WHICH NEURAL NETWORK LAYERS HAVE STABLE REPRESENTATIONS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Many applications that rely on deep neural network (DNN) representations, such as encoding models or transfer learning, require choosing which intermediate layers to use. For such applications, we would like to get similar results for different instances of a model, i.e. stable results. Here, we aim to find out which DNN layers fulfill this criterion. We analyze five instances each of six ImageNet-trained architectures, comparing representations from each layer with their counterparts in other instances. Later network stages tend to be less self-similar, while certain layer types maintain higher stability throughout the network. We identify three criteria that characterize these most stable layers: (i) they are not bypassed by a skip connection, (ii) they are normalized, and (iii) they are placed after the non-linear activation function. Based on these criteria we modify ConvNext to yield an additional stable layer type confirming the validity of our criteria. Furthermore, we predict a large-scale fMRI dataset from each layer's representation, finding that stable layers yield more consistent and improved results for these encoding models. Finally, we tested transfer learning for a fine grained classification task, again observing more stable and better performance for stable layers. To confirm that our observations are not caused by the specific comparison methods, we perform each analysis with two different sets of methods. Based on our results, it seems advisable to focus on stable layers for all analyses that are based on internal representations of DNNs.

1 INTRODUCTION

Many applications of deep neural networks (DNNs) require us to use internal representations of neural networks. For fine-tuning, transfer learning [41, 43], and using foundation models [6], we need to choose an intermediate layer of a pretrained network from which we map to the new task we want to solve. To understand what a DNN is doing internally for explainable artificial intelligence or to optimize DNN performance, we either analyze individual internal representations [4, 1] or compare them to each other [18, 13]. Finally, when we use DNNs as models of biological brains [8], we compare intermediate representations to brain measurements to find out which DNNs are good models of the brain and how to make them better. Most work so far has focused on how to analyze and use internal representations effectively. Here, we instead ask which layers' representations one should analyze.

A necessary condition for all uses of internal representations is their stability, i.e. that we get similar results for different instances of the same neural network trained for the same task. Without stability, successful transfer learning becomes a matter of chance, because different instances yield substantially different results. Also, on average, layers that produce unstable representations are expected to perform poorly, because most random representations are uninformative for machine learning tasks. Simultaneously, understanding what neural networks do and what makes them work well is substantially impaired, when statistics and comparisons between layers vary between instances. After all, a representation that is not reliably linked to a specific architecture or training method cannot really provide evidence whether the chosen training scheme is good. Similarly, comparisons to biological neural networks give unreliable guidance on what makes neural networks similar to the brain. Thus, we should focus on stable internal representations for all purposes, leading to the question of which representations fulfill this requirement.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

An important complication of our question is the increase in architectural variety of deep neural networks. Previous work has investigated which convolutional layers have stable representations in networks that exclusively use such layers [20, 28, 27]. Whether and how these results generalize to other architectures with other building blocks is unclear. Only a broader analysis across different architectures can reveal which layers in these more complex neural networks have stable representations. Furthermore, unless we are willing to test stability in all architectures, stability results need to be synthesised into criteria that allow us to infer which layers in an untested model architecture are expected to have stable representations.

Here, we evaluate the representational stability of intermediate layers of six neural network architectures trained on ImageNet [31]. For this, we use five instances from each network with minor modifications applied to their base training recipes. As different methods for comparing and analyzing internal representations of DNNs [e.g. 18, 5, 39] sometimes disagree [7], we employ at least two methods per analysis in a three-pronged study. First (Section 4), we compute representational similarities between all layers within each network architecture to identify the most self-similar layers and empirically define three criteria that allow us to find stable layers: The most stable layers are ones that (i) retain all information (i.e., are not bypassed by skip connections), (ii) follow normalization, and (iii) come after the application of non-linear activations. To test these criteria on an unseen architecture, we use a modified ConvNeXt model and demonstrate their validity. Second (Section 5), we predict fMRI data from the internal representations and test which layers yield stable model-to-brain comparisons for the different instances (Section 5.1). The most stable layers turn out to also be among the best performing ones in terms of brain similarity (Section 5.2). Third (Section 6), paralleling our findings in the brain encoding, we show that these stable layers also produce more consistent results across instances and outperform unstable layers on a transfer learning task, highlighting the functional advantages of representational stability.

2 Related Work

Several previous studies [20, 28, 27, 36] investigated whether different instances of a model learn similar representations. [20] found that most representations in classical convolutional networks are indeed stable across network instances, while [28] found that learning rate changes in otherwise identical networks could lead to different representations, and that differences increased in later stages of the networks. To a similar end, [27] used analysis tools from neuroscience to investigate representational differences in convolutional neural networks, changing only the initialization parameters, and also found that differences were more pronounced in later layers. Lastly, [36] simulated mouse neural activity and compared the intermediate representations of their models, arguing for a specific method to compare the representations after the non-linear activation function.

Another literature that has analyzed the similarity between network instances focuses on the metrics used to compare internal representations to each other. This literature is additionally relevant to our work as we need to choose metrics to perform our comparisons. A variety of methods exist, such as those based on canonical correlation analysis (CCA, [28, 30]), linear regression, and centered kernel alignment (CKA, [18]). While still focusing on convolutional layers, [18] showed cases where CKA correctly identifies the same layers across training variations for the same architecture, while CCA-based methods and linear regression did not. However, later work showed that both CCA-based methods and CKA can show inadequate performance under certain conditions [10]. In neuroscience applications, the aforementioned metrics are used together with others like representational similarity analysis (RSA, [19]) and voxel-encoding RSA (ve-RSA, [15]). It is unclear which methods are best, as recent large-scale evaluations [17] favor different metrics for different evaluation criteria and the discussion on the best methods is ongoing [32, 3, 36].

3 Analyzed Models

We consider six of the most widely used neural network architectures for image classification: ResNet-50 [14], Vision Transformer (ViT-B/16) [11], Swin Transformer Tiny [23], Swin Transformer V2 Small [24], ConvNeXt Tiny [25] and EfficientNet V2 Small [34]. We train 4 instances of each network on ImageNet [31] with minor modifications in their base training recipes, such as different image augmentation policies or learning rates, in addition to different random initializations (see

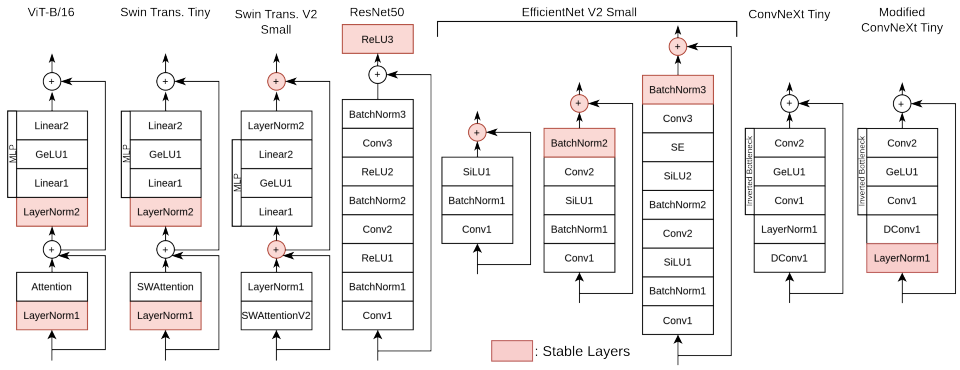

Figure 1: Blocks of each architecture. Different from other models, EfficientNet has three types of blocks. Also, in EfficientNet, if a block is at the beginning of a stage, it does not have a skip connection and an add layer at the end of the block. Stable layers we found in this work are colored.

Table 2 in Appendix A.3 for details). As an additional instance for each architecture, we use a pretrained model sourced from TorchVision (BSD-3-Clause License) [26]. Validation performances of the different instances are similar.

Each architecture consists of repeating blocks (see Fig. 1 for computational graphs of these blocks). Our main question below will be at which points in these repeating blocks stable representations occur. In addition to the repeating blocks, the architectures contain some initial and final layers that do not follow the block design, which we treat separately. Furthermore, the architectures contain rare blocks that reduce the spatial resolution (e.g. pooling, strided convolution or patch-merge layers, typically 3 occurrences per architecture). We did not include these layers in our main analyses, because their rarity hinders interpretation of the results. Results for these layers are provided in Appendices A.6 and A.8.

Unlike other architectures, EfficientNet contains three distinct types of repeating blocks. Furthermore, if a block is at the beginning of a stage (a group of blocks), then it does not have a skip connection. Since EfficientNet V2 Small has a high number of layers, we subsampled its blocks. We always got the first block from a stage and one other block from the same stage, to sample blocks with and without skip connections.

4 Identifying stable representations in DNNs

To obtain representations, we use a randomly selected 1000-image subset of images from the unlabeled set of MS COCO (CC BY 4.0) [22] (unlabeled2017) as input. To estimate layer stability, we compute the average self-distance of each layer, i.e. the average distance between instances of the layer.

We compute distances between layers using BayesCompare (MIT License) [32] and linear Centered Kernel Alignment (CKA) [18] (see definitions in Appendix A.1). We used BayesCompare with the square root of Jensen-Shannon Divergence (JSD) [33, 12] and a noise level parameter set to $b = 1/100$. We report and compare 1-CKA instead of CKA itself. BayesCompare and CKA are substantially different measures of similarity that differ in their equivalence classes. In particular, CKA is invariant to translations, while BayesCompare is not.

Self-distances of layers throughout ViT-B/16 are shown in Fig. 2 A. Consistent with earlier studies on purely convolutional neural networks trained for simpler tasks [27, 18], we find that self-distance increases with depth in all architectures for all layer types. It is also evident that different layers are stable to different degrees at similar depths and that there is a repetitive pattern corresponding to each block. The two metrics mostly agree on which layers within the blocks are most self-similar, and the same layer types are consistently more self-similar than the other layers throughout the network. A similar pattern occurs for the other architectures as well (see Appendices A.6 and A.8 for self-distance of layers of all model architectures).

To extract which layer types yield the most stable results, we compare the distribution of self-distances across layer types (Fig. 2 B). Unsurprisingly, initial layers produce the most stable results in all

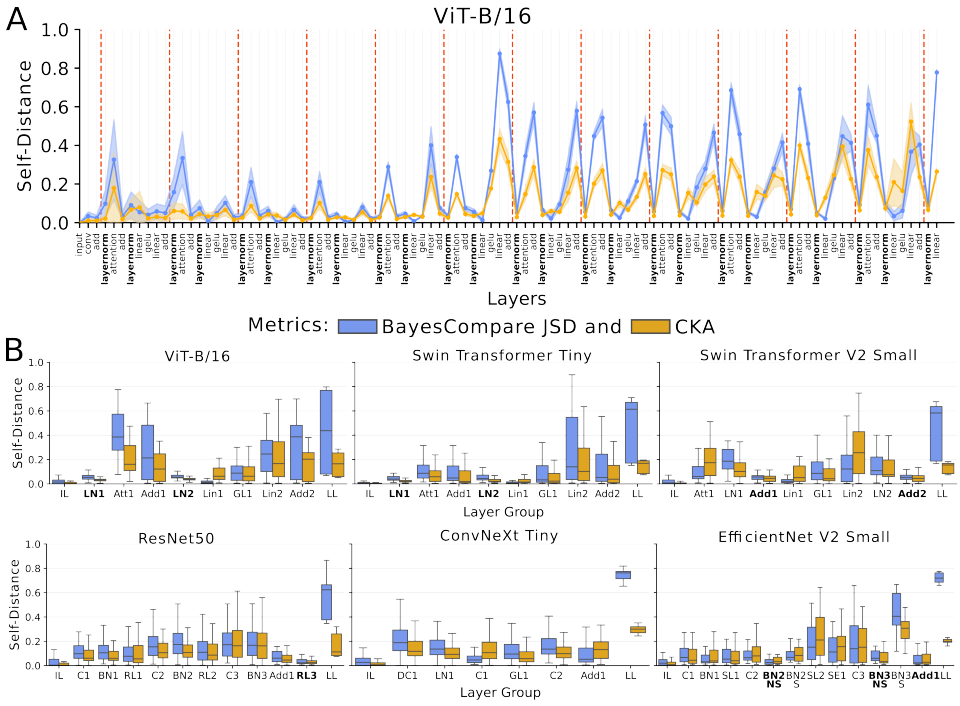

Figure 2: Self-distances of layers. **A:** Self-distances of all layers in ViT-B/16, for JSD and CKA, averaged over instances and shading indicates standard deviation, **B:** Self-distances per layer group in all architectures, for JSD and CKA. Layer groups are ordered according to their occurrence in the model. Most stable layers selected by both metrics are indicated with bold font. **Abbreviations (alphabetical):** Att: Attention; BN: Batch normalization; BN NS: Batch normalization not skipped; BN S: Batch normalization skipped; C: Convolutional; DC: Depth convolutional; GL: GeLU; IL: Initial layers; Lin: Linear; LL: Last layers; LN: Layer normalization; RL: ReLU; SE: Squeeze and Excitation; SL: SiLU.

architectures, matching the overall trend of increasing instability throughout the network. These few layers in the stem of the model appear to be stable without further conditions. However, these layers are rarely of interest for applications because their representations are still very similar to the input. The last layers after the repeating blocks are highly unstable in all tested architectures, consistently yielding substantially higher self-distances than the stable layers in the last repeating block.

For the repeating blocks, BayesCompare's JSD and CKA largely agree which layer types are most stable (see Fig. 2 B and Appendix Table 3 for listed results). We further supported their findings with a third neighborhood-based metric, Jaccard similarity (definition in Appendix A.1 and results in Appendices A.7 and A.9 and Table 4). Based on the most stable layer findings of these measures, we empirically derive the following model-independent criteria that stable layers follow:

1. Not bypassed by a skip connection
2. After normalization
3. After non-linearity

First, the most stable layers are always among the few layers where all information is represented in the same layer, i.e. the ones that are not bypassed by a skip connection. Thus, we see two points of low self-distance per block in the transformer architectures, one after the transformer part and one after the MLP part and only one in the convolutional neural networks, close to the point where the skip-connection is rejoined.

EfficientNet provides particularly clear evidence for this first criterion, as it includes the same type of blocks with and without a skip connection. We distinguish between the two types of blocks, specifically for their last batch normalization layers, by separating BatchNorm2 not-skipped (BN2 NS) from BatchNorm2 skipped (BN2 S), and BatchNorm3 not-skipped (BN3 NS) from BatchNorm3

skipped (BN3 S), respectively. The not-skipped versions of the layers are considerably more stable than the skipped versions of the same layer (see Figure 2 B and Table 3).

Second, when a normalization layer follows the skip-connection, its output is typically more stable. In early blocks, the un-normalized version can be similarly stable, but later in the network, the difference becomes obvious. For the ViT (Figure 2 A), this effect is quite visible in the latter half of the network, where the add layers are quite dissimilar between instances, while the subsequent layernorm results in a stable representation.

Third, if we are left with a choice between a representation before or after the non-linearity, the output of the non-linearity tends to be more stable than its input. For example, in ResNet-50, we are left with the representation after the add layer that combines the main path and the skip connection before or after applying the ReLU. The ReLU transformed representation is clearly more stable.

4.1 Understanding Exceptions

Expansion Layer in Transformer MLPs and ConvNeXt Inverted Bottleneck. The first linear layers in the transformer MLP blocks, 'Linear1', and the analogous layer 'Conv1' in ConvNeXt's inverted bottleneck block yield highly self-similar results according to JSD but not according to CKA (see Figure 2 B and Table 3).

To understand the difference between JSD and CKA, we examine the activation means of features over stimuli, as these values are subtracted from the activations when computing double-centered Gram matrices for CKA. The mean activations over stimuli are highly negative for most features in these layers (see Appendix A.10). This strong negativity is consistent across instances. This similarity is detected by JSD, but CKA does not capture the same effect. The low similarities according to CKA show that beyond this mean shift, these layers are not particularly stable.

These layers appear to produce negative means for their features to create a sparse representation in the subsequent GeLU layers. Pushing most features into negative values allows them to be eliminated by the subsequent GeLU non-linearity, thereby producing the sparse activations in this layer [21, 42].

ConvNeXt. ConvNeXt does not yield any highly stable layer group, compared to other models. Instead, all layer groups are similarly unstable and the two metrics disagree on which layers are least unstable. Notably, this is consistent with our criteria, as none of the layers from the repeating blocks in ConvNeXt fit the criteria. Outside the repeating blocks, we do find some stable layers in ConvNeXt, though (see Appendices A.6 and A.8): The first layer in the downsample blocks is most stable according to both metrics. These layers are also the only ones that conform to our criteria. On the downside, these downsample blocks occur only 3 times throughout the network.

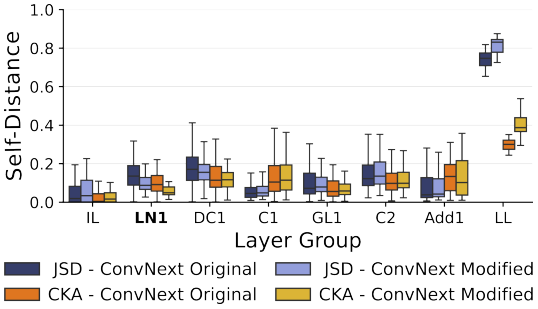

Figure 3: Self-distances per layer group in ConvNeXt Tiny (excluding the PyTorch trained model) and Modified ConvNeXt Tiny for JSD and CKA.

Test on a held-out architecture. The fact that no layers meet the criteria in ConvNeXt gives us a test bed to confirm our criteria on an unseen architecture. The criteria predict that if the LayerNorm and dConv layers are swapped, as shown in the rightmost diagram in Fig. 1, the LayerNorm layers should emerge as the most stable. To test this prediction, we train 4 instances of this modified ConvNeXt architecture on ImageNet and use the same recipes as for the original ConvNeXt instances. We compare the results of the modified ConvNeXts only with the 4 instances of the original ConvNeXts that we trained ourselves.

After this modification, the stability of 'LayerNorm1' layer indeed improved for both metrics and it became the most stable layer, with the exception of the 'Conv1' layer for JSD, which is explained to be an outlier for JSD above, see Fig. 3 (see Appendix A.11 for detailed plots). This confirms the validity of our criteria on an unseen model and shows that by following the criteria's predictions, we can obtain stable layers for models that lack them.

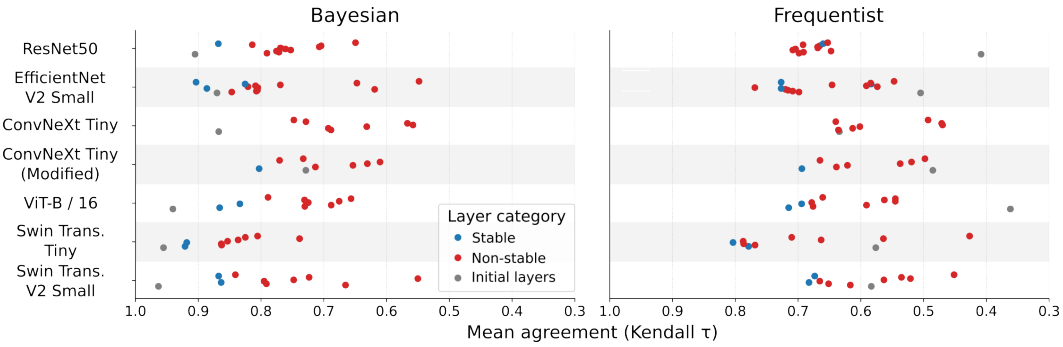

Figure 4: Mean agreement (Kendall's τ between instances) for the different architectures according to Bayesian (left) and frequentist (right) linear encoding models. Groups are divided into stable and non-stable, with their vertical arrangement within each architecture reflecting layer depth (deeper groups are higher).

5 Assessing stability for brain encoding

To test a common application of representational comparisons in neuroscience and confirm our layer stability results using independent methods, we examined whether evaluations are more stable for stable layers in a brain encoding model context. For this, we use a subset of the publicly available natural scenes dataset (NSD, [2]), which contains 7T fMRI data from 4 participants viewing the *shared1000* images from the validation set of MS COCO [22]. We include a liberal definition of brain areas activated during the NSD experiments. For more details on the dataset and voxel selection, see Appendix B.1. We evaluate each layer in each model instance as a model of each voxel using a linear encoding model, i.e. a L2-regularized linear regression [29]. Additionally, we use BayesCompare [32], which implements a Bayesian treatment of linear regression. See Appendix B.2 for more details.

5.1 Representational agreement in linear encoding

If stable layers are indeed stable in a general sense, their results for the different instances should be more similar to each other than for other layer types. To measure this, we compute a rank correlation (Kendall's τ) between all pairs of the five instances for each layer group and voxel. Overall stability differs strongly between voxels, but the ordering of layers groups is quite consistent over brain areas and noise levels (see Appendix Figure 12). We thus analyze only the mean agreement in the main text.

Within the repeated blocks, we mostly find stable results for the same layer groups that were found to be stable in our previous analysis (Fig. 4). The Bayesian linear regression agrees almost completely with the groups that follow the stability criteria (except for the non-skipped BatchNorm3 layers in EfficientNet that are at a disadvantage because they occur only deep in the network). The frequentist evaluation generally yields noisier results and lower agreement between instances, but still mostly agrees with the ranking of layer groups, except for three additional exceptions: For ResNet50, all layer groups give similarly stable results. For Swin Transformer Tiny, two layers groups that we classified as unstable (i.e. Linear1 and Add1) still yield stable results. In the case of EfficientNet, a layer group classified as unstable (i.e. the skipped BatchNorm2) yielded the most stable results. Due to the large number of voxels, our uncertainty about the average agreement is very small and virtually all pairwise differences between layer groups are significant (Wilcoxon signed-rank test, $p < 0.001$ for all but 3 pairs for Frequentist and 2 pairs for Bayesian analysis).

5.2 Are stable layers the best predictors of the brain?

Beyond discerning which model fits the brain best, we often also want to predict brain responses well. We should thus confirm that stable layers yield good prediction quality, too, before we can generally recommend using only the stable layers. To evaluate how well layers do, we ranked all layers of each model by prediction performance (with 1 being the best layer). To get a single score per layer, we

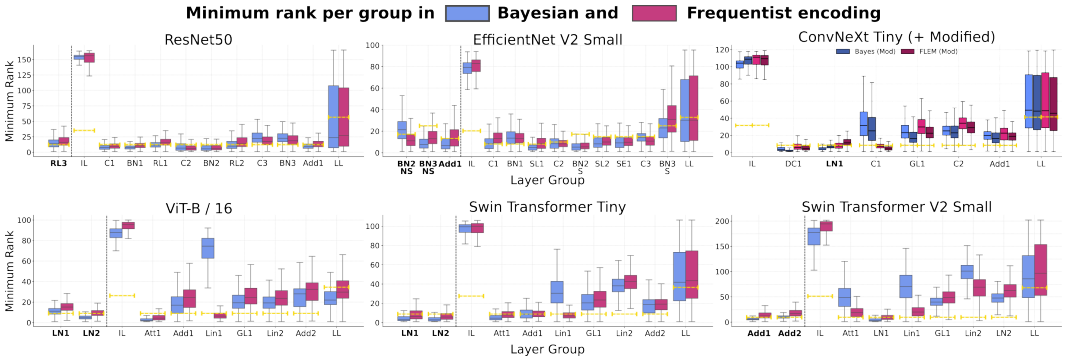

Figure 5: Voxel-wise minimum (best) ranking layer of each group according to Bayesian and frequentist encoding models. A lower position on the Y-axis indicates a better ranking. The dashed yellow line in each layer group corresponds to the median and IQR of the expected ranks if scores were random as computed using a permutation test. Layer groups that fulfill the stability criteria are grouped leftmost for each architecture and their label is presented in a bold font. The plots with the rare layer types are shown in App. B.5.

average across instances. We see a general U-shape trend, with better scores as the model progresses but a slight rebound towards worse scores at the very end of the model (see supplementary material B.4 for plots across depth). We again see a repeating pattern in each block of the network, with the stable layers among the best performers in each block.

Following the logic of maximizing performance, we use the score of the best layer within a layer group for each voxel to evaluate the layer groups. Due to the selection of the best layer, larger groups are expected to yield lower ranks. To give a reference for this effect, we compute the expected ranks if all scores were random. Both Bayesian and Frequentist encoding models find the stable groups among their best performers (Fig. 5), with considerably lower ranks than expected randomly. The Bayesian evaluation generally is less variable and thus yields a somewhat higher variance in ranks than the frequentist analysis.

The 'linear1' layers in the transformer architectures (ViT, Swin Transformer and Swin V2) again cause the biggest disagreement between the two methods. According to the Bayesian analysis, they perform very badly, while the frequentist analysis judges them to perform equally to the stable layer that precedes them. This effect is again due to the consistently negative representations in these layers which makes the Bayesian method predict strong correlations between the responses to all stimuli, which are not observed in the brain.

6 Representational stability for transfer learning

To show that the stability we describe is relevant for machine learning applications, too, we reuse the representations in our networks as features for transfer learning. We use the CUB-200-2011 dataset, which contains approximately 11k images categorized into 200 species of birds. For all analyses, we use the train/test split provided with the dataset. As features, we extract the activations from the last repeating block of each instance and architecture. Dimensionality reduction was applied using principal component analysis (PCA), targeting the number of components that account for 99% of the variance on the training set.

We used three classification methods to classify images into the new classes:

- **Nearest-centroid classification**: We obtain a prototype feature vector by averaging all feature vectors of a training class. For each test sample, we used cosine similarity to the prototypes to select the most similar one.
- **Linear probing**: We trained a logistic regression classifier using the training set and tested on the testing set.
- **Unsupervised learning**: Similarly, we trained a K-nearest neighbors (KNN) classifier on the dataset.

For both logistic regression and KNN, grid-search cross-validation was applied to find appropriate hyper-parameters (C and K, respectively) using a stratified 5-fold strategy. The three analyses were performed independently.

Table 1: Balanced test accuracy (mean and standard deviation across five instances) on the CUB-200-2011 fine grained bird classification task for stable vs. non-stable layers, across architectures and evaluation methods.

		Mean		Std.	
Architecture	Method	Stable	Non-stable	Stable	Non-stable
ResNet50	KNN	**.342**	.225	**.050**	.080
	Linear Probe	**.513**	.416	.044	**.038**
	Nearest Centroid	**.399**	.264	**.064**	.099
EfficientNet v2	KNN	**.330**	.232	**.032**	.042
	Linear Probe	.467	**.495**	**.018**	.020
	Nearest Centroid	**.409**	.307	**.043**	.048
ConvNext Tiny	KNN	N/A	.377	N/A	.042
	Linear Probe	N/A	.555	N/A	.035
	Nearest Centroid	N/A	.469	N/A	.043
ConvNext Tiny (Mod.)	KNN	.313	**.323**	**.024**	.035
	Linear Probe	**.545**	.518	**.014**	.020
	Nearest Centroid	**.424**	.405	**.025**	.036
ViT-B/16	KNN	**.169**	.086	**.016**	.024
	Linear Probe	**.458**	.412	**.011**	.034
	Nearest Centroid	**.245**	.131	**.027**	.035
Swin Transformer	KNN	**.345**	.212	**.021**	.022
	Linear Probe	**.593**	.582	.010	.010
	Nearest Centroid	**.460**	.282	.033	**.027**
Swin Transformer V2	KNN	.038	**.204**	**.005**	.022
	Linear Probe	**.621**	.495	**.024**	.045
	Nearest Centroid	.063	**.283**	**.007**	.024

The results in Table 1 show the balanced test accuracy and standard deviation for each architecture. The layer groups were grouped into stable and unstable layers.

First, we find that stable layers overall produce more similar results (as represented by their standard deviations across instances). This is analogous to our analysis in Section 5.1 where we evaluated the agreement for layer groups in brain encoding. Second, we observe that stable layers perform better in general (as represented by the mean accuracy). This parallels the brain decoding analysis depicted in Section 5.2.

There are some exceptions to these general trends, with the Swin Transformer V2 being the biggest. In this case, the performance for the KNN and nearest centroid classifiers based on the stable layers is very low, while the linear probing still shows an advantage of the stable layers. After observing these results, we noticed that this particular Add layer (i.e. in the last block) is different from those seen at the end of previous blocks of the network, because it is followed by a normalization layer that is part of the fully connected layer block. Also, our main analyses show its similarity and performance to be lower than those of preceding blocks and our criteria would have preferred the following normalization layer. Thus, the results for this network should be taken with a grain of salt.

These results clearly illustrate that layer stability is relevant for practical applications. Also, we confirm our results with entirely separate methods and data.

7 Discussion

We analyze the stability of layer representations in various DNN architectures and found considerable variability across different layer types. Layers consistently produce stable representations if they fulfill the following three criteria: They (i) retain all the information, (ii) follow normalization, and (iii) happen after the application of non-linear activations.

Our results are based on an extensive analysis of five instances each of six object classification architectures trained on ImageNet: ResNet, Vision Transformer, Swin Transformer, Swin Transformer V2, ConvNeXt and EfficientNet. Distances between representations in different instances are smaller in stable layers than in other layer types. Different instances give more consistent results for brain encoding models and for transfer learning tasks and performance on both tasks is better for stable layers. Furthermore, our criteria generalize to unseen architectures as shown by the modified ConvNeXt model producing a stable layer type. We confirmed each analysis using at least two different sets of methods, ensuring that the stability results are not due to idiosyncrasies of the methods.

While we did not originally derive the three criteria for stable layers from theory, we can provide an intuitive justification for them: (i) All information about the stimulus that is not strongly altered in a specific block may be passed through the processing block or through the skip connection with equal effect on the overall network behavior. Thus, training is not expected to favor either path and different instances will contain unmodified information in the skipped layer to a random extent, making the layer unstable. (ii) Whenever normalization occurs in a network, the scale of features before the normalization does not influence model behavior. Therefore, different instances will have different random scales for the features in layers before normalization steps, such that these layers are unstable. (iii) For ReLu-type non-linearities, small changes to negative activations do not affect overall network behavior. Thus, again, training will not constrain these activations effectively and different instances are expected to yield randomly different representations in the layers before non-linearities.

Our results will help guide layer selection in both machine learning and neuroscience applications. For understanding neural network computations or comparing them to brain processing, stable representations are obviously advantageous. Including unstable layers would yield unreliable and inconsistent results within a single study and when comparing results across studies. We observe good performance for transfer learning and brain encoding for the stable layers, too. Thus, it seems generally recommendable to focus on stable layers for applications that focus on performance, too. As an added bonus, considering only the stable layers is computationally cheaper than considering all layers. Based on our criteria, stable layers can be selected without further tests.

A particularly important result from our study, which confirms previous findings, is that the last layers are highly unreliable across all architectures. These layers have been a popular choice in studies evaluating similarity metrics [e.g. 16, 5, 40] and in studies on brain similarity [37]. Even though these layers may seem attractive due to their low dimensionality and deep placement in the network, our results strongly suggest that they should be avoided for these purposes in the future.

More broadly, our results highlight that we need to be careful and explicit about which exact representation is used for what analysis. For the analyses we report here, seemingly small differences—like whether the representation is extracted before or after the non-linearity or how much normalization is applied—yield categorically different results.

Even though we consider a wide variety of neural network architectures, our study is limited to vision neural networks, trained for the object classification task with supervised learning. Whether the criteria that we extract generalize to DNN architectures for other tasks and modalities remains untested.

AI USE STATEMENT

We have not used generative AI tools for any of the required disclosure tasks.

We only used generative AI tools as support tools for writing code and text, e.g. to polish writing, to identify relevant literature and to create or edit code. We have reviewed all AI-assisted work. We verified all LLM-generated code, checked the relevant literature suggestions provided by LLMs, and processed the writing suggestions before including them in the main text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics statement

This study uses only openly available datasets (ImageNet, MSCOCO, NSD and CUB-200-2011) and is in compliance with the ICLR Code of Ethics in every respect.

Reproducibility statement

We explain in detail our analyses and the models we trained and used in Sections 3, 4, 5 and 6. We provide the model training details in Appendix A.3 Table 2. Implementation details for model-to-model comparisons are provided in Appendix A.1 and model-to-brain comparisons in Appendices B.1 and B.2.

We provide the code for our model training and analyses, along with data specifications, in the supplemental materials, to enable replication of the results. All tools we used for these analyses are publicly available, including in particular the Python packages TorchLens, BayesCompare, and rsatoolbox.

References

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[2] Emily J Allen, Ghislain St-Yves, Yihan Wu, Jesse L Breedlove, Jacob S Prince, Logan T Dowdle, Matthias Nau, Brad Caron, Franco Pestilli, Ian Charest, et al. A massive 7t fmri dataset to bridge cognitive neuroscience and artificial intelligence. *Nature neuroscience*, 25(1):116–126, 2022.

[3] Itamar Avitan and Tal Golan. Model-behavior alignment under flexible evaluation: When the best-fitting model isn't the right one. *arXiv preprint arXiv:2510.23321*, 2025.

[4] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6541–6549, 2017.

[5] Enric Boix-Adsera, Hannah Lawrence, George Stepaniants, and Philippe Rigollet. Gulp: a prediction-based metric between representations. *Advances in Neural Information Processing Systems*, 35:7115–7127, 2022.

[6] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the Opportunities and Risks of Foundation Models, July 2022. URL `http://arxiv.org/abs/2108.07258`. arXiv:2108.07258 [cs].

[7] Nathan Cloos, Moufan Li, Markus Siegel, Scott L Brincat, Earl K Miller, Guangyu Robert Yang, and Christopher J Cueva. Differentiable optimization of similarity scores between models and brains. *arXiv preprint arXiv:2407.07059*, 2024.

[8] Colin Conwell, Jacob S Prince, Kendrick N Kay, George A Alvarez, and Talia Konkle. What can 1.8 billion regressions tell us about the pressures shaping high-level visual representation in brains and machines? *BioRxiv*, pp. 2022–03, 2022.

[9] Colin Conwell, Jacob S Prince, Kendrick N Kay, George A Alvarez, and Talia Konkle. A large-scale examination of inductive biases shaping high-level visual representation in brains and machines. *Nature communications*, 15(1):9383, 2024.

[10] Frances Ding, Jean-Stanislas Denain, and Jacob Steinhardt. Grounding representation similarity through statistical testing. *Advances in Neural Information Processing Systems*, 34:1556–1568, 2021.

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

[12] D.M. Endres and J.E. Schindelin. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, July 2003. ISSN 0018-9448. doi: 10.1109/TIT.2003.813506. URL http://ieeexplore.ieee.org/document/1207388/. Number: 7.

[13] Sarah E Harvey, David Lipshutz, and Alex H Williams. What representational similarity measures imply about decodable information. *arXiv preprint arXiv:2411.08197*, 2024.

[14] 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, 2016.

[15] Philipp Kaniuth and Martin N Hebart. Feature-reweighted representational similarity analysis: A method for improving the fit between computational models, brains, and behavior. *NeuroImage*, 257:119294, 2022.

[16] Meenakshi Khosla and Alex H Williams. Soft matching distance: A metric on neural representations that captures single-neuron tuning. In *Proceedings of unireps: the first workshop on unifying representations in neural models*, pp. 326–341. PMLR, 2024.

[17] Max Klabunde, Tobias Schumacher, Markus Strohmaier, and Florian Lemmerich. Similarity of Neural Network Models: A Survey of Functional and Representational Measures, August 2024. URL http://arxiv.org/abs/2305.06329. arXiv:2305.06329 [cs].

[18] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pp. 3519–3529. PMlR, 2019.

[19] Nikolaus Kriegeskorte, Marieke Mur, and Peter A Bandettini. Representational similarity analysis-connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:249, 2008.

[20] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations? *arXiv preprint arXiv:1511.07543*, 2015.

[21] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J. Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, and Sanjiv Kumar. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=TJ2nxciYCk-.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

[23] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021.

[24] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12009–12019, 2022.

[25] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11976–11986, 2022.

[26] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016.

[27] Johannes Mehrer, Courtney J Spoerer, Nikolaus Kriegeskorte, and Tim C Kietzmann. Individual differences among deep neural network models. *Nature communications*, 11(1):5725, 2020.

[28] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. *Advances in neural information processing systems*, 31, 2018.

[29] Thomas Naselaris, Kendrick N. Kay, Shinji Nishimoto, and Jack L. Gallant. Encoding and decoding in fMRI. *NeuroImage*, 56(2):400–410, May 2011. ISSN 10538119. doi: 10.1016/j.neuroimage.2010.07.073. URL `https://linkinghub.elsevier.com/retrieve/pii/S1053811910010657`. Number: 2.

[30] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. *Advances in neural information processing systems*, 30, 2017.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[32] Heiko H. Schütt. Bayesian comparisons between representations. In *8th Annual Conference on Cognitive Computational Neuroscience*, 2025. URL `https://openreview.net/forum?id=BEzJu8VsJ6`.

[33] Robin Sibson. Information radius. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 14:149–160, 1969. URL `https://api.semanticscholar.org/CorpusID:198179065`.

[34] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pp. 10096–10106. PMLR, 2021.

[35] JohnMark Taylor and Nikolaus Kriegeskorte. Extracting and visualizing hidden activations and computational graphs of pytorch models with torchlens. *Scientific Reports*, 13(1):14375, 2023.

[36] Imran Thobani, Javier Sagastuy-Brena, Aran Nayebi, Rosa Cao, and Daniel LK Yamins. Inter-animal transforms as a guide to model-brain comparison. In *Iclr 2024 workshop on representational alignment*, 2024.

[37] Aria Y Wang, Kendrick Kay, Thomas Naselaris, Michael J Tarr, and Leila Wehbe. Better models of human high-level visual cortex emerge from natural language supervision with a large and diverse dataset. *Nature Machine Intelligence*, 5(12):1415–1426, 2023.

[38] Chenxu Wang, Wei Rao, Wenna Guo, Pinghui Wang, Jun Liu, and Xiaohong Guan. Towards understanding the instability of network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 34(2):927–941, 2022.

[39] Alex H Williams, Erin Kunz, Simon Kornblith, and Scott Linderman. Generalized shape metrics on neural representations. *Advances in neural information processing systems*, 34:4738–4750, 2021.

[40] Jialin Wu, Shreya Saha, Yiqing Bo, and Meenakshi Khosla. Comparing and integrating different notions of representational correspondence in neural systems, 2026. URL https://arxiv.org/abs/2509.21628.

[41] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3320–3328. Curran Associates, Inc., 2014.

[42] Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 877–890, 2022.

[43] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning, June 2020. URL http://arxiv.org/abs/1911.02685. arXiv:1911.02685 [cs].

A Layer comparisons

A.1 Representational Similarity Measures

Suppose that we are interested in the representation in a deep neural network layer with k neurons for a list of n stimuli. The activities of these neurons can be formed into a matrix $\mathbf{X} \in \mathbb{R}^{n \times k}$. Then, the kernel matrix for this layer and the stimulus set is defined as $\mathbf{K_x} = \mathbf{X}\mathbf{X}^T$. Based on this, the double-centered kernel matrix is:

$$\bar{\mathbf{K}}_\mathbf{x} = \mathbf{C}\mathbf{X}\mathbf{X}^T\mathbf{C}, \quad \text{where} \quad \mathbf{C} = \mathbf{I}_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T. \tag{1}$$

Here, $\mathbf{I}_n$ is the $n \times n$ identity matrix and $\mathbf{1}\mathbf{1}^T$ is an $n \times n$ matrix of ones.

BayesCompare. [32] BayesCompare uses linear readouts to compare representations. With iid. Gaussian noise on the measurements and a zero-mean multivariate Gaussian assumption over the weights, this readout gives the following prior predictive distribution over all possible outcomes:

$$\mathbf{y} \sim \mathcal{N}(0, \sigma_\beta^2 \mathbf{K_x} + \sigma_\epsilon^2 I). \tag{2}$$

It is possible to use a wide range of distances or divergences over these distributions to compare representations. We selected JSD here since it is a metric and bounded, therefore, easier to compare with CKA. JSD corresponds to mutual information between samples and their generating distribution labels. The approximation used for the implementation is:

$$d^2_{\text{JSD}}(P_1, P_2) \approx \frac{1}{K}\sum_{i=1}^{K} \log_2 \frac{p_1(x_i^{(1)})}{p_1(x_i^{(1)}) + p_2(x_i^{(1)})} + \frac{1}{K}\sum_{i=1}^{K} \log_2 \frac{p_2(x_i^{(2)})}{p_1(x_i^{(2)}) + p_2(x_i^{(2)})} - 1, \tag{3}$$

where $x_i^{(1)}$ and $x_i^{(2)}$ are samples of the two distributions. For all evaluations, the square root of the Jensen Shannon Divergence, which is a metric and therefore sometimes called Jensen Shannon Distance. K is the sample size and is set to $K = 10000$ in our implementation.

In the final definition of the prior predictive distribution given in Eq. 4, the b parameter determines the sensitivity of the analysis by changing the signal-to-noise ratio. This is particularly important for

bounded metrics such as JSD. In our analysis, we set this value as $b = 1/100$ corresponding to the noise variance $\sigma_\epsilon^2 = 0.91$ for $n = 1000$ images we used in our model-to-model comparisons.

$$\Sigma_{\mathbf{y}} = \frac{1}{1+bn}\frac{n\mathbf{K}_{\mathbf{x}}}{\mathrm{tr}(\mathbf{K}_{\mathbf{x}})} + \frac{bn}{1+bn}I. \tag{4}$$

Centered Kernel Alignment (CKA). [18] CKA computes the similarity between two double-centered kernel matrices as follows,

$$s_{\mathrm{CKA}}(\bar{\mathbf{K}}_{\mathbf{x}}, \bar{\mathbf{K}}_{\mathbf{y}}) = \frac{\mathrm{tr}(\bar{\mathbf{K}}_{\mathbf{x}}\bar{\mathbf{K}}_{\mathbf{y}})}{\sqrt{\mathrm{tr}(\bar{\mathbf{K}}_{\mathbf{x}}^2)\,\mathrm{tr}(\bar{\mathbf{K}}_{\mathbf{y}}^2)}}. \tag{5}$$

Jaccard Similarity. [38] Jaccard is a neighborhood-based measure. For each pair of elements in the activation matrix, it computes the ratio between the number of shared k nearest neighbors (k-NN) (the intersection) and the total number of unique neighbors in both sets (the union). We used cosine similarity between elements for finding the k-NNs, obtained from the kernel matrices as follows,

$$\mathbf{D}_{\mathbf{x}ij}^{\text{CosSim}} = \frac{\mathbf{K}_{\mathbf{x}ij}}{\sqrt{\mathbf{K}_{\mathbf{x}ii}\mathbf{K}_{\mathbf{x}jj}}}. \tag{6}$$

Then, Jaccard is computed for a stimulus i, with the k-NNs of $\mathbf{X}$ and $\mathbf{Y}$, $\mathcal{N}_k^{\mathbf{X}}(i)$ and $\mathcal{N}_k^{\mathbf{Y}}(i)$, respectively.

$$s_{\mathrm{Jaccard}}(\mathbf{K}_{\mathbf{x}}, \mathbf{K}_{\mathbf{y}}, i) = \frac{|\mathcal{N}_k^{\mathbf{X}}(i) \cap \mathcal{N}_k^{\mathbf{Y}}(i)|}{|\mathcal{N}_k^{\mathbf{X}}(i) \cup \mathcal{N}_k^{\mathbf{Y}}(i)|}. \tag{7}$$

We set $k = 10$, in our implementations. The results reported for this metric are for $1 - s_{\mathrm{Jaccard}}$ rather than the similarity score.

A.2 Resources for Experiments

Model-to-model Comparisons. To obtain activations from neural networks, we used 4 NVIDIA Ampere 40GB GPUs on an HPC cluster node. Covariance extraction ranges from around 2 hours to a few minutes (including 5 instances), for 1000 images, depending on the number of layers in a model.

To compute distance matrices with different methods, we used an HPC cluster node with 2 AMD Rome CPUs, each with 64 cores at 2.6 GHz and the node has 512 GB of RAM. Distance computation also ranges between around 10 minutes and a couple of hours, depending on the method used. Therefore, model-to-model comparison results take about a day to fully compute for all models and both distances.

Model-to-brain Comparisons. To obtain activations from the neural networks, we used 4 Nvidia V100 SXM2 with 32GB of memory on an HPC cluster node. Features and covariance extraction ranged from 20 minutes to 2 hours (including 5 instances), for 1000 images, depending on the number of layers in the model. The extraction procedure is carried out only once for all subjects, as they all saw the same images. These images are different from the ones used in the model-to-model comparisons.

To run the brain similarity evaluations, we used 50 HPC cluster nodes with a Dual Intel Xeon Skylake CPU, each with 28 cores and 128GB of RAM for each of our two evaluations. Each node ran in parallel for around 9 hours for the Bayesian evaluation and 11 hours for the frequentist evaluation.

A.3 Models and Training Details

Table 2: Trained models with differences in their training recipes, other than different random initializations, Top-1 accuracies on ImageNet1k validation set, and selected layers from each model. Resnet50 instances are trained using four Tesla V100 SXM2 graphics cards, and all other model instances are trained using four NVIDIA Ampere graphics cards. The intermediate layer activations are obtained using TorchLens (ver. 0.1.36) [35].

Model Architecture	Training Variety	Accuracy (Top-1)	Selected Layers
ResNet50 [14]	-	76.01	Last ReLU
	-	76.38	
	Learning Rate and Augmentation Policy	76.52	
	Learning Rate and Augmentation Policy	76.78	
	Pre-trained by PyTorch	80.90	
ViT-B/16 [11]	-	80.73	Layernorms
	-	80.69	
	Augmentation Policy	80.59	
	Augmentation Policy	79.01	
	Pre-trained by PyTorch	81.00	
Swin Transformer Tiny [23]	-	81.45	Layernorms
	-	81.51	
	Augmentation Policy	81.53	
	Augmentation Policy and Interpolation	81.08	
	Pre-trained by PyTorch	81.50	
Swin Transformer V2 Small [24]	-	83.69	Adds
	-	83.53	
	Augmentation Policy	83.45	
	Augmentation Policy and Interpolation	83.22	
	Pre-trained by PyTorch	83.70	
ConvNeXt Tiny [25]	-	81.31	-
	Training Duration	81.27	
	Augmentation Policy	81.44	
	Augmentation Policy and Interpolation	81.12	
	Pre-trained by PyTorch	82.52	
ConvNeXt Tiny Modified	-	81.31	Layernorm
	Training Duration	81.27	
	Augmentation Policy	81.44	
	Augmentation Policy and Interpolation	81.12	
EfficientNet V2 Small [34]	-	83.76	The Add in each block if exists, Batchnorm otherwise
	-	83.65	
	Augmentation Policy	83.42	
	Augmentation Policy and Interpolation	83.35	
	Pre-trained by PyTorch	84.20	

A.4 Layer Group Distances for All Models

Table 3: Mean and standard deviation of self-distances according to JSD and CKA per layer group for each model. The layer groups identified as stable are shown in bold. Rows are ordered by the average of the means from the two metrics.

ResNet50

Layer Group	JSD	CKA
Initial Layers	0.0283 ± 0.0411	0.0208 ± 0.0346
ReLU3	**0.0496 ± 0.0853**	**0.0415 ± 0.0493**
Add1	0.1148 ± 0.1492	0.0868 ± 0.1277
BatchNorm1	0.1210 ± 0.0857	0.0875 ± 0.0710
ReLU1	0.1060 ± 0.0960	0.1175 ± 0.1432
Conv1	0.1340 ± 0.1189	0.1148 ± 0.1231
ReLU2	0.1425 ± 0.1175	0.1261 ± 0.1064
Conv2	0.1760 ± 0.1128	0.1457 ± 0.1267
BatchNorm2	0.1991 ± 0.1343	0.1418 ± 0.1172
BatchNorm3	0.2087 ± 0.1706	0.1893 ± 0.1578
Conv3	0.2032 ± 0.1546	0.1980 ± 0.1655
Last Layers	0.5978 ± 0.1834	0.1703 ± 0.0988

ConvNeXt Tiny

Layer Group	JSD	CKA
Initial Layers	0.0473 ± 0.0611	0.0236 ± 0.0361
Conv1	0.0662 ± 0.0668	0.1451 ± 0.1186
GeLU1	0.1362 ± 0.1201	0.0906 ± 0.1043
Add1	0.1035 ± 0.1057	0.1377 ± 0.0887
LayerNorm1	0.1631 ± 0.1077	0.1239 ± 0.1072
Conv2	0.1643 ± 0.0981	0.1259 ± 0.0965
DConv1	0.2263 ± 0.1420	0.1537 ± 0.1106
Last Layers	0.7479 ± 0.0402	0.2971 ± 0.0303

ViT-B/16

Layer Group	JSD	CKA
Initial Layers	0.0193 ± 0.0204	0.0070 ± 0.0084
LayerNorm1	**0.0642 ± 0.0478**	**0.0352 ± 0.0192**
LayerNorm2	**0.0628 ± 0.0299**	**0.0427 ± 0.0221**
Linear1	0.0200 ± 0.0232	0.0898 ± 0.0726
GeLU1	0.1053 ± 0.0866	0.0897 ± 0.0797
Add1	0.2648 ± 0.2304	0.1377 ± 0.1184
Add2	0.3006 ± 0.2330	0.1573 ± 0.1198
Linear2	0.2869 ± 0.2278	0.2092 ± 0.1648
Last Layers	0.4320 ± 0.3548	0.1654 ± 0.1031
Attention1	0.4214 ± 0.1866	0.2028 ± 0.1245

Swin Transformer V2 Small

Layer Group	JSD	CKA
Initial Layers	0.0184 ± 0.0240	0.0061 ± 0.0071
Linear1	0.0300 ± 0.0276	0.0805 ± 0.0689
Add1	**0.0863 ± 0.1057**	**0.0551 ± 0.0470**
Add2	**0.1036 ± 0.1441**	**0.0654 ± 0.0665**
GeLU1	0.1490 ± 0.1677	0.1056 ± 0.1397
Attention1	0.1053 ± 0.1134	0.1916 ± 0.1383
LayerNorm2	0.1790 ± 0.1857	0.1273 ± 0.1128
LayerNorm1	0.2291 ± 0.1888	0.1293 ± 0.1165
Linear2	0.1693 ± 0.1838	0.2672 ± 0.1941
Last Layers	0.4717 ± 0.2145	0.1363 ± 0.0410

EfficientNet V2 Small

Layer Group	JSD	CKA
Initial Layers	0.0358 ± 0.0436	0.0253 ± 0.0398
BN2_NoSkip	**0.0315 ± 0.0240**	**0.0668 ± 0.0856**
Add1	**0.0646 ± 0.0829**	**0.0578 ± 0.0625**
BN3_NoSkip	**0.0756 ± 0.0465**	**0.0626 ± 0.0609**
SiLU1	0.0874 ± 0.0990	0.0764 ± 0.0883
BN1	0.0828 ± 0.1361	0.0880 ± 0.1136
BN2_Skip	0.0848 ± 0.0498	0.0955 ± 0.0627
Conv1	0.1260 ± 0.1927	0.0910 ± 0.1224
Conv2	0.1139 ± 0.1194	0.1087 ± 0.0958
SqueezeExcit1	0.1296 ± 0.1137	0.1617 ± 0.1286
Conv3	0.2154 ± 0.2167	0.1780 ± 0.1474
SiLU2	0.1866 ± 0.1579	0.2287 ± 0.1671
BN3_Skip	0.4365 ± 0.1442	0.2920 ± 0.1015
Last Layers	0.6463 ± 0.1074	0.3283 ± 0.1638

ConvNeXt Tiny Modified

Layer Group	JSD	CKA
Initial Layers	0.0852 ± 0.1118	0.0391 ± 0.0571
LayerNorm1	**0.1426 ± 0.1627**	**0.0849 ± 0.0999**
Conv1	0.0777 ± 0.0955	0.1540 ± 0.1419
Add1	0.1053 ± 0.1277	0.1273 ± 0.0995
GeLU1	0.1435 ± 0.1682	0.1063 ± 0.1459
Conv2	0.1828 ± 0.1566	0.1422 ± 0.1378
DConv1	0.1899 ± 0.1476	0.1440 ± 0.1317
Last Layers	0.8141 ± 0.0496	0.4065 ± 0.0744

Swin Transformer Tiny

Layer Group	JSD	CKA
Initial Layers	0.0079 ± 0.0075	0.0036 ± 0.0038
Linear1	0.0269 ± 0.0518	0.0405 ± 0.0529
LayerNorm1	**0.0580 ± 0.0451**	**0.0270 ± 0.0192**
LayerNorm2	**0.0656 ± 0.0541**	**0.0324 ± 0.0284**
Add1	0.1169 ± 0.1351	0.0821 ± 0.1169
GeLU1	0.1237 ± 0.1879	0.0812 ± 0.1337
Attention1	0.1483 ± 0.1691	0.0741 ± 0.0657
Add2	0.1687 ± 0.2081	0.1108 ± 0.1464
Linear2	0.2945 ± 0.2953	0.1823 ± 0.1859
Last Layers	0.4877 ± 0.2359	0.1485 ± 0.0494

A.5 Layer Group Distances for All Models - Jaccard

Table 4: Mean and standard deviation of self-distances according to Jaccard similarity per layer group for each model. The layer groups identified as stable are shown in bold.

ResNet50

Layer Group	Jaccard
Initial Layers	0.01766 ± 0.1559
ReLU3	$\mathbf{0.4616 \pm 0.0978}$
Add1	0.5352 ± 0.1324
Last Layers	0.5374 ± 0.0582
Conv1	0.6059 ± 0.1529
ReLU1	0.6166 ± 0.1436
BatchNorm1	0.6301 ± 0.1361
ReLU2	0.6350 ± 0.1450
Conv2	0.6572 ± 0.1405
BatchNorm2	0.6737 ± 0.1270
BatchNorm3	0.7095 ± 0.1532
Conv3	0.7219 ± 0.1337

EfficientNet V2 Small

Layer Group	Jaccard
Initial Layers	0.2297 ± 0.1749
Add1	$\mathbf{0.4722 \pm 0.1548}$
BN3_NoSkip	$\mathbf{0.4832 \pm 0.1023}$
BN2_NoSkip	$\mathbf{0.4956 \pm 0.1297}$
Conv1	0.5524 ± 0.1609
SiLU1	0.5767 ± 0.1724
BN1	0.5807 ± 0.1562
Conv2	0.6723 ± 0.1124
SqueezeExcit1	0.6744 ± 0.1433
Last Layers	0.6776 ± 0.1600
Conv3	0.7129 ± 0.1632
BN2_Skip	0.7263 ± 0.0909
SiLU2	0.7311 ± 0.1538
BN3_Skip	0.8469 ± 0.0413

ConvNeXt Tiny

Layer Group	Jaccard
Initial Layers	0.1889 ± 0.1687
Last Layers	0.5886 ± 0.0452
LayerNorm1	0.6487 ± 0.1602
DConv1	0.6516 ± 0.1277
Add1	0.6608 ± 0.1470
Conv1	0.6832 ± 0.1681
GeLU1	0.7267 ± 0.1247
Conv2	0.7428 ± 0.0907

ConvNeXt Tiny Modified

Layer Group	Jaccard
Initial Layers	0.2381 ± 0.2001
LayerNorm1	$\mathbf{0.5888 \pm 0.1123}$
Last Layers	0.6598 ± 0.0608
Add1	0.6921 ± 0.1120
DConv1	0.6922 ± 0.1266
Conv1	0.7172 ± 0.1555
GeLU1	0.7465 ± 0.1183
Conv2	0.7673 ± 0.0938

ViT-B/16

Layer Group	Jaccard
Initial Layers	0.2046 ± 0.1713
LayerNorm2	$\mathbf{0.5101 \pm 0.0923}$
LayerNorm1	$\mathbf{0.5117 \pm 0.0919}$
Last Layers	0.6243 ± 0.0686
Linear1	0.6297 ± 0.1234
Attention1	0.6600 ± 0.0805
Add1	0.6679 ± 0.2151
Add2	0.6862 ± 0.2230
GeLU1	0.7059 ± 0.1246
Linear2	0.7569 ± 0.1308

Swin Transformer Tiny

Layer Group	Jaccard
Initial Layers	0.1287 ± 0.1089
LayerNorm1	$\mathbf{0.3950 \pm 0.0543}$
LayerNorm2	$\mathbf{0.4362 \pm 0.0696}$
Linear1	0.4514 ± 0.1315
Last Layers	0.4546 ± 0.0202
Add1	0.5042 ± 0.1975
Attention1	0.5411 ± 0.1588
Add2	0.5535 ± 0.2299
GeLU1	0.5702 ± 0.1405
Linear2	0.7113 ± 0.1712

Swin Transformer V2 Small

Layer Group	Jaccard
Initial Layers	0.1588 ± 0.1419
Last Layers	0.4300 ± 0.0146
Add1	$\mathbf{0.5167 \pm 0.1457}$
Add2	$\mathbf{0.5441 \pm 0.1515}$
Linear1	0.6020 ± 0.1564
LayerNorm1	0.6154 ± 0.1476
Attention1	0.7064 ± 0.1800
GeLU1	0.7094 ± 0.1732
LayerNorm2	0.7522 ± 0.1114
Linear2	0.8172 ± 0.1704

A.6 SELF-SIMILARITIES OF ALL MODELS - JSD AND CKA

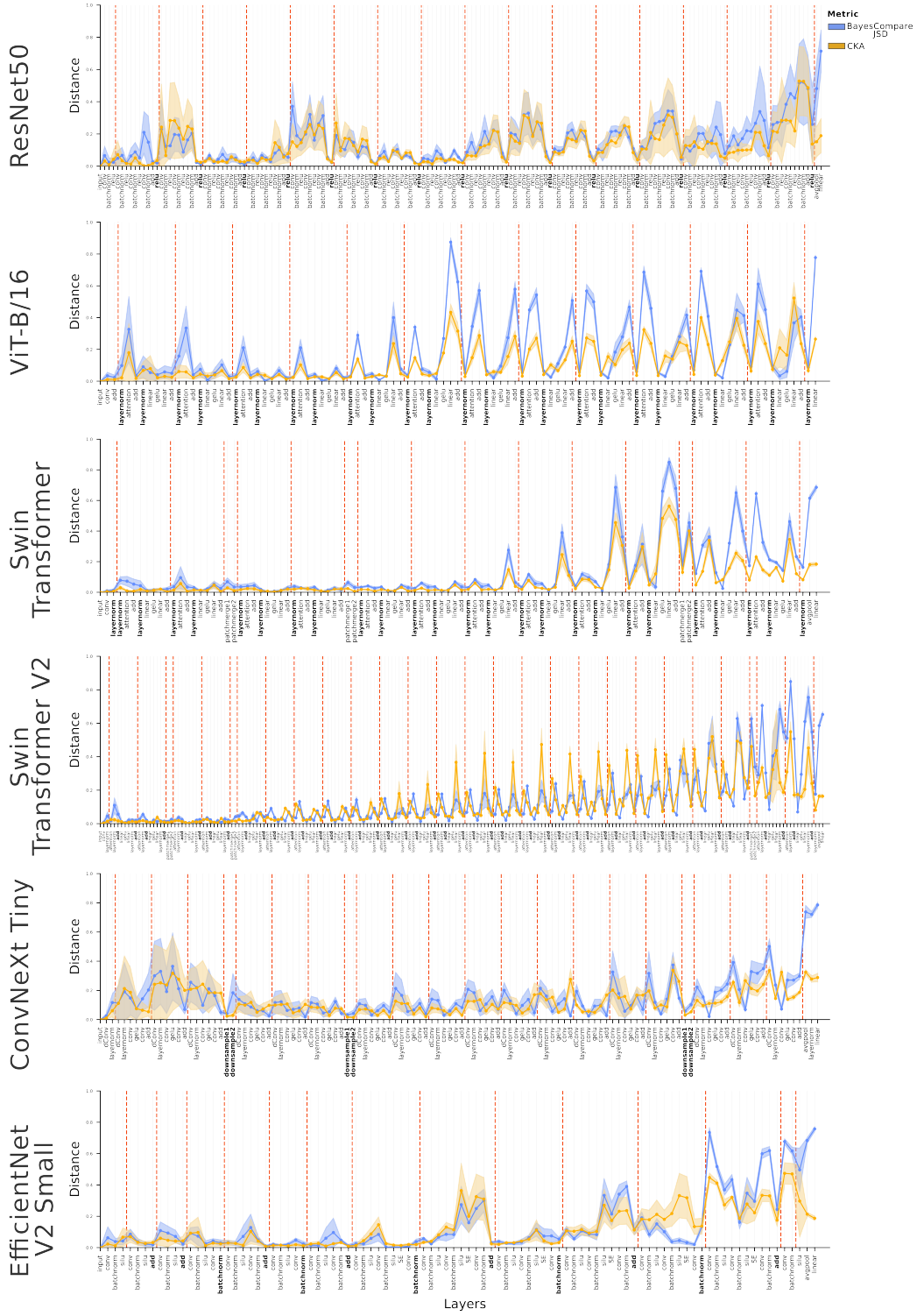

Figure 6: Self-distances of all layers in all model architectures for BayesCompare JSD and CKA. Selected stable layers are indicated with bold font.

A.7 SELF-SIMILARITIES OF ALL MODELS - JACCARD

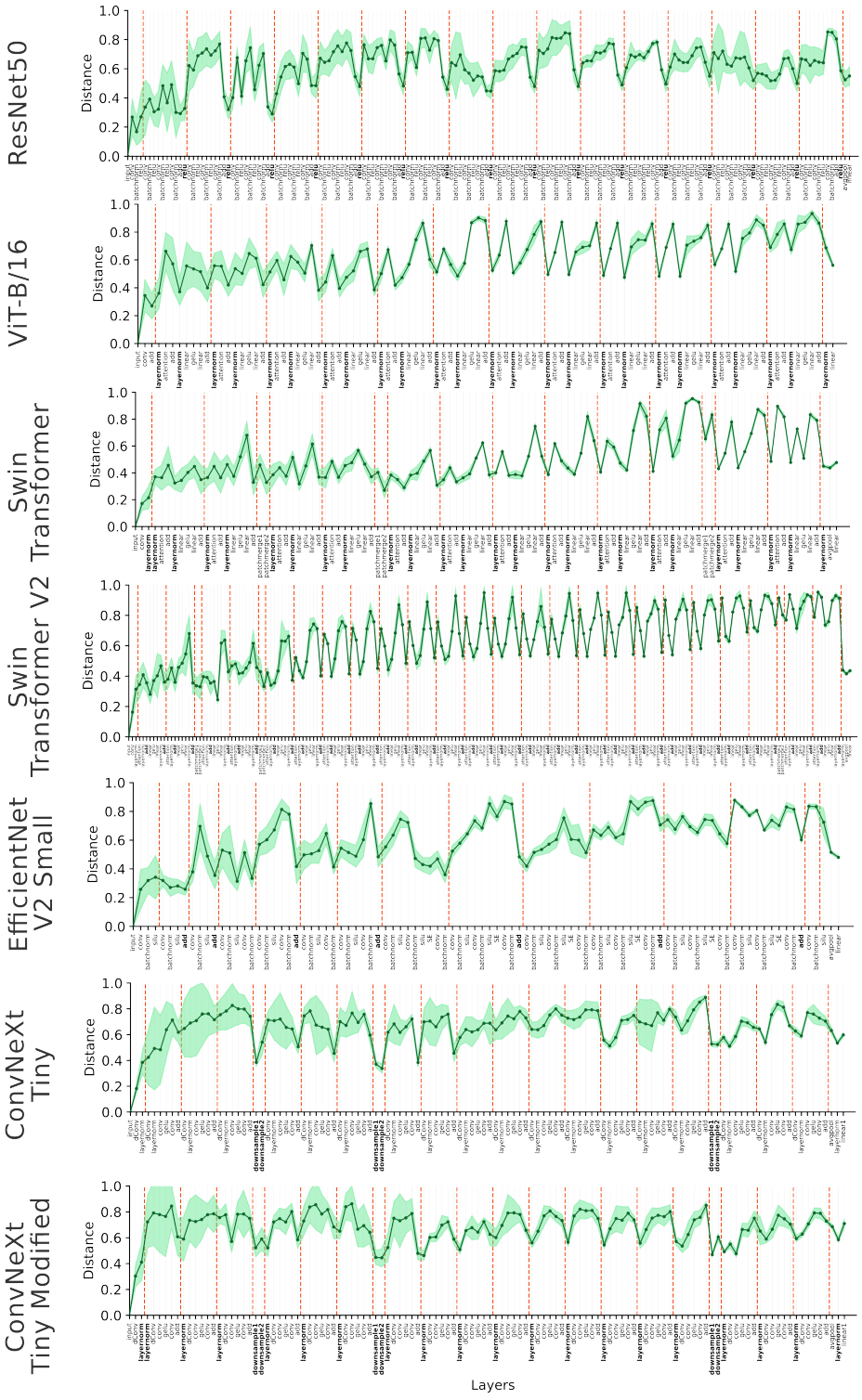

Figure 7: Self-distances of all layers in all model architectures for Jaccard. Selected stable layers are indicated with bold font.

A.8 Self-similarities of Layer Groups for All Models - JSD and CKA

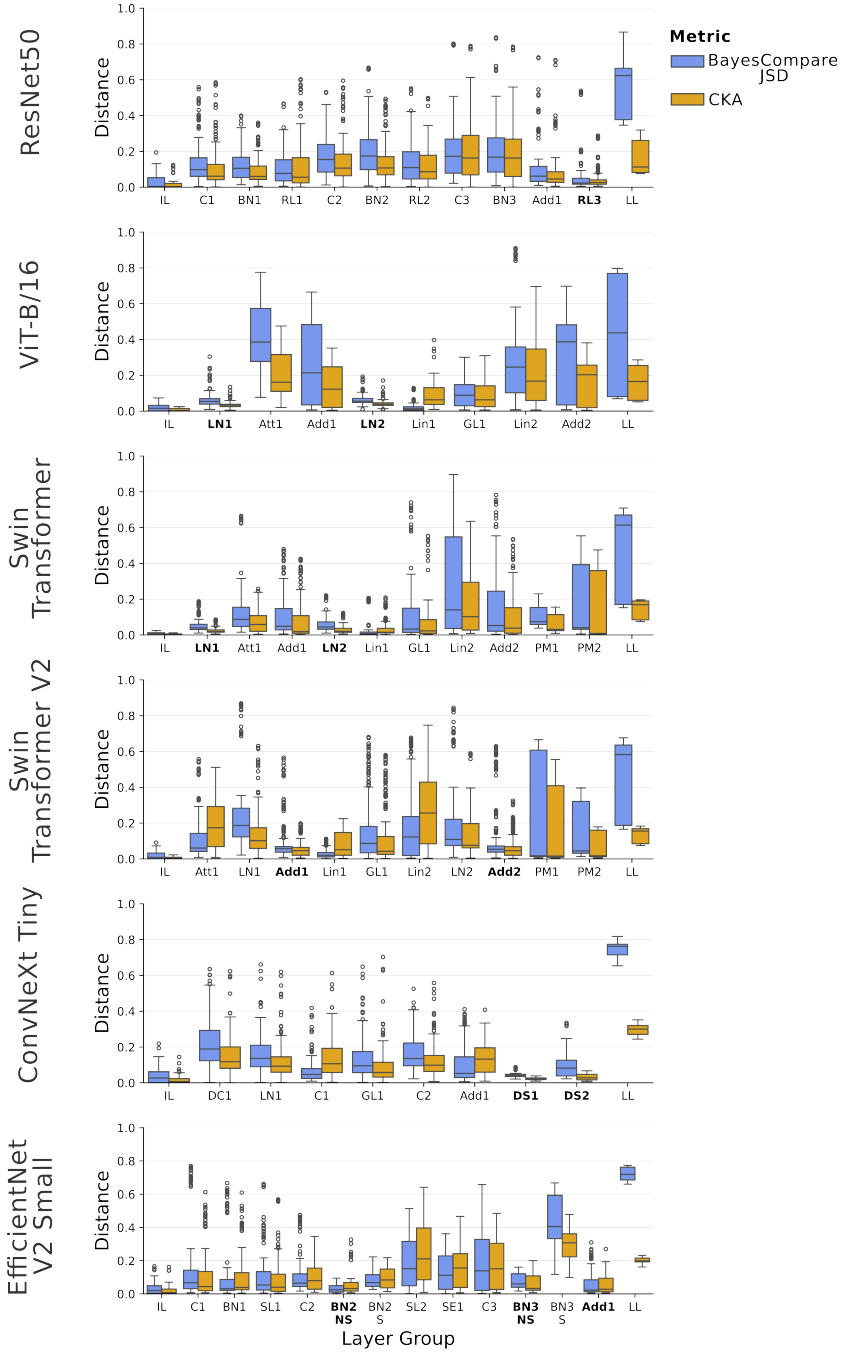

Figure 8: Self-distances of all layer groups in all model architectures for BayesCompare JSD and CKA. Selected stable layers are indicated with bold font. **Abbreviations (alphabetical):** Att: Attention; BN: Batch normalization; BN NS: Batch normalization not skipped; BN S: Batch normalization skipped; C: Convolutional; DS: Downsample; GL: GeLU; IL: Initial layers; Lin: Linear; LL: Last layers; LN: Layer normalization; PM: Patch Merging; RL: ReLU; SE: Squeeze and Excitation; SL: SiLU.

A.9 Self-similarities of Layer Groups for All Models - Jaccard

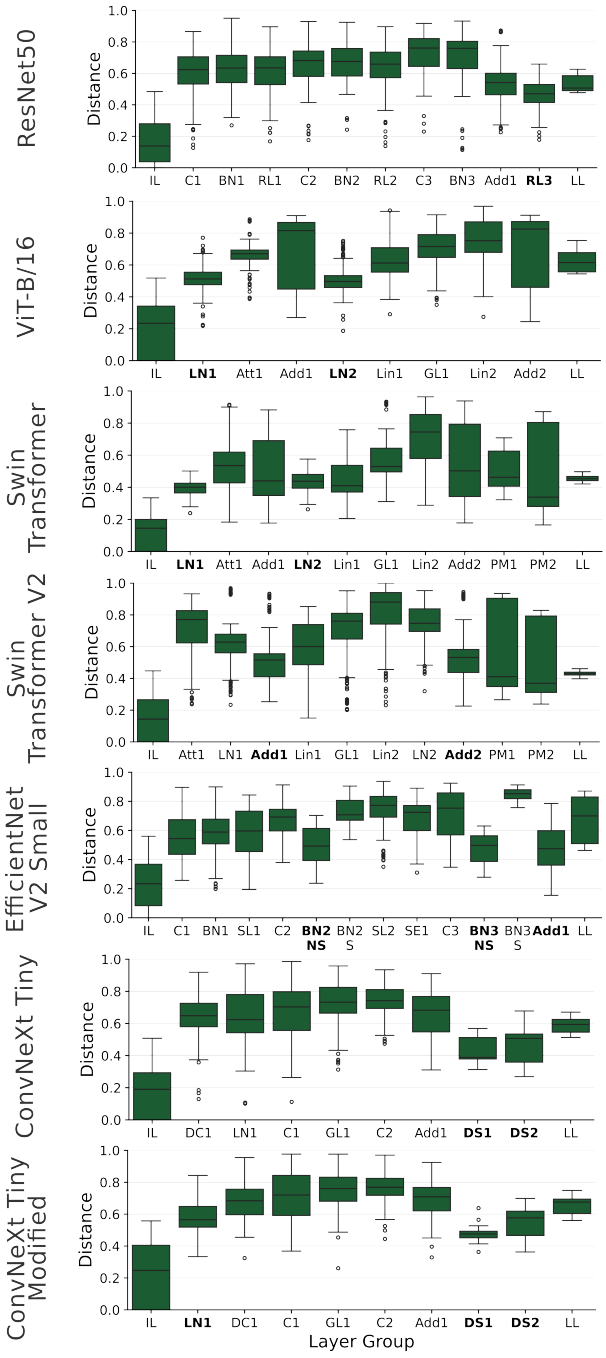

Figure 9: Self-distances of all layer groups in all model architectures for Jaccard. Selected stable layers are indicated with bold font. **Abbreviations (alphabetical):** Att: Attention; BN: Batch normalization; BN NS: Batch normalization not skipped; BN S: Batch normalization skipped; C: Convolutional; DS: Downsample; GL: GeLU; IL: Initial layers; Lin: Linear; LL: Last layers; LN: Layer normalization; PM: Patch Merging; RL: ReLU; SE: Squeeze and Excitation; SL: SiLU.

A.10 MEANS OF ACTIVATIONS OF EXPANSION LAYERS

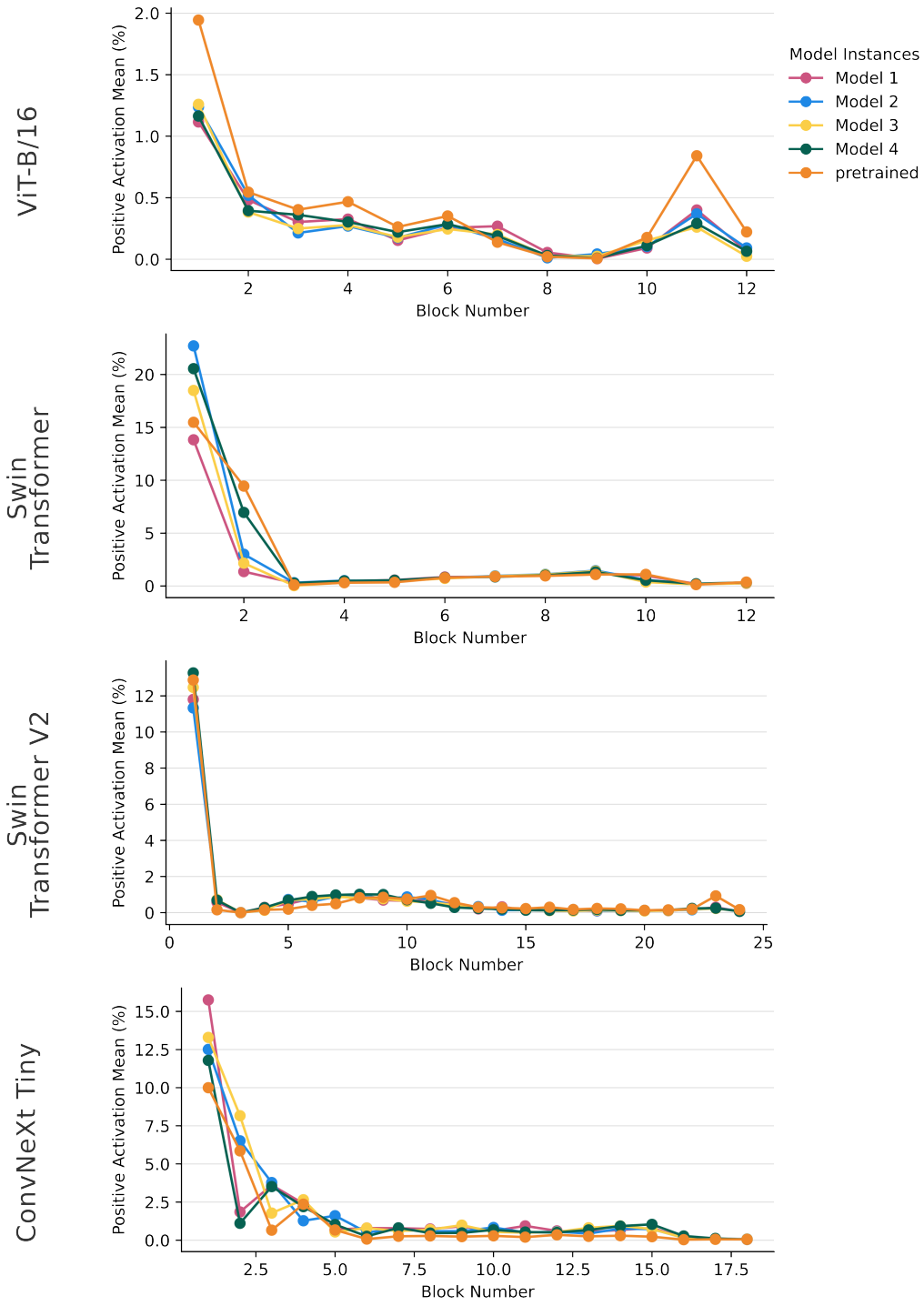

Figure 10: Percentage of positive values in means of activations across stimuli. 'Layer1' layers for the transformer models, and the 'Conv1' layer for the ConvNeXt are shown.

A.11 ConvNeXt and Modified ConvNeXt Comparisons

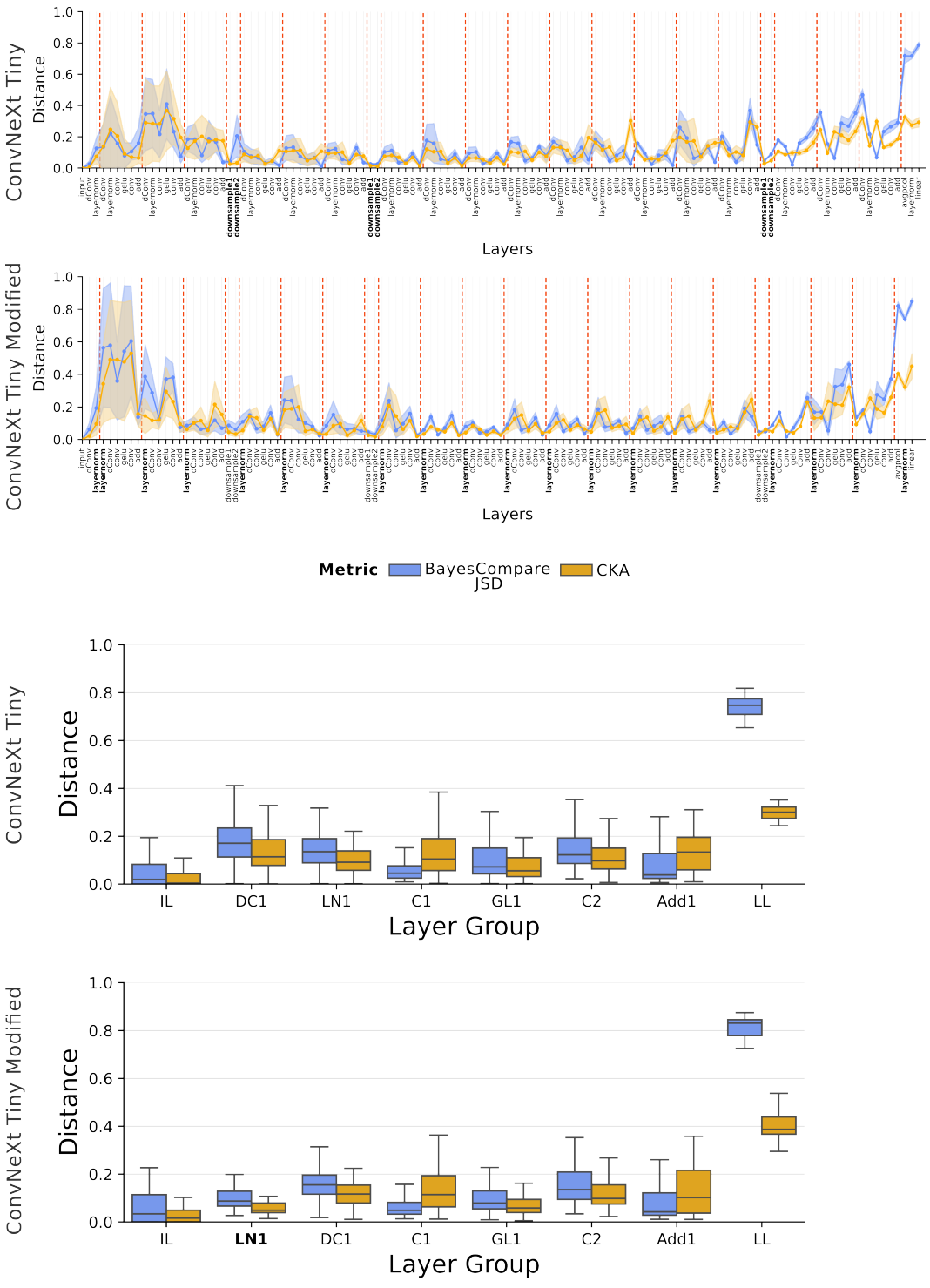

Figure 11: Self-distances of each layer and layer groups for 4 ConvNeXt Tiny and modified ConvNeXt Tiny instances that we trained (PyTorch trained model is excluded from ConvNeXt Tiny averages).

B MODEL-TO-BRAIN EVALUATIONS

B.1 FMRI DATA SELECTION AND PREPARATION

Voxel selection. The NSD dataset contains detailed meta-data that labels each of the voxel from each participant as part of one or more ROIs, including a number of functional localizers. Similarly to [9], we implemented a voxel selection procedure based on noise-ceiling signal-to-noise ratio (NCSNR), which is included in the NSD metadata. We excluded all voxels with NCSNR < 0.2. After filtering voxels based on SNR, we filtered the remaining voxels with an ROI mask comprising a general representation of relevant brain areas. For that, we included all voxels that are on any of the general NSD ROIs (i.e., 'nsdgeneral' as well as the 'stream' ROIs), as well as all the functional localizers (faces, bodies, words and places) and the early visual areas ROI (i.e. 'prf-visualrois'). The number of voxels included for each subject are as follows:

- Subject 01: 16821
- Subject 02: 15714
- Subject 05: 15354
- Subject 07: 11631

General evaluation procedure. We consider separately each layer of each model (independently for each instance) for all voxels in a subject as one evaluation. In each evaluation, the fMRI data of one subject and the features / covariance matrix of the corresponding DNN layer are loaded, and the corresponding evaluation is carried out. The results of each evaluation are saved separately and pooled across subjects/layers/instances at a later stage. The inter-rater agreement was measured using Kendall's tau across all five instances of each network (using the scipy[1] implementation).

B.2 BRAIN SIMILARITY MEASURES

Bayesian evaluation details. One of the brain similarity measures that we employ is a Bayesian linear encoding model proposed in [32]. As explained in 2, we assume a zero-mean multivariate Gaussian prior on the weights and iid. Gaussian noise. This makes the prior predictive distribution depend on the kernel matrix of the model $\mathbf{K_x} = \mathbf{X}\mathbf{X}^T$, and the two parameters σ_β^2 and σ_ϵ^2. The log-likelihood for each voxel under a given layer's covariance matrix is obtained using the Gaussian log-likelihood formula.

Noise estimation. Considering that each stimulus $s_i \in S = [s_1, s_2, ..., s_n]$ is repeated a number of times $r_i \in R$, we can estimate σ_ϵ^2 as the mean variance in each stimulus across its repetitions:

$$\hat{\sigma}_\epsilon^2 = \frac{1}{(\sum r) - n} \sum_{i=1}^{n} \sum_{j=1}^{r_i} (y_{ij} - \bar{y}_i)^2 \tag{8}$$

with $\sum r$ being the total number of presentations in all stimuli, y_{ij} being the j-th presentation of the i-th stimulus, and $\bar{y}_i$ being the mean response of stimulus i. This allows estimating the noise even when the number of repetitions for each stimulus is not the same. We then estimate the sample variance of $\bar{y}$ as the total variance $\hat{\sigma}_{tot}^2$ and the signal variance as $\hat{\sigma}_\beta^2 = \hat{\sigma}_{tot}^2 - \hat{\sigma}_\epsilon^2$.

Marginal log-likelihood. We perform the voxel-wise evaluation with different σ_β^2 and σ_ϵ^2 parameters, sampled from the distribution of noise proportion in the data for each subject. For this, we fit a beta distribution to it such that:

$$\hat{\sigma}_\epsilon^2 / \hat{\sigma}_{tot}^2 \sim \text{Beta}(\alpha_{MLE}, \beta) \tag{9}$$

where we fix $\beta = 1$ and estimate α_{MLE} using maximum-likelihood estimation. Then, we obtain its cumulative distribution function (CDF) and sample m points uniformly across the y-axis. The selected values for σ_ϵ^2 are the values corresponding to the sampled points on the x-axis of the CDF. This makes it so the selected noise values are sampled more closely where the likelihood of the fitted distribution is higher. Then, we sample 100 noise values that are used for our analyses (setting $\hat{\sigma}_\beta^2 = 1 - \hat{\sigma}_\epsilon^2$). This means that, for every voxel and model, we obtain 100 log-likelihoods. We marginalize over this dimension to obtain one log-likelihood per voxel and model

[1] https://scipy.org/

Frequentist evaluation details. For the frequentist linear regression / encoding model, we use a sparse random projection for dimensionality reduction. We use scikit-learn's[2] implementation. With the default parameters, the number of components is controlled by the Johnson-Lindenstrauss lemma. When the resulting number of components would be bigger than those present in the original representation, we do not perform dimensionality reduction. For the evaluation itself, we divide the input images into 5 folds using a K-Fold splitter (without shuffling), and each of the resulting folds is used once as testing data, while the rest is used as training. In each of these folds, the training data is fit using Ridge regression. We optimize the regularization parameter alpha using leave-one-out cross validation, implemented in sklearn as 'RidgeCV' and the '*alpha_per_target*' parameter, which selects a different alpha for each voxel. We tested 100 logarithmically spaced values from $1e-8$ to $1e10$. After obtaining predicted activations for each image and voxel, we compute and save the variance of the predictions and the variance of the errors, which are used to calculate the scores at a later stage.

B.3 Brain similarity agreement of all models

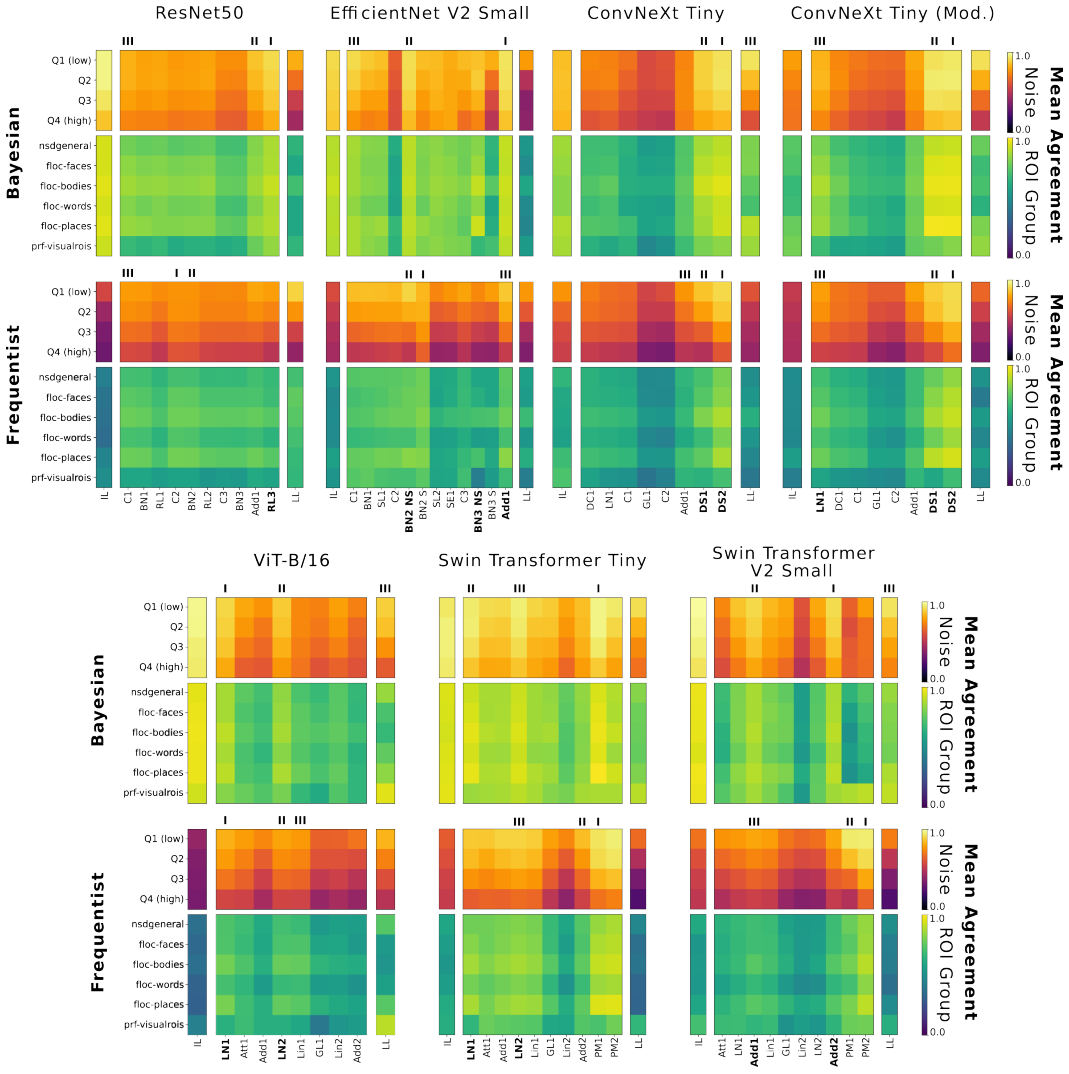

Figure 12: Mean agreement per ROI and noise level including non-repeating layer groups: Downsample (DS1 and DS2) and PatchMerge (PM1 and PM2).

[2]https://scikit-learn.org/

B.4 BRAIN SIMILARITY PERFORMANCE OF ALL MODELS (ALL LAYERS)

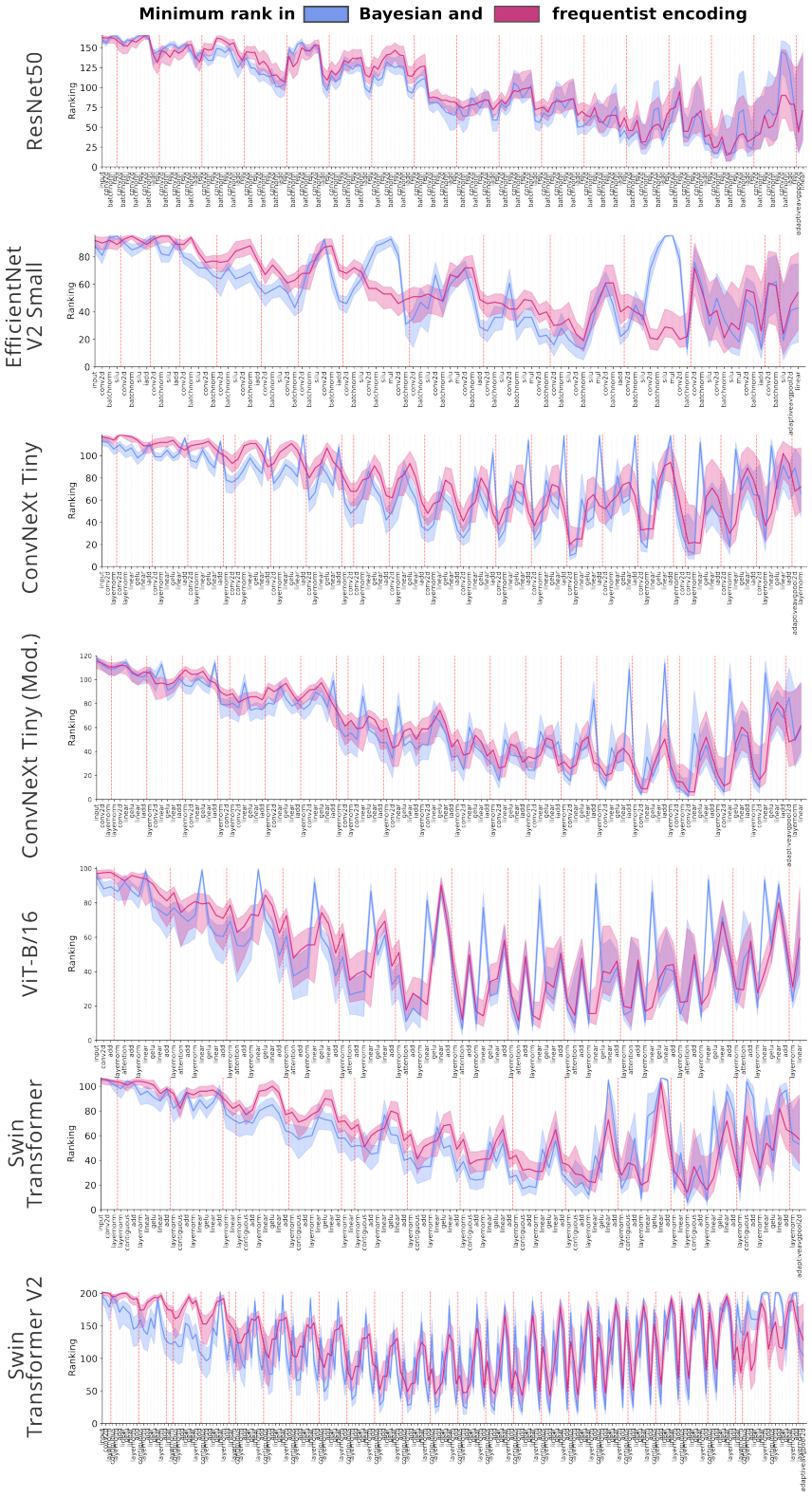

Figure 13: Layer-wise ranking for all architectures not present in the main text.

B.5 BRAIN SIMILARITY PERFORMANCE OF ALL MODELS (LAYER GROUPS)

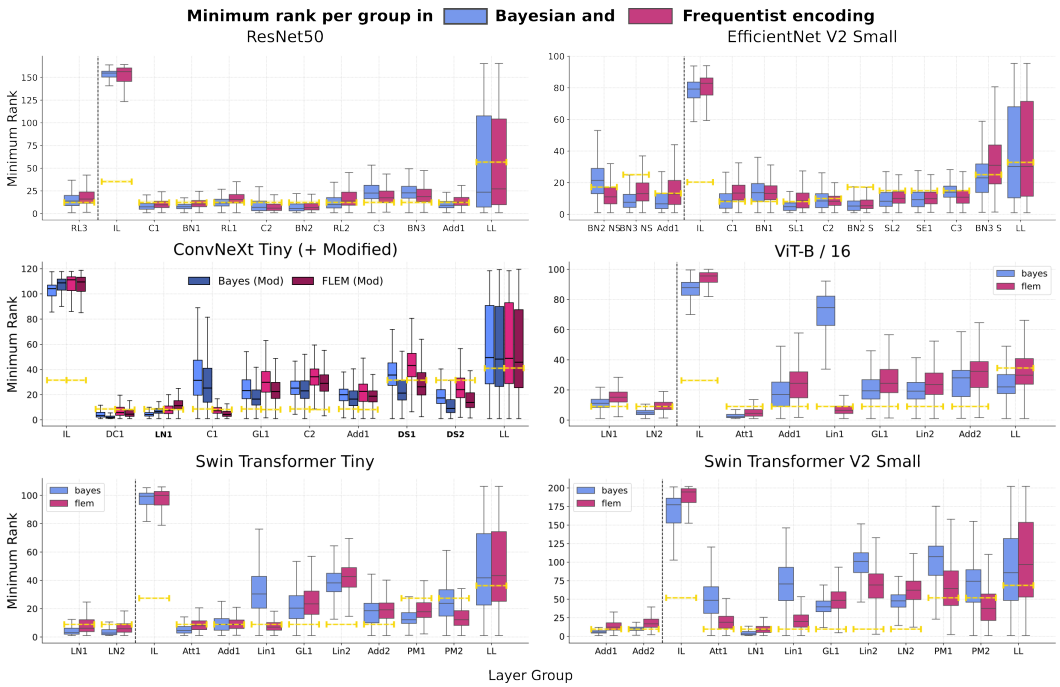

Figure 14: Voxel wise min rankings for all groups in architectures that include non-repeating layer groups. Stable layer groups are arranged on the left side of each figure, with the exception of ConvNeXt, where they are highlighted in bold. There, the group LayerNorm1 (LN1) is only considered stable in the modified ConvNeXt, while the two DownSample groups (DS1 and DS2) would be considered stable if not for the fact that they do not appear in every block in the network