Appendix

A. Implementation Details

Sampling the support mini-batches. In each iteration, PAWS randomly samples a small support mini-batch from the set of available labeled samples to compute the unsupervised consistency loss. Specifically, these support samples are used to determine the soft pseudo-labels for the unlabeled image views. To construct the support mini-batch in each iteration, we first sample a subset of classes, and then sample an equal number of images from each sampled class. Notably, we sample images *with replacement*. Therefore, while images in the same support mini-batch in a given iteration are always unique, some of the images may be re-sampled in the subsequent iteration's support mini-batch. This decision was made to simplify the implementation, although it is possible that epoch-based sampling of the support mini-batches (i.e., iterating through labeled samples with random reshuffling) could lead to improved performance.

Projection & prediction heads. The projection head is a 3-layer MLP with ReLU activations, consisting of three fully-connected layers of dimension 2048, and Batch Normalization applied to the hidden layers. The prediction head is a 2-layer MLP with ReLU activations, consisting of two fully-connected layers. The hidden layer has dimension 512, and the output layer has dimension 2048. Batch Normalization is applied to the input of the prediction head as well as to the hidden layer. The architectures of these projection and prediction heads are similar to those used in previous works on self-supervised learning [4, 32, 1].

Fine-tuning details. Following [1], we fine-tune a linear classifier from the first layer of the projection head in the pre-trained encoder f_θ, and initialize the weights of the linear classifier to zero. Specifically, we simultaneously fine-tune the encoder/classifier weights by optimizing a supervised cross-entropy loss on the small set of available labeled samples. We do not employ weight-decay during fine-tuning, and only make use of basic data augmentations (random cropping and random horizontal flipping). Following the experimental protocol of BYOL [4], we sweep the learning rate $\{0.01, 0.02, 0.05, 0.1, 0.2\}$ and the number of epochs $\{30, 50\}$. Similarly to BYOL, to avoid performing parameter selection on the ImageNet validation set (used for reporting), we use a local validation set (12000 images from the ImageNet train set). Optimization is conducted using SGD with Nesterov momentum. We use a momentum value of 0.9 and a batch size of 1024. All results are reported using a single center-crop.

Nearest neighbours classifier. We also report additional results without fine-tuning the encoder. Specifically, the PAWS-NN results in Table 1 are reported by directly applying a soft nearest neighbours classifier to the pre-trained representations. To determine a class prediction for an image $\mathbf{x}$, we compare its representation, $z = f_\theta(\mathbf{x})$, to the representations of the available labeled training samples, $\mathbf{z}_{\mathcal{S}} \in \mathbb{R}^{M \times d}$, and subsequently choose the class label with the highest probability under the similarity classifier; i.e., $\text{argmax}_{k \in [1000]} \left[\pi_d\left(z, \mathbf{z}_{\mathcal{S}}\right)\right]_k$. All results are reported using a single center-crop. Figure 4

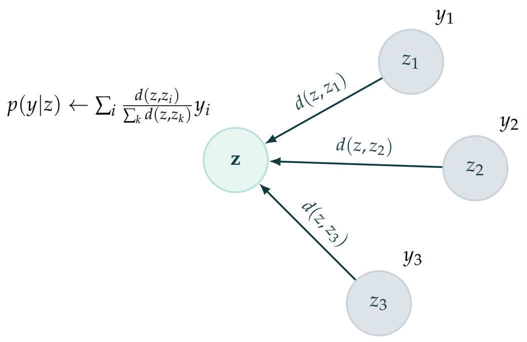


Figure 4: **Soft Nearest Neighbours classifier** π_d. For a K-way classification problem, and a scalar-valued similarity function $d(\cdot,\cdot) \geq 0$, the similarity classifier assigns a soft pseudo-label $y \in [0,1]^K$ to a representation z by measuring its similarity to a set of labeled representations $\{z_i\}_i$ with class labels $\{y_i \in [0,1]^K\}_i$. The soft pseudo-label y is a weighted average of the labels $\{y_i\}_i$, with labels corresponding to more similar representations assigned larger weights.

provides a schematic of the nearest neighbours classifier in an illustrative example with only only three labeled training images.

Momentum. When using momentum in our experiments, unless otherwise specified, we implicitly refer to classical momentum, commonly referred to as heavy-ball or Polyak momentum, given by

$$\begin{aligned} v_{t+1} &= \beta v_t - \eta_t \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \nabla_\theta \ell(x, \theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1}, \end{aligned} \tag{2}$$

where $\beta \geq 0$ is the momentum parameter and $\eta_t \geq 0$ is the learning rate. The model parameters are denoted by θ and the velocity buffer is denoted by v. Note that in some deep learning frameworks, such as PyTorch and Tensorflow, the update is instead written

$$\begin{aligned} v_{t+1} &= \beta v_t + \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \nabla_\theta \ell(x, \theta_t) \\ \theta_{t+1} &= \theta_t - \eta_t v_{t+1}. \end{aligned} \tag{3}$$

Specifically, in eq. (3), the learning rate is not incorporated into the velocity buffer update. Thus, under a trivial re-parameterization, the eq. (3) implementation can be interpreted as classical momentum with a time-varying momentum schedule, given by $\{\beta \frac{\eta_t}{\eta_{t-1}}\}_{t>0}$. Thus, training with learning rate warmup can result in momentum values > 1 during the warmup phase, leading to instability early on in training. Additionally, note that training using the implementation of momentum SGD in eq. (3) with an adaptive learning rate, e.g., as prescribed by the LARS optimizer, can lead to drastically different momentum values at consecutive iterations. However, it is worth pointing out that the LARS optimizer provided in the popular NVIDIA APEX package wraps around the optimizer, and applies learning-rate adaptation by directly scaling the gradient before the optimization step. Therefore, using the NVIDIA APEX implementation of LARS with the PyTorch implementation of momentum SGD, without accounting for the subtle implementation differences of PyTorch momentum, produces an odd hybrid of equations (2) and (3). In our experiments, we use the original version of classical momentum with a constant momentum parameter (i.e., equation (2)), and observe a non-trivial improvement in performance, especially when coupled with LARS adaptation.

Multi-Crop. Figure 2 illustrates the PAWS method when generating two views of each unlabeled image, however, as mentioned in Section 5, we use the multi-crop data-augmentation of SwAV [3] to generate more than two views of each image in all of our experiments. Given an unlabeled image, we generate several views of that image by taking two large crops (224×224) and six small crops (96×96). We use the `RandomResizedCrop` method from the `torchvision.transforms` module in PyTorch. The two large-crops (global views) are generated with scale $(0.14, 1.0)$, and the six small-crops (local views) are generated with scale $(0.05, 0.14)$, following the original implementation in [3].

When computing the PAWS loss, each small crop has two positive views (the two global views), and each large crop has one positive view (the other global view). Specifically, let $\mathbf{x} \in \mathbb{R}^{n \times (3 \times H \times W)}$ denote a mini-batch of n unlabeled images. For each image $\mathbf{x}_i$ in the mini-batch, we generate two large crop views, $\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)} \in \mathbb{R}^{3 \times 224 \times 224}$, and six small crop views, $\mathbf{x}_i^{(3)}, \ldots, \mathbf{x}_i^{(8)} \in \mathbb{R}^{3 \times 96 \times 96}$. Let $z_i^{(1)}, \ldots, z_i^{(8)} \in \mathbb{R}^d$ denote the representations computed from $\mathbf{x}_i^{(1)}, \ldots, \mathbf{x}_i^{(8)}$ respectively, and let $p_i^{(1)}, \ldots, p_i^{(8)}$ denote the predictions for representations $z_i^{(1)}, \ldots, z_i^{(8)}$ respectively. Lastly, let $\overline{p} := \frac{1}{8n} \sum_{i=1}^{n} \sum_{k=1}^{8} \rho(p_i^{(k)})$ denote the average of the sharpened predictions (recall $\rho(\cdot)$ is the sharpening function defined in Section 3). The overall PAWS objective to be minimized is

$$\frac{1}{8n} \sum_{i=1}^{n} \left(H(\rho(p_i^{(1)}), p_i^{(2)}) + H(\rho(p_i^{(2)}), p_i^{(1)}) + \sum_{k=3}^{8} H\left(\frac{\rho(p_i^{(1)}) + \rho(p_i^{(2)})}{2}, p_i^{(k)} \right) \right) - H(\overline{p}). \tag{4}$$

In equation (4), $p_i^{(1)}$ and $p_i^{(2)}$ correspond to the two large crop views, and $p_i^{(3)}, \ldots, p_i^{(8)}$ correspond to the six small crop views. Thus, from equation (4), the target for $p_i^{(1)}$ is the sharpened positive view prediction $\rho(p_i^{(2)})$, and similarly, the target for $p_i^{(2)}$ is the sharpened positive view prediction $\rho(p_i^{(1)})$. For the small views, $p_i^{(3)}, \ldots, p_i^{(8)}$, we use both $\rho(p_i^{(1)})$ and $\rho(p_i^{(2)})$ as positive view predictions and average those to produce a single target. This is similar to the use of multicrop in SwAV [3].

While the multi-crop augmentation makes the notation in equation (4) a little cumbersome, note that this objective is nearly identical to the objective in equation (1), except that (4) also includes a sum over the small crop-views, $\sum_{k=3}^{8}(\cdots)$.

Intuitively, by only using the large crops as positive samples (note that small crops are never positive samples for any of the other views), the method learns a global-to-local feature mapping, which maps local features in the small crops to global features in the large crops. The multi-crop augmentation is in fact an essential component of the PAWS algorithm. As will be shown in Appendix D, the multi-crop augmentation strategy in PAWS is not only important when training on large internet images, containing possibly obfuscated objects at various scales, such as ImageNet [41], but is also important for small object-centric images, such as CIFAR10 [43]. This observation suggests that the benefit of "local-to-global" matching induced by the multi-crop augmentation strategy in PAWS goes beyond simply inducing obfuscation or scale invariant image representations.

B. Additional Ablation Experiments

Longer training. The results reported in Section 6 illustrate the performance of PAWS after 100 and 200 pre-training epochs. We have not observed substantial benefits to training for longer than this. Results after pre-training longer are shown in Table 5 for ResNet-50 1× and 2× architectures. While PAWS does not seem to benefit from longer training, it is interesting

		Top-1	
Architecture	**Epochs**	**1%**	**10%**
ResNet-50	100	63.8	73.9
ResNet-50	200	66.1	75.0
ResNet-50	300	66.5	75.5
ResNet-50 (2×)	100	68.2	77.0
ResNet-50 (2×)	200	69.6	77.8
ResNet-50 (2×)	300	69.6	77.7

Table 5: **Longer Training.** Examining the impact of longer training for various ResNet architectures on ImageNet. In both 1% and 10% label settings, and across both ResNet-50 and ResNet-50 (2×) architectures, training for more than 200 epochs is generally not necessary and only yields marginal improvements.

to observe that, by contrast, PAWS-NN, which performs nearest neighbours classification (no fine-tuning), may benefit from longer training, as suggested by Table 1.

Prediction head. As noted in Section 5, we include a prediction head to facilitate comparison to previous work [4], where it was suggested as a mechanism to prevent representation collapse. Table 6 illustrates that this is not needed when pre-training with PAWS, and in fact the performance of PAWS is marginally better when the prediction head is omitted during pre-training.

	Top 1	
	100 epochs	**200 epochs**
With Prediction Head	73.9	75.0
Without Prediction Head	74.2	75.2

Table 6: **Prediction Head.** Examining the effect of the prediction-head when training a ResNet-50 on ImageNet and 10% of the training set is labeled. Our default setup is shaded in green. Unlike self-supervised methods that collapse without a prediction head [4, 32], PAWS still converges without a prediction head, as predicted by the theoretical result Proposition 1.

ME-Max regularization. Recall that PAWS pre-training uses a cross-entropy loss with sharpened targets to encourage representations of different views of the same image to be consistent (reducing cross-entropy), and it also uses the mean-entropy maximization regularizer to maximize the entropy of the average prediction, computed across the unlabeled samples in the mini-batch. Table 7 illustrates the effect of training with only the cross-entropy term and disabling the ME-MAX regularization. While the impact is more pronounced in the setting with only 1% labeled data, using ME-MAX regularization improves performance in all cases.

	Top 1	
	1%	**10%**
With ME-MAX	63.8	73.9
Without ME-MAX	52.9	73.6

Table 7: **ME-Max Regularization.** Examining the effect of the ME-MAX regularizer when training a ResNet-50 on ImageNet for 100 epochs. Our default setup is shaded in green. The ME-MAX regularizer is especially helpful in the 1% label setting, but only provides a marginal improvement in the 10% label setting.

C. Comparison to Supervised Learning

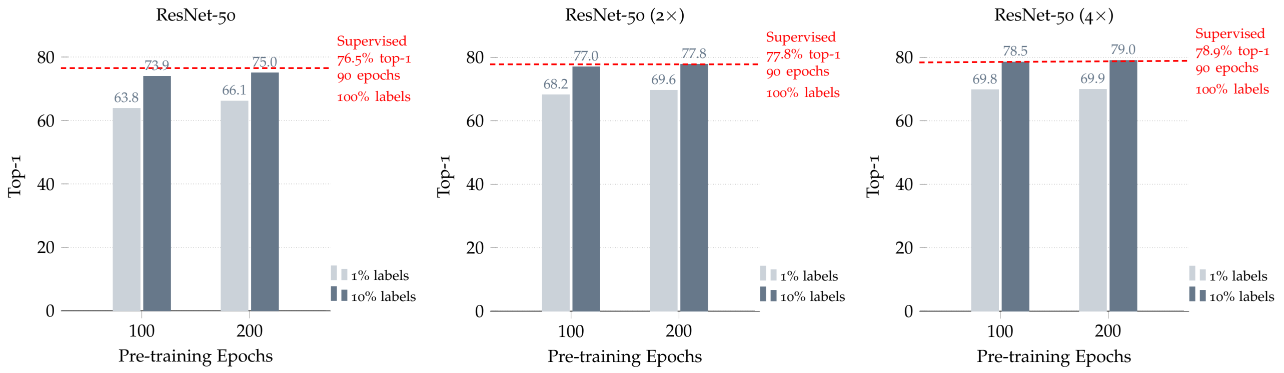


Figure 5: Comparing ResNet architectures trained using PAWS on ImageNet, with only a small fraction of the training instances labeled, to the same ResNet architectures trained in a fully supervised manner on ImageNet with *all* training instances labeled. Supervised models are reported from SimCLR [2, Appendix B.3], and ablated over the same data-augmentations used to train PAWS. We report results for the best supervised model found by [2] over the data-augmentation sweep. When training with ResNet-50 (2×) and ResNet-50 (4×) architectures, PAWS matches the performance of fully supervised learning. Specifically, PAWS is the first method to, with only 10% of training instances labeled, match fully supervised learning on ImageNet with 100% of training instances labeled, using the same architecture, and without distilling from a larger teacher model. Notably, this result is achieved with only 200 epochs of training.

Figure 5 compares PAWS semi-supervised training to supervised learning with the same architecture using a standard cross-entropy loss. The supervised baseline is trained on the full set of ImageNet labels, whereas the PAWS result is obtained by pre-training (and fine-tuning) with access to only a small fraction of the ImageNet labels. The supervised models are reported from SimCLR [2, Appendix B.3], where they are swept over the number of training epochs $\{90, 500, 1000\}$, and ablated over the data-augmentations used in PAWS pre-training $\{$crop/flip, crop/flip + color distortion, crop/flip + color distortion + Gaussian blur$\}$. Figure 5 reports results for the best supervised model found by [2], which corresponds to 90 epochs of training with random crop/flip for the ResNet-50, and 90 epochs of training with random crop/flip+color distortion for the wider ResNets. When training with ResNet-50 (2×) and ResNet-50 (4×) architectures, PAWS matches the performance of fully supervised learning. Specifically, PAWS is the first method to, with only 10% of training instances labeled, match fully supervised learning on ImageNet with 100% of training instances labeled, using the same architecture, and without distilling from a larger teacher model. Notably, this result is achieved with only 200 epochs of training. However, as a word of caution, this experiment should only be interpreted as a type of ablation, since the performance of supervised learning models can likely be improved by incorporating additional advanced supervised augmentation strategies like Mixup [44], CutMix [45], and AutoAugment [46], which simultaneously learns a data-augmentation policy during training.

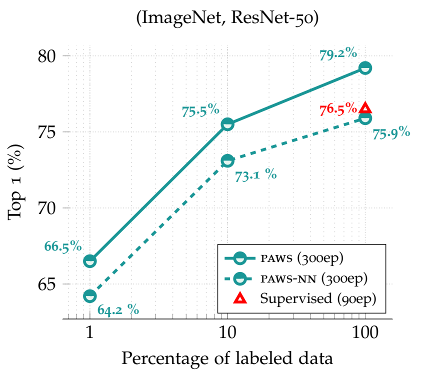


Figure 6: Examining PAWS scaling when training a ResNet-50 on ImageNet with various percentages of labeled data. PAWS-NN refers to performing nearest-neighbour classification directly using the PAWS-pretrained representations, with the labeled training samples as support, while PAWS refers to fine-tuning a classifier using the available labeled data after PAWS-pretraining. Supervised models are reported from SimCLR [2, Appendix B.3], and ablated over the same data-augmentations used to train PAWS. We report results for the best supervised model found by [2] over the data-augmentation sweep. When trained with 100% of the available labels, PAWS surpasses fully supervised learning and produces representations that are well calibrated for non-parametric classification (PAWS-NN).

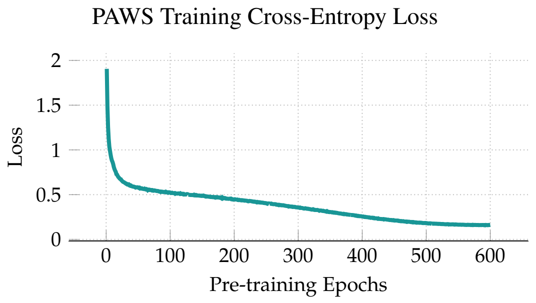
WideResNet-28-2, CIFAR10, 4000 labels

Method	Epochs	Top-1
Supervised Learning with full dataset [6]	1000	94.9 ± 0.2
Methods using label propagation:		
Temporal Ensemble [47]	300	83.6 ± 0.6
Mean Teacher [8]	300	84.1 ± 0.3
VAT + EntMin [13]	123	86.9 ± 0.4
LGA + VAT [48]	–	87.9 ± 0.2
ICT [14]	600	92.7 ± 0.0
MixMatch [9]	–	93.8 ± 0.1
ReMixMatch [10]	–	94.9 ± 0.0
EnAET [49]	1024	94.7 ± –––
UDA [15, 6]	2564	94.5 ± 0.2
FixMatch [11]	–	95.7 ± 0.1
MPL [6]	2564	**96.1** ± **0.1**
Non-parametric classification:		
PAWS-NN	**600**	**96.0** ± **0.2**

(a)

Additional Architectures, CIFAR10, 4000 labels

Method	Architecture	Params	Epochs	Top-1
SimCLRv2 [1]	ResNet-200 (+SK)	95M	800	**96.0**
SimCLRv2 [1]	ResNet-18 (+SK)	12M	800	92.1
Non-parametric classification:				
PAWS-NN	WideResNet-28-2	**1.5M**	**600**	**96.0**



(b)

Figure 7: Training a WideResnet-28-2 on CIFAR10.*For label propagation methods, the number of epochs is counted with respect to the unsupervised mini-batches. *For Meta Pseudo-Labels (MPL), the number of epochs only includes the student-network updates, and does not count the additional 1,000,000 teacher-network updates (computationally equivalent to roughly an additional 2564 epochs) that must happen sequentially (not in parallel) with the student updates. PAWS-NN refers to performing nearest-neighbour classification directly using the PAWS-pretrained representations, with the 4000 labeled training samples as support. We report the mean top-1 accuracy and standard deviation across 5 seeds for the 4000 label split.

D. Additional Experiments — CIFAR10

We also evaluate the PAWS pre-training scheme on the CIFAR10 [43] dataset using a single NVIDIA V100 GPU. We first pre-train a network using PAWS on CIFAR10 with access to 4000 labels, and then report the nearest-neighbour classification accuracy on the test set using the 4000 labeled training images as support. On CIFAR10 we only report PAWS-NN, and do not fine-tune a linear classifier on top of the network. For details on the Nearest Neighbours classifier, see Appendix A.

Implementation details. We adopt similar hyper-parameters to the ImageNet experiments. Specifically, for pre-training, we use the LARS optimizer with a momentum value of 0.9, weight decay 10^{-6}, cosine-similarity temperature of $\tau = 0.1$, and target sharpening temperature of $T = 0.25$. To construct the different image views, we use the multi-crop strategy, generating two large crops (32×32), and six small crops (18×18) of each unlabeled image. We use the `RandomResizedCrop` method from the `torchvision.transforms` module in PyTorch. The two large-crops (global views) are generated with scale $(0.75, 1.0)$, and the six small-crops (local views) are generated with scale $(0.3, 0.75)$. We use a batch-size of 256 and linearly warm-up the learning rate from 0.8 to 3.2 during the first 10 epochs of pre-training, and decay it following a cosine schedule thereafter. To construct the support mini-batch at each iteration, we also randomly sample 640 images, comprising 10 classes and 64 images per class, from the labeled set, and apply label smoothing with a smoothing factor of 0.1. For all sampled images (both unlabeled images and support images) we apply the basic set of SimCLR data-augmentations, specifically, random crop, horizontal flip, and color distortion (but no Gaussian blur). However, in contrast to the ImageNet setup, we also generate two views of each sampled support image. On CIFAR10 we find it much easier for the network to learn to classify the images than to perform instance discrimination. Generating two views of each sampled support image helps the network improve its instance discrimination ability and produce representations that are invariant to the data-augmentations used for training.

The encoder f_θ in our experiments is a WideResNet-28-2 [50] trunk without dropout, containing a 3-layer MLP projection head, consisting of three fully-connected layers of dimension 128, and Batch Normalization applied to the hidden layers. To simplify the implementation, we do not include a prediction head after the projection head. As shown in Table 6 on ImageNet, PAWS pre-training works well without a prediction head, and we find this to be true on CIFAR10 as well.

Following pre-training, we freeze the batch-norm layers, and fine-tune the trunk of the network for 180 optimization steps on the available labeled samples using the supervised contrastive loss of [40], and do not apply any data-augmentations during this phase. The point of these few optimization steps is to tighten the representation clusters of the labelled training samples before using them as support to classify the test images. For this phase, we use momentum SGD with a batch-size of 640 (comprising 64 images from 10 classes), and sample the mini-batches with replacement; i.e., while images in the same mini-batch in a given iteration are always unique, some of the images may be re-sampled in the subsequent iteration's mini-batch. We use a cosine-temperature of $\tau = 0.1$, momentum parameter 0.9, a learning rate of 0.1 with cosine-decay, and no weight-decay.

Results. Table 7a compares PAWS-NN to other semi-supervised learning methods trained using identical networks (WideResNet-28-2) on CIFAR10 with access to 4000 labels. Although the intention here is to simply validate PAWS on another dataset, the observations are similar to ImageNet. By using the pre-trained representations directly in a nearest neighbour classifier, PAWS can match the state-of-the-art on CIFAR10 with significantly less training. It is possible that carefully fine-tuning a linear classifier on top of the trunk and incorporating more advanced data-augmentations would further improve performance. Table 7b compares PAWS-NN to the self-supervised SimCLRv2 [1] method trained (and fine-tuned) with larger architectures. The PAWS method achieves superior performance in fewer pre-training epochs, using a residual network containing over $60\times$ fewer parameters.

E. Alternative Strategies for Non-Collapse

Proposition 1 provides a theoretical guarantee that the proposed method is immune to the trivial collapse of representations. The underlying principle is that collapsing representations result in high entropy predictions under the non-parametric similarity classifier, but the targets are always low-entropy (because we sharpen them), and so collapsing all representations to a single vector is not a stationary point of the training dynamics. In this section we demonstrate two simple alternative strategies to guarantee non-collapse of representations without making the target-sharpening assumption.

E.1. Semi-Supervised Prediction

If an image in the sampled mini-batch of image views has a class label, then we can directly use that class label as the target for its prediction p, rather than using the positive view prediction, p^+, as the target. Under such a scenario, Proposition 2 provides the theoretical guarantee.

Assumption 3 (Semi-Supervised Image Views)**.** Each mini-batch of image views contains at least one labeled sample.

Proposition 2 (Non-Collapsing Representations — Semi-Supervised)**.** Suppose Assumptions 1 and 3 hold. If the representations collapse, i.e., $z = z_i$ for all $z_i \in \mathcal{S}$, then $\|\nabla H(p^+, p)\| > 0$, and the solution is non-stationary.

Proof. The proof is identical to that of Proposition 1, up to the last step. At which point, letting z correspond to the labeled instance in the mini-batch of images views, we have that the target p^+ is not equal to the uniform distribution because it corresponds to the corresponding ground truth class label. From which it follows that $p \neq p^+$ and $\|\nabla H(p^+, p)\| > 0$. ■

Note that Proposition 2 is *only* presented as a theoretical alternative strategy to prevent collapse, but is not used in our experiments; instead, we always use the sharpened positive view prediction p^+ as the target for the anchor view prediction p.

E.2. Entropy Minimization

A third possible strategy to guarantee non-collapsing representations without using the target sharpening assumption is to add an entropy minimization term [12] to the loss. As shown in the proofs for Propositions 1 and 2, collapsing representations always result in high-entropy predictions p. These high-entropy predictions result in large non-zero gradients due to the entropy minimization term (which as the name implies is minimized when the entropy is low), and so, just as before, collapsing representations are not stationary points of the training dynamics. While adding an entropy minimization term to the loss is a conceptually simple strategy, target sharpening is arguably even simpler, and, by Proposition 1, suffices to guarantee non-collapsing representations, so we do not use entropy minimization in our experiments.

F. Ethical Considerations

Increasing model and dataset sizes is a proven approach to improving the performance of image recognition models. Depending on the intended application, more accurate image recognition models may yield substantial social benefits for society; e.g., improving the quality and safety of systems relying on image recognition. However, as with any engineering problem, there is no free lunch, and one must not stop grappling with the ethical concerns of more computationally expensive training pipelines, such as potentially larger environmental footprints (depending on the compute cluster used for training) and exclusionary ramifications. Computationally intensive training pipelines may exclude participation from researchers without access to the computational resources needed to conducted such experiments, which in-turn may lead to slower progress in the field.

The proposed method in this work matches the current state-of-the-art in data-efficient image recognition using considerably smaller models and fewer training epochs. While our method still benefits from wider and deeper architectures, we demonstrate that the performance of smaller models is not yet saturated, and that research targeting improvements on these smaller models may very well translate to larger-scale settings.

However, generally speaking, we caution against conflating increased computational effort with larger models, since we observe that this relationship is not always linear. For example, when training a ResNet-50 ($2\times$) for 12 hours (100 epochs) on 64 V100 GPUs, we obtain 68% top-1 accuracy in the 1% label setting and 77% in the 10% label setting. Conversely, when training a smaller ResNet-50 for 17 hours (200 epochs) on 64 V100 GPUs, we obtain 66% top-1 accuracy in the 1% label setting and 75% in the 10% label setting.

G. Historical Perspective

Constructivist learning theory—developed a near half-century ago by Jean Piaget and built on notions of schemata put forth by Immannuel Kant—has (surprisingly) withstood the test of time. Constructivism not only revolutionized school curricula in the 20^{th} century, but remains to this day a crucial element of many teaching philosophies—placing greater emphasis on spontaneous learning through self-regulation and concrete activities, often under the pseudonym of Project-Based Learning in primary and secondary schools, and Lab-Based Instruction in post-secondary institutions. At the heart of Constructivism is the idea that every individual possesses mental schemata—representations relating to distinct semantic concepts—and that learning occurs through the process of *assimilation and accommodation*.[5] During assimilation, the mind adapts its representation of new experiences to fit its existing schemata, while during accommodation, the existing schemata are updated to make sense of new experiences. In short, Constructivism purports that knowledge is "constructed" through self-guided exploration, and that mental representations of semantic concepts in sensorimotor observations are learned by conforming new observations to past experiences and vice versa.

It is of particular interest to us to note that one of Piaget's tenets was that sensorimotor development came about the process of optimizing a non-purposive mental objective using assimilation and accommodation. Non-purposive learning generally refers to the process of learning without working towards any particular purpose or goal. As such, non-purposive learning is generally concerned with deriving mental models, or schemata, of sensorimotor observations, under which all new observations can be readily explained in terms of past observations. Clearly, non-purposive learning is closely related to the idea embodied nowadays by task-agnostic self-supervised pre-training, but differs slightly. Whereas current task-agnostic self-supervised learning approaches predict inputs from inputs in a fully unsupervised manner, non-purposive learning approaches do not preclude the use of semantic information. To the contrary, semantic information can be used to aid in the construction of sensorimotor schemata; i.e., non-purposive learning can be unsupervised, semi-supervised, weakly-supervised, or fully supervised. This paper proposes a non-purposive method for semi-supervised learning.

Criticisms of Constructivist Learning Theory. Despite the widespread success of Constructivisim, one of the weaknesses of Piagetian theory is its lack of specificity in describing the mechanisms by which assimilation and accommodation occur to produce mental representations of semantic concepts in sensorimotor observations [51]. It is perhaps for this reason that Piaget was especially interested in the emerging field of cybernetics (a precursor to artificial intelligence developed in the 40's by Norbert Wiener) and has gone so far as to say that "Life is essentially auto-regulation," and "cybernetic models are, so far, the only ones throwing any light on the nature of auto-regulatory mechanisms" [52]. Piaget advocated for cybernetic models with great aplomb, "I wish to urge that we make an attempt to use it" [53], and may have attempted to use them himself had it not been for his advanced age. Unfortunately, despite the clear links to cybernetics, the connection to Constructivism did not readily carry over to artificial intelligence (AI) in the 70's due to the largely symbolic nature of AI approaches at the time; e.g., it was not obvious how to represent the near infinite variations of a hand-drawn curve in a single concise representation (i.e., a schema); an issue which is now largely resolved by gradient-based learning and modern neural network architectures.

[5]The term schema may be familiar to researchers working with relational database systems, where it has become standard jargon referring to the logical structure of a database (in close relation to its original meaning in psychology).