Supplementary Materials

We provide details omitted in the main paper.

- Appendix A: additional details of proposed TAPER method (cf. section 3 and section 4 of the main paper).
- Appendix B: additional experimental results and analysis (cf. section 4 of the main paper).

A. Additional Details of TAPER and Experiment

Improved three-stage training of TAPER. We provide more details about our proposed improved three-stage training pipeline in cf. subsection 3.4 in our experiments. Intuitively, each mixer vector will only select a subset of bases that are relevant to the task, and each basis should encode specialized knowledge. This leads to a dilemma that each basis model might only be trained on a small portion of the whole dataset/all the tasks. This makes it less generalized to test images, even if the test images are from its specialty, since it might learn less comprehensive low-level features (*e.g.*, texture, shapes, etc) given the small size of data it has seen. This motivates our design for the improved training in subsection 3.4. More details in each stage are as follows:

- **Stage 1: single basis pre-training.** We begin with a single network $\boldsymbol{\theta}^{(0)}$ to learn the general representation of the whole dataset in a standard way, *e.g.*, with cross-entropy. This step is exactly the same as training a conventional non-personalized predictor.
- **Stage 2: specialized basis models.** Next, given the pre-trained model in stage 1, we want to prepare Q specialized networks as the initialization for the Q bases. We split the dataset into Q shards based on classes or domains. For each shard $q = 1, \ldots, Q$, we copy $\boldsymbol{\theta}^{(0)}$ as the initialization, fine-tune it, and collect the "expert" model as $\boldsymbol{v}_q$. At the end of this stage, we will have $\{\boldsymbol{v}_q\}_{q=1}^{Q}$. We note that the purpose is just to burn in each basis different domain knowledge as warm starts. Our current way to define the shards is simple and intuitive (see the details in the next item), but we believe there is room for improvement in designing better engineering at this stage.
- **Stage 3: learning to mix the bases for tasks.** The previous two stages do not even have any "tasks" but just to pre-train the bases as good initialization that already encoded certain specialties. In this stage, we jointly learn both the bases $\{\boldsymbol{v}_q\}_{q=1}^{Q}$ and the mixer predictor $g(\boldsymbol{d}; \phi)$ to combine them for all the tasks, guided by the task descriptions. Note that, we use the classifier $\boldsymbol{w}_t$ selected for each task, building upon subsection 3.2.

We note that the "tasks" used in stage 3 are not directly tied to how we define the "shards" of datasets in stage 2. Instead, all about the bases $\{\boldsymbol{v}_q\}_{q=1}^{Q}$ and the mixer predictor $g(\boldsymbol{d}; \phi)$ are learned end-to-end without any hard constraints on how they should select the bases. By jointly learning, we let it automatically decide the best allocations of the bases across different tasks, classes, domains, etc. For instance, it is possible that the classes of the tasks are assigned to bases that are different from the stage 2 shards.

Detailed experiments setups. For datasets, we use the same data pre-processing/augmentation [17] and the resolutions are set to 224×224. To simulate the tasks, we construct the tasks as 20-way classification by sampling from the label space $\mathcal{Y}$. Each image from the training/validation set is randomly assigned with a task description as discussed in subsection 3.5 for training (sampled every epoch) and evaluation, respectively. The goal is to accurately predict the labels from the whole $\mathcal{Y}$ and the metric is the standard top-1 accuracy.

Task description. As defined in subsection 3.1, we currently constrain the "task" to be a set of classes. This is practical in many real-world applications where a typical pre-processing is to map a concept (*e.g.*, "kitchen") to its related classes (*e.g.*, cookers) using a commonsense knowledge graph (*e.g.*, WordNet). We leave different task descriptions such as high-level natural language instructions to be our future work. For the task description vector $\boldsymbol{d}_t$, we simply follow the script[3] in [32] to generate for each class a text embedding and average over $\mathcal{Y}_t$.

For ImageNet, we assign each image a k-way task ($k = 20$ by default) by sampling from classes that are the nearest $2k$ synsets in the WordNet knowledge graph [30] based on its ground-truth label (which is included in the task as well. Note that, we encode each class and average over classes as the task embedding thus it will not leak the ground-truths.); *e.g.*, *"coffee pot"* and *"espresso maker"* are more likely in the same task. This is to simulate a more realistic use case that the users might want a predictor for a certain environment, not for some random combinations. For the stage 2 training, we split 1000 classes into 100 classes per shard based on their synsets locations.

[3] https://github.com/openai/CLIP/blob/main/notebooks/Prompt_Engineering_for_ImageNet.ipynb

For iNaturalist, we construct each image a 20-way task description by sampling other classes from the same super-category; there are $10,000$ species from 11 super-categories such as *"Mammals"* and *"Reptiles"*. For the stage 2 training, we split the classes in the super-categories into 3 shards per super-category based on their biological distance.

For DomainNet, there are 6 domains of image styles over 345 common objects. Each task is from one domain but with different 20 class random combinations. To generate the task embedding, we create the task descriptions by attaching the domain name before each of the class names, *e.g.*, *"This is a **sketch airplane**."*, encoding each class to retrieve the textual embedding and averaging over the classes within the task. For the stage 2 training, we split the datasets based on the domains and then further split them by classes.

Training hyperparameters. We use the training process similar to the standard way on ImageNet in [17] for all the datasets, and the learning rate schedule (initial learning rate is 0.1 and decays by 0.1 every 30 epochs). We use the SGD optimizer with momentum $= 0.9$, batch size $= 128$, and weight decay $= 0.0001$. Our experiments are implemented using JAX [3]. We follow the official JAX/FLAX ImageNet training script[4]. We train on randomly-initialized **ResNet-18** networks [17] with cross-entropy by default. For TAPER, we train each stage sequentially for $100/5/20$ epochs, for the 3 stages, respectively. For a fair comparison, we, therefore, train 125 epochs for the baseline approaches (subsection 3.2). Our implementation will be shared in: https://github.com/google-research/google-research.

B. Additional Experimental Results and Analysis

B.1. More discussions on the design choices (cf. subsection 4.5)

Table A. **Ablation study for different design choices of TAPER (copied from Table 6 just for the ease of reference here).** The indentation with different symbols denotes adding (+) / removing (–) a component, or using a variant (○). We report the mean±std based on 3 runs on ResNet-18. *Accuracy here is averaged over examples.

Design choices	Methods / Datasets (#Bases Q)	ImageNet (10)	iNaturalist (33)	DomainNet* (18)	Avg. Accuracy
①	Standard, w/o personalization	69.9 ± 0.25	72.3 ± 0.52	65.8 ± 0.23	69.3
②	+ Classifier selection: a strong baseline	92.2 ± 0.36	90.8 ± 0.75	88.4 ± 0.54	90.5
③	TAPER w/ naive training & classifier selection	81.8 ± 2.45	75.7 ± 3.01	78.6 ± 1.44	78.7
④	TAPER at Stage 1	69.8 ± 0.34	72.3 ± 0.46	65.8 ± 0.26	69.3
⑤	+ Stage 2 & classifier selection	91.2 ± 1.56	89.3 ± 2.45	88.5 ± 1.16	89.7
⑥	+ uniform weight average	86.1 ± 1.52	15.8 ± 7.55	87.2 ± 2.66	63.0
⑦	+ fine-tuning w/o task description	92.1 ± 0.56	91.0 ± 0.76	88.4 ± 0.44	90.5
⑧	+ Stage 3 (complete TAPER)	95.8 ± 0.45	95.9 ± 0.72	94.1 ± 0.63	95.3
⑨	○ BoW task description	94.9 ± 0.51	93.1 ± 0.81	93.5 ± 0.74	93.8
⑩	– Block-wise mixers	94.0 ± 0.24	93.1 ± 1.20	91.7 ± 0.39	92.9
⑪	– classifier selection	84.3 ± 0.57	81.0 ± 1.75	87.5 ± 0.64	84.3

In subsection 4.5, we discuss our design choices proposed in subsection 3.3 and subsection 3.4. Due to the space limitations of the main paper, we provide a more detailed discussion here. Please refer to the indexes in Table A (same as Table 6). Note that, for the sake of consistency with the test loss and other datasets, the accuracy of the DomainNet dataset here is averaged over examples instead of over domains as in Table 5.

We observe:

- **Personalization baseline.** The classifier selection baseline (②) outperforms the non-personalized, plain ResNet-18 (①) significantly. This validates the value of personalization and the feasibility of train-once-for-all personalization.
- **TAPER with naive training (③).** TAPER with naive training (③) directly optimizes the basis models and the mixer predictor in Equation 2 from scratch. It outperforms a non-personalized network (①) but not the classifier selection baseline (②), even ③ is attached with classifier selection already. We hypothesize the bases are not properly trained and poor in generalization.
- **Sanity checks.** TAPER's stage 1 (④) is basically the same as ① that trains a plain ResNet in a standard way, but ④ is trained for 100 epochs, less than 125 epochs for ①. This is for the fair comparison as TAPER's stages 2 and 3 will train 25 epochs. This confirms that simply training more will not improve as the models are already converged. Stage 2 (⑤)

[4]https://github.com/google/flax/tree/main/examples/imagenet

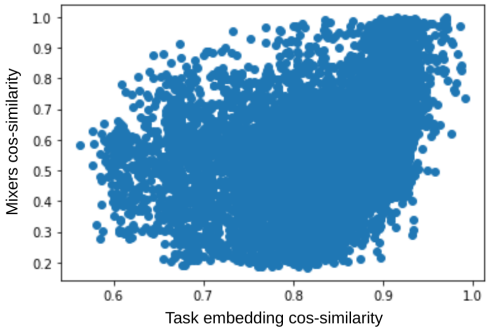

Figure A. **Mixers and the corresponding task embedding have high correlations (Pearson's $r = 0.46$).** Visualization on the ImageNet dataset. See more details in subsection B.2 about how to compute it.

is slightly worse than ② as expected since the models are specialized on a shard of the dataset. Note that, the evaluation of Table A are all on the official validation/test sets that contain all the classes. We further verify that the improvements of TAPER are not because in stage 2 we fine-tune many models and average them on weights. Simply averaging them on weights (⑥) will not become a stronger model. It still needs stage 3 to learn how to mix the bases.

- **Task descriptions help.** From ⑧ to ⑪, TAPER leverages task descriptions to personalize the features by the mixer predictor thus outperforming the baseline (②). We validate that the benefits are from the guidance of task descriptions by replacing the mixers with a fixed uniform vector and fine-tuning all the bases together (⑦), it will recover ⑥ to ② but still lower than ⑧. From the architecture's view, ⑥ is simply collapsed to a single-basis model like ②; from optimization's view, it is slightly different since we jointly learn several copies and average them every step. We verify this difference is negligible and the improvements are not due to this.
- **Minor choices.** We found some minor designs help TAPER to be even better. Text embedding is better for task descriptions compared to BoW vectors (⑨). It is preferred to have mixers block-wise (⑩) since it allows the bases to have more freedom to be mixed layer-wisely. We note while these alternative choices are worse than the complete TAPER (⑧), they are still effective compared to the baselines.
- **More discussions on DomainNet.** For DomainNet, we encode the tasks based on their image style domains and classes. We further consider one more baseline. Starting from ③ that the features are generic, we fine-tune it on each of the 6 domains to get 6 different domain-specific models and use the corresponding one in evaluation. We observe it hardly improves on ③ but drops by 1.9% on average. We hypothesize the reason might be each domain have much fewer images per class. On the contrary, using TAPER allows the bases to collaborate end-to-end and the mixer predictor can jointly consider sharing both the domains and classes information across different bases.
- **Classifier selection is important.** Removing classifier selection from TAPER (⑪) has a big impact. However, comparing ⑪ to ① and ④ that the features are generic without personalization, we validate that TAPER indeed learns personalized features.
- **Conclusion.** We thus conclude that the complete TAPER (⑧) performs the best consistently.

B.2. Correlations of the mixers and the task embedding

We provide one more evaluation as a sanity check that the mixers predicted from the mixer predictor are not random vectors but follow the task embedding. In cf. Figure 4 (a) we already visualize that the task embedding of similar classes inside results in similar mixers. Here we provide a more quantitative way based on cosine similarity. Intuitively, the mixers and their corresponding task embedding should have similar properties. However, it is not trivial to directly evaluate its distance by, *e.g.*, cosine similarity, since the mixers and the task embedding have a different number of dimensions and their coordinates are not aligned in a meaningful way. We, therefore, use another workaround, by assuming the pairwise similarities between the mixers themselves and the task embedding themselves should be similar. We first take a batch of tasks embedding and compute their pairwise similarities inside the batch, *i.e.*, we got a similarity matrix. We then input this batch of tasks embedding into the trained mixer predictor and in the same way compute the similarity matrix of the mixers. We then visualize the correlations between the two matrices (off-diagonals) in Figure A. We calculated the correlation coefficients and see a high 0.46 correlation. We note that we verify that this correlation will be almost 0 if using a randomly initialized mixer predictor.

B.3. Effects of different settings

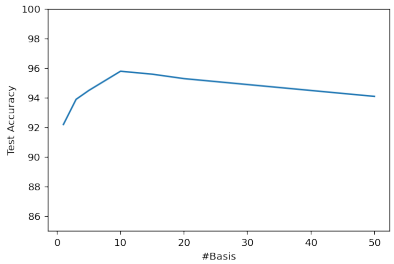

Figure B. **#Bases.**

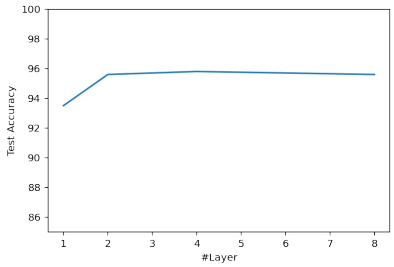

Figure C. **#Layers in the mixer predictor.**

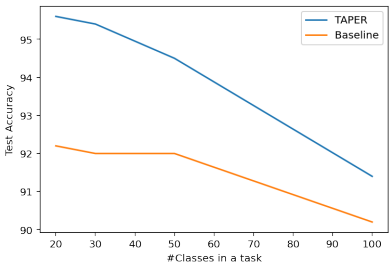

Figure D. **#Classes in a task.**

We now provide more details studied on some minor factors in our experiments or our complete TAPER algorithm. We focus on the ImageNet experiment (cf. Table 2) by default.

Number of bases. We investigate the effects of using more/fewer bases in Figure B. We observe using a small number of bases (*e.g.*, 10) is enough for ImageNet. We see using more bases saturates or even degrades the performance. This is understandable since we did not apply any advanced regularization except simple weight decays and image augmentation to prevent overfitting. We leave more dedicated designs to scale to larger datasets and more bases in our future work.

Number of layers in the mixer predictor. In the main experiments we fixed the mixer predictor to be a 4-layer MLP with ReLU activation functions and batchnorms for the latent layers. In Figure C, we validate that the number of layers in the MLP is not a sensitive choice — $2 \sim 4$-layer is strong enough to predict the mixers.

Number of classes in a task. In cf. Table 3, we validate that TAPER can handle scenarios of that different numbers of classes in a task. Here we provide one more evaluation. We take the TAPER model trained with tasks dynamically drawn with $5 \sim 100$ classes per task and evaluate tasks with different classes per task in Figure D by comparing to the baseline with classifier selection. We see TAPER consistently outperforms the baselines, and both accuracies drops as more classes are included in a task, *i.e.*, the tasks become harder.