A. Experimental and Implementation details

In this section, we provide more details about our experiments. We conduct all experiments using PyTorch [22]. We use two ImageNet pretrained base models for fine-tuning. One of these is the default `ResNet50_Weights.IMAGENET1K_V1` from PyTorch, while we pre-train the other starting from random initialization following the same pipeline. For fine-tuning the models on each domain, we use the Adam [17] optimizer, and sweep the learning rates logarithmically between [1e-4,1e-1], testing out 4 values for LR. We validate on the validation subset wherever available, and on 10% of the training dataset where an explicit val set is not provided. We use standard image augmentation techniques. Our MoE model has a light-weight router, which is a 3 layer CNN trained to predict which model to use for classifying an image.

For finding permutation symmetries, we use the official implementation of Git Re-Basin at this url. We also rely on the implementation of ZipIt! and MuDSC for the comparisons in Sec. 5.

For solving the least squares objective for `PLeaS`, we use SGD with a batch size of 32, a learning rate of 10^{-3}. We sample equally from both datasets in each batch for experiments involving data. We run our algorithm for 100 steps, and find that it converges quickly. For the first step of `PLeaS`, we similarly compute the activations on 100 batches of data for matching and finding the optimal permutations. We also reset batch norm parameters using 100 batches of data from the actual domains for all methods.

For evaluations concerning the same label space setting, we ensure that the final model produces a distribution over the output classes. For ZipIt!, we achieve this by ensembling the predictions across multiple task specific heads. `PLeaS` on the other hand already produces models with the same output dimensions as the original models.

For evaluations on different label spaces, we train a linear probe on the final layer representations for each merged model. We use training data from the target domains to train this linear probe, run Adam with a learning rate of 10^{-3}, with a batch size of 64 for 50 epochs.

As an example evaluation for Domainnet, we have 8 models, two each on Clipart, Infograph, Painting and Real domains. We merge these pair-wise. We hence have 12 (6 domain pairs and two models per pair) merged models. For each merged model, we compute the accuracy on its component domains. Hence, for each domain, we have 6 performances (3 domain pairs and two models per pair). We report the average of these 6 numbers in Tab. 1.

ViT To obtain models for merging with CIFAR-50+50 for Sec. 5.5.1, we follow the protocol from [32]. In particular, we use ImageNet pretrained ViT models, and train two such models on disjoint 50 class subsets of CIFAR-100. These are trained with CLIP language embeddings for the final layer. This process is repeated thrice to get different sets of classes and pairs of models. After merging, the performance of the merged model is measured on CIFAR-100 as well as the two subsets of classes that the moders were trained on. The average of these results is reported in Tab. 3. For the other datasets, we start off with ViT-B/16 model from the OpenCLIP project[11] which is pretrained on the LAION-400M dataset. We fix the text encoder, and fine-tune the image encoder on various datasets. This fine-tuned model achieves an accuracy of (72%, 91%, 68%, 24%) on CUB, Pets, Dogs and NABirds datasets resp. We then merge the model pairwise and report the results.

A.1. Compute time and cost

All our experiments (apart from the pretraining and fine-tuning runs to get the original models) are run on a single RTX 2080 Ti GPU. The first step of our method runs in 2 minutes, with the majority of time devoted to computing the activations. This is commensurate with ZipIt! [28] and Git Re-Basin [1] The second step takes around 4 minutes, which is similar to RegMean [14]. We believe that this can be significantly reduced with better dataloading strategies and more efficient implementation, but that is beyond the scope of this paper.

A.2. Computing the permutation matrices

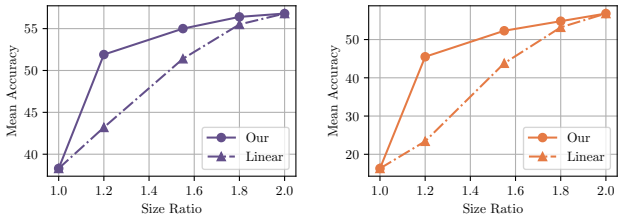
We use the algorithms of Git Re-Basin to compute the permutation matrices P_i. For activation matching, we collect the representations for both models over a batch of data, and measure the alignment between two neurons as the squared distance between their representations (closer $\rightarrow$ better alignment). Then for each layer, we find the bipartite matching (i.e. permutation) between the two models that minimizes the total distance using a standard algorithm (`scipy.optimize.linear_sum_assignment`). For weight matching, the alignment between two neurons is the squared distance between its input and output vectors. This time, the permutations at adjacent layers interact, so we perform an alternating minimization, solving for the permutations one layer at a time until we reach a fixed point.

A.3. Computing the layerwise merging ratio

Note that k_i can be different for each layer. Given a configuration $K = \{\frac{k_i}{d_i} : i \in [L]\}$, we can model the FLOPs/memory of the merged model as a quadratic function of k_i, which we denote as Footprint(K). For a given relative memory/FLOPs budget B, we want to find K s.t. Footprint(K) $\leq B$ to maximize the accuracy of a model merged with the configuration K. We scale everything so that $B = 1$ corresponds to the footprint of a single model. This problem is NP-Hard. We propose a relaxation of the problem in order to get an approximate solution. First, we measure the performance of a set of models merged with "leave one out" configurations of K, where for each layer i, we construct $K_i^0 = \{k_j : k_j = d_i \text{ if } j = i,\ 0 \text{ otherwise}\}$ and $K_i^1 = \{k_j : k_j = 0 \text{ if } j = i,\ d_i \text{ otherwise}\}$. K_i^0 corresponds to merging only layer i, keeping all other layers unmerged, and K_i^1 corresponds to merging every other layer while keeping i unmerged. We also compute the accuracies of the fully merged model (denoted by K^0) and the ensemble (denoted by K^1). Then, we approximate the accuracy of any given K with a linear function as

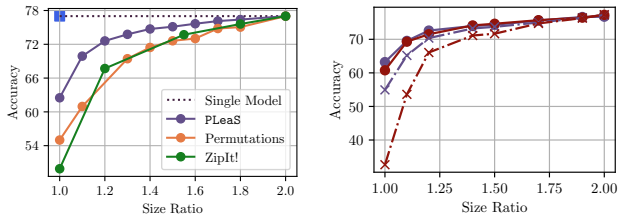
$$\text{Acc}(K) = \sum_{i=1}^{L} \frac{k_i}{d_i} \Big((2-B)(\text{Acc}(K^1) - \text{Acc}(K_i^0)) - (1-B)(\text{Acc}(K_i^0) - \text{Acc}(K^0)) \Big)$$

This approximates the effect of k_i on model performance at budget B by linearly interpolating between the performance with fully merging layer i and keeping it separate. We then propose to solve



(a) PLeaS (b) Permutations

Figure 5. **Comparing our strategy for layer-wise merging with a linear baseline**: We merge models using PLeaS and permutations using the strategy described in Appendix A.3 and a linear strategy where $\frac{k}{d}$ is held constant.



(a) ImageNet performance (b) Data-free setting

Figure 6. **Merging models trained on ImageNet**: In Fig. 6a, we demonstrate how PLeaS-free can reduce the accuracy barrier by 8% for merging independently trained ImageNet models. This is further reduced for larger target model sizes. In Fig. 6b, we show the effect of using synthetic data for computing the activations on this task, and find that synthetic data is a viable alternative at larger model sizes.

a quadratically constrained linear program to maximize $\text{Acc}(K)$ subject to $\text{Footprint}(K) \leq B$. This program is non-convex however Gurobi [8] is able to solve the program to global optimality in a few seconds. To faithfully compute the performance of the merged model, one would require validation samples from the target domain. However, we empirically observe that using the accuracy of a configuration K on ImageNet is a good proxy for its performance on other merging tasks as well, and we hence use it to compute the layer-wise merging ratio for all our experiments.

A.3.1. Empirical Results

In Figs. 5a and 5b, we compare the QP method with a baseline strategy which assigns the number of units in each layer to be a constant ratio. We find that our strategy outperforms this baseline for both PLeaS and Permutations.

B. Additional Results

B.1. What to optimize for Least Squares?

In Eq. (1), we propose to solve a least squares problem involving the permuted average activations from each layer of the component models. In Tab. 4, we demonstrate that this choice is not only natural, but also performs better than other alternatives. It is also interesting to note that the second row in the table corresponds to a permuted version of RegMean[14]. This formulation performs better than RegMean, indicating that using permutations is necessary to align features for networks which were differently initialized. Further, row 3 is similar to the objective proposed by [10], but we show that PLeaS outperforms this objective as well.

B.2. Reducing the accuracy barrier on ImageNet

In this section, we show the performance of PLeaS while merging ResNet-50 models trained independently on ImageNet. The accuracy of a single model on this task is 77.5%. As seen from Fig. 6a, current methods including ZipIt! [28] and Git Re-Basin [1] struggle on merging models for this task, with the accuracy of the merged model being significantly lower than the accuracy of a single model. This has been referred to as the accuracy barrier on ImageNet in prior work. PLeaS makes some progress towards lowering this barrier, and improves over Git Re-Basin by over 9% at $1.0\times$ FLOPs budget. For context, this accuracy is at par with that obtained by merging WideResNet-50 models with a width multiplier of 2 using Git Re-Basin. More promisingly, the flexibility afforded by partially permuting and merging models gives another avenue to lower the accuracy barrier, with a model of size $1.4\times$ having an accuracy barrier of 2% with PLeaS. However, further work is needed to reduce this accuracy barrier. In Fig. 6b, we compare using synthetic data from [6] for all purposes of activation computation while merging ImageNet trained models. We find that using PLeaS-free with synthetic data can come close to using actual data, being within 1% in terms of accuracy at $1.2\times$ model size.

B.3. Detailed Results

Each of our evaluation was run across three random restarts. These random restarts shuffle the data used for computing activations and merging the models. They also affect the initialization of the merged model. Each pair evaluation was also run twice, swapping the order of pre-trained models used for either of the datasets of the pair. We hence have 6 runs for each dataset pair. In Tabs. 5 and 6, we provide the results for each dataset pair, reporting the average and standard deviation across the 6 runs.

B.4. Using task specific heads

In Tab. 7, we report the results computed using the protocol mentioned in [28]. We find that PLeaS outperforms ZipIt! in this evaluation across model budgets.

B.5. PLeaS with weight and activation matching

In Fig. 7, we compare PLeaS-Weight with PLeaS, and find that PLeaS is better for larger models, while at small sizes the two methods give a comparable performance.‘

C. Broader Impact

Advances in model merging, especially through methods which do not require training data, can help further democratize machine learning by helping practitioners improve the capabilities of open source models. However, the risk of merged models inheriting biases of the component models still remains.

Table 4. **Comparing different objectives for `PLeaS`** : We compare the performance of different loss functions for the Least Squares component of `PLeaS` . We find that Eq. (1) gives the best performance on DomainNet and ImageNet when merging models completely. Here $\tilde{Z}_i = 0.5(Z^a_{:,i} + P_i Z^b_{:,i})$ and $\tilde{Z}_{i+1} = 0.5(Z^a_{:,i+1} + P_{i+1} Z^b_{:,i+1})$

Optimization Objective	DomainNet	ImageNet
$\|Z^a_{:,i}W - \tilde{Z}_{i+1}\|^2 + \|P_i Z^b_{:,i}W - \tilde{Z}_{i+1}\|^2$	22.3	45.1
$\|Z^a_{:,i}W - (Z^a_{:,i+1})\|^2 + \|P_i Z^b_{:,i}W - (P_{i+1}Z^b_{:,i+1})\|^2$	30.6	53.2
$\|\tilde{Z}_i W - Z^a_{:,i+1}\|^2 + \|\tilde{Z}_i W - P_{i+1}Z^b_{:,i+1}\|^2$	34.3	58.1
$\|\tilde{Z}_i W - \tilde{Z}_{i+1}\|^2$	**40.1**	**63.1**

Table 5. **Detailed Results on DomainNet** We report the results for ResNet-50 here

Method	Budget	Data	in-re		cl-pa		cl-in		pa-in		cl-re		pa-re	
PLeaS-Act	1.2	Original	26.3 ±1.2	69.8 ±0.5	57.3 ±0.7	53.6 ±0.1	55.4 ±0.0	26.6 ±0.0	52.7 ±0.2	25.6 ±0.1	59.1 ±1.3	69.2 ±1.2	56.8 ±1.2	70.3 ±0.9
PLeaS-Act	1.2	Imagenet	26.6 ±0.9	69.6 ±0.8	56.4 ±0.9	53.9 ±0.2	55.8 ±0.7	26.3 ±0.5	53.3 ±0.2	25.6 ±0.4	59.1 ±1.3	69.3 ±0.9	56.7 ±1.0	70.2 ±0.9
PLeaS-Act	1.0	Original	17.4 ±1.3	55.0 ±1.5	40.2 ±2.0	39.9 ±1.1	39.5 ±1.5	17.8 ±0.8	35.9 ±0.0	17.3 ±0.0	42.8 ±2.0	54.9 ±2.0	42.3 ±1.7	56.1 ±1.2
PLeaS-Act	1.0	Imagenet	17.0 ±1.3	51.9 ±1.3	39.7 ±2.4	38.8 ±0.8	37.6 ±2.1	16.6 ±0.7	37.1 ±1.6	16.5 ±0.7	42.2 ±2.9	53.9 ±1.5	40.9 ±2.6	54.4 ±0.9
PLeaS-Act	1.55	Original	29.0 ±0.8	72.4 ±0.3	60.6 ±0.6	57.3 ±0.0	59.7 ±0.6	28.7 ±0.3	56.4 ±0.1	27.9 ±0.4	62.5 ±0.8	72.3 ±0.9	60.6 ±1.0	72.7 ±0.7
PLeaS-Act	1.55	Imagenet	29.0 ±0.8	72.5 ±0.6	60.3 ±0.7	57.8 ±0.5	59.9 ±0.6	29.0 ±0.2	56.9 ±0.2	28.1 ±0.3	62.4 ±0.9	72.6 ±0.7	60.1 ±0.9	72.9 ±0.8
PLeaS-Act	1.8	Original	29.5 ±0.8	73.7 ±0.3	62.3 ±0.5	59.0 ±0.2	61.3 ±0.4	29.8 ±0.2	58.3 ±0.2	29.1 ±0.0	63.7 ±0.6	73.8 ±0.6	62.0 ±1.0	73.9 ±0.7
PLeaS-Act	1.8	Imagenet	30.1 ±0.8	73.6 ±0.6	62.1 ±0.2	59.9 ±0.4	61.6 ±0.3	29.9 ±0.1	58.4 ±0.2	28.8 ±0.1	63.7 ±0.7	74.0 ±0.7	61.7 ±0.5	73.9 ±0.7
Permutation-Act	1.2	Original	22.3 ±0.9	62.9 ±1.2	50.7 ±1.2	45.9 ±0.2	48.8 ±0.0	22.1 ±0.0	45.4 ±0.4	21.9 ±0.4	51.4 ±1.0	61.6 ±1.7	49.1 ±1.3	63.9 ±1.1
Permutation-Act	1.0	Original	7.6 ±0.2	24.5 ±1.9	15.4 ±1.7	15.5 ±1.8	15.6 ±0.5	7.6 ±0.5	15.4 ±0.9	6.8 ±0.0	17.1 ±1.4	24.6 ±1.7	18.7 ±0.6	26.6 ±2.0
Permutation-Act	1.55	Original	27.1 ±0.7	69.7 ±0.7	58.2 ±1.0	53.6 ±0.4	56.6 ±0.6	26.8 ±0.1	53.3 ±0.4	25.9 ±0.3	59.3 ±1.1	68.8 ±0.8	57.2 ±0.8	70.3 ±1.0
Permutation-Act	1.8	Original	28.9 ±0.8	71.8 ±0.6	60.7 ±0.6	56.8 ±0.3	59.7 ±0.6	28.6 ±0.3	56.5 ±0.4	27.7 ±0.5	62.4 ±1.0	71.4 ±0.8	60.2 ±0.8	72.4 ±0.9
PLeaS-Weight	1.2	Original	27.3 ±1.0	70.5 ±0.5	58.8 ±0.2	54.6 ±0.3	56.6 ±0.8	27.2 ±0.3	53.5 ±0.1	26.1 ±0.5	60.9 ±0.2	68.8 ±0.1	58.3 ±0.9	70.5 ±0.8
PLeaS-Weight	1.2	Imagenet	27.4 ±0.9	70.2 ±0.6	57.5 ±0.9	54.3 ±0.3	56.9 ±0.4	27.3 ±0.3	53.7 ±0.4	26.0 ±0.6	59.6 ±1.0	69.9 ±1.1	57.6 ±0.8	70.8 ±0.9
PLeaS-Weight	1.0	Original	19.2 ±2.0	56.8 ±1.4	45.6 ±0.2	39.9 ±0.2	41.0 ±1.5	19.0 ±1.5	39.6 ±1.3	17.1 ±0.9	45.7 ±3.5	57.3 ±2.1	43.2 ±2.5	58.1 ±1.4
PLeaS-Weight	1.0	Imagenet	17.8 ±1.9	55.0 ±1.2	41.0 ±2.9	40.8 ±1.0	40.2 ±1.8	17.9 ±1.2	38.7 ±1.4	16.7 ±0.7	43.7 ±3.7	56.4 ±1.7	40.7 ±2.7	56.1 ±0.7
PLeaS-Weight	1.55	Original	28.5 ±0.9	72.9 ±0.4	61.3 ±0.0	58.2 ±0.0	60.0 ±0.4	29.1 ±0.2	56.7 ±0.1	28.1 ±0.2	63.2 ±0.6	72.1 ±0.7	60.4 ±0.8	73.0 ±0.6
PLeaS-Weight	1.55	Imagenet	29.1 ±1.1	72.7 ±0.5	60.4 ±1.0	57.4 ±0.6	60.4 ±0.2	29.4 ±0.4	56.9 ±0.3	28.1 ±0.4	62.9 ±0.7	72.2 ±0.8	60.6 ±0.9	72.9 ±0.6
PLeaS-Weight	1.8	Original	29.7 ±1.1	73.8 ±0.4	62.7 ±0.2	59.4 ±0.3	61.2 ±0.2	30.1 ±0.2	58.2 ±0.4	29.1 ±0.3	64.1 ±0.5	73.2 ±0.5	62.3 ±0.9	73.8 ±0.7
PLeaS-Weight	1.8	Imagenet	29.7 ±1.0	74.0 ±0.5	62.0 ±0.8	58.9 ±0.5	61.5 ±0.1	30.3 ±0.1	58.7 ±0.1	29.1 ±0.1	63.6 ±0.7	73.7 ±0.5	61.7 ±0.6	74.4 ±0.6
Permutation-Weight	1.2	Original	24.8 ±1.1	65.0 ±0.7	55.0 ±0.0	48.4 ±0.0	51.8 ±0.8	24.0 ±0.3	48.5 ±0.9	23.3 ±0.3	56.3 ±0.3	63.7 ±0.2	52.3 ±0.5	65.5 ±1.1
Permutation-Weight	1.0	Original	10.5 ±0.7	34.6 ±2.7	28.3 ±0.1	22.9 ±0.1	21.8 ±1.2	9.7 ±0.7	21.1 ±1.9	9.4 ±0.1	29.1 ±1.3	38.1 ±2.9	26.5 ±1.5	38.1 ±1.2
Permutation-Weight	1.55	Original	27.5 ±0.9	69.7 ±0.7	59.9 ±0.2	54.9 ±0.1	57.7 ±0.8	27.7 ±0.5	54.1 ±0.7	26.3 ±0.2	60.6 ±0.8	69.4 ±0.8	57.4 ±0.9	70.4 ±0.9
Permutation-Weight	1.8	Original	28.9 ±0.8	71.7 ±0.5	61.6 ±0.1	57.3 ±0.1	60.5 ±0.7	29.0 ±0.3	56.8 ±0.7	28.1 ±0.6	63.1 ±0.9	71.1 ±0.7	60.8 ±0.8	72.2 ±1.0
ZipIt!	1.2	Original	60.0 ±1.2	21.9 ±1.1	52.4 ±1.4	44.8 ±0.7	49.8 ±0.3	21.4 ±0.6	45.7 ±0.8	20.6 ±0.2	54.1 ±2.2	59.6 ±1.3	50.3 ±1.3	63.6 ±0.7
ZipIt!	1.0	Original	35.8 ±1.3	13.2 ±0.6	29.3 ±2.6	26.1 ±0.5	26.1 ±1.2	12.4 ±0.8	24.4 ±1.2	10.9 ±0.7	33.4 ±2.8	37.0 ±0.5	31.0 ±1.8	39.4 ±1.5
ZipIt!	1.55	Original	66.3 ±0.7	26.6 ±1.0	61.2 ±1.0	53.6 ±0.2	58.6 ±0.2	25.9 ±0.3	52.6 ±0.4	25.5 ±0.3	62.9 ±1.1	67.5 ±0.3	58.3 ±1.1	69.9 ±0.2
ZipIt!	1.8	Original	68.4 ±0.1	28.7 ±1.2	63.1 ±0.7	56.6 ±0.4	60.8 ±0.3	27.9 ±0.3	55.1 ±0.7	27.3 ±0.2	65.0 ±0.6	69.8 ±0.2	61.0 ±0.9	72.5 ±0.6

C.1. Using synthetic data for merging

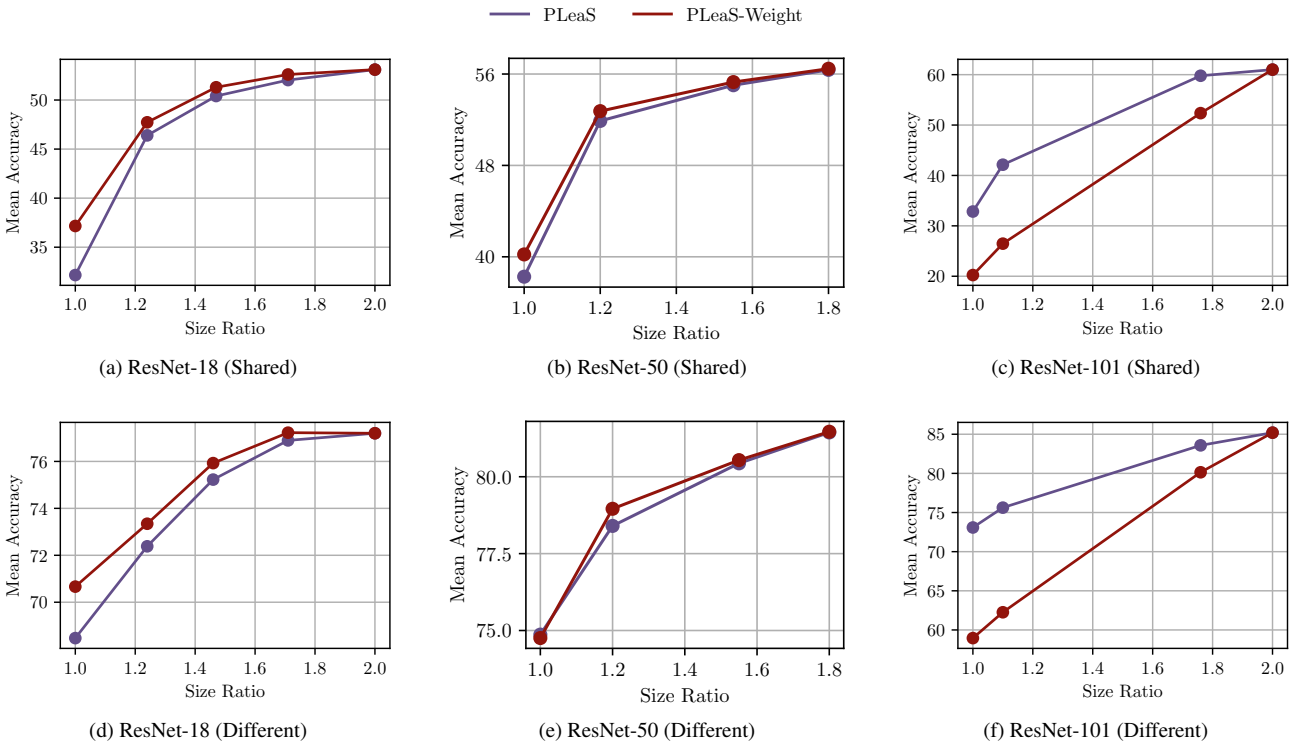
We use synthetic images (e.g. procedurally generated data which can mimic the broad structure of real images) for merging. In Fig. 8, we present the results of using such images with `PLeaS-free`. This leads to a slight drop over using images from ImageNet, but the performance is close, and the gap closes as model size increases.

Table 6. **Detailed Results on Different Label Spaces** We report the results for ResNet-50 here

Method	Budget	Data	na-ox		na-st		cu-na		cu-st		cu-ox		st-ox	
PLeaS-Act	1.2	Imagenet	69.2 ± 0.2	88.6 ± 0.2	67.0 ± 0.3	68.6 ± 0.5	78.5 ± 0.6	69.4 ± 0.3	71.9 ± 0.5	71.3 ± 0.7	75.0 ± 1.1	89.4 ± 0.7	77.9 ± 0.4	90.5 ± 0.4
PLeaS-Act	1.2	Original	71.6 ± 0.2	89.5 ± 0.2	70.0 ± 0.5	71.0 ± 1.0	80.1 ± 0.5	72.0 ± 0.3	74.0 ± 0.5	75.2 ± 0.6	76.4 ± 0.8	90.2 ± 0.5	79.9 ± 0.5	91.9 ± 0.6
PLeaS-Act	1.0	Imagenet	66.2 ± 0.7	80.3 ± 1.1	63.6 ± 0.8	56.1 ± 0.5	76.6 ± 0.4	65.4 ± 0.6	67.4 ± 1.2	62.5 ± 1.4	70.9 ± 0.6	84.1 ± 1.1	73.1 ± 0.8	87.4 ± 0.3
PLeaS-Act	1.0	Original	70.2 ± 0.2	81.1 ± 0.2	68.2 ± 0.5	61.2 ± 0.6	79.7 ± 0.4	69.8 ± 0.4	71.5 ± 0.7	69.3 ± 1.2	74.4 ± 0.6	87.2 ± 0.5	78.2 ± 0.4	90.6 ± 0.4
PLeaS-Act	1.8	Imagenet	73.4 ± 0.2	91.6 ± 0.2	71.3 ± 0.4	75.7 ± 0.2	80.2 ± 0.3	72.3 ± 0.4	75.9 ± 0.4	77.4 ± 0.3	77.1 ± 1.1	91.8 ± 0.5	81.3 ± 0.3	92.3 ± 0.4
PLeaS-Act	1.8	Original	75.3 ± 0.3	91.3 ± 0.2	74.0 ± 0.6	76.9 ± 0.3	81.5 ± 0.4	75.0 ± 0.3	77.0 ± 0.4	79.5 ± 0.4	78.8 ± 0.5	92.4 ± 0.5	82.9 ± 0.5	92.8 ± 0.4
PLeaS-Act	1.55	Imagenet	71.5 ± 0.3	90.7 ± 1.1	69.8 ± 0.7	73.7 ± 1.1	79.5 ± 0.5	71.1 ± 0.6	74.9 ± 0.3	75.4 ± 0.4	76.3 ± 0.8	91.2 ± 0.5	80.3 ± 0.5	91.7 ± 0.3
PLeaS-Act	1.55	Original	73.2 ± 0.2	90.4 ± 0.2	72.5 ± 0.4	75.5 ± 0.7	81.2 ± 0.6	73.8 ± 0.3	75.9 ± 0.4	78.3 ± 0.6	78.0 ± 0.3	92.1 ± 0.5	81.9 ± 0.4	92.8 ± 0.5
RegMean	1.0	Original	44.1 ± 0.2	56.0 ± 0.2	51.8 ± 14.8	39.0 ± 16.4	54.1 ± 10.1	42.5 ± 13.4	45.5 ± 21.2	37.7 ± 22.3	55.1 ± 20.8	61.7 ± 21.4	37.4 ± 15.5	56.4 ± 16.2
PLeaS-Weight	1.2	Imagenet	71.0 ± 0.2	88.1 ± 0.2	68.5 ± 1.0	70.1 ± 0.5	79.2 ± 0.6	70.4 ± 0.3	73.7 ± 1.2	73.4 ± 0.7	73.5 ± 1.8	89.1 ± 0.5	77.5 ± 0.6	89.9 ± 0.3
PLeaS-Weight	1.2	Original	72.5 ± 0.2	88.5 ± 0.2	70.9 ± 0.5	72.5 ± 0.5	80.5 ± 0.4	72.5 ± 0.3	75.2 ± 0.8	76.4 ± 0.4	77.2 ± 0.9	90.7 ± 0.6	79.9 ± 0.3	91.5 ± 0.9
PLeaS-Weight	1.0	Imagenet	68.6 ± 0.5	79.4 ± 1.0	66.2 ± 0.8	58.9 ± 0.6	76.5 ± 0.6	65.8 ± 1.0	69.1 ± 1.3	63.9 ± 0.7	71.7 ± 1.1	83.7 ± 0.7	70.9 ± 0.4	84.6 ± 0.7
PLeaS-Weight	1.0	Original	70.6 ± 0.3	79.2 ± 0.4	69.6 ± 0.6	62.2 ± 0.4	79.4 ± 0.6	69.8 ± 0.4	72.4 ± 0.7	69.1 ± 0.5	75.2 ± 0.8	87.0 ± 0.4	76.1 ± 0.4	89.4 ± 0.5
PLeaS-Weight	1.8	Imagenet	73.4 ± 0.6	91.7 ± 0.5	71.2 ± 0.7	75.8 ± 0.5	80.0 ± 0.6	72.2 ± 0.2	75.5 ± 0.8	77.2 ± 0.5	76.1 ± 0.7	92.0 ± 0.7	80.9 ± 0.5	91.9 ± 0.6
PLeaS-Weight	1.8	Original	74.8 ± 0.2	91.6 ± 0.2	73.9 ± 0.3	77.5 ± 0.6	81.7 ± 0.4	74.8 ± 0.4	76.5 ± 0.5	80.2 ± 0.3	78.8 ± 0.4	92.3 ± 0.5	82.8 ± 0.5	92.9 ± 0.5
PLeaS-Weight	1.55	Imagenet	72.5 ± 0.2	89.9 ± 0.3	70.1 ± 1.0	74.1 ± 0.8	79.4 ± 0.6	71.6 ± 0.3	74.9 ± 0.6	75.8 ± 0.7	75.8 ± 1.1	91.0 ± 0.7	79.8 ± 0.7	91.4 ± 0.5
PLeaS-Weight	1.55	Original	73.9 ± 0.4	90.0 ± 0.2	72.8 ± 0.5	76.0 ± 0.6	81.4 ± 0.3	74.0 ± 0.3	76.1 ± 0.5	78.7 ± 0.5	77.9 ± 0.7	91.9 ± 0.4	81.7 ± 0.5	92.3 ± 0.7
SimpleAvg	1.0	Original	5.2 ± 0.9	21.3 ± 1.7	5.0 ± 0.7	11.2 ± 0.9	8.7 ± 0.5	4.1 ± 0.4	6.8 ± 1.0	9.0 ± 0.4	6.9 ± 0.8	18.0 ± 1.2	8.7 ± 0.4	17.9 ± 0.6
Permutation-Weight	1.2	Original	67.9 ± 0.7	87.5 ± 0.7	66.8 ± 0.3	69.6 ± 0.4	78.2 ± 0.5	68.3 ± 0.3	72.0 ± 0.8	72.3 ± 0.5	73.9 ± 0.9	88.9 ± 0.3	77.2 ± 0.7	89.7 ± 0.7
Permutation-Weight	1.0	Original	60.6 ± 0.4	78.1 ± 0.5	57.0 ± 0.7	59.1 ± 0.5	72.3 ± 0.6	59.9 ± 0.3	59.8 ± 0.8	60.4 ± 0.8	66.5 ± 0.6	82.1 ± 0.8	67.0 ± 0.7	82.5 ± 0.5
Permutation-Weight	1.8	Original	74.0 ± 0.6	91.0 ± 0.7	73.0 ± 0.4	76.3 ± 0.5	81.2 ± 0.3	73.8 ± 0.2	76.0 ± 0.5	78.5 ± 0.5	77.7 ± 0.6	92.1 ± 0.6	82.2 ± 0.4	92.5 ± 1.0
Permutation-Weight	1.55	Original	71.4 ± 0.6	90.4 ± 0.5	71.1 ± 0.4	73.9 ± 0.5	80.1 ± 0.5	71.9 ± 0.3	75.0 ± 0.7	76.2 ± 0.4	76.5 ± 0.9	91.0 ± 0.5	80.4 ± 0.3	91.9 ± 0.5
Permutation-Act	1.2	Original	67.1 ± 0.6	85.7 ± 0.6	64.6 ± 0.7	65.3 ± 0.9	77.4 ± 0.4	67.0 ± 0.5	70.2 ± 0.9	69.4 ± 0.8	73.0 ± 0.7	87.7 ± 0.4	76.7 ± 0.5	89.8 ± 0.4
Permutation-Act	1.0	Original	61.2 ± 0.7	75.9 ± 1.1	55.8 ± 0.7	54.8 ± 0.9	73.9 ± 0.4	61.2 ± 0.7	60.9 ± 0.8	59.4 ± 1.3	66.6 ± 2.7	80.0 ± 2.2	71.6 ± 0.5	86.4 ± 0.6
Permutation-Act	1.8	Original	74.1 ± 0.4	91.2 ± 0.5	73.0 ± 0.5	76.1 ± 0.8	81.0 ± 0.3	73.9 ± 0.4	76.4 ± 0.4	78.6 ± 0.5	78.5 ± 0.4	92.1 ± 0.6	82.4 ± 0.4	93.0 ± 0.4
Permutation-Act	1.55	Original	71.6 ± 0.4	89.6 ± 0.5	70.4 ± 0.3	72.4 ± 0.9	79.7 ± 0.4	71.5 ± 0.2	74.4 ± 0.5	75.5 ± 0.5	76.3 ± 0.5	90.9 ± 0.4	80.9 ± 0.5	92.3 ± 0.6
ZipIt!	1.2	Original	62.6 ± 0.4	85.0 ± 0.6	62.0 ± 0.9	63.1 ± 0.8	74.9 ± 0.7	63.6 ± 0.7	70.3 ± 0.7	66.5 ± 0.4	71.1 ± 0.9	85.1 ± 0.6	72.2 ± 0.5	88.2 ± 0.3
ZipIt!	1.0	Original	56.7 ± 0.6	76.7 ± 1.3	55.1 ± 0.6	52.3 ± 1.1	72.9 ± 0.4	57.8 ± 0.4	64.6 ± 0.6	58.3 ± 0.9	67.1 ± 0.6	80.8 ± 0.3	67.6 ± 1.3	85.5 ± 0.9
ZipIt!	1.8	Original	70.0 ± 0.4	91.0 ± 0.4	69.4 ± 0.8	75.0 ± 0.7	78.9 ± 0.6	70.4 ± 0.5	75.7 ± 0.4	76.0 ± 0.5	75.9 ± 0.7	91.0 ± 0.5	78.7 ± 0.6	92.2 ± 0.6
ZipIt!	1.55	Original	67.0 ± 0.4	90.0 ± 0.7	66.4 ± 0.3	72.9 ± 0.5	77.7 ± 0.4	67.3 ± 0.8	73.7 ± 0.6	73.2 ± 0.5	74.0 ± 0.5	89.9 ± 0.6	76.6 ± 0.7	91.4 ± 0.5

Table 7. **Evaluating merged models with task specific heads**: For each pair of datasets, we merge models and compute the final accuracy on the pair. To compute the final accuracy on a dataset, we average the accuracies across the pairs that it is a part of. These accuracies are computed by using the task specific head for each pair of models.

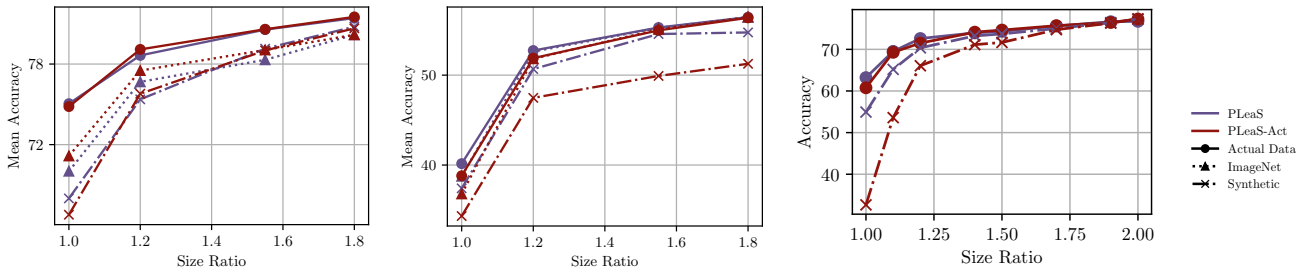
Method	FLOPs	CUB	NABird	Dogs	Pets
ZipIt[28]	1.0	47.8	46.3	35.7	68.6
Git Re-Basin[1]	1.0	35.3	32.7	26.1	50.4
`PLeaS`	1.0	63.1	63.2	45.3	68.4
ZipIt	1.2	68.7	64.1	60.9	84.8
Permutation	1.2	67.8	64.7	60.9	83.7
`PLeaS`	1.2	71.6	69.6	70.2	86.9
ZipIt	1.55	72.9	69.2	71.9	89.7
Permutation	1.55	71.8	69.8	71.8	88.2
`PLeaS`	1.55	74.5	72.4	74.8	89.8
ZipIt	1.8	75.1	72.7	74.0	91.6
Permutation	1.8	74.1	72.6	75.8	89.9
`PLeaS`	1.8	77.0	74.3	75.5	89.6
Ensemble	2.0	77.0	76.0	80.8	91.6



(a) ResNet-18 (Shared) (b) ResNet-50 (Shared) (c) ResNet-101 (Shared)

(d) ResNet-18 (Different) (e) ResNet-50 (Different) (f) ResNet-101 (Different)

Figure 7. **Comparing `PLeaS` and `PLeaS-Weight`** : We find that `PLeaS` is better for larger models while `PLeaS-Weight` is better for smaller models.



(a) Different Label Spaces (b) Same Label Spaces (c) Imagenet

Figure 8. `PLeaS-free` with procedurally generated synthetic data. We find that using synthetic data is similar to using ImageNet