SCALABLE TRANSFER LEARNING WITH EXPERT MODELS

Joan Puigcerver[*] Google Research
Carlos Riquelme[*] Google Research
Basil Mustafa Google Research
Cedric Renggli[†] ETH Zurich

André Susano Pinto Google Research
Sylvain Gelly Google Research
Daniel Keysers Google Research
Neil Houlsby Google Research

ABSTRACT

Transfer of pre-trained representations can improve sample efficiency and reduce computational requirements for new tasks. However, representations used for transfer are usually generic, and are not tailored to a particular distribution of downstream tasks. We explore the use of expert representations for transfer with a simple, yet effective, strategy. We train a diverse set of experts by exploiting existing label structures, and use cheap-to-compute performance proxies to select the relevant expert for each target task. This strategy scales the process of transferring to new tasks, since it does not revisit the pre-training data during transfer. Accordingly, it requires little extra compute per target task, and results in a speed-up of 2–3 orders of magnitude compared to competing approaches. Further, we provide an adapter-based architecture able to compress many experts into a single model. We evaluate our approach on two different data sources and demonstrate that it outperforms baselines on over 20 diverse vision tasks in both cases.

1 INTRODUCTION

Deep learning has been successful on many computer vision tasks. Unfortunately, this success often requires a large amount of per-task data and compute. To scale deep learning to new vision tasks, practitioners often turn to transfer learning. Transfer learning involves re-using models trained on a large *source* task, and tuning them on the *target* task. This can improve both convergence rates (Ben-David et al., 2007; 2010; Blitzer et al., 2008; Du et al., 2017; Kuzborskij & Orabona, 2013; Mansour et al., 2009) and empirical performance (Dai et al., 2007; Donahue et al., 2014; Oquab et al., 2014; Tan et al., 2018). Transfer learning reduces *per-task* data or compute requirements, given a large one-off pre-training cost. In practice, this one-off down payment may not be made by the practitioner, since pre-trained networks are made available through platforms like PyTorch and TensorFlow Hub[1]. For instance, ImageNet pre-training is popular since it is freely available and works well for many tasks (Donahue et al., 2014; Oquab et al., 2014; Sharif Razavian et al., 2014).

In contrast to generic homogeneous models (e.g. most pre-trained ImageNet networks), Mixture of Experts (MoE) include multiple heterogeneous sub-models ("experts") that specialize to sub-problems of the full task. MoEs have been studied for decades (Eigen et al., 2013; Jacobs & Jordan, 1993), and have also been successful in deep learning (Shazeer et al., 2017). Yet, the application of experts for deep transfer learning has been less explored. We study visual transfer with experts, and present a simple, scalable, yet effective strategy.

Transfer of specialist models has been studied before. However, they either require expensive re-training on the source dataset for every target task (Ngiam et al., 2018; Yan et al., 2020), or operate at a small scale where all experts can be applied simultaneously (Dvornik et al., 2020). Further, most of them are tested only on a limited suite of natural single-object classification tasks. We lift these

[*] Equal contribution. Order decided by a coin toss.
[†] Work done while interning at Google Research.
[1] `https://pytorch.org/hub/` and `https://tfhub.dev/`, respectively

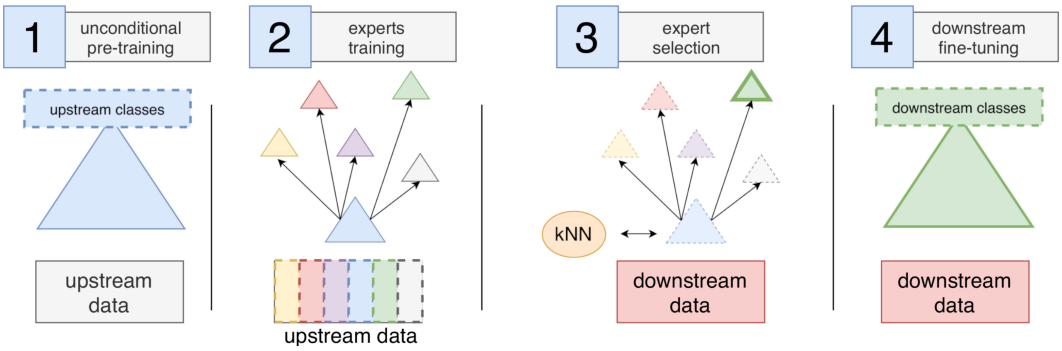


Figure 1: Transfer Learning with Per-Task Routing of Experts. **Step 1.** A single baseline model **B** is trained on the entire upstream dataset. **Step 2.** The upstream data is divided in semantic subsets (possibly overlapping). One expert is trained on each subset using the weights from **B** as initialization. **Step 3.** Given a new downstream task $\mathbf{D}_T = (X_T, Y_T)$, we compute the image representations $M_e(X_T)$ from each expert e. We use kNN to compute the accuracy on the supervised problem $\mathbf{D}_{T,e} = (M_e(X_T), Y_T)$, and select the expert e^* with highest accuracy. **Step 4.** We add a new head to e^* and fine-tune its whole network with the downstream data, leading to the final model.

constraints, and present a practical approach that scales to hundreds of large experts, while requiring relatively little compute *per target task*.

Our strategy consists of four stages (fig. 1). (1) *Unconditional pre-training*. A single baseline model is trained on the entire upstream data. (2) *Experts training*. Multiple experts are pre-trained by exploiting the label hierarchy present in many large-scale image datasets, such as ImageNet and JFT. In addition to entire expert networks, we explore residual adapters that allow all of the expertise to be packed into a single model that can be loaded into memory. These two stages may be expensive, but are done only once. (3) *Expert selection*. Applying all experts to each task does not scale well; some sort of sparsification is required. We focus on inexpensive model selection that can be applied to hundreds or thousands of experts. (4) *Downstream fine-tuning*. We take the output of the model selection phase and tune it on the target task. Importantly, this phase does not require revisiting the source dataset, which may be unavailable or expensive to train on.

We show that this approach yields remarkably strong performance on many diverse tasks. We evaluate not only on classic vision tasks, but also on the diverse VTAB benchmark of 19 tasks (Zhai et al., 2019). Our contributions can be summarized as follows.

- We propose a transfer learning algorithm with a large number of experts based on per-task routing via nearest neighbors selection. Once we have amortized the pre-training cost, this algorithm requires little compute *per target task*, achieving an speed-up of $500\times$–$1000\times$ compared to competing strategies. Also, it can be easily replicated with any large upstream multilabel dataset.
- We achieve a mean accuracy improvement of 3.6% over the state-of-the-art performance on 19 VTAB datasets using ResNet50 networks. Our algorithm offers improvements on every group of tasks: natural, specialized, and structured. Figure 2 summarizes these results.
- We explore using sub-networks as experts via residual adapters, allowing all experts to be packed into a single model. Surprisingly these perform almost as well as their full-network counterparts.

2 Related Work

Transfer Learning. Tasks with little training data can benefit from other larger datasets, often from a similar domain. Transfer learning concerns the link between the source and target dataset (Pan & Yang, 2009; Weiss et al., 2016; Tan et al., 2018; Wang, 2018). One family of methods creates a single training dataset, where source instances are re-weighted according to their relevance (Dai et al., 2007; Pardoe & Stone, 2010; Wan et al., 2011; Xu et al., 2017). A popular method consists of fine-tuning a model that was pre-trained on the source data (Donahue et al., 2014; Oquab et al., 2014; Sharif Razavian et al., 2014). Some transfer learning algorithms condition the initial source model on the target dataset itself (Ngiam et al., 2018; Xie et al., 2019; Yalniz et al., 2019), while

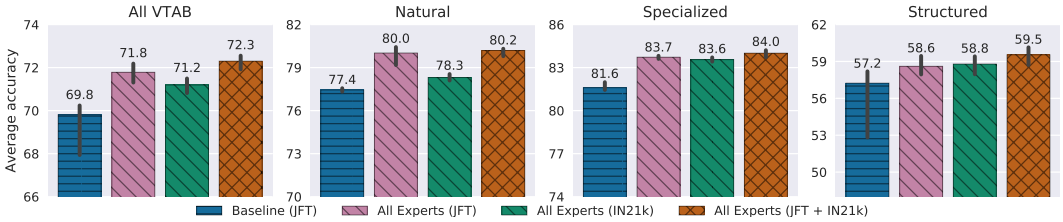


Figure 2: Summary of results on the VTAB-1k benchmark, combining experts with different architectures trained on two different data sources (JFT, ImageNet21k). In each of the 19 datasets, we use the median accuracy over 30 runs. The average of the accuracies in each group is shown, as well as (percentile) bootstrap confidence intervals at the 95% level.

others (like ours) are agnostic about the downstream task when the initial model is trained on the source data (Kolesnikov et al., 2019). We offer an in-depth comparison with (Ngiam et al., 2018) in section 6.6. In the context of few-shot learning, where out-of-the-box fine-tuning may not work, generic representations are sometimes frozen, and simple feature selection (Dvornik et al., 2020) or model training (Chen et al., 2019) techniques are applied on top. Instead of relying on fixed universal representations, (Rebuffi et al., 2017; 2018) use small additional modules, or adapters, that incorporate knowledge from several visual domains. Our work also explores this idea.

Multi-task Learning. MTL tries to leverage the common aspects of several learning tasks (Caruana, 1997). A prominent approach uses explicit parameter sharing; for instance, by means of common low-level layers leading to different heads. Among others, this has been successfully applied to vision (Zhang et al., 2014), language (Liu et al., 2015), and reinforcement learning (Fedus et al., 2019) tasks. In addition, a variety of ways to combine task-specific representations have arisen, such as cross-stitch networks (Misra et al., 2016), or lateral connections (Rusu et al., 2016). A different family of methods impose joint constraints on the –possibly different– models corresponding to each task. We can combine the learning problems via regularization and shared sparsity patterns (Argyriou et al., 2007; Lounici et al., 2009), or by imposing some prior knowledge regarding the task structure (Evgeniou et al., 2005; Jacob et al., 2009; Kim et al., 2012).

3 THE TRANSFER LEARNING FRAMEWORK

In this section, we describe our transfer learning setup of interest. The high-level goal is to train strong models for arbitrary downstream tasks, possibly under severe data and compute limitations. To do so efficiently, one can offload computation to a previous upstream phase which is executed *a priori*, without knowing the downstream tasks in advance. Accordingly, the upstream model should not depend on any specific target data. We are mostly interested in the *low data* regime where downstream tasks contain few datapoints. These restrictions have a practical motivation: we would like to build and deploy universal representations that are easily transferred to a wide range of downstream settings. Any transfer algorithm must implement the following three stages.

Upstream Training. Given the upstream data $\mathbf{D}_U$, the algorithm first outputs a source model $\mathbf{M}$. The goal is to provide useful initial representations for various new tasks. This stage could actually produce a family of models $\{\mathbf{M}_e\}$ rather than a single one. These models might not be disjoint, and could share parameters. The upstream learning problems are auxiliary; accordingly, $\mathbf{D}_U$ could include a diverse set of classification, regression, or even synthetic learning instances.

Model Selection. When a new downstream task is given, a selection algorithm is applied to choose the upstream model(s) to transfer, possibly depending on the downstream data. This phase should be computationally cheap; thus, the upstream data is no longer available. Sometimes, there is no choice to make (say, with a single ImageNet representation). Alternatively, in models with a complex structure, one may choose which parts, routes, or modules to keep in a data-dependent fashion.

Downstream Training. The final stage fine-tunes the selected model using the downstream data, either fully or partially. For neural nets, a new head is added as the output classes are task-specific.

Our overall algorithm is depicted in fig. 1. We give details about each step in the following sections.

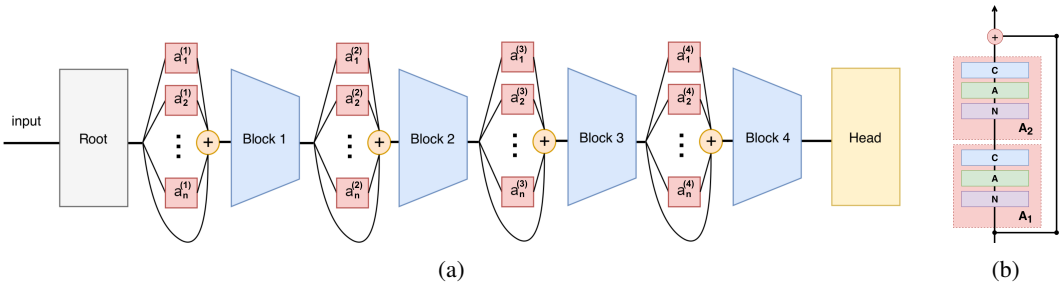


Figure 3: (a) ResNet with expert adapters before all blocks. A layer of experts is placed before every block. (b) Each individual adapter including the overall skip connection. N, A, C stand for (Group) Normalization, (ReLU) Activation, and Convolution layers, respectively.

4 Upstream Training

4.1 Expert Architectures

Our experts should provide feature extractions that are a good starting point to learn future tasks related to the expert's upstream training data. We explore two different model architectures to train such experts. As an obvious choice, we first look at ResNets (He et al., 2016b). These are powerful models; however, storing and deploying many of them can be challenging. As an alternative, we also develop more compact adapter modules that can all be assembled in a single architecture. Also, their individual size can be easily customized to meet memory and computational constraints, which makes them an ideal candidate for combining multiple experts in a single model, when needed. We informally refer to these as *full* and *adapter* modules (or experts), respectively.

Full ResNet Modules. As a base architecture for full experts we use ResNets. In particular, all of our experiments focus on the ResNet50-v2 architecture (R50) (He et al., 2016a), which sequentially stacks a root block and 4 blocks with (3, 4, 6, 3) residual units. The initial step in every experiment consists of training a baseline model $\mathbf{B}$ on the whole upstream data (see stage 1 in fig. 1). This baseline is subsequently fine-tuned by both full and adapter experts, but in different ways. A full expert trained on a slice of data is simply the baseline $\mathbf{B}$ fine-tuned on that data. The head will later be discarded for transfer. This approach requires as many R50s as there are experts.

Adapter Modules. Residual adapters were proposed to adapt a neural network to a particular downstream task without needing to fine-tune the entire network (Rebuffi et al., 2017). Instead, we use them to adapt the baseline architecture to slices of the *upstream data*. Originally, they were 1×1 convolutions placed after each 3×3 convolution, with a residual connection. Instead we place them before each of the R50's blocks. Finally, our adapters have a bottleneck and are non-linear, as in (Houlsby et al., 2019). We insert several in parallel into the backbone $\mathbf{B}$. When creating an expert, only the adapters are tuned and the backbone weights are frozen.

Figure 3a depicts the ResNet architecture with multiple expert adapters $(a_1^{(i)}, \ldots, a_n^{(i)})$. Let F_i be the function implemented by the i-th block of the backbone network. We *adapt its input* by computing the output as $x_i := F_i(x_{i-1} + a_e^{(i)}(x_{i-1}))$, where $e = R(x)$ is the identity of the selected expert, given by some routing function R, and x is the original input. During upstream training, the function R may also use the labels in addition to the image, as we discuss in section 4.3.

Figure 3b shows the adapter's bottleneck architecture. An adapter sequentially applies components A_1 and A_2. Each component performs a group normalization (N) (Wu & He, 2018), a ReLU activation (A) (Glorot et al., 2011), and a convolution (C) (LeCun et al., 1989), in that order. Due to the skip connection, the output dimension of $A_2 \circ A_1$ must match that of its input, c. However, we can change the output channels k of A_1, in order to limit the amount of parameters. Thus, we set $k = \frac{c}{2}$ so that the number of parameters equals that of a linear adapter. Each adapter only increases the parameter count of the R50 backbone by 6%. We briefly explored placing these adapters in other locations, or using other variations (Rebuffi et al., 2018), but we did not observe any significant improvement.

It's important to emphasize that the names *Full* and *Adapters* refer to the parameters that are specialized to a given subset of the upstream data *before any downstream fine-tuning*. In both cases, we fine-tune the entire network, after the model selection phase, to the downstream task.

4.2 Upstream Data and Expert Definition

We train our upstream models on large vision datasets with thousands of classes. Moreover, the datasets include an expressive hierarchy, linking classes and ancestor concepts via "is-a" relationships. Our experts' domains are nodes in this hierarchy, which are selected automatically based on the number of images. Due to the multi-label nature of the datasets, several experts could simultaneously apply to an image. For example, for an image of a lion, all of *organism, animal, carnivore, felidae*, and *lion* could be relevant expert domains. In particular, we use two different upstream image datasets, and independently train a set of experts on each. We further describe them in section 6.1.

4.3 Expert Training

Recall we denote by $\mathbf{B}$ the baseline R50 model trained on the whole upstream dataset $\mathbf{D}_U$. As shown in fig. 1, the second step of upstream training consists of training each expert individually on different subsets of the upstream dataset. Let $\mathbf{D}_e := (X_e, Y_e) \subseteq \mathbf{D}_U$ be the data corresponding to expert e. The subsets corresponding to different experts may overlap (e.g. for the *animal* and *dog* experts).

As mentioned before, the *full* experts completely fine-tune $\mathbf{B}$ on $\mathbf{D}_e$. For the *adapter* experts the weights corresponding to the adapter e (modules in red in fig. 3) are trained on $\mathbf{D}_e$, but the shared blocks and head parameters are *frozen*. Note that, due to the sharing scheme, we can train all experts independently in parallel. We train all experts for the same number of steps, regardless of the size of $\mathbf{D}_e$. Instead of learning a routing function, we exploit the structure of the upstream labels and use a hard-coded routing. We found this makes learning easier, and leads to powerful specialized models.

5 Expert Selection

Given a new downstream dataset $\mathbf{D}_T = (X_T, Y_T)$, we must choose an expert to use. We consider three approaches: *domain prediction*, *label matching*, and *performance proxy*.

Domain Prediction. This strategy selects the expert solely based on the images X_T. It effectively selects the expert whose domain best matches the target images. We implement this by training an auxiliary network (the "Expert Prediction Network" or EPN) to classify the expert from the image (i.e. learn the hard-coded routing mentioned previously). The EPN is trained upstream using the pre-training data and expert assignments. During transfer, an expert is selected using the highest geometric mean EPN predictive probability across the dataset. Details are in the Appendix A.

Label Matching. Matching the task with an expert can be done in the label space as opposed to the input space, by computing the affinity of each expert to a new downstream task in the label space of the upstream dataset. We first use a generic network trained on all upstream labels to predict upstream labels on the downstream images. We compute the KL-divergence between the distribution of labels on the downstream task images, and the prior distribution of labels for each expert. This per-expert prior is computed as the empirical distribution of labels on the images used to train that expert. We select the expert with the smallest KL-divergence. Details are in the Appendix B.

Performance Proxy. The aforementioned two strategies are simple, but do not use the training labels Y_T available for downstream tasks, which may contain key information. It would be too expensive to fine-tune every expert to every new task and select the best with hindsight, so we propose a proxy for the final performance. For this, we use a k-nearest neighbors classifier (Altman, 1992) with the image embeddings produced by each expert. In the case of full experts, we simply apply the corresponding full network to compute these embeddings. For adapter-based experts, we apply the specific expert and ignore the remaining ones. Concretely, let $M_e(x)$ be the embedding corresponding to expert e on input x, and let $\mathbf{D}_T = \{(x_i, y_i)_{i=1}^{N_T}\}$ be our downstream task. In order to score each expert, we apply a kNN classifier on the embedded dataset $\mathbf{D}_{T,e} = \{(M_e(x_i), y_i)_{i=1}^{N_T}\}$, with $k = 1$ and Euclidean distance. The accuracy $\mathrm{acc}(\mathbf{D}_{T,e})$ is computed via leave-one-out cross-validation. Finally, we select the expert with highest accuracy: $e^* = \arg\max_e \mathrm{acc}(\mathbf{D}_{T,e})$. There are other alternative proxies that

are cheaper than full fine-tuning, for example fitting a logistic regression, SVM, or decision trees to the features. Although these proxies may better match the fine-tuning accuracy, we use kNN since it is computationally cheap — it only requires a forward pass through the data, and leave-one-out cross-validation requires no additional inference per-fold — and it performs well (section 6).

5.1 Downstream transfer

The expert selection algorithm could choose several experts to be combined to solve any target task. However, we limit the scope of our work to transferring a single expert per task, since this approach is simple and turns out to be effective. Thus, the downstream transfer procedure is straightforward: it simply involves fine-tuning the selected expert model. We fine-tune the entire expert network to the downstream dataset, including the adapters when applicable. While it was valuable to restrict the scope of upstream training to focus on specializing the expert adapter parameters, we found fine-tuning the whole network downstream to be greatly beneficial. This differs from the original residual adapters work (Rebuffi et al., 2017), where only the adapters were fine-tuned.

6 Experimental Results

6.1 Upstream Training

ImageNet21k (Deng et al., 2009) is a public dataset containing 13 million images, and 14 million labels of 21 843 classes, which are WordNet synsets (Fellbaum, 2012). In addition to the 21k classes, we consider the 1 741 synsets that are their ancestors. We use the 50 synsets of ImageNet21k with the largest number of images to train the expert models. These include e.g. *animal, artifact, organism, food, structure, person, vehicle, plan*, or *instrument*. We released 48 of these ImageNet21k models[2].

JFT (Sun et al., 2017) is an even larger dataset containing 300 million images and 18 291 classes. Each image can belong to multiple classes, and as for ImageNet21k, the classes are organized in a hierarchy. We select as expert domains the classes with a sufficiently large number of examples: the 240 classes with more than 850 000 images. Some of the automatically chosen experts are *animal, arts, bird, food, material, person, phenomenon, plant*, or *product*.

We pre-train generic models on a Cloud TPUv3-512, as done in (Kolesnikov et al., 2019). Then fine-tune them briefly on each slice to create the expert models. Check additional details in appendix D.

6.2 Downstream Tasks

We evaluate on two suites of tasks, each consisting of several datasets. The first is the Visual Task Adaptation Benchmark (VTAB) (Zhai et al., 2019), which consists of 19 datasets. We evaluate on VTAB-1k, where each task contains only 1k training examples. The tasks are diverse, and divided into three groups: natural images (single object classification), structured tasks (count, estimate distance, etc.), and specialized ones (medical, satellite images). Appendix E.1 contains further details. The second suite is a collection of popular natural datasets commonly used in transfer learning literature and, particularly in (Ngiam et al., 2018), to which we compare our work.

6.3 Transfer Evaluation Protocol

When transferring to new tasks we need to perform expert selection and choose other hyperparameters (e.g. learning rate for fine-tuning). For each downstream task, we use the following three step protocol.

Expert Transfer. We select the expert to transfer using one of the methods presented in section 5. In both sets of tasks, we use 1k training examples per dataset. Details are provided in appendix C.1.

Hyperparameter Selection. In VTAB-1k we use the recommended hyperparameter sweep and 800-training/200-validation split. We independently repeat the hyperparameter selection procedure 10 times for confidence intervals. For the other datasets we perform a single random search over 36 hyperparameter sets and select the best set based on the validation performance. This is a similar computational budget to that of (Ngiam et al., 2018). See appendices E.2 and F.1 for sweep details.

[2] `https://tfhub.dev/google/collections/experts/bit/1`

Table 1: VTAB-1k results of different selection algorithms, using full experts trained on JFT. The average accuracy across each group of tasks and across all VTAB is reported. In each dataset, the median accuracy over 30 runs is used. Bootstrapped confidence intervals at 95% level are included.

	NATURAL	SPECIALIZED	STRUCTURED	ALL
No Experts	77.4 [77.3–77.6]	81.6 [81.5–82.0]	**57.2** [52.8–58.2]	69.8 [68.0–70.2]
Random	60.6 [59.1–63.9]	81.2 [80.9–81.8]	56.8 [54.9–57.8]	63.3 [62.3–64.6]
Domain Prediction	75.9 [74.4–77.4]	81.5 [81.3–82.2]	57.0 [56.1–57.4]	69.1 [68.4–69.8]
Label Matching	77.6 [77.8–78.1]	80.3 [79.1–82.5]	56.9 [55.6–57.2]	69.6 [68.9–70.0]
Performance Proxy	**79.7** [79.5–80.0]	**83.6** [83.3–83.8]	55.3 [52.1–56.3]	**70.2** [68.9–70.6]

Table 2: VTAB-1k results of the baseline models and different expert architectures using kNN selection, pre-trained on ImageNet21k (IN21k) and JFT. The average accuracy across each group of tasks and across all 19 tasks is shown. In each dataset, the median accuracy over 30 runs is used.

		NATURAL	SPECIALIZED	STRUCTURED	ALL
IN21k	Baseline	77.7 [77.4–77.8]	82.0 [78.4–83.9]	56.8 [55.9–57.2]	69.8 [68.8–70.3]
	Adapters	78.1 [78.0–78.3]	83.5 [83.1–83.6]	57.5 [56.8–58.2]	70.6 [70.3–70.9]
	Full	78.3 [78.1–78.6]	83.4 [83.2–83.6]	59.4 [58.7–59.8]	71.4 [71.1–71.6]
	All Experts	78.3 [78.1–78.6]	83.6 [83.4–83.7]	58.8 [58.0–59.4]	71.2 [70.8–71.5]
JFT	Baseline	77.4 [77.3–77.6]	81.6 [81.5–82.0]	57.2 [52.8–58.2]	69.8 [68.0–70.2]
	Adapters	79.0 [78.6–79.1]	81.3 [79.2–82.5]	59.1 [58.3–60.1]	71.1 [70.5–71.6]
	Full	79.7 [79.5–80.0]	83.6 [83.3–83.8]	55.3 [52.2–56.2]	70.2 [68.9–70.6]
	All Experts	80.0 [79.2–80.4]	83.7 [83.6–83.8]	58.6 [58.0–59.4]	71.8 [71.3–72.2]
IN21k + JFT	All Experts	**80.2** [79.8–80.3]	**84.0** [83.7–84.2]	**59.5** [58.7–60.1]	**72.3** [71.9–72.6]

Final Re-training. Using the hyperparameters from the previous step, we re-train the selected expert on the entire task (training plus validation set). In VTAB-1k, we repeat this step 3 times for each of the 10 trials of hyperparameter selection and compute the test accuracy, yielding 30 outcomes per method per task. We compute the median of these 30 outcomes as the final accuracy in the dataset.

6.4 PERFORMANCE OF DIFFERENT EXPERT SELECTION STRATEGIES

We first establish which of the expert selection strategies presented in section 5 performs best. As a baseline we also try selecting a random, uniformly drawn, expert per task. Table 1 shows the results on VTAB-1k, using full experts trained on JFT. Table 5 show the results with adapters.

Overall, all methods perform better than random selection, particularly on the NATURAL group. This confirms that selecting good experts is essential. Overall, the performance proxy (kNN) selection performs better than the other alternatives. kNN's average accuracy is 11% (relative) and 5.5% higher than that of the domain prediction and label matching, respectively. Thus, making use of the downstream labels offers a significant advantage in expert prediction. Therefore, in all subsequent experiments we use the kNN-based selection. We did not see a strong difference for the STRUCTURED datasets. We provide an extensive analysis of the kNN accuracy distribution per expert in appendix C. Appendix G shows how training experts on *random* subsets of the upstream data does not work well.

6.5 RESULTS ON VTAB

Table 2 shows the average accuracy across all the 19 VTAB-1k datasets broken down by type (natural, specialized, and structured). We summarize our findings as follows:

Improvement over Non-expert Baseline. All the algorithms, trained on either JFT or ImageNet21k, improve their corresponding baseline. Differences are most pronounced on the NATURAL datasets. While we also see improvements in SPECIALIZED and STRUCTURED datasets, some of the confidence intervals overlap. The performance of both JFT and ImageNet21k models is fairly similar in general. This is not unexpected; it has been observed before that, with restricted model capacity, they perform very similarly (Kolesnikov et al., 2019). Appendix C.6 shows the selected experts.

Table 3: Accuracy on the datasets used by DAT (Ngiam et al., 2018), and their average accuracy. Bootstrapped confidence intervals at 95% level are shown next to the accuracy where available. DAT uses Inception-v3 (In-v3) and a larger network, AmoebaNet-B (Am-B). Models pre-trained on JFT.

	AIRCRAFT	BIRDS	CARS	CIFAR10	FOOD	PETS*	AVG.
Baseline	91.4 [91.0–91.7]	78.8 [78.0–79.4]	95.6 [95.4–95.7]	97.8 [97.7–97.9]	91.3 [91.2–91.5]	94.5 [94.4–94.6]	91.6 [91.4–91.7]
Adapters	92.5 [92.2–92.8]	79.4 [78.7–80.1]	95.9 [95.8–96.0]	97.9 [97.8–98.0]	91.6 [91.5–91.7]	94.6 [94.4–94.8]	92.0 [91.9–92.1]
Full	**94.8** [94.5–95.1]	83.6 [83.1–83.9]	**96.1** [96.0–96.3]	97.8 [97.7–97.9]	93.1 [92.8–93.2]	**97.0** [96.9–97.1]	93.7 [93.6–93.8]
DAT (In-v3)	94.1	81.7	95.7	98.3	94.1	**97.1**	93.5
DAT (Am-B)	92.8	**85.1**	95.8	**98.6**	**95.3**	96.8	**94.1**

*Pets results are mean per class accuracy as opposed to accuracy.

Quality of Natural Representations. The upstream datasets used to train the experts mostly contain natural images. Consequently, the spectrum of representations offered by our models seem very effective in downstream natural datasets. More concretely, all models lead to improvements over the baseline performance, with average gains ranging from 1% to over 3.3% on the 7 natural datasets.

Full vs. Adapters. *JFT Experts.* Full models outperform adapters convincingly in NATURAL and SPECIALIZED datasets. However, they do a poor job on STRUCTURED datasets –mainly due to the failure on one specific dataset. *ImageNet21k Experts.* In this case, the advantage of full experts comes precisely from STRUCTURED datasets. Appendix E provides results broken down by each dataset.

Combining All Experts. The previous results suggest that adding all types of expert models to the selection pool (full or adapter trained on JFT or ImageNet), since the optimal expert architecture and upstream dataset change from task to task. The results are remarkable: kNN is able to select good candidates to fine-tune from a pool of almost 600 models, and the relative improvement over the Baseline accuracy, across all VTAB datasets is 3.6%, showing gains on all dataset types (see last row in table 2).

6.6 OUR APPROACH VS. DOMAIN ADAPTIVE TRANSFER

Domain Adaptive Transfer (Ngiam et al., 2018) (DAT) also relies on specialist models pre-trained on JFT. First it trains a generalist model on the upstream data, similar to our $\mathbf{B}$. For any new task, then re-weights the *upstream* images based on a forward pass on the downstream data, and fine-tunes a new specialist model using the re-weighted upstream data. Finally, the model is further tuned on the target task. DAT falls outside of our transfer setup presented in section 5, as the downstream data *directly* influences the upstream training. This incurs a significant cost learning *every* new target task.

Remarkably, our algorithm works in setups where access to upstream data is not available (e.g. for privacy or proprietary reasons). We also use downstream labels, which proved to carry key information about the task (see section 6.4). And most importantly, our method is more practical by amortizing the cost of expert pre-training as more downstream tasks are served. Under same models and hardware, running kNN (with 240 models) is between $500\times$–$1000\times$ faster than fine-tuning the baseline model with the re-weighted upstream data. Appendix F includes additional details.

Table 3 shows the mean accuracy over 30 trials per dataset, on the same datasets and under a similar hyperparameter budget as DAT. These tasks are close to VTAB's NATURAL group and yield similar results: full experts outperform adapters. Our results are not directly comparable to those of DAT since they use Inception-v3 (Szegedy et al., 2016), and AmoebaNet-B (Real et al., 2019) architectures. Inception-v3 and R50 are similar in performance and size; the former has 24M parameters, attaining 78.8% top-1 on ILSVRC2012 (from-scratch), whereas the latter has 26M parameters and attains 76.0%. The AmoebaNet-B (N=18, F=512) is 22 times larger, with more than 550M parameters. Despite the differences, our method is competitive and matches or beats DAT in half the datasets.

7 DISCUSSION

Algorithm. Our results suggest that there are strong potential benefits to using smartly routed pre-trained experts *when* the domain of the experts broadly matches that of the downstream tasks. We have clearly seen this with natural images. Instead, as expected, when there is a skill mismatch (e.g.

trying to solve a counting task with diverse single-object recognition experts) we have not observed any significant gain or loss. Still, in these cases, the expert selector can fall back on the generic model or representation. When there is an extremely relevant expert for a task –say, our *flower* or *plant* models for the Oxford Flowers 102 task–, using full network experts proved beneficial. In contrast, many datasets did not have a perfect match, and adapters seemed easier to fine-tune in these cases.

Impact. In the near future, we foresee large computer vision systems composed by a wide range of pre-trained specialist modules. These modules may be based on huge amounts of data, small but high-quality curated repositories, or even on private and proprietary content, and they would cover a diverse spectrum of canonical tasks (object recognition, some way of narrow reasoning, counting, sorting, etc.). Some of them may not even need to be end-to-end learned from data.

Future Directions. There are a number of exciting follow-up research directions. Selecting and combining multiple experts for any downstream task is a natural extension of our work. This could be especially useful for tasks that require understanding several concepts, not necessarily captured by a single expert. Per-example routing (i.e. applying routes tailored to each individual data-point) could also lead to improvements based on targeted processing, for example, in the context of tasks with instances of various difficulties. Finally, moving beyond our experts based on label hierarchies, and towards automatic discovering and training of experts could unlock even further gains.

ACKNOWLEDGMENTS

We would like to thank Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai and Jessica Yung, who worked on the original BiT, that we use as baseline, and shared their insights with us. Josip Djolonga and Maxim Neumann provided very useful feedback on an earlier version of this work. Finally, we thank whole Google Brain team in Zürich for many useful discussions and support.

REFERENCES

Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in neural information processing systems*, pp. 41–48, 2007.

Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Advances in neural information processing systems*, pp. 137–144, 2007.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman. Learning bounds for domain adaptation. In *Advances in neural information processing systems*, pp. 129–136, 2008.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

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.

Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*, pp. 193–200, 2007.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pp. 647–655, 2014.

Simon S Du, Jayanth Koushik, Aarti Singh, and Barnabás Póczos. Hypothesis transfer learning via transformation functions. In *Advances in neural information processing systems*, pp. 574–584, 2017.

Nikita Dvornik, Cordelia Schmid, and Julien Mairal. Selecting relevant features from a universal representation for few-shot classification. *arXiv preprint arXiv:2003.09338*, 2020.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

Theodoros Evgeniou, Charles A Micchelli, and Massimiliano Pontil. Learning multiple tasks with kernel methods. *Journal of machine learning research*, 6(Apr):615–637, 2005.

William Fedus, Carles Gelada, Yoshua Bengio, Marc G Bellemare, and Hugo Larochelle. Hyperbolic discounting and learning over multiple horizons. *arXiv preprint arXiv:1902.06865*, 2019.

Christiane Fellbaum. Wordnet. *The encyclopedia of applied linguistics*, 2012.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, pp. 630–645, 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*, pp. 770–778, 2016b.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. *arXiv preprint arXiv:1902.00751*, 2019.

Laurent Jacob, Jean-Philippe Vert, and Francis R Bach. Clustered multi-task learning: A convex formulation. In *Advances in neural information processing systems*, pp. 745–752, 2009.

Robert A Jacobs and Michael I Jordan. Learning piecewise control strategies in a modular neural network architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):337–345, 1993.

Seyoung Kim, Eric P Xing, et al. Tree-guided group lasso for multi-response regression with structured sparsity, with an application to eqtl mapping. *The Annals of Applied Statistics*, 6(3): 1095–1117, 2012.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (BiT): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 2019.

Ilja Kuzborskij and Francesco Orabona. Stability and hypothesis transfer learning. In *International Conference on Machine Learning*, pp. 942–950, 2013.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2015.

Karim Lounici, Massimiliano Pontil, Alexandre B Tsybakov, and Sara Van De Geer. Taking advantage of sparsity in multi-task learning. *arXiv preprint arXiv:0903.1468*, 2009.

Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation with multiple sources. In *Advances in neural information processing systems*, pp. 1041–1048, 2009.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3994–4003, 2016.

Jiquan Ngiam, Daiyi Peng, Vijay Vasudevan, Simon Kornblith, Quoc V Le, and Ruoming Pang. Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*, 2018.

Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1717–1724, 2014.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

David Pardoe and Peter Stone. Boosting for regression transfer. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 863–870, 2010.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pp. 506–516, 2017.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8119–8127, 2018.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pp. 270–279. Springer, 2018.

Chang Wan, Rong Pan, and Jiefei Li. Bi-weighting domain adaptation for cross-language text classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

Zirui Wang. Theoretical guarantees of transfer learning. *arXiv preprint arXiv:1810.05986*, 2018.

Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):9, 2016.

Yuxin Wu and Kaiming He. Group normalization. In *European Conference on Computer Vision (ECCV)*, September 2018.

Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252*, 2019.

Yonghui Xu, Sinno Jialin Pan, Hui Xiong, Qingyao Wu, Ronghua Luo, Huaqing Min, and Hengjie Song. A unified framework for metric transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 29(6):1158–1171, 2017.

I Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.

Xi Yan, David Acuna, and Sanja Fidler. Neural data server: A large-scale search engine for transfer learning data, 2020.

Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. The visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pp. 94–108. Springer, 2014.

A EXPERT PREDICTOR NETWORKS

An expert predictor network (EPN) tries to directly predict the relevant expert for an input image, using only the image itself as input. We first train the EPN upstream, and then we apply it downstream to select the most relevant expert for a new task by aggregating its output on all the images in the task.

EPN Upstream Training. As we described in section 4, we have split the upstream dataset D_U into a collection of subsets $\{D_e : 1 \leq e \leq E\}$, with $D_e = (X_e, Y_e) \subseteq D_U$. In order to train the EPN, we simply assign the expert identity e as the label of all images in X_e, and train the network in a supervised manner (using softmax cross-entropy) to predict the expert. Expert slices $\{D_e\}$ are not disjoint, thus, it is possible that an individual image appears multiple times in the training data for the EPN with different expert identities. Because subsets sizes are different, and in order not to favor any particular expert, we resample the training images so that each expert is seen equally often. Intuitively, this classification problem should be substantially easier than predicting the upstream classes y directly, as there are much fewer experts than upstream classes.

Downstream Expert Selection. Suppose we are given a downstream task, containing images $X_T = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_T}\}$. We first apply a forward pass on X_T using the EPN. Let $Q_{\text{EPN}}(e \mid X = \mathbf{x}_i)$ be the probability assigned by the EPN to expert e for input image $\mathbf{x}_i$. In order to make a single decision for the whole downstream task, we combine those probabilities using a log-linear transformation.

We select the expert as follows:

$$\hat{e} = \arg\max_e \frac{1}{N_T} \sum_{i=1}^{N_T} \log Q_{\text{EPN}}(e \mid X = \mathbf{x}_i). \tag{1}$$

The log-linear combination of per-example probabilities was obtained after several experiments with a number of functions. Intuitively, this transformation penalizes experts that only apply to a subset of the downstream data, but are not relevant to other downstream examples.

A major drawback of the EPN is the fact that it does *not* use or benefit from the downstream labels. Imagine there is an image dataset with pictures containing simultaneously both lions and elephants. Suppose we are faced with two different downstream tasks based on the same inputs: one is to count lions, the other is to count elephants. Furthermore, imagine our experts happen to include *lion* and *elephant*. Depending on the task, it would be reasonable to choose one or the other expert. Unfortunately, the basic EPN approach is agnostic to the outputs, and –as the input images are identical– it would return the same selected expert in both cases.

B KULLBACK–LEIBLER DIVERGENCE

Since our expert datasets $\mathbf{D}_e$ were built based on the hierarchy of labels in the upstream dataset, it is reasonable to assume that the prior distribution of the labels in each $\mathbf{D}_e$ differ across experts e. Let P_e be this prior distribution for expert e. Then, we can use a divergence measure, such as the Kullback–Leibler (KL), to determine which expert to use. If one assumes that the downstream dataset is well represented by the upstream dataset $\mathbf{D}_U$ (although not necessarily by an individual $\mathbf{D}_e$), one can use the baseline neural network $\mathbf{B}$ to approximate the distribution of *upstream* labels conditioned to the set of downstream images:

$$Q(Y) \coloneqq \frac{1}{N_T} \sum_{j=1}^{N_T} Q_{\mathbf{B}}(Y \mid X = \mathbf{x}_j), \tag{2}$$

where $Q_{\mathbf{B}}(Y \mid X = \mathbf{x}_j)$ is the probability distribution given by $\mathbf{B}$ over each image in the set $X_T = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_T}\}$ of downstream images. Then, we simply select the expert with the lowest KL divergence:

$$\hat{e} \coloneqq \arg\min_e D_{\text{KL}}(P_e \parallel Q) \tag{3}$$

This allows us to leverage the baseline model that we already trained, and not train an auxiliary neural network to predict the expert to use, like the EPN in appendix A does. In addition, this has the benefit

of using information about the distribution of upstream classes, which may be useful when the target classes are well represented among the upstream ones.

In our case, the upstream datasets consists of multi-labeled images. The distribution of labels given to a particular image is modelled by the neural network as a joint distribution of independent Bernoulli random variables. Assuming this independence also holds for P_e, one can then compute eq. (3) very efficiently. Of course, this assumption is not true in either case (e.g. the presence/absence of the *dog* and *animal* labels is not independent), but it is standard practice for multi-label classification.

C FURTHER RESULTS ON k-NEAREST NEIGHBORS

In this section, we present the kNN accuracy distribution per dataset that we found for both JFT and ImageNet21k experts. A flat curve indicates differences across experts may not be very relevant for the downstream task, while steep regions suggest strong decreases in value among expert models.

C.1 kNN HYPERPARAMETERS

We select the expert to transfer using the kNN transfer proxy with $k = 1$ and a Euclidean distance metric. We use 1 000 training examples in all datasets (including those in the comparison with Domain Adaptive Transfer Ngiam et al. (2018)), and compute kNN leave-one-out cross-validation to compute the accuracy per expert. Finally, we select the one with highest accuracy. For VTAB-1k this corresponds to the entire training set per task; whereas for the other tasks, we randomly sample 1k training examples. We do not perform special data pre-processing, and simply resize and crop to 224×224, as done in upstream evaluation.

We used a NVDIA V100 GPU to perform the kNN selection for each dataset, with this hardware, selecting among 240 models takes less than 2 hours.

C.2 ARCHITECTURE COMPARISONS

In this section we look at the kNN accuracy *before* transferring the experts, and –in particular– at how it depends on the architecture choice. Recall each expert is associated with one slice of the upstream data. For any given slice, we have trained both a full ResNet50 network, and adapters attached to a pretrained ResNet50. We plot the accuracy achieved by these representations (scatter-plot, full at x-axis; adapters at y-axis) for all experts for each group of VTAB datasets. We look at JFT experts (Figure 4) and at ImageNet21k experts (Figure 5). Ideally, we would expect some positive correlation if expert representations were somewhat similar regardless of the architecture.

JFT Experts. The first seven plots in Figure 4 show the results in natural datasets. While most experts do not seem relevant –and performance seems a bit uncorrelated between both types of models–, in most datasets we see that there are a few good experts (top right corner) which offer the strongest performance despite of the selected architecture. SVHN seems to be an exception.

Similar plots are displayed in the following 4 and the last 8 plots in Figure 4 for specialized and structured datasets, respectively. Few datasets show agreement on the most promising expert slices, such as Eurosat or Resisc45. Unfortunately, there is no clear agreement in most specialized and structured datasets.

ImageNet21k Experts. We see a reasonable agreement among the best ImageNet21k experts in natural datasets (see first 7 plots in Figure 5). While not as correlated as in the case of natural datasets, we still see some positive relationship in some specialized (Eurosat, Resisc45, Patch Camelyon) and structured (Clevr Count, DSprites Position, Smallnorb Azimuth) datasets.

C.3 kNN ACCURACY DISTRIBUTION FOR JFT EXPERTS

Figure 6 shows the distribution of the kNN accuracy obtained from the embedding of each of the experts trained on JFT. In each case (full and adapters), the kNN accuracy of the 240 experts has been sorted in a decreasing manner. Note that we pick the single expert with highest-score (although other approaches are possible).

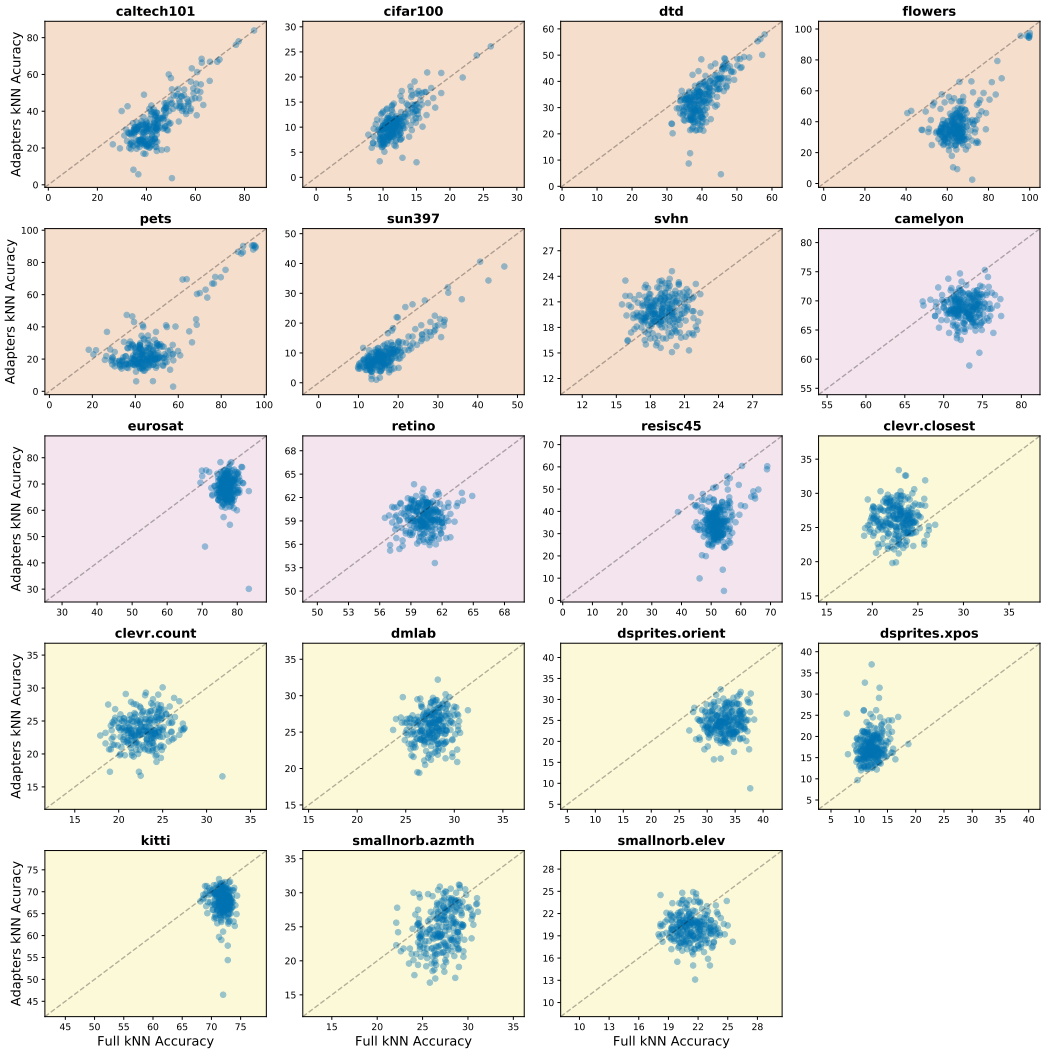


Figure 4: **JFT Experts**. Each point is one expert (upstream data slice); the x-axis represents the kNN accuracy before downstream finetuning of the expert trained on a *full* network. The y-axis displays the kNN accuracy before downstream finetuning of the expert trained with an *adapter* module. The background color indicates the dataset group: ● NATURAL, ● SPECIALIZED, and ● STRUCTURED. There are 240 experts. Identity dashed line shown too.

In most ● NATURAL datasets, we observe that full JFT experts are on average better than their adapter counter-parts. In datasets like Caltech101, Cifar100, or DTD, it seems these differences do not affect the top experts, while in others (such as Flowers, Pets, and Sun397) the differences still apply to the best expert. Also, overall, we see that in natural datasets there are usually *strong* differences between good and bad experts. The range of kNN accuracies is pretty large for Caltech101, Flowers, or Pets, where some experts seem to already solve the task, while others lead to quite poor accuracies. The latter may be fixed to some extent by downstream fine-tuning.

In the ● SPECIALIZED group we see a similar pattern in the comparison between full and adapter-based experts. However, the accuracy range of variations (except, maybe, at the very worst end) is narrower.

The story for ● STRUCTURED datasets with JFT experts is a bit different. In some datasets, adapters models lead on average to better initial representations (such as Clevr Closest and Clevr Count, or dSprites Position). As with structured datasets, the difference between the best and the worst experts is shorter. This may be in part explained by the hardness of the task itself (the average accuracy *after*

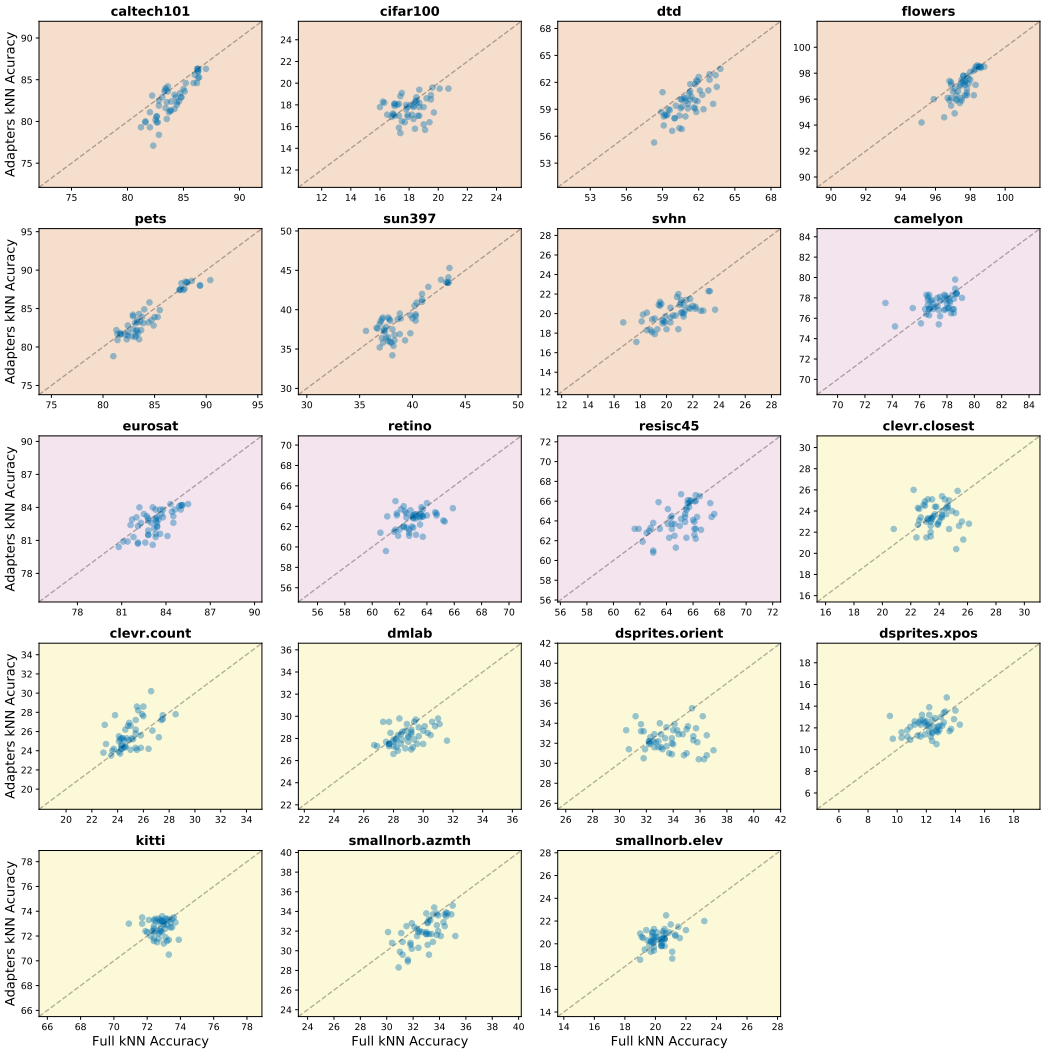


Figure 5: **ImageNet21k Experts**. Each point is one expert (upstream data slice); the x-axis represents the kNN accuracy before downstream finetuning of the expert trained on a *full* network. The y-axis displays the kNN accuracy before downstream finetuning of the expert trained with an *adapter* module. The background color indicates the dataset group: ● NATURAL, ● SPECIALIZED, and ● STRUCTURED. There are 50 experts. Identity dashed line shown too.

fine-tuning is definitely lower than in the natural case), but there are some counter-examples to this, like dSprites Position where final accuracies go up to around 90%.

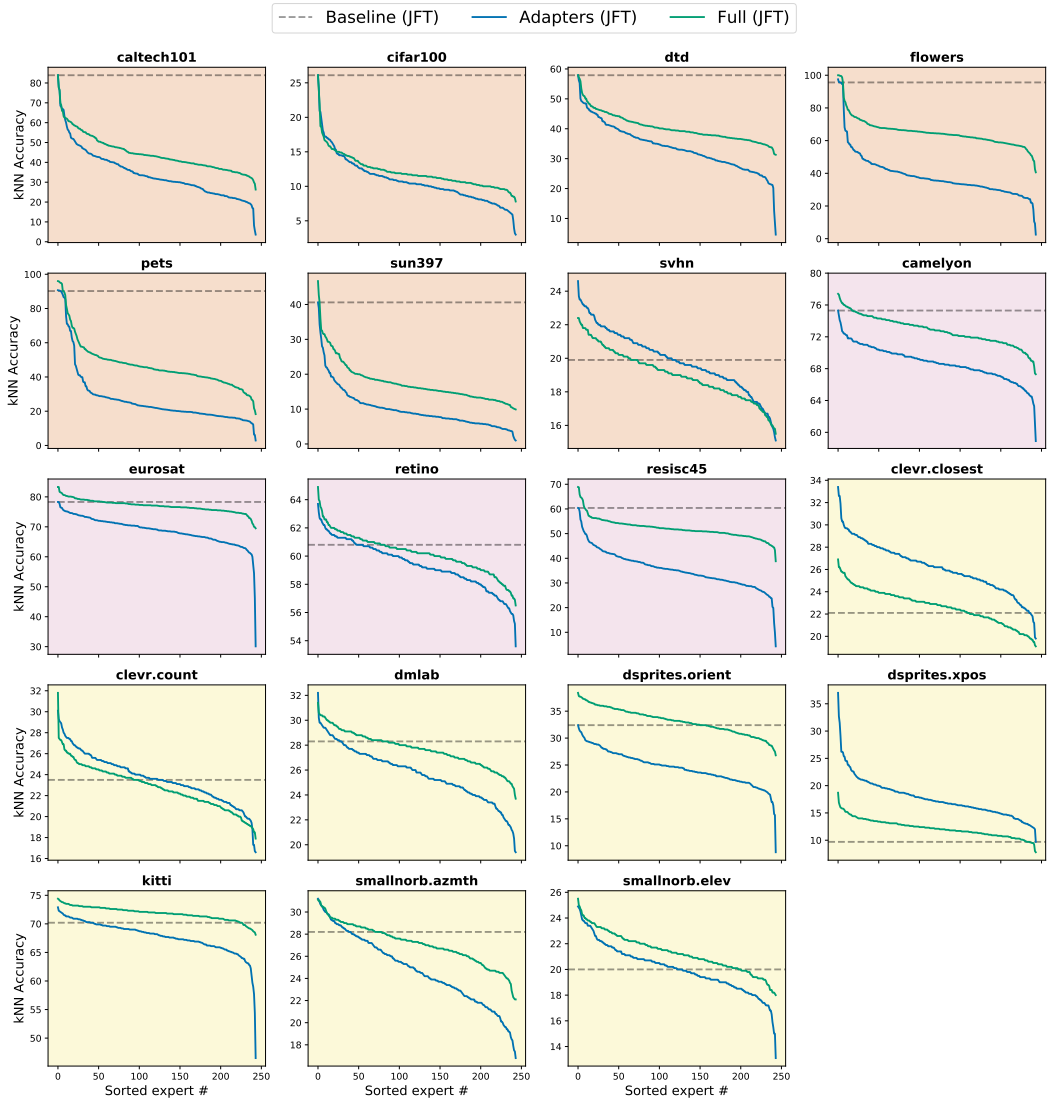


Figure 6: Distribution of the kNN accuracy from experts trained on JFT. The dashed lines shows the kNN accuracy of the baseline model. In each dataset, the experts are sorted according to their accuracy. The background color of the plot represents the group of the dataset: ● NATURAL; ● SPECIALIZED; ● STRUCTURED.

C.4 kNN ACCURACY DISTRIBUTION IMAGENET21K EXPERTS

Figure 7 presents the distribution of the kNN accuracy obtained from the embedding of each of the experts trained on the ImageNet21k dataset. For context, in all the plots we also show the Top-50 JFT full experts.

For ● NATURAL tasks, we observe that the quality of the ImageNet21k expert representations is way more homogeneous than the JFT one. Accordingly, finding the right expert in JFT may be more important (as accuracy decreases fast), while it may provide even more target-tailored representations (see Oxford Flowers and Oxford Pets). Overall, ImageNet21k accuracies seem more stable, and differences between full and adapters are modest.

Overall, both full and adapter ImageNet21k experts seem to perform similarly on ● SPECIALIZED tasks. The plots suggest that ImageNet21k experts are a bit ahead of the full JFT ones (even though the gap at the top tends to close).

We see a few distinct behaviors in ● STRUCTURED. There tend not to be very remarkable winner experts, and full experts may provide a small boost compared to adapter-based ones. In most datasets, the Top-50 full JFT experts outperform the ImageNet21k ones.

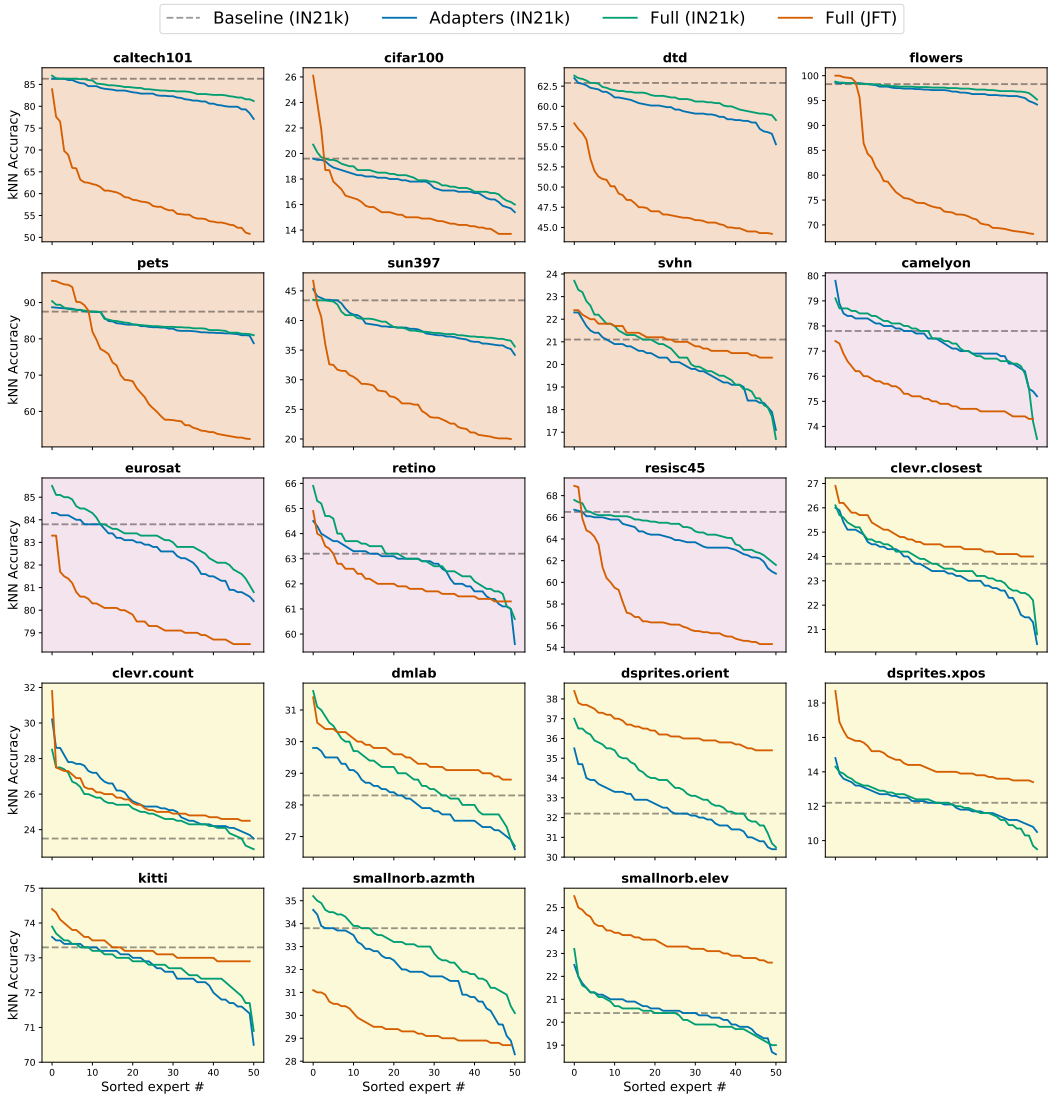


Figure 7: Distribution of the kNN accuracy from experts trained on ImageNet21k. The dashed lines shows the kNN accuracy of the baseline model. The performance of the top-50 JFT full experts on each dataset is also shown. In each dataset, the experts are sorted according to their accuracy. The background color of the plot represents the group of the dataset: ● NATURAL; ● SPECIALIZED; ● STRUCTURED.

C.5 kNN Accuracy Distribution for Consecutive Checkpoints of ImageNet21k Baseline

In this subsection, we study how representations evolve during training. In order to do that, we stored 157 checkpoints –equally spaced– over the training of our ImageNet21k baseline. We trained the model for 90 epochs. For each dataset, we compute the kNN accuracy of the checkpoints, and display the curves in fig. 8. As an auxiliary line, we also show the mean kNN accuracy across all the checkpoints.

There are some clear differences depending on the *type* of dataset. In the case of ● Natural images, it seems that more training leads to better representations. The kNN accuracy tends to increase (Cifar100 and SVHN are exceptions). ● Specialized datasets behave in a different way; while there is an initial boost in accuracy (i.e. trained models are better than randomly initialized ones), long training only leads to very minor improvements in representation quality for these tasks. Finally, ● Structured datasets have extremely flat footprints. This probably means that our semantic experts are not a good fit for this type of task.

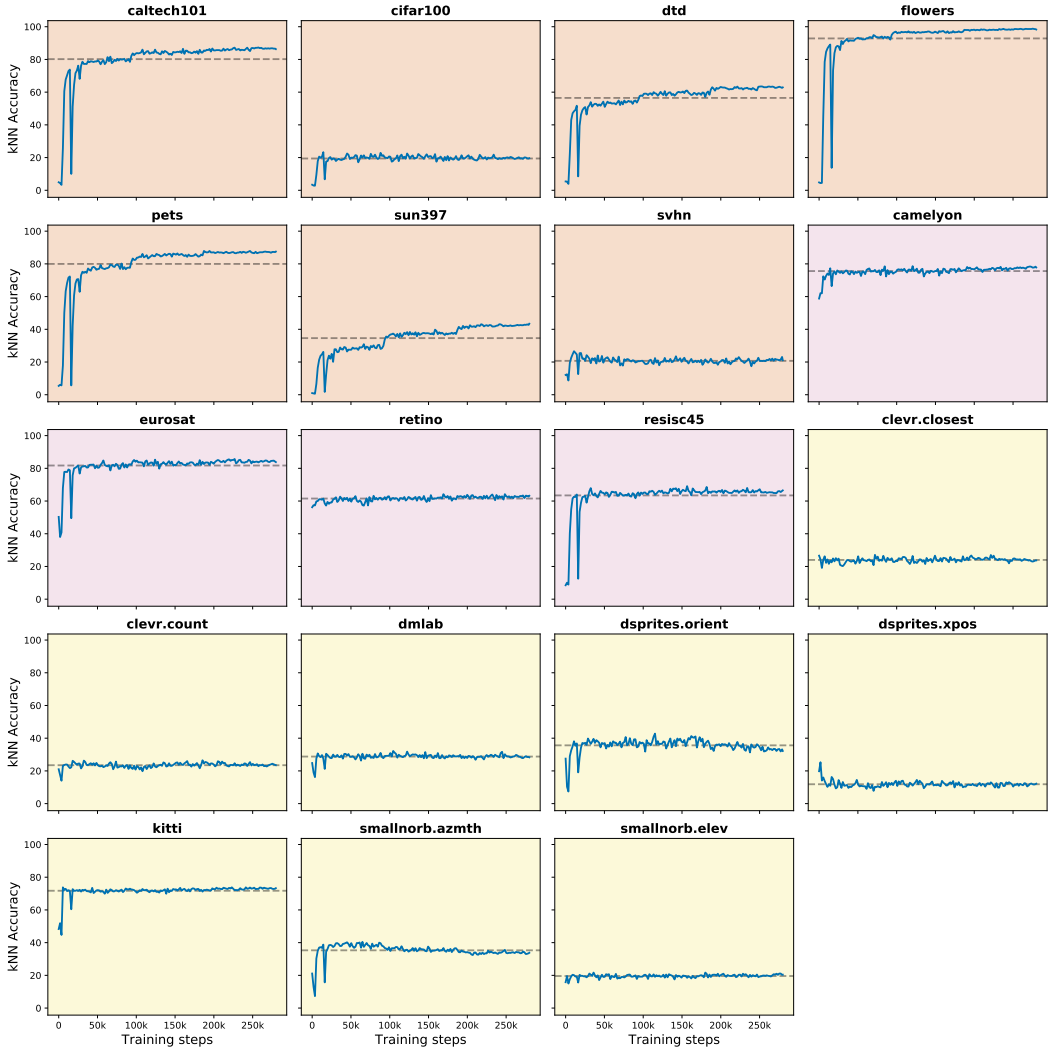


Figure 8: Accuracy of kNN using consecutive checkpoints stored during the ImageNet21k baseline training (90 epochs). The dashed line represents the mean value across all checkpoints and it is useful to point out lack of improvement over time in some cases. The different types of datasets are highlighted by the background color: ● Natural, ● Specialized and ● Structured.

C.6 SELECTED EXPERTS

The following table presents the experts selected by kNN in each of the individual datasets of the ● NATURAL, ● SPECIALIZED and ● STRUCTURED groups.

Table 4: Selected experts by kNN using different expert architectures, in each of the VTAB-1k datasets. Datasets are grouped by ● NATURAL, ● SPECIALIZED and ● STRUCTURED.

Dataset	Full-JFT	Adapter-JFT	Full-INet	Adapter-INet
● Caltech101	*Baseline*	*Baseline*	Physical Entity	*Baseline*
● Cifar100	*Baseline*	*Baseline*	Object	*Baseline*
● DTD	*Baseline*	*Baseline*	Artifact	Artifact
● Oxford Flowers	Plant	Flower	Organism	Physical Entity
● Oxford Pets	Mammal	Carnivore	Carnivore	Carnivore
● Sun397	Structure	*Baseline*	Structure	Structure
● SVHN	Textile	Staple food	Implement	Artifact
● Diabetic Retinopathy	Paper	Material	Food	Plant
● Eurosat	Snow	*Baseline*	Whole	Breathe
● Patch Camelyon	Tree	*Baseline*	Whole	Woody Plant
● Resisc45	Geographical feature	*Baseline*	Instrument	Arthropod
● Clevr Closest	Mode of transport	Canis	Mammal	Relation
● Clevr Count	Snow	Adventure	Spermatophyte	Flower
● DMLab	Sports equipment	Art	Implement	Vertebrate
● dSprites Orientation	Flowering plant	Mode of transport	Clothing	Implement
● dSprites Position	Dish	Toyota	Matter	Chordate
● Kitti	Shoe	Geographical feature	Plant	Abstraction
● Smallnorb Azimuth	Home and garden	Food	Abstraction	Device
● Smallnorb Elevation	Bag	Artwork	Carnivore	Mammal

D UPSTREAM TRAINING

D.1 UPSTREAM TRAINING DETAILS

Unconditional pre-training. We pre-train generic JFT and ImageNet21k models using a similar protocol to the one described in Kolesnikov et al. (2019). In particular, we use SGD with momentum of 0.9, with a batch size of 4096, an initial learning rate of 0.03 (scaled by a factor of $\frac{\text{batch size}}{256}$), and weight decay of 0.001. The JFT backbone model is trained for a total of 30 epochs, while the ImageNet21k is trained for 90 epochs. In both cases we perform training warm-up during the first 5 000 steps by linearly increasing learning rate, and then decay the learning rate by a factor of 10 at $\{\frac{1}{3}, \frac{2}{3}, \frac{5}{6}\}$ of the total duration. During this phase we used a Cloud TPUv3-512 to train each of the baseline models, which takes about 25 hours in the case of JFT and .

Experts training. In order to obtain the expert models, we then further tune these baselines (adding the residual adapters, when applicable) on different subsets of the original upstream dataset. We use a similar setting to the one described before, although we train for much shorter times, use a batch size of 1 024, and use different learning rates. In particular, the full experts use an initial learning rate 10^{-4}, since all the parameters were pre-trained in the earlier phase. The experts with adapters use a larger learning rate of 10^{-1}, as these components are trained from scratch and are the only ones that are tuned. We use the same learning scaling factor and decay schedule as in the previous step. The initial learning rate in each case was decided based on average upstream performance across the different expert datasets. We fine-tune the full experts for 2 epochs, and the adapters for 4 epochs, relative to the size of the entire dataset. The only exception is for the results reported in the comparison with Ngiam et al. (2018), for which we observed in the validation data that full experts trained for 4 epochs performed better.

In both stages, we perform standard data augmentation during training, which includes random image cropping as in Szegedy et al. (2015), random horizontal mirroring, and finally image resize to a fixed resolution of 224×224. When we need to evaluate these models on upstream data (i.e. upstream learning rate selection), we simply resize and crop the images to a fixed resolution of 224×224. Pixel values are converted to the $[-1, 1]$ range.

During this phase we used a Cloud TPUv3-32 to train each of the experts. Training one of the JFT experts for 2 epochs takes about 11 hours, while this is reduced to 30 minutes in the case of the ImageNet21k experts.

D.2 Upstream Freezing

Note that many expert datasets $\mathbf{D}_e$ do not contain any instances of some original upstream classes (for example, the data for the expert *elephant* may not contain any image with the label *vehicle*). As the head is frozen and shared among all experts, the adapters need to find other ways to ignore classes that do not apply at all to the expert. We found this to be beneficial in practice, as we avoided too much upstream dependence on the head (which is later discarded in the transfer stage).

E Visual Task Adaptation Benchmark Details

E.1 Classes per Task

The number of classes in the VTAB tasks varies significantly, see Figure 9. As we are most interested in the low-data regime, we fix the number of downstream examples to 1 000, implying that some downstream datasets only contain 3-10 examples per class –like Sun397 or Caltech.

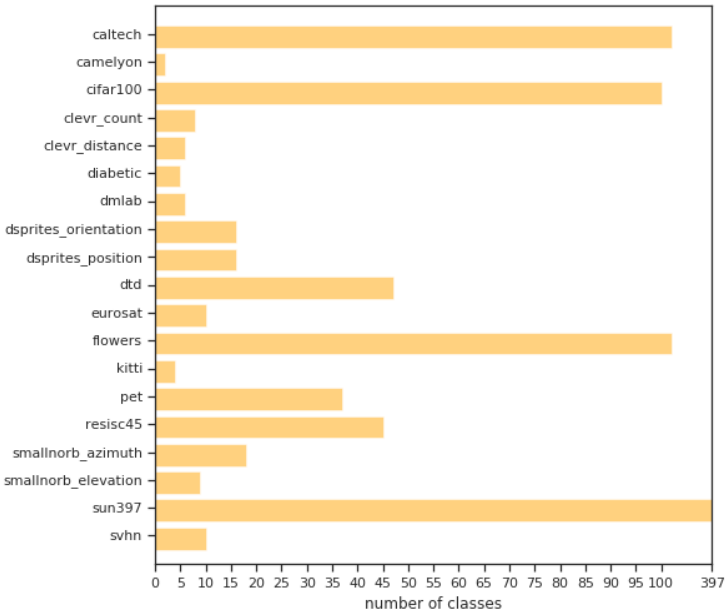


Figure 9: Number of classes per Downstream Task in VTAB.

E.2 Downstream Hyperparameters on VTAB

We use SGD with momentum of 0.9 and a batch size of 512. The initial learning rate is scaled by $\frac{\text{batch size}}{256}$. We don't perform any data augmentation, and resize all the images to a fixed resolution of 224×224. Pixel values are converted to the $[-1, 1]$ range. We perform a restricted hyperparameter search, in particular we follow the *lightweight* suggestion from Zhai et al. (2019). The sweep tries in total 4 different sets of hyperparameters for each dataset.

- Initial learning rate: the values of $\{0.1, 0.01\}$.
- Training schedule: we try with a total training duration of $2\,500, 10\,000$ steps, with a linear warm up of the scaling rate of 200 and 500 steps, respectively. For both durations, the learning rate is reduced by a factor of 10 after $\{\frac{1}{3}, \frac{2}{3}, \frac{9}{10}\}$ of the total number of steps.

Note that the hyperparameter sweep is done by training the models on 800 examples (out of the 1 000 available), and selecting over 200 validation examples. Then, the best combination of hyperparameters is used to re-train the models on the full 1 000 data points.

Because the variability due to random initialization with so few data points is large in some datasets, we perform 10 independent runs of hyperparameter selection, and then re-training the models 3 times for each of the 10 selected hyperparameters. This yields a total of 30 outcomes for each of the VTAB-1k datasets. For each dataset, we report the median over these 30 trials and compute (percentile) bootstrapped confidence intervals at 95% level.

For downstream training we use a Cloud TPUv3-16. In VTAB-1k, the running time depends on the number of steps. It takes 12 minutes to fine-tune one of our experts for 10 000 steps (the longest duration), and just 3 minutes for the shorter schedule of 2 500 steps.

E.3 Additional Results of Different Expert Selection Strategies

In section 6.4, and more precisely in table 1, we studied the performance of random transfer (expert selection uniformly at random). There, we only reported the results corresponding to full experts trained on JFT. For completeness, table 5 shows also the results with adapters. The pattern is fairly similar: random transfer leads to massive losses in Natural datasets (we obtain an almost 35% improvement by applying kNN with respect to random transfer). Domain Prediction and Label Matching also heavily help in these settings. In Specialized and Structured tasks, both Domain Prediction and Label Matching seem to offer little-to-no gains, whereas kNN still leads to a modest boost on Structured, and a decent one on Specialized.

Overall (last column of the table), the improvement is significant and strong for all routing methods.

Table 5: VTAB-1k results of different selection algorithms, using full and adapter experts trained on JFT. The average accuracy across each group of tasks and across all VTAB is reported. In each dataset, the median accuracy over 30 runs is used. Bootstrapped confidence intervals at 95% level.

	Natural	Specialized	Structured	All
Adapters				
Random	58.6 [56.1–59.6]	78.3 [76.8–79.2]	58.6 [57.8–59.6]	62.8 [61.7–63.3]
Domain Prediction	70.8 [69.3–71.6]	75.5 [63.7–78.0]	59.7 [58.2–61.0]	67.1 [64.5–67.9]
Label Matching	75.3 [75.1–75.4]	80.5 [78.2–81.3]	56.1 [51.8–57.0]	68.3 [66.4–68.7]
Performance Proxy	79.0 [78.6–79.1]	81.3 [79.2–82.5]	59.1 [58.3–60.1]	71.1 [70.5–71.6]
Full				
Random	60.6 [59.1–63.9]	81.22 [80.9–81.8]	56.8 [54.9–57.8]	63.3 [62.3–64.6]
Domain Prediction	75.9 [74.4–77.4]	81.5 [81.3–82.2]	**57.0** [56.1–57.4]	69.1 [68.4–69.8]
Label Matching	78.0 [77.8–78.1]	80.3 [79.1–82.5]	56.9 [55.6–57.2]	69.6 [68.9–70.0]
Performance Proxy	**79.7** [79.5–80.0]	**83.6** [83.3–83.8]	55.3 [52.1–56.3]	**70.2** [68.9–70.6]

E.4 Per-Task Results

All VTAB results presented so far were averaged over dataset types (natural, specialized, and structured). In this subsection, we break down the outcomes per dataset. Table 6 shows the mean accuracy (and confidence intervals) for 13 algorithms and 19 datasets. The datasets are sorted according to the data type. The table can be used for reference. Some datasets showcase a wide range of outcomes for any fixed algorithm. In order to expose this in a clear fashion, we present in fig. 10 the individual trial accuracies for the best algorithms and baselines, in all of the VTAB datasets. The fine-tuning process on datasets like Clver Count, dSprites xPosition, or SVHN definitely shows a high variance of test accuracies. The median estimator partially mitigates this effect.

Table 6: Accuracy on the individual datasets of the VTAB-1k benchmark. Algorithms include experts trained on both JFT and ImageNet21k, with adapters and full architectures, and by means of different selection methods (Expert Predictor Network, EPN; Kullback–Leibler, KL; and kNN). In each dataset, the median accuracy over 30 runs is used. Bootstrapped confidence intervals at 95% are shown. Color indicates dataset group: ● NATURAL; ● SPECIALIZED; ● STRUCTURED.

	● caltech101	● cifar100	● dtd	● flowers	● pets	● sun397	● svhn	● camelyon	● eurosat	● retino	● resisc45	● clevr.closest	● clevr.count	● dmlab	● dsprites.orient	● dsprites.xpos	● kitti	● smallnorb.azmth	● smallnorb.elev
JFT																			
Baseline	91.7 [91.5–91.8]	68.6 [68.3–68.7]	72.1 [72.0–72.2]	97.2 [97.1–97.2]	91.5 [91.4–91.5]	49.9 [49.9–50.0]	71.2 [70.5–72.0]	81.6 [81.4–83.1]	93.0 [92.9–93.1]	70.0 [69.6–70.2]	81.8 [81.5–81.9]	54.9 [54.5–55.8]	62.8 [60.7–68.0]	45.1 [45.0–45.3]	61.6 [61.1–62.1]	94.9 [93.7–96.2]	79.8 [43.9–80.8]	25.1 [22.0–30.5]	33.6 [33.1–35.5]
Adapters (EPN)	91.7 [91.6–91.7]	34.0 [32.2–34.3]	58.3 [58.0–58.6]	98.2 [98.1–98.2]	91.4 [91.3–91.5]	48.3 [48.1–48.4]	73.7 [63.3–79.2]	79.6 [71.7–83.1]	93.1 [92.2–93.6]	60.0 [53.7–66.4]	69.3 [23.3–74.3]	60.6 [57.3–61.9]	63.3 [56.7–73.3]	45.1 [43.7–46.0]	59.3 [56.9–59.5]	95.8 [95.0–96.8]	79.2 [73.7–80.4]	32.6 [32.1–33.1]	41.8 [37.7–42.6]
Adapters (KL)	91.4 [91.1–91.6]	68.3 [68.2–68.3]	57.5 [57.0–57.7]	98.1 [97.8–98.2]	92.0 [91.9–92.0]	48.3 [48.3–48.4]	71.6 [70.9–72.1]	83.0 [74.1–83.6]	93.0 [92.9–93.1]	68.2 [64.5–70.7]	77.7 [77.5–79.7]	52.2 [49.3–53.8]	59.9 [58.3–62.3]	43.8 [41.7–45.0]	61.2 [59.9–62.1]	94.1 [90.8–96.1]	77.9 [47.0–79.0]	24.9 [16.8–30.4]	34.4 [33.4–35.4]
Adapters (kNN)	91.6 [91.5–91.7]	68.4 [68.3–68.6]	72.2 [72.1–72.2]	97.7 [97.7–97.8]	91.9 [91.8–92.0]	49.8 [49.8–49.9]	81.1 [78.7–81.8]	83.3 [82.8–83.7]	93.0 [93.0–93.1]	67.3 [58.8–71.8]	81.7 [81.5–81.8]	61.2 [59.1–62.1]	78.0 [75.0–79.5]	44.9 [44.3–46.2]	61.8 [61.3–62.2]	89.3 [87.5–92.8]	78.8 [77.6–79.4]	24.1 [23.7–26.6]	34.5 [30.3–40.7]
Full (EPN)	91.6 [91.2–91.7]	53.2 [53.1–53.4]	63.3 [63.0–63.6]	99.5 [99.5–99.5]	96.1 [96.1–96.2]	55.1 [55.0–55.1]	72.3 [61.7–82.6]	83.3 [82.9–84.1]	94.0 [93.9–94.1]	74.1 [73.9–74.2]	74.5 [74.3–77.1]	56.8 [55.2–57.9]	62.9 [60.6–65.5]	38.2 [37.4–39.5]	64.2 [63.1–64.5]	96.5 [95.5–97.0]	75.5 [74.4–76.1]	22.8 [19.1–23.3]	39.1 [38.7–39.7]
Full (KL)	91.6 [91.5–91.7]	68.4 [68.2–68.6]	64.3 [63.8–64.7]	99.5 [99.5–99.5]	95.9 [95.9–95.9]	55.1 [55.1–55.2]	70.7 [70.2–71.8]	83.1 [81.8–83.5]	93.0 [92.9–93.0]	61.8 [57.4–70.4]	83.4 [83.4–83.5]	54.0 [50.3–55.3]	60.6 [56.5–61.3]	45.1 [41.5–45.2]	64.5 [62.9–65.2]	97.3 [96.8–97.5]	75.0 [74.8–75.4]	24.9 [18.3–25.2]	34.0 [32.0–35.2]
Full (kNN)	91.4 [91.2–91.6]	68.7 [68.4–68.9]	72.2 [72.0–72.3]	99.5 [99.5–99.5]	95.4 [95.3–95.4]	55.1 [55.1–55.2]	75.3 [74.4–77.4]	82.8 [82.5–83.4]	94.8 [94.4–95.1]	73.1 [72.1–73.8]	83.4 [83.4–83.5]	57.3 [55.3–58.1]	54.4 [51.0–57.7]	42.2 [41.8–43.0]	60.6 [58.6–62.2]	93.7 [88.9–95.7]	67.0 [44.0–71.3]	25.5 [25.3–29.0]	41.5 [39.9–42.2]
All Experts (kNN)	91.6 [91.4–91.7]	68.5 [68.4–68.6]	72.1 [72.0–72.3]	99.5 [99.5–99.5]	95.4 [95.3–95.4]	55.1 [55.1–55.2]	77.9 [71.8–80.7]	82.9 [82.6–83.2]	94.9 [94.7–95.0]	73.7 [73.4–73.9]	83.4 [83.3–83.5]	61.0 [58.4–61.8]	77.6 [74.2–80.7]	45.6 [44.8–46.4]	62.3 [60.0–62.9]	87.6 [85.7–90.6]	67.6 [66.8–71.8]	25.3 [25.1–25.6]	41.8 [40.9–42.1]
IN21k																			
Baseline	90.8 [90.7–91.0]	72.5 [72.4–72.6]	71.1 [71.0–71.2]	98.5 [98.4–98.5]	87.6 [87.4–87.7]	48.6 [48.6–48.7]	74.9 [72.6–75.5]	84.3 [84.1–84.5]	87.7 [73.4–94.6]	73.2 [72.9–73.8]	82.8 [82.2–83.0]	52.2 [51.1–53.4]	59.6 [58.1–61.8]	42.1 [37.5–43.2]	61.3 [60.0–62.2]	95.4 [94.4–96.0]	80.5 [78.7–81.6]	30.6 [27.7–30.7]	32.4 [29.7–33.3]
Adapters (kNN)	89.9 [89.7–90.9]	72.4 [72.3–72.6]	71.2 [71.1–71.3]	98.4 [98.4–98.4]	89.4 [89.3–89.6]	49.5 [49.4–49.6]	75.8 [75.3–76.2]	83.9 [83.1–84.3]	94.6 [94.5–94.6]	73.8 [73.2–74.0]	81.8 [80.5–82.0]	54.9 [53.5–55.5]	64.0 [61.1–68.4]	44.9 [44.2–45.2]	61.3 [60.2–61.9]	93.6 [91.4–94.3]	78.6 [77.0–79.5]	27.8 [27.7–31.2]	34.9 [34.4–35.6]
Full (kNN)	90.7 [90.3–90.9]	72.5 [72.5–72.5]	71.4 [71.4–71.5]	98.6 [98.5–98.6]	89.9 [89.4–90.1]	49.7 [49.6–49.7]	74.9 [74.5–77.2]	83.8 [83.5–84.2]	94.9 [94.8–94.9]	73.6 [72.7–74.0]	81.5 [81.5–81.7]	58.2 [55.6–59.4]	68.9 [65.8–70.6]	45.1 [44.5–45.7]	60.9 [59.5–62.0]	95.3 [93.5–96.2]	80.4 [79.7–80.9]	31.1 [30.5–31.8]	35.2 [34.7–35.8]
All Experts (kNN)	90.8 [90.7–90.9]	72.4 [72.2–72.5]	71.4 [71.3–71.4]	98.5 [98.4–98.6]	89.6 [89.4–90.1]	49.5 [49.5–49.6]	76.0 [74.7–77.5]	84.5 [84.0–84.8]	94.8 [94.8–94.9]	73.5 [73.2–74.1]	81.4 [81.3–81.5]	57.3 [55.2–58.5]	68.3 [62.6–72.1]	44.4 [44.3–44.6]	61.4 [60.4–62.0]	93.8 [92.1–95.1]	80.2 [79.6–80.7]	30.6 [30.1–31.3]	34.3 [33.9–35.4]
JFT + IN21k																			
All Experts (kNN)	90.7 [90.6–90.8]	68.5 [68.4–68.7]	71.4 [71.3–71.5]	99.5 [99.5–99.5]	95.4 [95.3–95.4]	55.2 [55.1–55.2]	80.6 [78.1–81.5]	84.5 [84.3–84.8]	94.9 [94.8–94.9]	73.2 [72.1–73.9]	83.5 [83.4–83.6]	61.4 [60.6–62.2]	78.1 [73.2–81.8]	44.8 [44.0–45.4]	62.5 [61.8–63.0]	91.0 [88.1–93.6]	66.9 [66.5–67.9]	30.8 [27.7–31.2]	40.9 [39.4–41.7]

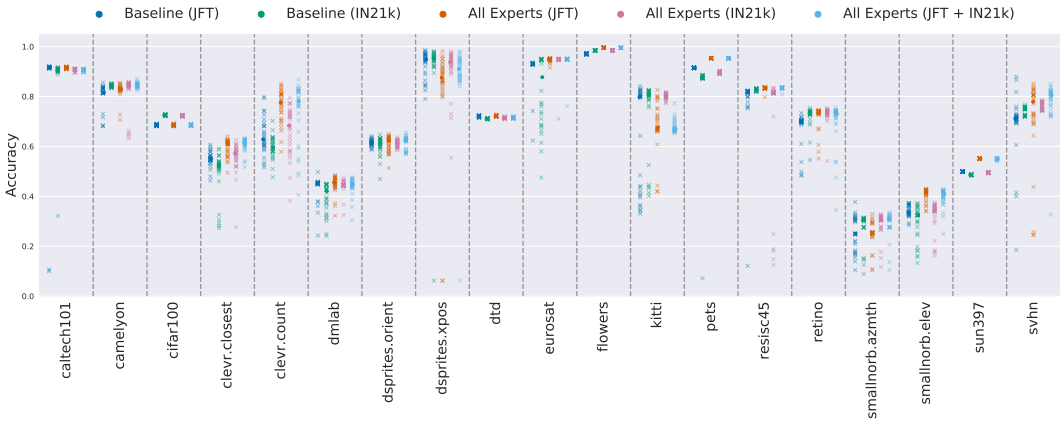


Figure 10: VTAB-1k accuracy in all datasets for 30 runs using different baselines and experts models trained on JFT and ImageNet21k. The median is represented by a darker point. Best seen in color.

F Details on the Comparison with Domain Adaptive Transfer

F.1 Hyperparameters

We randomly explored the space of hyperparameters drawing 36 samples from the following distributions:

- Initial learning rate: Log-uniform in $[2 \cdot 10^{-4}, 2 \cdot 10^{-1}]$.
- Total training steps: Uniform in $\{2000, 4000, 8000, 16000, 32000\}$.
- Weight decay: Log-uniform in $[10^{-6}, 10^{-2}]$.
- Mixup α: Uniform in $\{0, 0.05, 0.1, 0.2, 0.4\}$.

We use a batch size equal to 512 and decay the learning rate by 0.1 at 30%, 60% and 90% of the training duration. During the first 10% of training steps, we linearly warm up the learning rate. Standard data augmentation techniques are applied to prevent overfitting. During training, for all datasets except CIFAR10 (which has a smaller resolution), we resized the images to a fixed size of 512 pixels on both sides, then randomly cropped it to a patch of 480 pixels, randomly flipped the image horizontally, and converted the pixel values to the $[-1, 1]$ range. For CIFAR10, we used a resolution of 160 pixels during the resize and crops of 128 pixels. During evaluation, we simply resize the images and convert the pixel values analogously.

We find the best hyperparameter for each dataset based on the accuracy on the validation data, and then, applying the corresponding set of hyperparameters, the selected expert is fine-tuned again on the union of the training and validation examples of the dataset. The accuracy on a test set is reported. We re-trained the models 30 times using different random seeds.

For downstream training we use a Cloud TPUv3-16. In DAT, the running time depends on the number of steps selected as the hyperparameter and the resolution of the images. At most, it takes 150 minutes to fine-tune one of our experts on one downstream dataset, and at least it takes 8 minutes.

F.2 Detailed results

Table 7 shows the mean accuracy across those 30 trials and the 95% bootstrapped confidence intervals, for each of our experts and selection algorithms, for each dataset as well as the average across the six datasets.

F.3 Differences in asymptotic running time

Table 8 contains an asymptotic analysis of the different phases of each approach. In our case, upstream training includes both the cost of training the generic backbone network for S_U steps, and the cost of

Table 7: Accuracy on the datasets used in Ngiam et al. (2018), and the average accuracy across the six of them. Bootstrapped confidence intervals at 95% are shown below the accuracy, when available. The first two rows are Inception-v3 models, as reported in Ngiam et al. (2018). The rest of the rows are produced by our own models, based on Resnet-50-v2, grouped by expert selection method. The suffixes "2e" and "4e" denote that the expert modules were trained for 2 or 4 epochs, respectively.

	Aircraft	Birds	Cars	CIFAR10	Food	Pets*	Avg.
Baseline	91.4 [91.0–91.7]	78.8 [78.0–79.4]	95.6 [95.4–95.7]	97.8 [97.7–97.9]	91.3 [91.2–91.5]	94.5 [94.4–94.6]	91.6 [91.4–91.7]
kNN							
Adapters, 4e	92.5 [92.2–92.8]	79.4 [78.7–80.1]	95.9 [95.8–96.0]	97.9 [97.8–98.0]	91.6 [91.5–91.7]	94.6 [94.4–94.8]	92.0 [91.9–92.1]
Full, 2e	94.5 [94.2–94.7]	83.5 [83.1–83.9]	96.0 [95.8–96.2]	97.9 [97.8–98.0]	92.9 [92.8–93.1]	96.8 [96.7–96.9]	93.6 [93.5–93.7]
Full, 4e	**94.8** [94.5–95.1]	83.6 [83.1–83.9]	96.1 [96.0–96.3]	97.8 [97.7–97.9]	93.1 [92.8–93.2]	**97.0** [96.9–97.1]	93.7 [93.6–93.8]
KL							
Adapters, 4e	92.1 [91.8–92.5]	80.0 [79.5–80.4]	95.9 [95.8–96.0]	97.9 [97.8–98.0]	91.6 [91.5–91.8]	94.5 [94.3–94.7]	92.0 [91.9–92.1]
Full, 2e	94.6 [93.6–95.0]	83.1 [82.6–83.5]	96.1 [96.0–96.2]	97.9 [97.8–98.1]	92.9 [92.9–93.1]	96.6 [96.5–96.7]	93.5 [93.4–93.7]
Full, 4e	94.4 [94.1–94.7]	83.7 [83.3–84.3]	**96.4** [96.3–96.4]	97.9 [97.8–98.0]	93.1 [92.9–93.3]	96.6 [96.6–96.6]	93.7 [93.6–93.8]
EPN							
Adapters, 4e	92.1 [91.7–92.5]	79.7 [79.1–80.2]	95.8 [95.6–96.0]	97.3 [97.1–97.4]	91.5 [91.3–91.7]	93.1 [91.8–93.6]	91.6 [91.4–91.8]
Full, 2e	94.0 [93.6–94.3]	83.3 [82.7–83.9]	96.2 [96.1–96.2]	96.9 [96.7–97.1]	92.5 [92.3–92.6]	**97.0** [97.0–97.1]	93.3 [93.2–93.4]
Full, 4e	94.2 [94.1–94.4]	84.3 [83.9–84.7]	96.0 [96.0–96.1]	96.6 [96.4–96.7]	92.4 [92.3–92.6]	96.9 [96.9–97.0]	93.4 [93.3–93.5]
Dom-Ad (In-v3) Ngiam et al. (2018)	94.1	81.7	95.7	98.3	94.1	**97.1**	93.5
Dom-Ad (Am-B) Ngiam et al. (2018)	92.8	**85.1**	95.8	**98.6**	**95.3**	96.8	**94.1**

*Pets results are mean per class accuracy as opposed to mean accuracy.

Table 8: Asymptotic running times of Domain Adaptive Transfer (DAT) Ngiam et al. (2018) and our work, where P is the number of parameters of the network, B is the batch size, S_U is the number of training steps of the baseline model, S_A is the number of training steps for adapting the baseline model, S_F is the number of training steps for fine-tuning the specialist model to the downstream task, and E is the number of pre-trained experts in our approach.

	DAT Ngiam et al. (2018)	Ours
Upstream training	$O(S_U \cdot B \cdot P)$	$O((S_U + S_A \cdot E) \cdot B \cdot P)$
Downstream		
Expert preparation	$O((N_T + S_A \cdot B) \cdot P)$	$O((N_T \cdot P + N_T^2) \cdot E)$
Fine-tuning	$O(S_F \cdot B \cdot P)$	$O(S_F \cdot B \cdot P)$

training each of the E experts for S_A steps, with a batch size of B. In the case of Domain Adaptive Transfer (DAT), only the first cost is incurred in this phase. Observe that in our case the cost of upstream training is amortized over the number of tasks that one has to learn, since it's only incurred once.

In the downstream phase, both methods need a forward pass over the number of downstream examples, N_T. Then, leaving the number of parameters of the model aside, the cost of Ngiam et al. (2018) is dominated by $S_A \cdot B$, which is the total number of examples used to fine-tune the baseline model to a weighted/resampled version of the upstream data, and ours is dominated by $N_T \cdot E$, the cost of running a forward pass of the downstream data in each of the pre-trained experts. In practice, because $S_A \cdot B$ (roughly $1.2 \cdot 10^9$, when fine-tuning for 4 epochs on JFT) is much larger than $N_T \cdot E$ (roughly $5 \cdot 10^5$, when using 1 000 downstream examples for selecting over 500 experts), our approach is much faster. Thus, our approach should be roughly three orders of magnitude faster than that of Ngiam et al. (2018), when learning a new task. The final cost of fine-tuning to the downstream dataset is the same in both cases, and it's negligible in comparison.

As described in appendix C.1, we use *a single NVIDIA V100 GPU* as the hardware accelerator when selecting the expert model to use for a given downstream task. It requires less than 2 hours[3] to select a single expert model among a pool of 240 experts, based on 1 000 images from the downstream task. This is the real duration of the "Expert preparation" step, following our method, depicted in table 8.

[3]Most of the time is actually spent reading the model parameters from disk.

Under the same hardware constraints, fine-tuning a R50 on JFT for 4 epochs, using the details described in appendix D, requires around 1780 hours. This is essentially the duration of the "Expert prepartion" step, following the Domain Adaptive Transfer approach[4].

Given that fine-tuning on the downstream dataset takes only a few minutes (see appendix E.2), our transfer approach is $890\times$ faster than that of Domain Adaptive Transfer. Depending on different optimizations and bottlenecks, we expect our approach to be $500\times - 1000\times$ faster in other scenarios.

G THE VALUE OF SEMANTIC EXPERTS

In section 6.4, we have seen that a smart choice of experts leads to substantial gains with respect to a single model trained on all the upstream data. Here, we rule out the possibility that these gains come merely from the fact that we are able to select a representation among a wide range of choices by *directly* testing their initial predictive power on the downstream task. To do so, instead of training our experts in subsets of JFT based on its label hierarchy, we fully fine-tuned the baseline model on 240 uniformly random subsets of JFT, with sizes matching the size of our original semantic experts. We did this independently with both adapter and full experts. Then, we applied kNN to select the best random expert on each downstream dataset. Note this is not at all equivalent to applying transfer at random as in section 6.4.

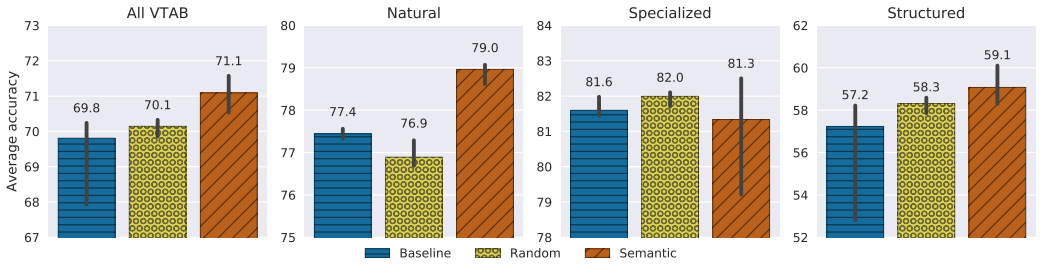
In principle, it was not even clear if this approach with random experts would outperform the baseline. Figure 11 (a) and (b) show our results for adapter and full experts respectively.

In both cases, the overall performance drops when we replace semantic experts with random ones. This difference seems stronger in the case of adapter-based experts. However, notice that the full expert results are very influenced by strong negative results in one of the structured datasets (Clevr Count), as shown in table 6. Also, random experts results are comparable to the baseline. Note that random experts are not dumb; they are just a diverse set of models with the general flavor of the upstream dataset. The algorithm can still benefit from their diversity when confronted with a new task, whereas we expect them to be more similar to each other than in the semantic case.

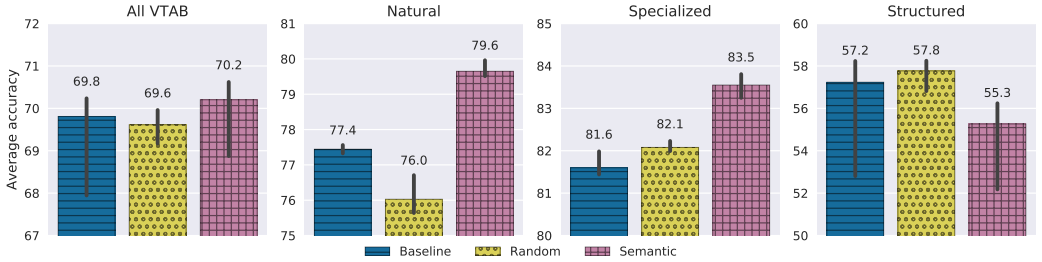
The semantic experts are trained mostly on natural slices, and we see a large improvement in the NATURAL tasks when we use them (2.7% and 4.7% gains for adapters and full, respectively, compared to random experts). This reinforces the idea that experts in the right domain can be very helpful. Moreover, in NATURAL tasks, the baseline outperforms random experts; this suggests that here more data is better unless data is smartly selected.

As we have hypothesized before, it seems that our natural-image experts do not provide a meaningful expertise or competitive edge on STRUCTURED tasks. We see a large improvement on SPECIALIZED tasks when using full experts, while the effect is not there for adapters. Accordingly, we would not read too much into these results.

[4]In fact, we are ignoring the cost of the forward pass through the 1 000 downstream datasets, but that cost is negligible compared to fine-tuning on JFT.



(a) JFT Adapter Experts



(b) JFT Full Experts

Figure 11: Results on VTAB with 1 000 examples per dataset achieved by experts trained on random subsets of JFT and experts trained on semantically meaningful subsets. For each dataset in VTAB, the median accuracy over 30 trials is considered. The results of the datasets in each group are averaged. The error bars show the (percentile) bootstrapped confidence intervals at 95% level.