LEARNING USEFUL REPRESENTATIONS FOR SHIFTING TASKS AND DISTRIBUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Does the dominant approach to learn representations (as a side effect of optimizing an expected cost for a single training distribution) remain a good approach when we are dealing with multiple distributions. Our thesis is that such scenarios are better served by representations that are "richer" than those obtained with a single optimization episode. This is supported by a collection of empirical results obtained with an apparently naïve ensembling technique: concatenating the representations obtained with multiple training episodes using the same data, model, algorithm, and hyper-parameters, but different random seeds. These independently trained networks perform similarly. Yet, in a number of scenarios involving new distributions, the concatenated representation performs substantially better than an equivalently sized network trained from scratch. This proves that the representations constructed by multiple training episodes are in fact different. Although their concatenation carries little additional information about the training task under the training distribution, it becomes substantially more informative when tasks or distributions change. Meanwhile, a single training episode is unlikely to yield such a redundant representation because the optimization process has no reason to accumulate features that do not incrementally improve the training performance.

1 INTRODUCTION

Although the importance of features in machine learning systems was already clear when the Perceptron was invented (Rosenblatt, 1957), learning features from examples was often considered a hopeless task (Minsky and Papert, 1969). Some researchers hoped that random features were good enough, as illustrated by the Perceptron. Other researchers preferred to manually design features using substantive knowledge of the problem (Simon, 1989). This changed when Rumelhart et al. (1986) showed that the possibility of feature learning as a side effect of the risk optimization. Despite reasonable concerns about the optimization of nonconvex cost functions, feature discovery through optimization has driven the success of deep learning methods.

There are however many cues suggesting that learning features no longer can be solely understood through the optimization of the expected error for a single data distribution. First, adversarial examples (Szegedy et al., 2014) and shortcut learning (Geirhos et al., 2020) illustrate the need to make learning systems that are robust to certain changes of the underlying data distribution and therefore involve multiple expected errors. Second, the practice of transferring features across related tasks (Bottou, 2011; Collobert et al., 2011; Oquab et al., 2014) is now viewed as foundational (Bommasani et al., 2021) and intrinsically involves multiple data distributions and cost functions. It is therefore timely to question whether the optimization of a single cost function creates and accumulates features in ways that make the most sense in this broader context.

This contribution reports on experiments showing how the out-of-distribution performance of a deep learning model can benefit from internal representations that are richer and more diverse than those computed through the usual optimization process. For instance, in a variety of changing distribution scenarios, merely concatenating the penultimate layer representations obtained with several training episodes can outperform networks of equivalent size trained from scratch.

Limitations This work only considers the representations extracted by the penultimate layer of a deep network with a linear elementary classifier, or, in a single case, a distance-based classifier. Our experimental results also focus image recognition problems. Whether the same findings hold for inner representations or other applicative domains is left to future work. Despite this limited focus, it is worth noticing that our experiments cover a variety of model sizes, ranging from a couple millions

to a couple billions of parameters. Finally, we only consider two very simple ways to create rich representations, leaving considerable room for improved variants.

Organization of the manuscript Section 3 argues that a vast subspace of the penultimate layer representation of a deep network can contain information that has no bearing on the optimization cost and yet can have a substantial impact on the transfer learning performance of the network. Section 4 presents simple techniques that can be used to produce networks with slightly different representations and to accumulate them in order to construct a richer and internally redundant representation vector. Sections 5, 6, 7, and 8, present experimental results pertaining respectively to supervised transfer learning, self-supervised transfer learning, meta-learning, and out-of-distribution learning.

2 Related work

Several authors (e.g. Huang et al., 2020; Teney et al., 2022) propose to work around the shortcut learning problem (Geirhos et al., 2020) by shaping the last-layer classifier or introducing penalty terms in the optimization process in a manner that favors richer representations. Zhang et al. (2022) shows that many of these additions make the optimization challenging but more manageable if one initializes the networks with rich features constructed by alternate methods. The methods discussed in our work resemble the RFC algorithm of Zhang et al. (2022) because they also rely on multiple training episodes. However, we find that merely training with different random seeds provides sufficient diversity to achieve excellent performances for a variety of problems involving changes in data distribution, revealing the limitations of representations constructed with a single training run.

Our experimental approach is related to deep ensembles (e.g. Lakshminarayanan et al., 2017) which combine the predictions of separately trained networks in order to achieve superior in-distribution performance. Ensembling techniques work best when one makes sure that the individual models are diverse and have weakly correlated errors. Engineering diversity is often an expression of prior knowledge about the problem at hand. In contrast, we purposely refrain from engineering diversity (other than changing the seed) and we focus on the performance of our ensembles for new tasks and data distributions.

The idea of rich representation is also related to recent work (Wang et al., 2022; Dvornik et al., 2020; Bilen and Vedaldi, 2017; Gontijo-Lopes et al., 2021; Li et al., 2021; 2022) on "universal representations" that combine the representations obtained with different tasks, datasets, network architectures, hyper-parameters, etc. Because we purposely refrain from engineering such a diversity and still observe a substantial effect, we cast a new light on both the desirable properties of a robust representation and the limitations of the usual training process.

3 The representation nullspace

For the purposes of this work, we view a deep learning network as the composition of a feature extractor Φ that maps a pattern x into a representation $\Phi(x) \in \mathbb{R}^d$ and a linear layer that produces an output of dimension k that we assume is substantially smaller than d (i.e. $k < d$). The network output is then expressed as $w \circ \Phi(x)$ where w is the $k \times d$ weight matrix of the linear output layer. Since we assume that $k < d$, the matrix w has a substantial nullspace $\mathcal{N}$ of dimension greater or equal to $d - k$. Therefore the function implemented by the network remains unchanged when one replaces the feature extractor Φ by any Φ' such that $\Phi(x) - \Phi'(x) \in \mathcal{N}$ with high probability. This implies that there is a considerable choice of potential feature extractors that produce different representations and yet have no impact on the network output.

Since such equivalent representations do not affect the empirical risk, they can only be differentiated by regularization, either explicit or implicit. In order to improve the in-distribution generalization performance of the network, common regularization techniques, such as the ubiquitous weight decay, aim at zeroing the nullspace component of the representations $\Phi(x)$. For instance, Papyan et al. (2020) show how the penultimate layer representation of various deep network collapses to a "simplex equi-angular tight frame" of dimension $k - 1$ when trained for a very long time using a cross-entropy loss and a slight weight decay. When this happens, the representation carries very little information other than a noise-tolerant encoding of the network output.

Although this situation can be beneficial for in-distribution generalization, we argue that it is often problematic when the task or the data distribution changes.

- In scenarios such as transfer learning and meta-learning, one is allowed to adapt the network after the task or distribution change. Because the gradient back-propagated through the linear layer is orthogonal to the nullspace $\mathcal{N}$, the only pressure to change the nullspace component of the representation $\Phi(x)$ comes from fluctuations of the nullspace itself. Such fluctuations occur when the linear layer weights w are adapted to exploit a temporary correlation between the desired output and potentially interesting features presented in the nullspace. This is easier when the initial training leaves more information in the nullspace.
- In other scenarios, one seeks to construct networks that are robust with respect to targeted data distribution changes. This is usually achieved by special optimization objectives such as distributional robust optimization (Sagawa et al., 2020) or such as invariant learning (Arjovsky et al., 2020). In order to tame these more challenging optimization problems, it is common to pretrain the networks with empirical risk minimization. However, Zhang et al. (2022) show that constructing rich representations gives far better results than pretraining with empirical risk minimization.

Table 1 reports on a simple experiment consisting of pre-training a RESNET18 on the CIFAR10 task and transferring its learned features to a CIFAR100 task by simply retraining the last linear layer (see setups in Appendix A). Although the best in-distribution performance, 94.9%, is achieved using a slight weight decay, the transfer learning performance of the features learned without weight decay is far superior (49.6% versus 29.1%). The same observation holds when one reverses the role of the CIFAR10 and CIFAR100 datasets. Because we are far from a full network collapse, this effect disappears when one fine-tunes the transferred feature extractor, effectively recovering the performance of a network directly trained on the target task. Sections 5 and 6 offer a more thorough discussion of fine-tuning.

Table 1: Impact of L2 weight decay on supervised transfer learning between CIFAR10 and CIFAR100.

L2 weight decay	CIFAR10	CIFAR10→CIFAR100	CIFAR100	CIFAR100→CIFAR10
0	91.41±0.81	**49.68±0.72**	70.37±1.49	**78.87±0.98**
5e-4	**94.89±0.23**	29.17±0.50	**76.78±0.36**	75.92±0.54

Things become of course more complicated when we abandon the assumptions that allowed us to define the representation nullspace. For instance, the final layer might still be linear but with a number of outputs or classes k that exceeds the dimension d of the penultimate layer. Alternatively the final layer might not be linear at all, as, for instance, in the siamese networks Bromley et al. (1993) increasingly used in self-supervised scenarios. Yet, the work of Papyan et al. (2020), the information bottleneck approach of Shwartz-Ziv and Tishby (2017), and the gradient starvation observations (Pezeshki et al., 2021) of, still suggest that in the long run, the regularized training process produces upper layer representations that hardly contain any information other than what is necessary for the task encoded by the optimization criterion. In fact, once the representation contains enough information to fulfill the training task, the optimization process has no reason to create and accumulate features that no longer help improve the training objective but might yet become useful when the data distribution changes.

4 FEATURE ACCUMULATION

How then can we construct a deep learning procedure that accumulates as many potentially useful features as possible into the internal representations of the network? The answer depends of course on how we determine that a feature is potentially useful before knowing how the data distribution might change.

The Rich Feature Construction (RFC) algorithm of Zhang et al. (2022, § 4.1) seeks features that are useful for at least some subset of the training examples. The algorithm first performs multiple training episodes using adversarially re-weighted training data in a manner that ensures that these training episodes yield substantially different representations. Then a multiple head distillation process (see Figure 1, Left) constructs a single feature extractor that combines all these distinct representations into a single representation vector of equivalent size. This cumbersome process has been shown to vastly improve the out-of-distribution performance of several invariant training algorithms.

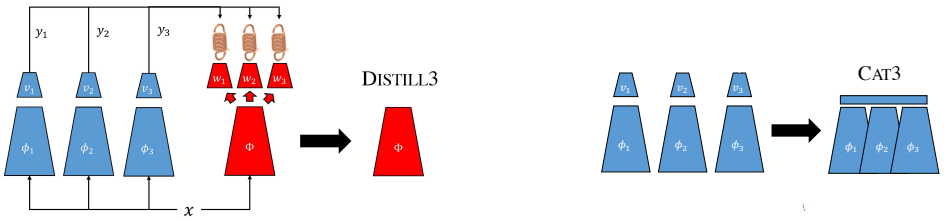

Figure 1: **Left:** (DISTILLn) A multiple head network (red) trained to predict the outputs of the pre-trained networks $\Phi_1, \Phi_2, \cdots$ (blue) must develop a representation Φ that subsumes those of all the blue networks. The same distillation process is used by the RFC algorithm (Zhang et al., 2022) but after training the networks with adversarially re-weighted data. **Right:** (CATn) The feature extractors of the pre-trained networks are concatenated, producing a larger network.

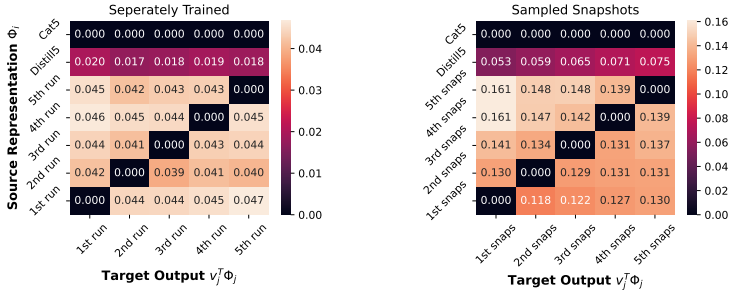

Figure 2: How well can we predict the output of network i using the feature extractor Φ_j of network $j \neq i$, and using a feature extractor Φ_d of equal dimension obtained by distillation? Left: Five networks trained from scratch with different seeds. Right: Five snapshots during a single training episode with relatively high stepsize. The comparison is only applicable on each column because Eq 1 isn't invariant to invertible linear transformation on $v_j^\top \Phi_j(x)$.

The present contribution exploits a far simpler intuition. Although the optimization process has no reason to accumulate features that no longer help the training objective, these same features might have been found useful when the network representation was still wanting. Therefore we posit that we can obtain substantially different representations by simply performing multiple training episodes with different seeds, that is, different initial weights and different data orderings.

In order to validate this intuition, we train five RESNET18 networks on the Caltech-UCSD Birds (CUB) dataset (Wah et al., 2011) using different seeds, yielding five feature extractors Φ_i and final linear layers v_i. We then compute the normalized least square errors (computed via QR decomposition) obtained when predicting the output of network j using the features of each network $i \neq j$,

$$\min_v \frac{\mathbb{E}[\|v^\top \Phi_i(x) - v_j^\top \Phi_j(x)\|^2]}{\mathbb{E}[\|v_j^\top \Phi_j(x)\|^2]} \,. \tag{1}$$

Figure 2 (left) shows how these errors are substantially higher than those obtained when predicting the network outputs using the combined feature extractor obtained by distillation (Figure 1, left). This indicates that merely training with the same data but different seeds produces different representations which can be combined into a richer representation that predicts the outputs of all the original networks better than any individual network representation. Figure 2 (right) shows the same effect using instead five regularly spaced snapshots taken during a single training episode with a high step size. Although this is computationally more attractive, the experimental results reported in the rest of this paper always use the more easily interpretable and independently reproducible multiple training approach.

Although distillation provides the means to combine the representations of multiple networks into a single feature extractor of equivalent dimension (Figure 1, left), the distillation process depends on the training data distribution and therefore might still exclude features that could become useful when the distribution changes. The simplest distribution-independent way to combine representations consists of concatenating them into a vector whose dimension is the sum of the dimensions of the

Table 2: Supervised transfer learning from IMAGENET to INAT18, CIFAR100, and CIFAR10 using linear probing. The ERM (empirical risk minimization) rows provide baseline results. The CATn rows use the concatenated representations of n separately trained networks. The DISTILL5 row uses the representations of five separately trained networks combined by distillation. Performances should be compared between architectures with comparable numbers of parameters.

method	architecture	params	ID IMAGENET	Linear Probing (OOD) INAT18	CIFAR100	CIFAR10
ERM	RESNET50	23.5M	75.58	37.91	90.57	73.23
ERM	RESNET50W2	93.9M	77.58	37.34	90.86	72.65
ERM	RESNET50W4	375M	78.46	38.71	92.13	74.81
ERM	2×RESNET50	47M	75.03	39.34	90.94	74.36
ERM	4×RESNET50	94M	75.62	41.89	90.61	74.06
CAT2	2×RESNET50	47M	77.57	43.26	91.86	76.10
CAT4	4×RESNET50	94M	78.15	46.55	93.09	78.19
CAT5	5×RESNET50	118M	78.27	47.78	93.21	78.53
CAT10	10×RESNET50	235M	78.36	49.65	93.75	79.61
DISTILL5	RESNET50	23.5M	76.39	40.75	92.54	76.50

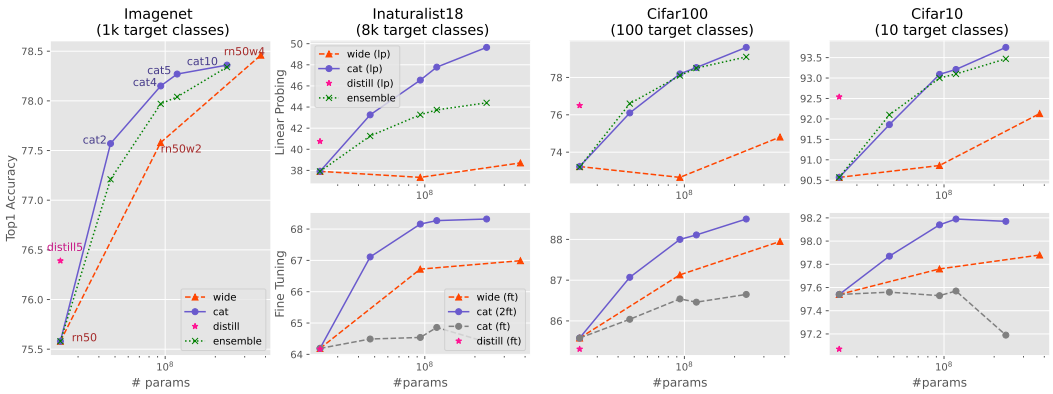

Figure 3: Supervised transfer learning from IMAGENET to INAT18, CIFAR100, and CIFAR10. The top row shows the superior linear probing performance of the CATn networks (blue, "cat(lp)"). The bottom row shows the performance of fine-tuned CATn, which is poor with normal fine-tuning (brown, "cat(ft)") and excellent for two-stage fine tuning (blue, "cat(2ft)"). Because all models are trained using the same data and protocol, the training time grows proportionally with the number of parameters, except for the largest wide models which suffer from model parallelization overhead.

source representations (Figure 1, right). However, we must then compare the out-of-distribution performance of the concatenated representation with that of single networks of equivalent complexity. The following sections show that substantial improvements can still be achieved under this condition.

5 SUPERVISED TRANSFER LEARNING EXPERIMENTS

This section focuses on supervised transfer learning scenarios in which the representations learned using an auxiliary supervised task, such as the IMAGENET(1k) object recognition task (Deng et al., 2009), are then used for the target tasks, such as, for our purposes, the CIFAR10, CIFAR100, and INATURALIST18 (INAT18), object recognition tasks (Krizhevsky, 2009; Van Horn et al., 2018). We distinguish the linear probing scenario where the penultimate layer features of the pre-trained network are used as inputs for linear classifiers trained on the target tasks, and the fine tuning scenario which uses back-propagation to further update the transferred features using the target task training data.

The first three rows of Table 2, labeled ERM, provide baselines for the linear probing scenario, using respectively a RESNET50 network (He et al., 2016a), as well as larger variants RESNET50Wn with n times wider internal representations and roughly n^2 times more parameters. The following

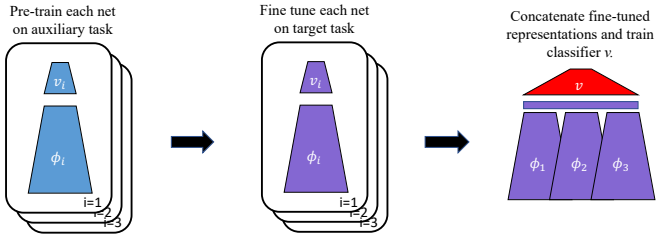

Figure 4: Two-stage fine-tuning consists of fine-tuning each network separately, then concatenating their feature extractors, now frozen, and training a final classifier.

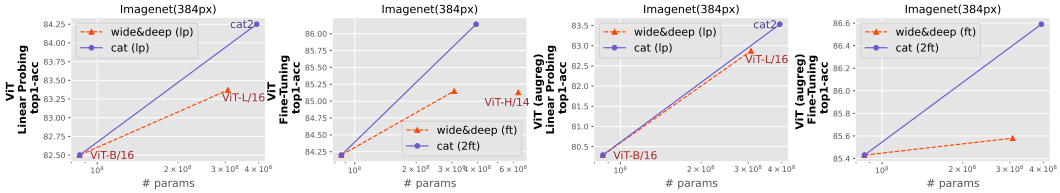

Figure 5: Supervised transfer learning from IMAGENET21K to IMAGENET on vision transformers.

two rows of Table 2 provides additional baseline results using networks $n\times$RESNET50 composed of respectively n separate RESNET50 networks joined by concatenating their penultimate layers. Although these networks perform relatively poorly on the pre-training task IMAGENET, their linear probing performance is substantially better than that of the ordinary RESNETs.

The following four lines of Table 2, labeled CATn, are obtained by training n separate RESNET50 networks on IMAGENET using different random seeds and using their concatenated representations as inputs for a linear classifier trained on the target tasks. This approach yields linear probing performances that substantially exceed the performances of comparably sized baseline networks. It is interesting to note how CATn, with separately trained components, outperforms the architecturally similar $n\times$RESNET50 trained as a single network. The final line of Table 2, labeled DISTILL5, is obtained by combining the representation of five separately trained RESNET50 by distillation and training a linear classifier on the target task. The DISTILL5 linear probing performance exceeds that of the comparable RESNET50 network but does not match CATn.

These results are succinctly represented in the top row of Figure 3. For each target task INAT18, CIFAR100, and CIFAR10, two curves respectively show the linear probing performance of the baseline RESNET50Wn (red, labeled "wide(lp)") and of the CATn networks (blue, "cat(lp)") as a function of the number of parameters of their inference architecture. An additional point (pink star, "distill(lp)") describes the performance of DISTILL5. The left plot (double height) of Figure 3 provides the same information when the target task is the same as the pre-training task. In order to save space, all further results in the main text of this contribution are presented with such plots, with result tables provided in the appendix B and C.

The top row of Figure 3 also plots the performance of deep ensemble averaging (logits averaging, green, labeled "ensemble"). Such ensembles rely on the same concatenated representation as the CATn but differ because the final classifier is an average of n separately trained classifiers of dimension d instead of a full linear classifier of dimension nd. We view this as a difference in training capacity that explains why ensembles match the CATn performance on the CIFAR tasks but lag behind on the more challenging INAT18 task.

The bottom row of Figure 3 presents results for the fine-tuning scenario. For each target task, the red curve (labeled "wide(ft)") plots the fine-tuning transfer learning performance of the baseline RESNET50Wn networks. The brown curves ("cat(ft)") represent the poor transfer performance obtained by concatenating n separately trained RESNET50 feature extractors, adding a linear layer, and fine-tuning everything by back-propagation on the target task. The poor performance of plain fine-tuning has already been pointed out (Kumar et al., 2022; Kirichenko et al., 2022). In contrast, the blue curves ("cat(2ft)") represent the superior performance of two-stage fine tuning (Figure 4), which consists of separately training n RESNET50 on IMAGENET, separately fine-tuning these RESNET50 on the target task, and finally training a linear classifier on top of the concatenation of the n separately fine-tuned representations.

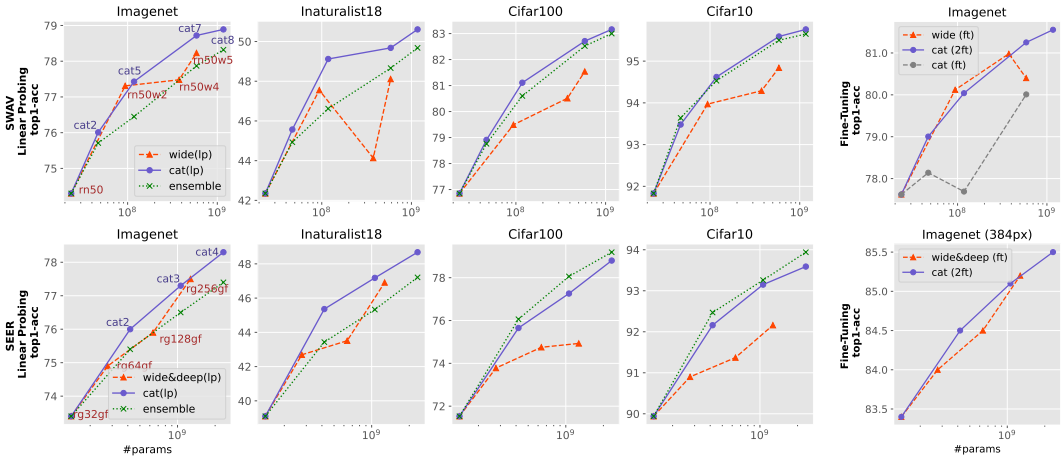

Figure 6: Self-supervised transfer learning with SWAV on unlabeled IMAGENET (top row) and with SEER in INSTAGRAM1B (bottom row). CATn yields the best linear probing performance ("cat(lp)") for supervised IMAGENET, INAT18, CIFAR100, and CIFAR10 target tasks. CATn with two-stage fine tuning ("cat(2ft)") matches equivalently sized baseline models, but with much easier training. Due to the space limitation, we put other fine-tuning curves in appendix C.1.1.

Figure 5 shows that these observations also hold for large self-attention networks. We carried out supervised transfer experiments using the original vision transformers (ViT) (Dosovitskiy et al., 2020) and using a more advanced version (ViT(augreg)) with tuned data augmentations and regularization (Steiner et al., 2021). We use two transformers of two different sizes, ViT-B/16 and ViT-L/16, pretrained on IMAGENET21K.[1] Supervised transfer baselines are obtained by linear-probing and by fine-tuning on IMAGENET1K. These baselines are outperformed by respectively linear-probing and two-stage fine tuning on top of the concatenation of their final representations (CAT2). Note that an even larger transformer architecture, ViT-H/14, yields about the same IMAGENET1K fine-tuning performance as ViT-L/16, despite having twice as many parameters Dosovitskiy et al. (2020).

6 SELF-SUPERVISED TRANSFER LEARNING EXPERIMENTS

In self-supervised transfer learning (SSL), transferable representations are no longer constructed using a supervised auxiliary task, but using a training criterion that does not involve tedious manual labeling. We focus on schemes that rely on the knowledge of a set of acceptable pattern transformations. The training architecture then resembles a siamese network whose branches process different transformations of the same pattern. The SSL training objective must then balance two terms: on the one hand, the representations computed by each branch must be close or, at least, related; on the other hand, they should be prevented from collapsing partially (Jing et al., 2021) or catastrophically (Chen and He, 2020). Although this second term tends to fill the representation with useful features, what is necessary to balance the SSL training objective might still exclude potentially useful features for the target tasks.

This section presents results obtained using SWAV pre-training using 1.2 million IMAGENET images (Caron et al., 2020) and using SEER pre-training using 1 billion INSTAGRAM1B images (Goyal et al., 2022). These experiments leverage the pre-trained models made available by the authors: four RESNET50, one RESNET50W2, one RESNET50W4 and one RESNET50W5 for the SWAV experiments;[2] and one REGNET32GF, one REGNET64GF, one REGNET128GF, and one REGNET256GF (1.3B parameters) for the SEER experiments.[3].

The first four columns of Figure 6 present linear probing results for four target object recognition tasks: supervised IMAGENET, INATURALIST18, CIFAR100, and CIFAR10. The baseline curves (red, labeled "wide(lp)" or "wide&deep(lp)") plots the performance of linear classifiers trained on top of the pre-trained SSL representations. The CATn curves (blue, labeled "cat(lp)") were obtained by training a linear classifier on top of the concatenated representations of the n smallest SSL pre-trained

[1] Checkpoints provided at https://github.com/google-research/vision_transformer.
[2] https://github.com/facebookresearch/swav
[3] https://github.com/facebookresearch/vissl/tree/main/projects/SEER

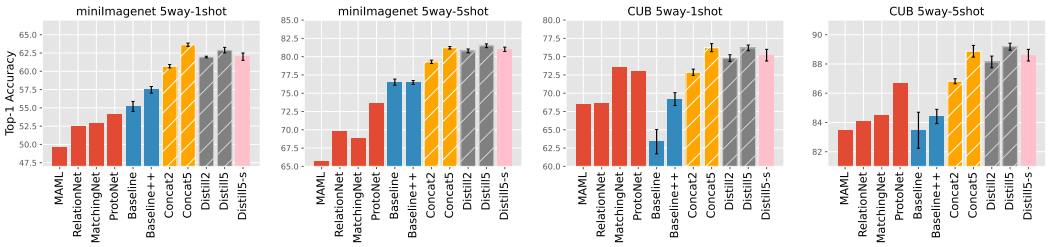

Figure 7: Few-shot learning performance on MINIIMAGENET and CUB. Four common few-shot learning algorithms are shown in red (results from Chen et al. (2019)). Two supervised transfer methods, with either a linear classifier (BASELINE) or cosine-based classifier (BASELINE++) are shown in blue. The DISTILL and CAT results, with a cosine-base classifier, are respectively shown in orange and gray. The DISTILL5-S results were obtained using five snapshots taken during a single training episode with a relatively high step size. The standard deviations over 5 runs are reported.

representations. The deep ensemble curve (green, labeled "ensemble") was obtained by averaging the outputs (before softmax) of linear classifiers separately trained on top of each of the n smallest SSL pre-trained representations. The CATn approach offers the best overall performance.

The last column of Figure 6 presents results with fine-tuning for the supervised IMAGENET task. The CATn approach with two-stage fine-tuning (Figure 4) matches the performance of equivalently sized baseline networks. In particular, the largest CAT4 model, with 2.3B parameters, achieves 85.5% correct classification rate, approaching the 85.8% rate of the largest network of Goyal et al. (2022), REGNET10B with 10B parameters. Of course, separately training and fine-tuning the components of the CAT4 network is far easier than training a single REGNET10B network.

Additional results using SIMSIAM (Chen et al., 2020) and with DISTILL are provided in appendix C.3. Other experiment details are provided in appendix C.

7 META-LEARNING AND FEW-SHOTS LEARNING EXPERIMENTS

Each target task in the few-shots learning scenario comes with only a few training examples. One must then consider a large collection of target tasks to obtain statistically meaningful results.

We follow the setup of Chen et al. (2019) in which the base task is an image classification task with a substantial number of classes and examples per class, and the target tasks are five-way classification problems involving novel classes that are distinct from the base classes and come with only a few examples. Such a problem is often cast as a meta learning problem in which the base data is used to learn how to solve a classification problem with only a few examples. Chen et al. (2019) find that excellent performance can be achieved using simple baseline algorithms such as supervised transfer learning with linear probing, (BASELINE, as in Section 5), or with a cosine-based final classifier (BASELINE++). These baselines match and sometimes exceed the performance of common few shots algorithm such as MAML (Finn et al., 2017), RELATIONNET (Sung et al., 2018), MATCHINGNET (Vinyals et al., 2016), and PROTONET (Snell et al., 2017).

Figure 7 reports results obtained with a RESNET18 architecture on both the MINIIMAGENET and CUB) five ways classification tasks with either one or five examples per class as set up by Chen et al. (2019). The MAML, RELATIONNET, MATCHINGNET, and PROTONET results (red bars) are copied verbatim from (Chen et al., 2019, table A5). The BASELINE and BASELINE++ results were further improved by a systematic L2 weight decay search procedure (see appendix D.2). All these results show substantial variations across runs, about 4% for CUB and 2% for MINIIMAGENET.

The DISTILLn and CATn results were then obtained by first training n RESNET18 on the base data with different seeds, constructing a combined feature extractor by either distillation or concatenation, then, for each task, training a cosine distance classifier using these features as inputs. Despite the high replication variance of the competing results, both DISTILL and CAT show very strong performance.

The pink bar (DISTILL5-S) in Figure 7 shows that similar results can be obtained when one combines the feature extractors of five snapshots taken at regular intervals during a single training episode with a relatively high step size (0.8). More results and details are shown in appendix D.

Table 3: Test accuracy on the CAMELYON17 dataset with DENSENET121. We compare various initialization (ERM, CATn, DISTILLn, and RFC) for two algorithms VREX and ERM using either the IID or OOD hyperparameter tuning method. The standard deviations over 5 runs are reported.

representation construction method	IID Tune vREx	IID Tune ERM	OOD Tune vREx	OOD Tune ERM
ERM	69.6±10.5	66.6±9.8	70.6±10.0	70.2±8.7
CAT2	74.3±8.0	74.3±8.0	73.7±8.1	74.2±8.1
CAT5	75.2±2.9	75.0±2.7	74.9±3.3	75.1±2.8
CAT20	76.4±0.5	76.5±0.5	76.8±0.9	76.4±0.9
DISTILL2	67.1±4.7	66.9±4.8	67.4±4.3	66.7±4.2
DISTILL5	69.9±7.4	69.9±6.9	71.8±5.0	69.9±6.3
DISTILL20	73.3±2.5	73.2±2.3	74.8±3.2	73.1±2.7
RFC2	77.9±2.7	78.2±2.6	79.5±2.7	78.6±2.6

8 OUT-OF-DISTRIBUTION GENERALIZATION EXPERIMENTS

In the out-of-distribution generalization scenario, we seek a model that performs well on a family of data distributions, also called environment, on the basis of a finite number of training sets distributed according to some of these distributions. Arjovsky et al. (2020) propose an invariance principle to solve such problems and propose the IRMV1 algorithm which searches a good predictor whose final linear layer is simultaneously optimal for all training distributions. Since then, a number of algorithms exploiting similar ideas have been proposed, such as VREX (Krueger et al., 2020), FISHR (Rame et al., 2022), or CLOVE (Wald et al., 2021). Theoretical connections have been made with multi-calibration (Hebert-Johnson et al., 2018; Wah et al., 2011). Alas the performance of these algorithms remains wanting (Gulrajani and Lopez-Paz, 2021). Zhang et al. (2022) attribute this poor performance to the numerical difficulty of optimizing the complicated objective associated with these algorithms. They propose to work around these optimization problems by providing initial weights that already extract a rich palette of potentially interesting features constructed using the RFC algorithm.

Following Zhang et al. (2022), we use the CAMELYON17 tumor classification dataset (Bandi et al., 2018) which contains medical images collected from five hospitals with potentially different devices and procedures. As suggested in Koh et al. (2021), we use the first three hospitals as training environments and the fifth hospital for testing. OOD-tuned results are obtained by using the fourth hospital to tune the various hyper-parameters. IID-tuned results only use the training distributions (see details in appendix E). The purpose of our experiments is to investigate whether initializing with the DISTILL or CAT algorithm provides a computationally attractive alternative to RFC.

Table 3 compares the test performance achieved by two algorithms, VREX and ERM, after initializing with ERM, CATn, DISTILLn, and RFC2, in both the IID-tune and OOD-tune scenarios. The CAT and DISTILL initialization perform better than ERM but not as well as RFC. This result clearly shows the need to research better ways to train networks in a manner that yields diverse representations. Although this contribution shows that simply changing the seed (as in CAT and DISTILL) can achieve good results, the experience of deep ensembles (Gontijo-Lopes et al., 2022) suggests that more refined diversification methods might yield substantially better representations.

9 CONCLUSION

We have presented an ensemble of experimental results for scenarios involving changing task and distributions such as transfer learning, few shots learning, and cross-domain robust learning. These results are quite good in their own right. But more importantly, they show that such scenarios are better served by representations that are richer than those obtained with a single optimization episode. This observation provides a lot of room for new representation learning algorithms that move away from the currently dominant scheme, that is, relying solely on a single optimization episode.

ACKNOWLEGMENTS

Anonymized.

REFERENCES

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. arXiv, 2020.

Peter Bandi, Oscar Geessink, Quirine Manson, Marcory Van Dijk, Maschenka Balkenhol, Meyke Hermsen, Babak Ehteshami Bejnordi, Byungjae Lee, Kyunghyun Paeng, Aoxiao Zhong, et al. From detection of individual metastases to classification of lymph node status at the patient level: the camelyon17 challenge. IEEE Transactions on Medical Imaging, 2018.

Hakan Bilen and Andrea Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. arXiv preprint arXiv:1701.07275, 2017.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. On the opportunities and risks of foundation models. CoRR, abs/2108.07258, 2021. URL https://arxiv.org/abs/2108.07258.

Léon Bottou. From machine learning to machine reasoning. Technical report, arXiv:1102.1808, February 2011.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. Siam Review, 60(2):223–311, 2018.

Jame Bromley, Jim W. Bentz, Léon Bottou, Isabelle Guyon, Yann Le Cun, C. Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a siamese time delay neural network. International Journal of Pattern Recognition and Artificial Intelligence, 7(4), 1993. URL http://leon.bottou.org/papers/bromley-bentz-93.

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. Advances in Neural Information Processing Systems, 33:9912–9924, 2020.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In International conference on machine learning, pages 1597–1607. PMLR, 2020.

Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. arXiv preprint arXiv:1904.04232, 2019.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. arXiv preprint arXiv:2011.10566, 2020.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. Journal of Machine Learning Research, 12: 2493–2537, Aug 2011.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.

Nikita Dvornik, Cordelia Schmid, and Julien Mairal. Selecting relevant features from a multi-domain representation for few-shot classification. In European Conference on Computer Vision, pages 769–786. Springer, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning, pages 1126–1135. PMLR, 2017.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. Nature Machine Intelligence, 2(11):665–673, 2020.

Raphael Gontijo-Lopes, Yann Dauphin, and Ekin D Cubuk. No one representation to rule them all: Overlapping features of training methods. arXiv preprint arXiv:2110.12899, 2021.

Raphael Gontijo-Lopes, Yann Dauphin, and Ekin Dogus Cubuk. No one representation to rule them all: Overlapping features of training methods. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=BK-4qbGgIE3.

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.

Priya Goyal, Quentin Duval, Isaac Seessel, Mathilde Caron, Mannat Singh, Ishan Misra, Levent Sagun, Armand Joulin, and Piotr Bojanowski. Vision models are more robust and fair when pretrained on uncurated images without supervision. arXiv preprint arXiv:2202.08360, 2022.

Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=lQdXeXDoWtI.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016b.

Ursula Hebert-Johnson, Michael Kim, Omer Reingold, and Guy Rothblum. Multicalibration: Calibration for the (Computationally-identifiable) masses. In Jennifer Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 1939–1948. PMLR, 10–15 Jul 2018.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2(7), 2015.

Zeyi Huang, Haohan Wang, Eric P Xing, and Dong Huang. Self-challenging improves cross-domain generalization. In European Conference on Computer Vision, pages 124–140. Springer, 2020.

Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning. arXiv preprint arXiv:2110.09348, 2021.

Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Last layer re-training is sufficient for robustness to spurious correlations. In ICML 2022: Workshop on Spurious Correlations, Invariance and Stability, 2022. URL https://openreview.net/forum?id=THOOBy1uWVH.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In International Conference on Machine Learning, pages 5637–5664. PMLR, 2021.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex). arXiv, 2020.

Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=UYneFzXSJWh.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. Advances in neural information processing systems, 30, 2017.

Wei-Hong Li, Xialei Liu, and Hakan Bilen. Universal representation learning from multiple domains for few-shot classification. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 9526–9535, 2021.

Wei-Hong Li, Xialei Liu, and Hakan Bilen. Universal representations: A unified look at multiple task and domain learning. arXiv preprint arXiv:2204.02744, 2022.

M. Minsky and S. Papert. Perceptrons. MIT Press, Cambridge, MA, 1969.

Maxime Oquab, Léon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In Proceedings of Computer Vision and Pattern Recognition (CVPR). IEEE, 2014.

Vardan Papyan, X. Y. Han, and David L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. Proceedings of the National Academy of Sciences, 117(40): 24652–24663, 2020. doi: 10.1073/pnas.2015509117. URL https://www.pnas.org/doi/abs/10.1073/pnas.2015509117.

Mohammad Pezeshki, Oumar Kaba, Yoshua Bengio, Aaron C Courville, Doina Precup, and Guillaume Lajoie. Gradient starvation: A learning proclivity in neural networks. Advances in Neural Information Processing Systems, 34:1256–1272, 2021.

Alexandre Rame, Corentin Dancette, and Matthieu Cord. Fishr: Invariant gradient variances for out-of-distribution generalization. In International Conference on Machine Learning, pages 18347–18377. PMLR, 2022.

F. Rosenblatt. The perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Project PARA, Cornell Aeronautical Lab, 1957.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Parallel distributed processing: Explorations in the microstructure of cognition, volume I, pages 318–362. Bradford Books, Cambridge, MA, 1986.

Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. In International Conference on Learning Representations (ICLR), 2020.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. arXiv preprint arXiv:1703.00810, 2017.

J.C. Simon. From Pixels to Features. North Holland, August 1989.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf.

Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. arXiv preprint arXiv:2106.10270, 2021.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1199–1208, 2018.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In International Conference on Learning Representations, 2014. URL http://arxiv.org/abs/1312.6199.

Damien Teney, Ehsan Abbasnejad, Simon Lucey, and Anton van den Hengel. Evading the simplicity bias: Training a diverse set of models discovers solutions with superior ood generalization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 16761–16772, 2022.

Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 8769–8778, 2018.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray Kavukcuoglu, and Daan Wierstra. Matching Networks for One Shot Learning. In D Lee, M Sugiyama, U Luxburg, I Guyon, and R Garnett, editors, Advances in Neural Information Processing Systems, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 dataset. Technical report, California Institute of Technology, 2011.

Yoav Wald, Amir Feder, Daniel Greenfeld, and Uri Shalit. On calibration and out-of-domain generalization. arXiv preprint arXiv:2102.10395, 2021.

Hongyu Wang, Eibe Frank, Bernhard Pfahringer, Michael Mayo, and Geoffrey Holmes. Cross-domain few-shot meta-learning using stacking. arXiv preprint arXiv:2205.05831, 2022.

Jianyu Zhang, David Lopez-Paz, and Leon Bottou. Rich feature construction for the optimization-generalization dilemma. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, Proceedings of the 39th International Conference on Machine Learning, volume 162 of Proceedings of Machine Learning Research, pages 26397–26411. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/zhang22u.html.

LEARNING USEFUL REPRESENTATIONS FOR SHIFTING TASKS AND DISTRIBUTIONS

Supplementary Material

A CIFAR SUPERVISED TRANSFER LEARNING

CIFAR10 supervised transfer learning experiments train a RESNET18 network on the CIFAR10 dataset with/without L2 weight decay (4e-5) for 200 epochs. During training, we use a SGD optimizer (Bottou et al., 2018) with momentum=0.9, initial learning rate=0.1, cosine learning rate decay, and batch size=128. As to data augmentation, we use RANDOMRESIZEDCROP (crop scale in $[0.8, 1.0]$), aspect ratio in $[3/4, 4/3]$) and RANDOMHORIZONTALFLIP. During testing, the input images are resized to 36×36 by bicubic interpolation and CENTERCROPED to 32×32. All input images are normalized by $mean = (0.4914, 0.4822, 0.4465)$, $std = (0.2023, 0.1994, 0.2010)$ at the end.

Then transfer the learned representation to CIFAR100 dataset by training a last-layer linear classifier (linear probing). The linear layer weights are initialized by Gaussian distribution $\mathcal{N}(0, 0.01)$. The linear probing process shares the same training hyper-parameters as the supervised training part except for a zero L2 weight decay in all cases.

The CIFAR100 supervised transfer learning experiments swap the order of CIFAR100 and CIFAR10.

B IMAGENET SUPERVISED TRANSFER LEARNING

B.1 EXPERIMENT SETTINGS

Image Preprocessing: Following He et al. (2016b), we use RANDOMHORIZONTALFLIP and RANDOMRESIZEDCROP augmentations for all training tasks. For IMAGENET and INAT18, the input images are normalized by $mean = (0.485, 0.456, 0.406)$, $std = (0.229, 0.224, 0.225)$. For CIFAR, we use the same setting as Appendix A.

IMAGENET Pretraining: The RESNETs are pre-trained on IMAGENET with the popular protocol of Goyal et al. (2017): a SGD optimizer with momentum=0.9, initial learning rate=0.1, batch size=256, L2 weight decay=1e-4, and 90 training epochs. The learning rate is multiplied by 0.1 every 30 epochs. By default, the optimizer in all experiments is SGD with momentum=0.9.

DISTILL : To distill the CATn representations $[\phi_1, \dots \phi_n]$ ($n\times$RESNET50) into a smaller representation Φ (RESNET50), we use the multi-head architecture as Figure 2. Inspired by Hinton et al. (2015), we use the Kullback–Leibler divergence loss to learn Φ as:

$$\min_{\Phi, w_0, \dots, w_n} \sum_{i=0}^{n} \sum_{x} \left[\tau^2 \mathcal{L}_{kl} \Big(s_\tau \big(v_i \circ \phi_i(x) \big) \;||\; w_i \circ \Phi(x) \Big) \right], \tag{2}$$

where $s_\tau(v)_i = \frac{e^{v_i/\tau}}{\sum_k e^{v_k/\tau}}$ is a softmax function with temperature τ, v_i is the learned last-layer classifier of i^{th} sub-network of CATn.

In the DISTILL experiments, we distill five separately trained RESNET50 into one RESNET50 according to Eq 2 with $\tau = 10$. We use a SGD optimizer with momentum=0.9, batch size=2048, and weight decay=0. The initial learning rate is 0.1 and warms up to 0.8 within the first 5 epochs. Then learning rate decays to 0.16 and 0.032 at 210^{th} and 240^{th} epochs, respectively. The total training epochs is 270.

Linear probing:

- **IMAGENET:** The IMAGENET linear probing experiments train a linear classifier with the same hyper-parameters as IMAGENET pretraining. By default, the last linear classifier in all linear probing experiments is initialized by $\mathcal{N}(0, 0.01)$.

- **INAT18, CIFAR100, CIFAR10:** Following the settings of Goyal et al. (2022), the linear probing experiments (on INAT18, CIFAR100, CIFAR10) adds a BATCHNORM layer before the linear classifier to reduce the hyper-parameter tuning difficulty. The learning rate is initialized to 0.01 and multiplied by 0.1 every 8 epochs. Then train these linear probing tasks for 28 epochs by SGD Nesterov optimizer with momentum=0.9, batch size 256. Note that BATCHNORM + a linear classifier is still a linear classifier during inference. We tune L2 weight decay from {1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2} for CIFAR100 and CIFAR10, {1e-6, 1e-5, 1e-4} for INAT18.

Fine-tuning: As to the fine-tuning experiments (on CIFAR100, CIFAR10, and INAT18), we tune the initial learning rate from {0.005, 0.01, 0.05}, training epochs from {50, 100}. We further tune L2 weight decay from {0, 1e-5, 1e-4, 5e-4} for CIFAR100 and CIFAR10, {1e-6, 1e-5, 1e-4} for INAT18. A cosine learning rate scheduler is used in fine-tuning experiments. A 0.01 learning rate and 100 training epochs usually provide the best performance for these three datasets. So we fix these two hyperparameters in the following supervised learning two-stage fine-tuning experiments and self-supervised learning experiments.

Two-stage fine-tuning: For the two-stage fine-tuning experiments, we separately fine-tune each sub-network (i.e. RESNET50) of the CATn architecture by the same protocol as the normal fine-tuning above. Then train a last-layer linear classifier on top of the concatenated fine-tuned representation. The last-layer linear classifier training can be very efficient with a proper weights initialization strategy. In this work, we initialize the last-layer classifier w (including the bias term) by concatenating the last-layer classifier of each fine-tuned sub-network w_i, $w \leftarrow [w_0^\top, \ldots, w_n^\top]^\top / n$, Then we only need to train the last-layer classifier w for 1 epoch with a learning rate = 0.001.

B.2 EXPERIMENTS ON A DEEPER ARCHITECTURE: RESNET152

Similar to table 2 in section 5, table 4 provides similar experiments on a deeper architecture RESNET152. CATn exceeds ERM on IMAGENET, CIFAR10, CIFAR100 , and INAT18 linear probing tasks.

Table 4: Imagenet supervised transfer learning performance on a deep architecture RESNET152.

method	architecture	ID	Linear Probing (OOD)		
		IMAGENET	CIFAR10	CIFAR100	INAT18
ERM	RESNET152	77.89	92.50	76.23	39.70
CAT2	2×RESNET152	79.34	94.26	79.15	45.42
CAT5	5×RESNET152	80.14	94.91	81.35	50.32
CAT10	10×RESNET152	80.18	95.38	82.39	52.73

B.3 FINE TUNING EXPERIMENTS

For reference, table 5 provides numerical results for the fine-tuning experiments of Figure 3.

B.4 VISION TRANSFORMER EXPERIMENT SETTINGS

For all vision transformer experiments, we keep the input image resolution at 384 × 384. For linear-probing and (2-stage) fine-tuning, we follows a similar protocol as the one in {CIFAR10, CIFAR100, INAT18}. Specifically, we use a weight decay=5e-4 and a batch size=256 for linear probing, a weight decay=0 and a batch size=512 (following the same settings as Dosovitskiy et al. (2020)) for (2-stage) fine-tuning. Following Dosovitskiy et al. (2020), all input images are normalized by $mean = (0.5, 0.5, 0.5), std = (0.5, 0.5, 0.5)$.

Table 5: Supervised transfer learning by either normal fine-tuning or proposed two-stage fine-tuning. Various representations are pre-trained on IMAGENET and then fine-tuned or two-stage fine-tuned on CIFAR10, CIFAR100, INAT18 tasks.

			fine-tuning			two-stage fine-tuning		
method	architecture	params	CIFAR10	CIFAR100	INAT18	CIFAR10	CIFAR100	INAT18
ERM	RESNET50	23.5M	97.54	85.58	64.19	-	-	-
ERM	RESNET50W2	93.9M	97.76	87.13	66.72	-	-	-
ERM	RESNET50W4	375M	97.88	87.95	66.99	-	-	-
ERM	2RESNET50	47M	97.39	85.77	62.57	-	-	-
ERM	4RESNET50	94M	97.38	85.56	61.58	-	-	-
CAT2	RESNET50	47M	97.56	86.04	64.49	97.87	87.07	67.11
CAT4	RESNET50	94M	97.53	86.54	64.54	98.14	88.00	68.16
CAT5	RESNET50	118M	97.57	86.46	64.86	98.19	88.11	68.27
CAT10	RESNET50	235M	97.19	86.65	64.39	98.17	88.50	68.32
DISTILL5	RESNET50	23.5M	97.07	85.31	64.17	-	-	-

C SELF-SUPERVISED TRANSFER LEARNING

C.1 SWAV ON IMAGENET

SWAV is a contrastive self-supervised learning algorithm proposed by Caron et al. (2020). We train RESNET50 on IMAGENET[4] by the SWAV algorithm four times, which gives us four pretrained RESNET50 models. As to the rest four SWAV pre-trained models in this work, we use the public available RESNET50[5], RESNET50W2[6], RESNET50W4[7], and RESNET50W5[8] checkpoints.

Linear probing: Following the settings in Goyal et al. (2022), the linear probing experiments (on IMAGENET, INAT18, CIFAR100, CIFAR10) adds a BATCHNORM layer before the last-layer linear classifier to reduce the hyper-parameter tuning difficulty. The learning rate is initialized to 0.01 and multiplied by 0.1 every 8 epochs. Then train these linear probing tasks for 28 epochs by SGD Nesterov optimizer with momentum=0.9. We search L2 weight decay from $\{5e-4\}$, $\{5e-4, 1e-3, 5e-3, 1e-2\}$, and $\{1e-6, 1e-5, 1e-4\}$ for IMAGENET, CIFAR, and INAT18 tasks, respectively.

Fine-tuning:

- **IMAGENET**: Inspired by the semi-supervised IMAGENET fine-tuning settings in Caron et al. (2020), we attach a randomly initialized last-layer classifier on top of the SSL learned representation. Then fine-tune all parameters, using a SGD optimizer with momentum=0.9 and L2 weight decay=0. Low-layers representation and last-layer classifier use different initial learning rates of 0.01 and 0.2, respectively. The learning rate is multiplied by 0.2 at 12^{th} and 16^{th} epochs. We train 20 epochs for networks: RESNET50, RESNET50W2, RESNET50W4. We further search training epochs from $\{10, 20\}$ for the wide network (due to overfitting), RESNET50W5 and then select the best one with 10 training epochs.
- **CIFAR10, CIFAR100, INAT18**: Same as the fine-tuning settings in supervised transfer learning in Appendix B.1.

Two-stage fine-tuning:

[4] https://github.com/facebookresearch/swav/blob/main/scripts/swav_400ep_pretrain.sh

[5] https://dl.fbaipublicfiles.com/deepcluster/swav_400ep_pretrain.pth.tar

[6] https://dl.fbaipublicfiles.com/deepcluster/swav_RN50w2_400ep_pretrain.pth.tar

[7] https://dl.fbaipublicfiles.com/deepcluster/swav_RN50w4_400ep_pretrain.pth.tar

[8] https://dl.fbaipublicfiles.com/deepcluster/swav_RN50w5_400ep_pretrain.pth.tar

- **IMAGENET**: Similar to the two-stage fine-tuning settings in supervised transfer learning, we initialize the last-layer classifier w by concatenation and then train 1 epoch with learning rate=0.001, L2 weight decay=0.
- **CIFAR10, CIFAR100, INAT18**: For CIFAR10, CIFAR100, we use same two-stage fine-tuning settings as in supervised transfer learning in Appendix B.1. For INAT18, we attach a BATCHNORM layer before the last-layer linear classifier to reduce the training difficulty. Note that BATCHNORM + a linear classifier is still a linear classifier during inference. Following the linear probing protocol, we train the BATCHNORM and linear layers by a SGD optimizer with momentum=0.9, initial learning rate=0.01, and a 0.2 learning rate decay at 12^{th} and 16^{th} epochs. As to L2 weight decay, we use the same searching space as in the fine-tuning.

C.1.1 ADDITIONAL RESULTS

Beside the SWAV IMAGENET fine-tuning experiments of figure 6, Figure 8 provides further fine-tune / two-stage fine-tune various SWAV pretrained RESNETS on NATURALIST18, CIFAR100, and CIFAR10 tasks. We give the "cat(ft)" curve on the IMAGENET task, but omit the curves on other tasks (NATURALIST18, CIFAR100, and CIFAR10) because they are computational costly.

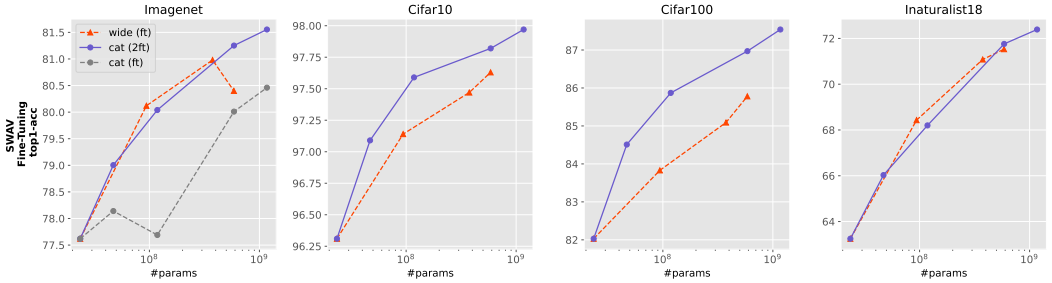

Figure 8: Fine-tuning performance of SWAV on IMAGENET, NATURALIST18, CIFAR100, and CIFAR10 tasks. SWAV is trained on unlabeled IMAGENET. "cat(2ft)" and "cat(ft)" indicate our two-stage fine-tuning strategy and the normal fine-tuning strategy on n concatenated networks. "wide(ft))" refers to the normal fine-tuning strategy on wide networks, i.e. RESNET50, RESNET50W2, RESNET50W4, and RESNET50W5.

C.2 SEER ON INSTAGRAM1B

SEER (Goyal et al., 2022) trains large REGNET{32GF, 64GF, 128GF, 256GF, 10B} architectures on the INSTAGRAM1B dataset with 1 billion Instagram images, using the SWAV contrastive self-supervised learning algorithm.

Linear Probing: Same as the linear probing settings in SWAV.

Fine-tuning: We use SEER checkpoints [9] fine-tuned on IMAGENET with 384×384 resolutions. It is fine-tuned on IMAGENET for 15 epochs using SGD momentum 0.9, weight decay 1e-4, learning rate 0.04 and batch size 256. The learning rate is multiplied by 0.1 at 8^{th} and 12^{th} epochs.

Two-stage Fine-tuning: We keep L2 weight decay 1e-4 the same as fine-tuning. Then follow the other settings as in SWAV.

C.3 ADDITIONAL EXPERIMENT: SIMSIAM ON CIFAR

SIMSIAM Chen and He (2020) is a non-contrastive self-supervised learning algorithm. In this section, we pre-train the networks using SIMSIAM on CIFAR10, when transfer the learned representation by linear probing to CIFAR10, CIFAR100, CIFAR10 with 1% training examples, and CIFAR100 with 10% training examples.

[9] https://github.com/facebookresearch/vissl/tree/main/projects/SEER

SimSiam Pretraining Following Chen and He (2020) we pretrain RESNET18, RESNET18W2, RESNET18W4, 2RESNET18, and 4RESNET18 on CIFAR10 (32×32 resolution) by Simsiam for 800 epochs, using a SGD optimizer with momentum = 0.9, initial learning rate = 0.06, batch size = 512, L2 weight decay = $5e-4$, and cosine learning rate scheduler. The data augmentations include RANDOMRESIZEDCROP (crop scale in $[0.2, 1]$), RANDOMHORIZONTALFLIP, RANDOMGRAYSCALE ($p = 0.2$), and a random applied COLORJITTER $(0.4, 0.4, 0.4, 0.1)$ with probability 0.8. All images are normalized by $mean = (0.4914, 0.4822, 0.4465), std = (0.2023, 0.1994, 0.2010)$ before training.

DISTILL Since self-supervised learning tasks don't contain target labels as supervised learning, we apply knowledge distillation on representation directly. Specifically, we set $v_1, \dots v_n$ in Figure 2 (left) as Identity matrices. Then we distill $[\phi_1, \dots, \phi_n]$ into Φ by use a cosine loss:

$$\min_{\Phi, w_0, \dots, w_n} \sum_{i=0}^{n} \sum_{x} \left[1 - \cos\Big(\phi_i(x) \ , \ w_i \circ \Phi(x) \Big) \right] \tag{3}$$

Linear Probing: Following again the settings of Goyal et al. (2022), the linear probing experiments (on CIFAR100, CIFAR10, CIFAR100(1%) with 10% training data, and CIFAR10(1%) with 1% training data) adds a BATCHNORM layer before the last-layer linear classifier to reduce the hyper-parameter tuning difficulty. We use batch size = 256 for CIFAR100 and CIFAR10, use batch size = 32 for corresponding sampled (10%/1%)version. Then we search initial learning rate from $\{0.1, 0.01\}$, L2 weight decay from {1e-4, 5e-4, 1e-3, 5e-3}. The learning rate is multiplied by 0.1 every 8 epochs during the total 28 training epochs. As to the optimizer, all experiments use a SGD Nesterov optimizer with momentum=0.9.

Results Table 6 shows the linear probing accuracy of Simsiam learned representation on various datasets and architectures. When linear probing on the same CIFAR10 dataset as training, the CATn method performs slightly better than width architectures (e.g. RESNET18W2 and RESNET18W4). When comparing them on the CIFAR100 dataset (OOD), however, CATn exceeds width architectures.

Table 6: Linear probing accuracy of SIMSIAM (Chen and He, 2020) CIFAR10 learned representation on CIFAR100, CIFAR10, CIFAR100(1%), and CIFAR10(10%) tasks. CATn concatenates n learned representation before linear probing. DISTILLn distills n learned representation into RESNET18 before linear probing. RESNET18Wn contains around n^2 parameters as RESNET18.

method	architecture	Linear Probing (ID)		Linear Probing (OOD)	
		CIFAR10	CIFAR10(1%)	CIFAR100	CIFAR100(10%)
Simsiam	RESNET18	91.88	87.60	55.29	42.93
Simsiam	RESNET18W2	92.88	88.95	59.41	45.39
Simsiam	RESNET18W4	93.50	90.45	59.28	44.98
Simsiam	2RESNET18	91.62	87.14	55.67	43.07
Simsiam	4RESNET18	92.54	85.65	64.42	49.65
CAT2	2×RESNET18	92.94	88.32	59.40	46.06
CAT4	4×RESNET18	93.42	88.81	63.06	47.48
CAT5	5×RESNET18	93.67	88.78	63.71	48.31
CAT10	10×RESNET18	93.75	88.65	66.19	49.90
DISTILL2	2×RESNET18	93.04	88.59	59.65	45.10
DISTILL5	5×RESNET18	93.02	88.56	60.79	46.41
DISTILL10	10×RESNET18	93.11	88.72	61.35	46.75

C.4 NUMERICAL RESULTS

For reference, Tables 7 and 8 provide the numerical results for the linear probing, fine-tuning, and two-stage fine-tuning plots of Figure 6.

Table 7: Linear probing, fine-tuning, and two-stage fine-tuning performance of SWAV pre-trained representation and corresponding CATn representations.

method	architecture	params	Linear Probing IMAGENET	CIFAR10	CIFAR100	INAT18	finetune IMAGENET	two-stage ft IMAGENET
SWAV	RESNET50	23.5M	74.30	91.83	76.85	42.35	77.62	-
SWAV	RESNET50W2	93.9M	77.31	93.97	79.49	47.55	80.12	-
SWAV	RESNET50W4	375M	77.48	94.29	80.51	44.13	80.98	-
SWAV	RESNET50W5	586M	78.23	94.84	81.54	48.11	80.40	-
CAT2	-	47M	76.01	93.48	78.91	45.57	78.14	80.40
CAT5	-	118M	77.43	94.62	81.11	49.12	77.69	80.04
CAT7	-	587M	78.72	95.59	82.71	49.68	80.05	81.25
CAT9	-	1170M	78.89	95.76	83.16	50.61	80.46	81.55

Table 8: Linear probing, fine-tuning, and two-stage fine-tuning performance of SEER pre-trained representation and corresponding CATn representations.

method	architecture	params	Linear Probing IMAGENET	CIFAR10	CIFAR100	INAT18	finetune IMAGENET (384px)	two-stage ft IMAGENET (384px)
SEER	REGNET32GF	141M	73.4	89.94	71.53	39.10	83.4	-
SEER	REGNET64GF	276M	74.9	90.90	73.78	42.69	84.0	-
SEER	REGNET128GF	637M	75.9	91.37	74.75	43.51	84.5	-
SEER	REGNET256GF	1270M	77.5	92.16	74.93	46.91	85.2	-
CAT2	-	418M	76.0	92.16	75.65	45.36	-	84.5
CAT3	-	1060M	77.3	93.15	77.26	47.18	-	85.1
CAT4	-	2330M	78.3	93.59	78.80	48.68	-	85.5

D META-LEARNING / FEW-SHOTS LEARNING

D.1 DATASETS

CUB (Wah et al., 2011) dataset contains $11,788$ images of 200 birds classes, 100 classes ($5,994$ images) for training and 100 classes ($5,794$ images) for testing.

MINIIMAGENET (Vinyals et al., 2016) dataset contains $60,000$ images of 100 classes with 600 images per class, 64 classes for training, 36 classes for testing.

D.2 BASELINE AND BASELINE++ EXPERIMENT SETTINGS

For BASELINE and BASELINE++ experiments, following Chen et al. (2019), we use RANDOMSIZEDCROP, IMAGEJITTER(0.4, 0.4, 0.4), and HORIZONTALFLIP augmentations, as well as a image normalization $mean = (0.485, 0.456, 0.406)$, $std = (0.229, 0.224, 0.225)$. Then use an ADAM optimizer with learning rate = 0.001, batch size = 16, input image size = 224×224. Finally, train RESNET18 on CUB and MINIIMAGENET datasets for 200 and 400 epochs, respectively. We further tune L2 weight decay from {0, 1e-5, 1e-4, 1e-3, 1e-2} and choose 1e-4 for CUB, 1e-5 for MINIIMAGENET experiments. Compared with the BASELINE and BASELINE++ performance reported by Chen et al. (2019) (table A5), this L2 weight decay tuning process provides $\sim 5\%$ and $\sim 1\%$ improvement on MINIIMAGENET 5way-1shot and 5way-5shot, respectively. In this work, we use this stronger setting in baseline methods.

As to the few-shots learning evaluation, following Chen et al. (2019), we scale images by a factor of 1.15, CENTERCROP, and normalization. Then randomly sample 1 or 5 images from 5 random classes from the test set (5way-1shot and 5way-5shot). Finally, train a linear classifier on top of the learned representation with a SGD optimizer, momentum = 0.9, dampening = 0.9, learning rate = 0.1, L2 weight decay = 1e-3, batch size = 4, and epochs = 100. We take the average of 600 such evaluation processes as the test score.

The BASELINE and BASELINE++ results in Figure 7 report the mean of five runs with different training and evaluating seeds.

Implementation details of the cosine classifier Here we summarize the technical details of the cosine classifier implementation used in this work which follows Chen et al. (2019)[10].

Denote the representation vector as z. The cosine classifier calculates the i^{th} element of logits by:

$$h_i = g_i \frac{\langle u_i, z \rangle}{||u_i|| ||z||} \tag{4}$$

where u_i is a vector with the same dimension of z, g_i is a scalar, h_i is i^{th} element of logits h.

Then minimize the cross entropy loss between the target label y and softmax output $s(h)$ by updating w and g: $\min_{w,g} \mathcal{L}_{ce}(y, s(h))$.

D.3 CAT AND DISTILL EXPERIMENT SETTINGS

For CAT, we concatenate n representation separately trained by either BASELINE or BASELINE++ as the settings above. For DISTILL, we use the same multi-head architecture as figure 2 together with a cross-entropy loss function:

$$\min_{\Phi, w_0, \ldots, w_n} \sum_{i=0}^{n} \sum_{x} \left[(1-\alpha) \mathcal{L}_{ce}\Big(s\big(w_i \circ \Phi(x)\big), y\Big) + \alpha \tau^2 \mathcal{L}_{kl}\Big(s_\tau\big(v_i \circ \phi_i(x)\big) \;||\; w_i \circ \Phi(x)\Big)\right] \tag{5}$$

, where $\mathcal{L}_{ce}$ indicates a cross-entropy loss, α is a trade-off parameter between cross-entropy loss and kl-divergence loss. We set L2 weight decay = 0, $\tau = 10$, search $\alpha \in \{0.8, 0.9, 1\}$, and keep the other hyper-parameters as Appendix D.2. We find the impact of α is limited in both CUB ($\leq 1\%$) and MINIIMAGENET ($\leq 0.5\%$) tasks.

D.4 SNAPSHOTS EXPERIMENT SETTINGS

The previous section shows the CATn and DISTILLn methods applied on n separately trained RESNET18 representations. In this section, we apply DISTILLn on n snapshots sampled from one training episode.

We train CUB and MINIIMAGENET respectively for 1000 and 1200 epochs by naive SGD optimizer with a relevant large learning rate 0.8. Then we sample 5 snapshots, $\{200^{th}, 400^{th}, 600^{th}, 800^{th}, 1000^{th}\}$ and $\{400^{th}, 600^{th}, 800^{th}, 1000^{th}, 1200^{th}\}$, for CUB and MINIIMAGENET, respectively. The other hyper-parameters are the same as Appendix D.2.

D.5 MORE EXPERIMENTAL RESULTS

Table 10 provides the exact number in Figure 7, as well as additional CATn and DISTILLn few-shots learning results with a linear classifier (The orange and gray bars in figure 7 report the few-shots learning performance with a cosine classifier).

Table 9 provides more DISTILL5-s results with either a linear classifier or a cosine-based classifier.

E OUT-OF-DISTRIBUTION LEARNING

Following Zhang et al. (2022), we use the CAMELYON17 (Koh et al., 2021) task to showcase the CAT and DISTILL constricted representation in out-of-distribution learning scenario. The first row of table 3 is copied from Zhang et al. (2022). The rest results use a frozen pre-trained representation, either by concatenating n ERM pre-trained representations (CATn), distilling of n ERM pre-trained representations (DISTILLn), or RFC constructed representations (RFC2). Then train a linear classifier on top of the representation by vREx or ERM algorithms.

For the vREx algorithm, we search the penalty weights from {0.5, 1, 5, 10, 50, 100}. For DISTILLn representations in the CAMELYON17 task, we follow the Algorithm 2 in Zhang et al. (2022), but

[10] https://github.com/wyharveychen/CloserLookFewShot/blob/master/backbone.py#L22

Table 9: Few-shot learning performance of the distillation method on CUB and MINIIMAGENET for both the linear and cosine-based linear classifier. The DISTILL5 results were obtained using five RESNET18 trained from scratch with different seeds. The DISTILL5-S results were obtained using five snapshots taken during a single training episode with a relatively high step size.

		MINIIMAGENET		CUB	
	classifier	5way 1shot	5way 5shot	5way 1shot	5way 5shot
DISTILL5	linear	59.7 ± 0.6	80.5 ± 0.3	71.0 ± 0.3	88.5 ± 0.1
DISTILL5	cosine	62.9 ± 0.4	81.5 ± 0.3	75.7 ± 0.7	89.2 ± 0.2
DISTILL5-S	linear	59.9 ± 0.5	80.8 ± 0.4	68.4 ± 0.5	87.2 ± 0.4
DISTILL5-S	cosine	62.0 ± 0.5	81.0 ± 0.3	75.2 ± 0.8	88.6 ± 0.4

Table 10: Few-shot learning performance on CUB and MINIIMAGENET dataset with either a linear classifier or cosine-distance based classifier. Standard deviations over five repeats are reported.

			CUB		MINIIMAGENET	
	architecture	classifier	5way 5shot	5way 1shot	5way 5shot	5way 1shot
supervised	RESNET18	linear	63.37±1.66	83.47±1.23	55.20±0.68	76.52±0.42
CAT2	2×RESNET18	linear	66.25±0.85	85.50±0.34	57.30±0.31	78.42±0.17
CAT5	5×RESNET18	linear	67.00±0.18	86.80±0.10	58.40±0.25	79.59±0.17
DISTILL2	RESNET18	linear	69.93±0.74	87.72±0.31	58.99±0.32	79.73±0.21
DISTILL5	RESNET18	linear	70.99±0.31	88.52±0.14	59.66±0.59	80.53±0.27
supervised	RESNET18	cosine	69.19±0.88	84.41±0.49	57.47±0.45	76.47±0.27
CAT2	2×RESNET18	cosine	72.87±0.43	86.82±0.17	60.69±0.24	79.29±0.23
CAT5	5×RESNET18	cosine	76.23±0.55	88.87±0.40	63.63±0.23	81.22±0.17
DISTILL2	RESNET18	cosine	74.81±0.45	88.14±0.40	61.95±0.11	80.79±0.26
DISTILL5	RESNET18	cosine	76.20±0.39	89.18±0.24	62.89±0.38	81.49±0.26

use a slightly different dataset balance trick in the loss function (Zhang et al. (2022) Algorithm 2 line 13-14). We instead balance two kinds of examples: one share the same predictions on all ERM pre-trained models, one doesn't. We keep other settings to be the same as Zhang et al. (2022)[11].

F REPLICABILITY

Code will of course be provided on github to replicate these experiments.

As explained in the paper, many of our experiments are based the setups defined by other authors and rely on the code and the data files they provide. We are therefore very conscious of the value of making well organized replication resources available to the community.

[11] https://github.com/TjuJianyu/RFC