Learning on Model Weights using Tree Experts

Supplementary Material

Contents

1. Proofs

1.1. Proposition 1

Proposition 1. *Assume $\mathcal{E}_1, \dots, \mathcal{E}_{r_U}$ are all linear operations and a sufficient number of probes. The dense expert ($y_k = \sum_{ij} W_{ijk} X_{ij}$, Eq. 1 in the manuscript) and linear probing network ($\boldsymbol{y} = \mathcal{T}(\boldsymbol{e})$, Eq. 2 in the manuscript) have identical expressivity.*

Proof. We will prove both that the dense expert entails linear probing (1), and that probing entails linear experts (2).

Direction (1) is trivial, as linear probing is a composition of linear operations, it follows that the operation is a linear operation from $\mathbb{R}^{d_W \times d_H} \rightarrow \mathbb{R}^{d_Y}$. As the dense expert, parameterized as $W \in \mathbb{R}^{d_W \times d_H \times d_Y}$, can express all linear operations in $\mathbb{R}^{d_W \times d_H} \rightarrow \mathbb{R}^{d_Y}$, it clearly entails linear probing.

Direction (2) requires us to prove that we can find a set of matrices $U, \mathcal{E}[1], \mathcal{E}[2], \cdots, \mathcal{E}[r_U], T$ such that $\mathbf{y} = T \sum_l \mathcal{E}[l] X \mathbf{u}_l = \sum_{ij} W_{ijk} X_{ij}$ for every $X \in \mathbb{R}^{d_W \times d_H}$ and any $W \in \mathbb{R}^{d_W \times d_H \times d_Y}$. We show a proof by construction. Let $T = I$ (the identity matrix), $U = I$ and $\mathcal{E}[l]_{ik} = W_{ilk}$. We have:

$$y_k = (T \sum_l \mathcal{E}[l] X \mathbf{u}_l)_k = \sum_{ijl} W_{ilk} X_{ij} \delta_{jl} \tag{1}$$

Where δ_{jl} is 1 in the diagonal and 0 otherwise, the T is the identity matrix and cancels out. Summing over l, we obtain:

$$y_k = \sum_{ij} W_{ijk} X_{ij} \tag{2}$$

This proves that linear probing can express any dense expert. □

1.2. Proposition 2

Proposition 2. *The linear ProbeX ($\boldsymbol{y} = T \sum_l M_l V^T X^T \boldsymbol{u}_l$, Eq. 4 in the manuscript) has identical expressivity as using the dense predictor ($y_k = \sum_{ij} W_{ijk} X_{ij}$, Eq. 1 in the manuscript), when the weight tensor W obeys the Tucker decomposition:*

$$W_{\mathit{Tucker}} = \sum_{nml} M_{nml} \cdot \boldsymbol{t}_n \otimes \boldsymbol{v}_m \otimes \boldsymbol{u}_l$$

Proof. The Tucker decomposition expresses a 3D tensor $W \in \mathbb{R}^{d_W \times d_H \times d_Y}$ by the product of a smaller tensor $M \in \mathbb{R}^{r_T \times r_V \times r_U}$ and three matrices $U \in \mathbb{R}^{d_H \times r_U}, V \in \mathbb{R}^{d_W \times r_V}, T \in \mathbb{R}^{d_Y \times r_T}$ as follows:

$$W = \sum_{nml} M_{nml} \cdot \mathbf{t}_n \otimes \mathbf{v}_m \otimes \mathbf{u}_l \tag{3}$$

Where $\otimes$ is the tensor product, and $\mathbf{u}_q, \mathbf{v}_q, \mathbf{t}_q$ are the q^{th} column vectors of matrices U, V, T respectively.

The expression for the Tucker decomposition in index notation is:

$$W_{ijk} = \sum_{nml} T_{kn} M_{nml} V_{im} U_{jl} \tag{4}$$

By linearity, we can reorder the sums as:

$$W_{ijk} = \sum_n T_{kn} \sum_{ml} M_{nml} V_{im} U_{jl} \tag{5}$$

We can equivalently split tensor M into r matrices $M[1], M[2], \cdots, M[r]$, so that:

$$W_{ijk} = \sum_n T_{kn} \sum_{ml} M[l]_{nm} V_{im} U_{jl} \tag{6}$$

Multiplying tensor W by input matrix $X \in \mathbb{R}^{d_W \times d_H}$, the result is:

$$\tilde{y}_k = \sum_{ij} X_{ij} W_{ijk} = \sum_{ij} X_{ij} \sum_n T_{kn} \sum_{ml} M[l]_{nm} V_{im} U_{jl} \tag{7}$$

By linearity, we can reorder the sums:

$$\tilde{y}_k = \sum_n T_{kn} \sum_{ml} M[l]_{nm} \sum_{ij} V_{im} X_{ij} U_{jl} \tag{8}$$

Rewriting U using its column vectors this becomes:

$$\tilde{y}_k = \sum_n T_{kn} \sum_{ml} M[l]_{nm} \sum_i V_{im} (X \mathbf{u}_l)_i \tag{9}$$

Rewriting the sum over i as a matrix multiplication:

$$\tilde{y}_k = \sum_n T_{kn} \sum_{ml} M[l]_{nm} (V^T X \mathbf{u}_l)_m \tag{10}$$

Rewriting the sum over m as a matrix multiplication:

$$\tilde{y}_k = \sum_n T_{kn} \sum_l (M[l] V^T X \mathbf{u}_l)_n \tag{11}$$

Rewriting the sum over n as a matrix multiplication, we finally obtain:

$$\tilde{y} = T \sum_l M[l] V^T X \mathbf{u}_l \tag{12}$$

□

2. Additional discussion

2.1. Mechanistic vs. functional weight learning

Herrmann et al. [1] distinguished between two approaches to weight-space learning. The mechanistic approach treats the weights as input data and learns directly from them, while the functionalist approach (e.g., probing) interacts only with a model's inputs and outputs. Although the functionalist approach bypasses weight-space-related nuisance factors such as permutations or Model Trees, it treats the entire model as a black box, limiting its scope. ProbeX can be seen as a blend of both approaches, enabling us to operate at the weight level while engaging with the function defined by the weight matrix. This approach may facilitate the study of different model layers' functionalities. For instance, in the case of the MAE and Sup. ViT Model Trees, which share the same architecture, the most effective layer for our task differed between the two. This suggests that, despite having the same architecture, the two models utilize their layers for different functions.

Similarly, for our aligned representations, the best-performing layer is a "query" layer in the U-Net's encoder. However, examining the top 10 best-performing layers by in-distribution accuracy reveals that the specific "query" layer chosen is critical, resulting in a 6.6% difference in zero-shot accuracy between the best and second-best layers. Additionally, while two of the top 10 layers are "out" layers and perform well on in-distribution samples, their performance drops sharply on the zero-shot task, causing a rank decrease of five places. Table 1 lists the top 10 layers by in-distribution validation accuracy alongside their zero-shot task results.

2.2. Self-supervised learning vs. aligning representations

Here, we align model weights with existing representations. While weight-space self-supervised (SSL) learning [4, 5, 7] do not depend on external representations, they typically require carefully crafted augmentations and priors. Designing such augmentations for model weights is non-trivial as key nuisance factors are still being identified. We hope our work accelerates research on new weight-space SSL methods.

2.3. Other weight-space learning tasks

In this paper we focused on predicting the categories in a model's training dataset. However, many more weight-space learning tasks exists. As demonstrated in Prop. 1, our probing formulation is equivalent to the weight formulation, suggesting that ProbeX can potentially perform any task achievable by other mechanistic approaches. Since our focus has been on predicting the model's training dataset categories and their connection to text-based representations, extending ProbeX to these additional tasks is left for future work.

Table 1. ***Best performing layers of*** SD_{200}***:*** Rankings differ significantly between in-distribution and zero-shot tasks. Numbers in $(\cdot)$ indicate the amount the layer moved up or down in rank.

Layer Name	**In Dist. ↑ Acc.**	**Zero-shot ↑ Acc.**
down.2.attentions.1.attn2.q	0.974	0.898 $(0^{\uparrow})$
up.1.attentions.1.attn2.q	0.958	0.832 $(2_{\downarrow})$
up.1.attentions.0.attn2.q	0.952	0.850 $(1^{\uparrow})$
up.1.attentions.1.attn2.out.0	0.946	0.665 $(5_{\downarrow})$
up.1.attentions.2.attn2.q	0.944	0.822 $(1_{\downarrow})$
up.1.attentions.2.attn2.out.0	0.920	0.641 $(5_{\downarrow})$
down.2.attentions.0.attn2.q	0.916	0.830 $(2^{\uparrow})$
up.2.attentions.2.attn2.k	0.872	0.735 $(0^{\uparrow})$
mid.attentions.0.attn2.q	0.866	0.848 $(6^{\uparrow})$
up.2.attentions.1.attn2.k	0.830	0.760 $(3^{\uparrow})$

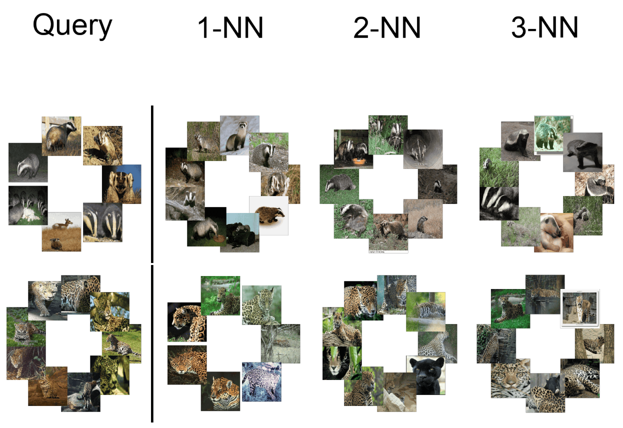

Figure 1. ***Additional model retrieval results:*** Retrieval is performed using model weights, to visualize each model we use the set of all the images that were used to fine-tune the model.

3. Mixture-of-Experts router

As described in the manuscript, when handling Model Graphs with multiple Model Trees, we use a mixture-of-experts approach. This involves first learning a routing function and then training a separate ProbeX model for each Model Tree.

To implement the routing function, we perform hierarchical clustering on the ℓ_2 pairwise distances between models in the Model Graph. By calculating distances for a single model layer, this stage is significantly accelerated, enabling us to cluster Model Graphs with up to 10,000 models in under 5 minutes. Once clustering is complete, the routing function assigns each model to the nearest cluster based on ℓ_2 distance. The number of Model Trees is determined using the dendrograms produced by hierarchical clustering. We

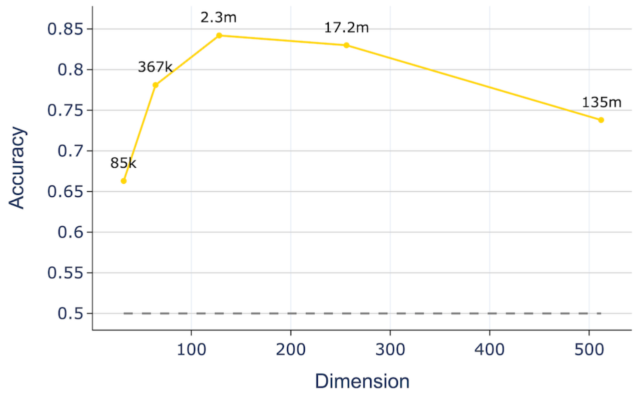

Figure 2. $r_{(\cdot)}$ ***dimension ablation:*** We ablate the effect of changing the dimension of all r_U, r_V and r_T jointly. We can see that beyond some point the performance drops.

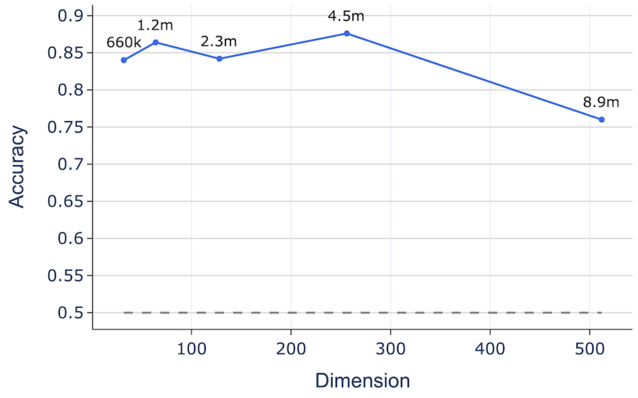

Figure 3. ***Number of probes*** (r_U) ***ablation:*** We fix r_V and r_T to 128 and change the number of probes (r_U). We can see that too many probes decreases the performance.

use the `scipy` [8] implementation with default hyperparameters.

In practice, this simple routing function achieved *perfect accuracy* every time.

4. Additional model retrieval results

In Sec. 6.2.2 of the manuscript, we presented results for the task of model retrieval. Here, we provide results for *all* held-out models in SD_{200}. These results are not cherry-picked, and each model is visualized using the full set of images that were used for its fine-tuning. In Fig. 1, we display two additional results, in Figs. 18 to 20 present the remaining results.

5. Additional ablations

We provide additional ablations and expand on the ones from the manuscript.

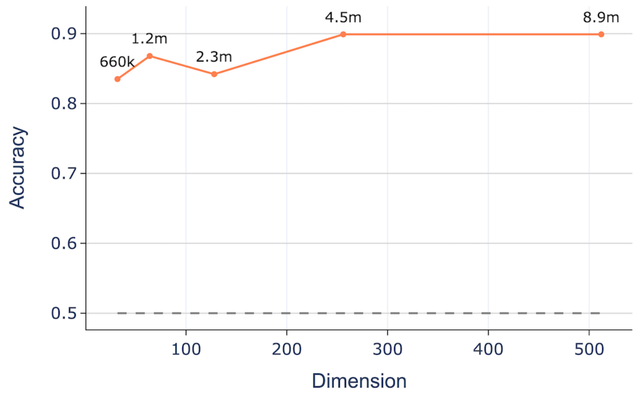

Figure 4. ***Probe dimension*** (r_V) ***ablation:*** We fix r_U and r_T to 128 and change the probe dimension (r_V). We can see that even a small probe dimension already results in good performance and that increasing it does not help beyond some point.

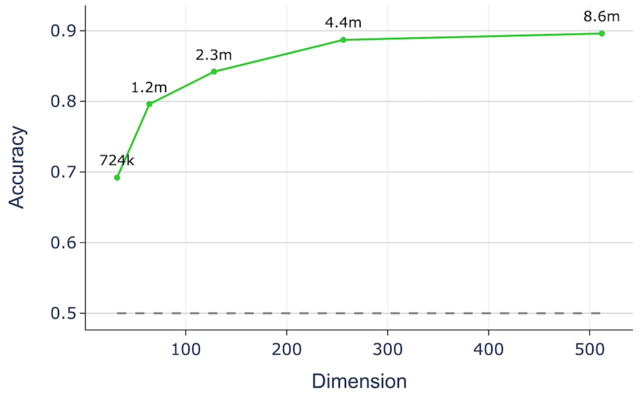

Figure 5. ***Encoding dimension*** (r_T) ***ablation:*** We fix r_U and r_V to 128 and change the encoding dimension (r_T). We can see that the size of the model encoding plays an important role in the performance of our method.

5.1. r_U, r_V, r_T size ablation

We ablate the effect of the dimensions r_U, r_V, and r_T using the Sup. ViT Model Tree. We begin by examining the impact of jointly increasing all dimensions. As shown in Fig. 2, increasing the size improves performance up to a point (128), after which performance begins to decline. When jointly adjusting all dimensions, the larger model size appears to be responsible for this drop. However, when we vary each dimension independently while fixing the other two at 128, we observe a different pattern.

Starting with the number of probes (r_U), as shown in Fig. 3, increasing the number of probes has minimal effect on performance until a threshold (256), beyond which performance drops significantly. This decline may explain the performance drop in Fig. 2, even without an extreme increase in the parameter size.

In Fig. 4, we observe that changing the dimension of the

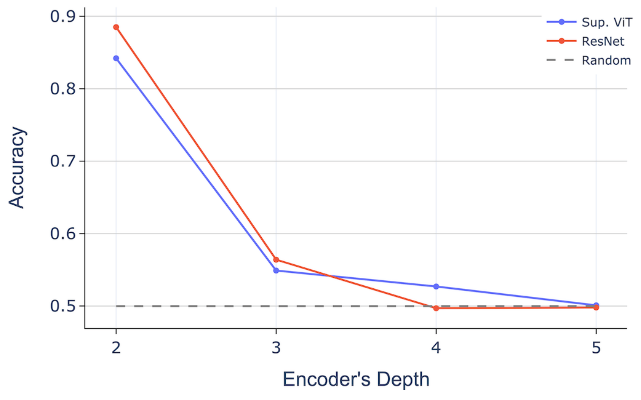

Figure 6. ***Deeper ProbeX encoder ablation - discriminative***

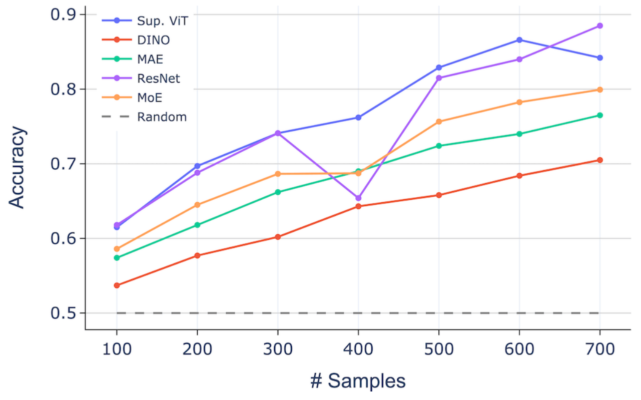

Figure 7. ***Dataset size ablation***

probes (r_V) has little impact on performance. Lastly, Fig. 5 shows that increasing the dimension of the encoding (r_T) has a dramatic effect, significantly improving performance.

5.2. Deeper ProbeX encoders

Here, we evaluate whether deeper, non-linear ProbeX encoders outperform our single hidden-layer encoder. Specifically, we stack additional dense layers followed by non-linear activations and assess their performance. This experiment is conducted for each architecture in the Model-J dataset (i.e., ViT, ResNet, and Stable Diffusion). As shown in Figs. 6 and 8, deeper encoders tend to overfit, leading to reduced performance.

5.3. Dataset size

We examined the effect of dataset size on accuracy. Indeed, in Fig. 7 we see that as discussed in the motivation, models that belong to the same Model Tree have positive transfer.

5.4. Text encoder

We ablate whether our success in aligning model weights with CLIP rerepresentations is due to the fact Stable Diffusion was originally trained with CLIP. We perform the zero-shot experiment using SD_{200} and the following text encoders. The results in Tab. 2, suggest that while CLIP performs best, our approach remains effective across different text encoders. This shows the robustness of ProbeX to the choice of text backbone.

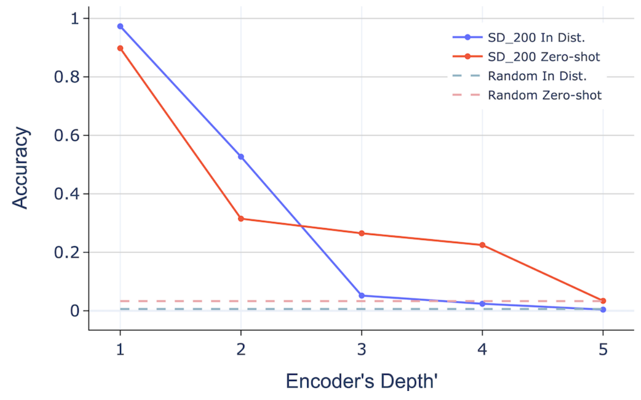

Figure 8. ***Deeper ProbeX encoder ablation -*** SD_{200}

Table 2. ***Text-Encoder Ablation on*** SD_{200}***:*** We ablate the sensitivity of the representation alignment to different text encoders using the zero-shot experiment. While CLIP performs best, as expected due to Stable Diffusion's training, our approach remains effective across various text encoders, demonstrating robustness to the choice of text backbone

Encoder	**Acc. ↑**
BLIP2	0.564
OPENCLIP	0.860
CLIP	**0.898**

6. Hugging Face Model Graph analysis

Our presented statistics regarding Model Trees are based on the "hub-stats" Hugging Face dataset[1]. This dataset, maintained by Hugging Face, is automatically updated daily with statistics about Hugging Face models, datasets, and more. We used a version from late September 2024, when there were "only" about 800,000 models hosted on Hugging Face. We utilized the `base_model` property from model cards and aggregated based on it. However, since not all models on Hugging Face use this property, these statistics are not 100% accurate and may contain some bias. Additionally, Fig. 1 in the manuscript (Growth in Hugging Face models) also uses the "hub-stats" dataset and is based on the graphs shown at https://huggingface.co/spaces/cfahlgren1/hub-stats.

[1] https://huggingface.co/datasets/cfahlgren1/hub-stats

Table 3. ***Hyperparameters overview - Discriminative split***

Name	Value
`lr`	$[5e-4, 3e-4, 1e-4, 9e-5, 7e-5, 5e-5, 3e-5]$
`lr_scheduler`	linear, cosine, cosine-with-restarts, constant, constant-with-warmup
`weight_decay`	$[5e-2, 3e-2, 1e-2, 9e-3, 7e-3, 5e-3]$
`epochs`	$[2-9]$
`random_crop`	T,F
`random_flip`	T,F
`batch_size`	64
`fine-tuning type`	Full Fine-tuning

7. Dataset details

Note: The dataset will be made publicly available as an Hugging Face dataset.
Existing weight-space learning datasets and model zoos [2, 6] primarily consist of models that are randomly initialized. This means that each model in such datasets serves as the root of a distinct Model Tree containing only that model. As demonstrated in the motivation section of the manuscript, learning from such Model Graphs is significantly more challenging, highlighting the need for our approach of learning within Model Trees. Furthermore, existing datasets primarily consist of small models, typically with only thousands of parameters per model. *As such, we cannot utilize the existing and established weight-space learning datasets.*

To address this, we introduce the Model Jungle dataset (Model-J), which simulates the structure of public model repositories. Each of our fine-tuned models is trained using a set of hyperparameters sampled uniformly at random. Discriminative models share the same set of possible hyperparameters, summarized in Tab. 3. Generative models, in contrast, use a different set of hyperparameters detailed in Tab. 4. Notably, in the generative split, our Model Trees have multiple levels of hierarchy, as models were fine-tuned from SD1.2 to SD1.5. This structure is designed to simulate public model repositories, where Model Trees often exhibit multiple levels of hierarchy. In Figs. 10 and 12 to 14 we provide a summary of the test accuracy the models in the discriminative split converged to. In Figs. 11 and 15 to 17 we plot these accuracies as a function of the model's learning rate.

For the discriminative split, we use the following models as the Model Tree roots taken from *Hugging Face*:

- `https://huggingface.co/google/vit-base-patch16-224`
- `https://huggingface.co/facebook/vit-mae-base`
- `https://huggingface.co/facebook/dino-vitb16`
- `https://huggingface.co/microsoft/resnet-101`

Table 4. ***Hyperparameters overview - generative Split***

Name	Value
`# images`	[5-10]
`lr`	$[5e-4, 3e-4, 1e-4, 9e-5, 7e-5, 5e-5, 3e-5]$
`prompt_template`	a photo of a, a picture of a, a photograph of a, an image of a, cropped photo of the, a rendering of a
`base_model`	[SD1.2, SD1.3, SD1.4, SD1.5]
`steps`	$[450, 500, 550, 600, 650, 700]$
`fine-tuning type`	LoRA
`random_crop`	T,F
`rank`	16
`batch_size`	1

Table 5. ***Model Jungle dataset summary.*** We train over 14,000 models, covering different architectures, tasks and model sizes. Each model uses randomly sampled hyper parameters

Name	FT Type	Task	Size	# Classes
DINO	Full FT	Att. classification	1000	50/100
MAE	Full FT	Att. classification	1000	50/100
Sup. ViT	Full FT	Att. classification	1000	50/100
ResNet	Full FT	Att. classification	1000	50/100
SD_{200}	LoRA	Fine-grained	5000	200
SD_{1k}	LoRA	Few shot	5000	1000

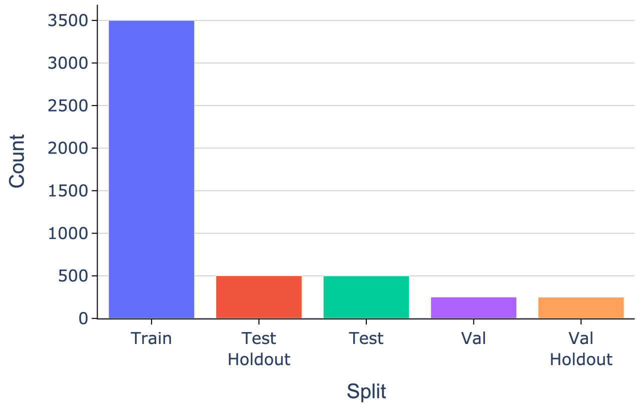

Figure 9. ***Distribution of splits in the generative split***

8. Implementation details

8.1. Experimental setup

We use the Model-J dataset presented in Sec. 5 of the manuscript, split into 70/10/20 for training, validation, and testing. Given the significant variation in results between layers, we train ProbeX for 500 epochs on each layer and select

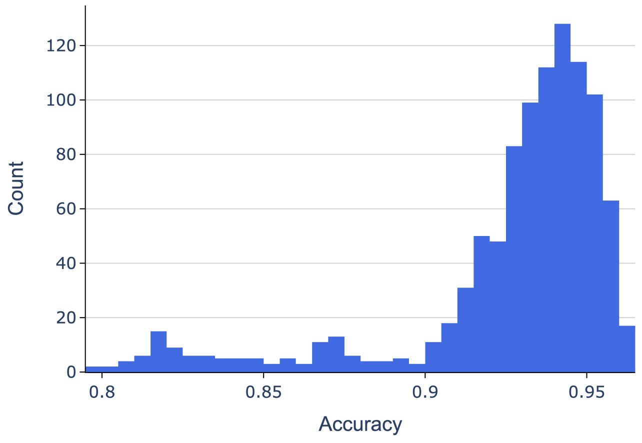

Figure 10. ***Sup. ViT - Test accuracy distribution***

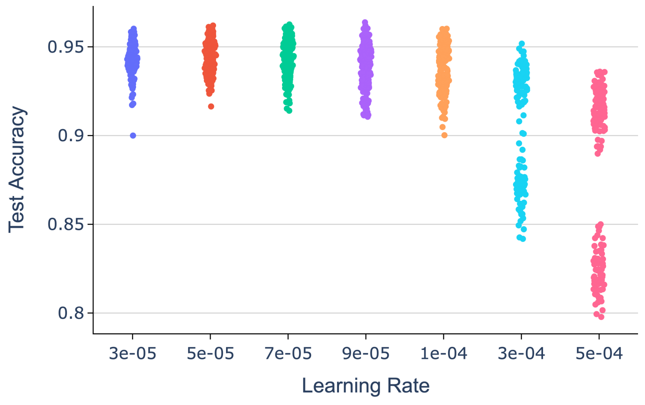

Figure 11. ***Sup. ViT - Effect of learning rate on test accuracy***

the best layer and epoch based on the validation set. We use the Adam optimizer with a weight decay of $1e{-}5$ and a learning rate of $1e{-}3$. The number of probesm probe dimensions, and encoder dimension are set to $r_U = r_V = r_T = 128$. We reshape higher-dimensional weight tensors (e.g., convolutional layers) into 2D matrices, with the first dimension being the output channels.

8.2. Baselines

8.2.1. Neural Graphs baseline

As mentioned in the manuscript, we attempted to use [3] as a baseline but were unable to scale the method to models in our dataset. Here, we provide additional details about this attempt. Neural Graphs [3] is a graph-based approach that treats each bias in the network as a node and each weight as an edge. These methods scale quadratically with the number of neurons, leading to computational challenges when applied to larger models. To address this, we adapted the Neural Graphs approach to the single-layer case, but even in this simplified scenario, it required a relatively low hidden dimension to run on a 24GB GPU. Since this baseline yielded near-random results in our experiments with discriminative models, we chose not to include its results in the tables.

8.2.2. StatNN

For the discriminative split, we used StatNN as a baseline by training XGBoost on the StatNN features. To compare StatNN in the case of text-aligned representations, we replaced XGBoost with an MLP, allowing the baseline to be trained with the same contrastive objective used for ProbeX. We developed two StatNN variants: i) $StatNN_{Linear}$: A single linear layer trained on top of the StatNN features. ii) $StatNN_{MLP}$: A deeper architecture designed to match the parameter count of our method.

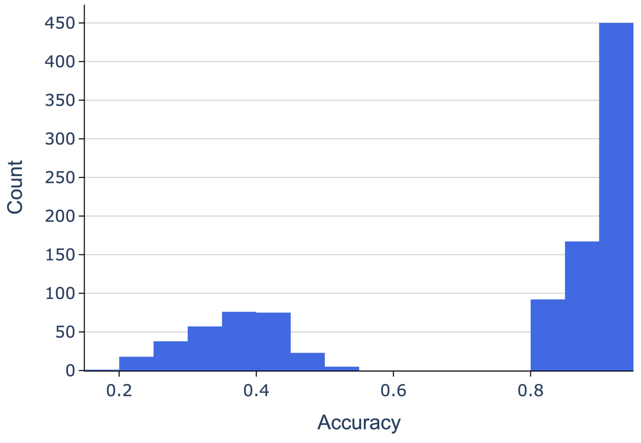

Figure 12. ***DINO - Test accuracy distribution***

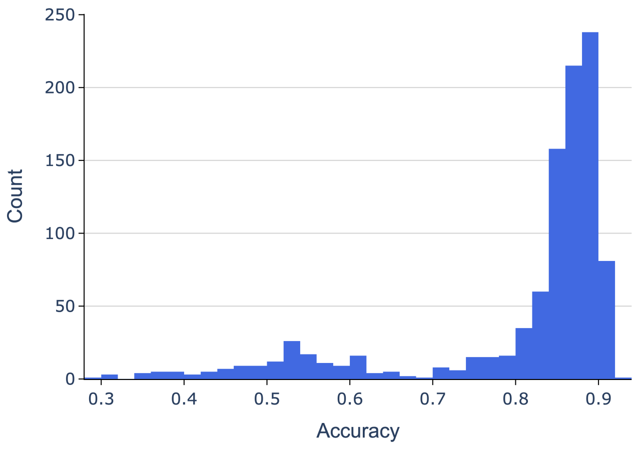

Figure 13. ***MAE - Test accuracy distribution***

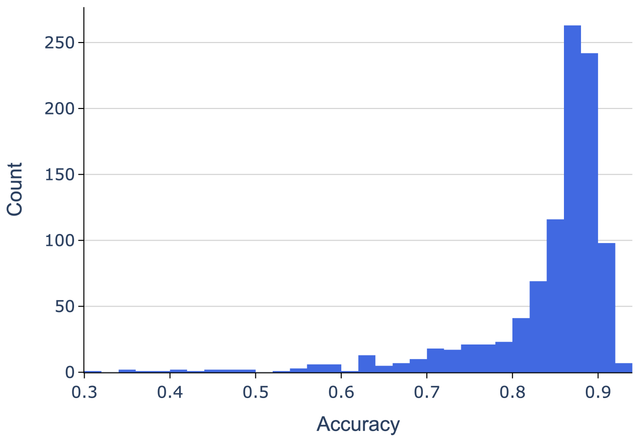

Figure 14. ***ResNet - Test accuracy distribution***

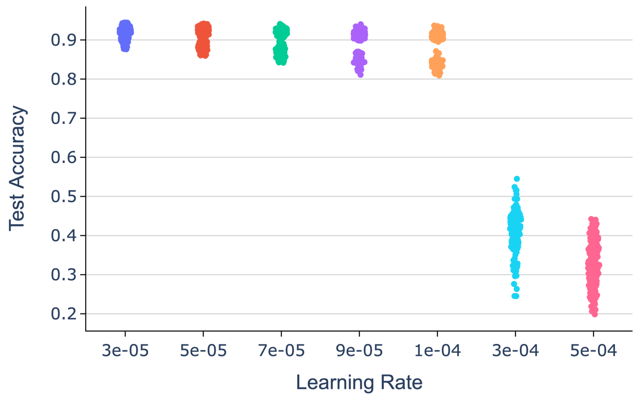

Figure 15. ***DINO - Effect of learning rate on test accuracy***

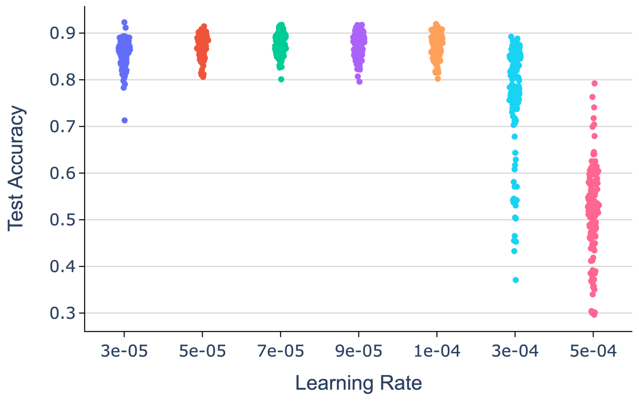

Figure 16. ***MAE - Effect of learning rate on test accuracy***

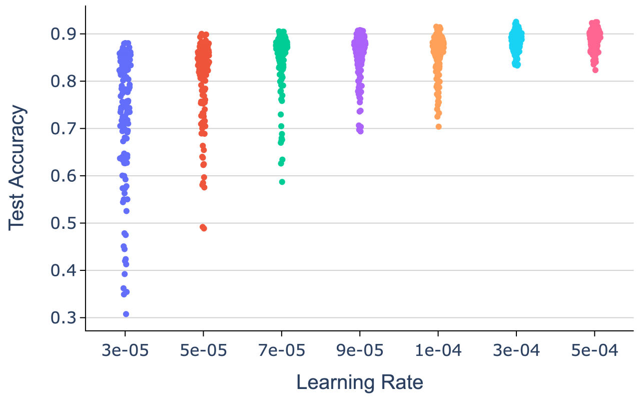

Figure 17. ***ResNet - Effect of learning rate on test accuracy***

References

[1] Vincent Herrmann, Francesco Faccio, and Jürgen Schmidhuber. Learning useful representations of recurrent neural network weight matrices. In *Forty-first International Conference on Machine Learning*, 2024. 3

[2] Dominik Honegger, Konstantin Schürholt, and Damian Borth. Sparsified model zoo twins: Investigating populations of sparsified neural network models. *arXiv preprint arXiv:2304.13718*, 2023. 6

[3] Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu

Chen, Gertjan J Burghouts, Efstratios Gavves, Cees GM Snoek, and David W Zhang. Graph neural networks for learning equivariant representations of neural networks. *arXiv preprint arXiv:2403.12143*, 2024. 7

[4] Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34: 16481–16493, 2021. 3

[5] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. *Advances in Neural Information Processing Systems*, 35: 27906–27920, 2022. 3

[6] Konstantin Schürholt, Diyar Taskiran, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Model zoos: A dataset of diverse populations of neural network models. *Advances in Neural Information Processing Systems*, 35: 38134–38148, 2022. 6

[7] Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *Forty-first International Conference on Machine Learning*, 2024. 3

[8] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 4

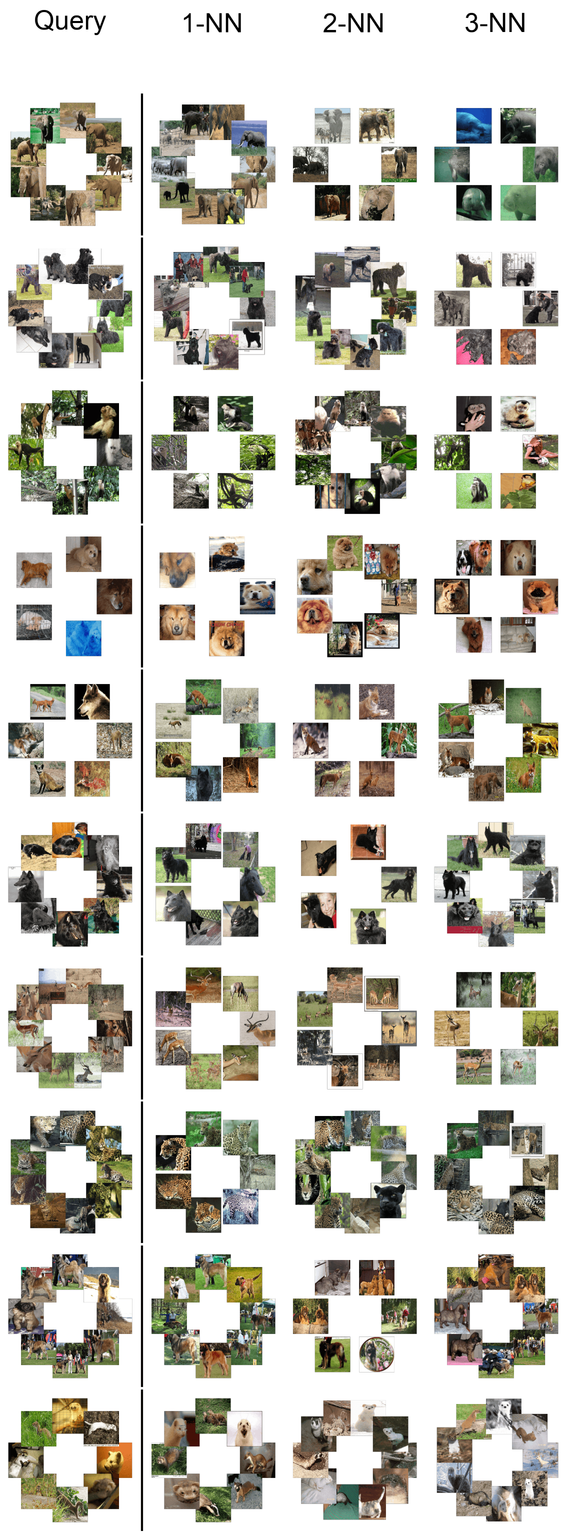

Figure 18. ***Additional non-cherry picked retrieval results (1/3):*** Retrieval is performed using model weights, to visualize each model we use the set of all the images that were used to fine-tune the model.

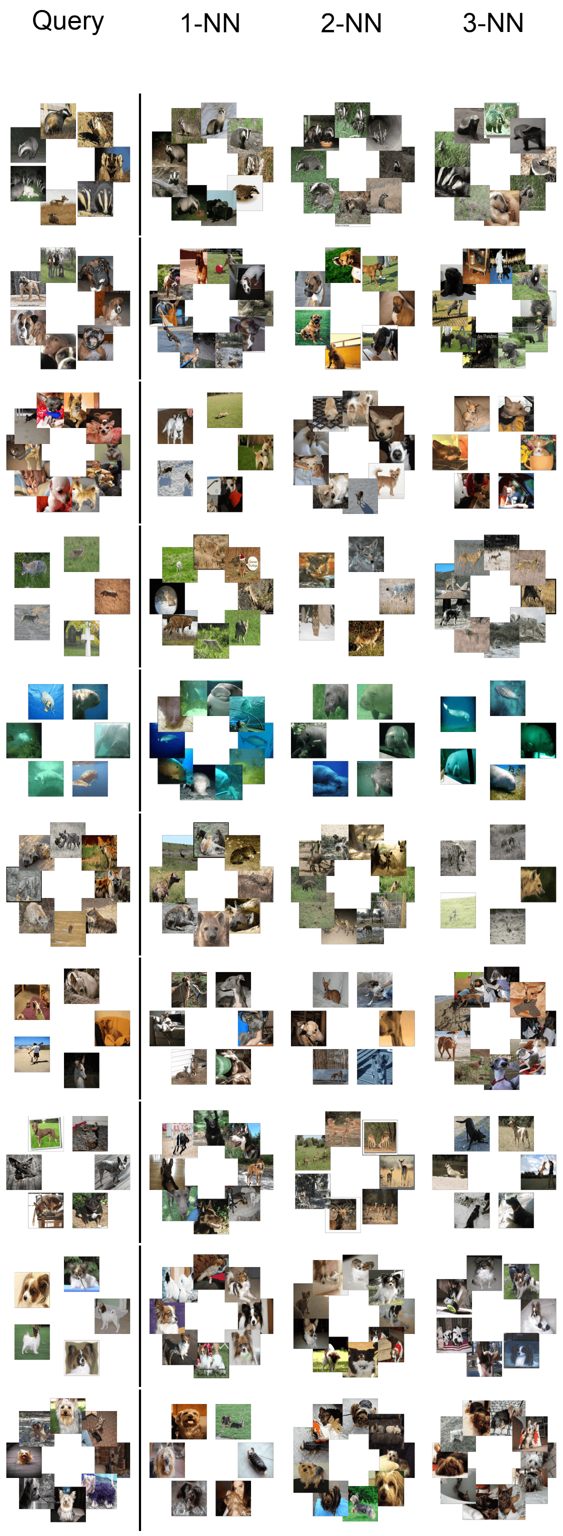

Figure 19. ***Additional non-cherry picked retrieval results (2/3):*** Retrieval is performed using model weights, to visualize each model we use the set of all the images that were used to fine-tune the model.

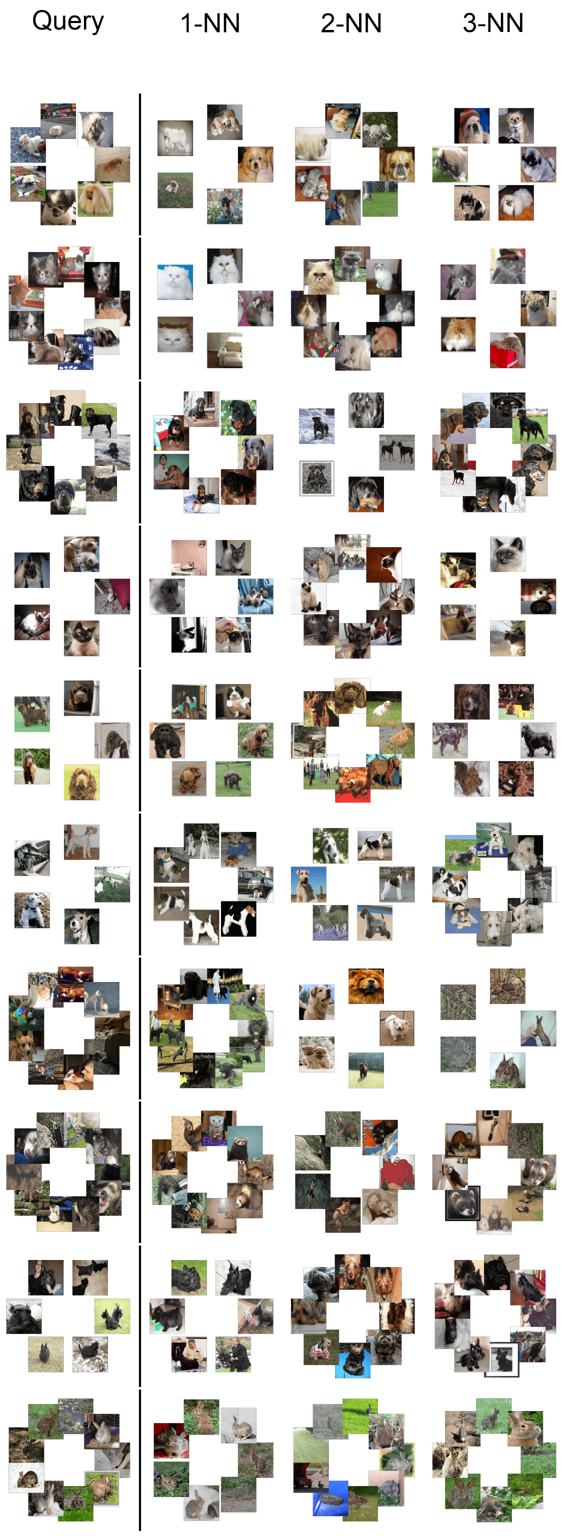

Figure 20. ***Additional non-cherry picked retrieval results (3/3):*** Retrieval is performed using model weights, to visualize each model we use the set of all the images that were used to fine-tune the model.