contrastive representation learning methods:

- DIM/AMDIM (Hjelm et al., 2018; Bachman et al., 2019) achieve global-to-local/local-to-neighbor prediction by predicting the middle layer of ConvNet. The ConvNet is a ResNet that has bewen modified to place significant constraints on the receptive fields of the network (e.g. replacing many 3x3 Convs with 1x1 Convs). In our framework, we decouple the prediction task and encoder architecture, by random cropping (with resizing) and using the final

representations of two augmented views for prediction, so we can use standard and more powerful ResNets. Our NT-Xent loss function leverages normalization and temperature to restrict the range of similarity scores, whereas they use a tanh function with regularization. We use a simpler data augmentation policy, while they use FastAutoAugment for their best result.

- CPC v1 and v2 (Oord et al., 2018; Hénaff et al., 2019) define the context prediction task using a deterministic strategy to split examples into patches, and a context aggregation network (a PixelCNN) to aggregate these patches. The base encoder network sees only patches, which are considerably smaller than the original image. We decouple the prediction task and the encoder architecture, so we do not require a context aggregation network, and our encoder can look at the images of wider spectrum of resolutions. In addition, we use the NT-Xent loss function, which leverages normalization and temperature, whereas they use an unnormalized cross-entropy-based objective. We use simpler data augmentation.

- InstDisc, MoCo, PIRL (Wu et al., 2018; He et al., 2019; Misra & van der Maaten, 2019) generalize the Exemplar approach originally proposed by Dosovitskiy et al. (2014) and leverage an explicit memory bank. We do not use a memory bank; we find that, with a larger batch size, in-batch negative example sampling suffices. We also utilize a nonlinear projection head, and use the representation before the projection head. Although we use similar types of augmentations (e.g., random crop and color distortion), we expect specific parameters may be different.

- CMC (Tian et al., 2019) uses a separated network for each view, while we simply use a single network shared for all randomly augmented views. The data augmentation, projection head and loss function are also different. We use larger batch size instead of a memory bank.

- Whereas Ye et al. (2019) maximize similarity between augmented and unaugmented copies of the same image, we apply data augmentation symmetrically to both branches of our framework (Figure 2). We also apply a nonlinear projection on the output of base feature network, and use the representation before projection network, whereas Ye et al. (2019) use the linearly projected final hidden vector as the representation. When training with large batch sizes using multiple accelerators, we use global BN to avoid shortcuts that can greatly decrease representation quality.