# A DATA-FREE METRIC FOR ESTIMATING THE SIMILARITY OF NEURAL NETWORKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

The information learned by a neural network is encoded entirely in its weights and computation graph, but this information is viewed as practically inaccessible without probing the model's functionality with sample data. For this reason, existing methods for characterizing model similarity draw comparisons between outputs or internal representations, using a shared set of inputs. A weakness of such methods, however, is that evaluating model similarity based on a finite probing set does not guarantee similarity across all possible inputs. Here, we provide evidence that training-relevant information can, in fact, be gleaned from model weights directly and used to compare what models have learned. We present and validate the weight-based Ahmad RV coefficient (wARV): a novel, data-independent metric for quantifying neural network model similarity. We show that wARV captures training-relevant variation across different architectures, weight initializations, and batch ordering. Our results demonstrate the utility of wARV as a general-purpose similarity metric and provide a starting point for further applications leveraging what models learn in a data-agnostic manner.

## 1 INTRODUCTION

A fundamental challenge in modern deep learning is understanding how artificial neural networks (NNs) behave when faced with unpredictable inputs that may fall outside the training distribution. Analyzing the similarity of NN functionality is typically accomplished by comparing output logits or internal representations across a shared set of inputs (Klabunde et al., 2025). Such data-dependent comparisons, however, are not necessarily reliable across a broader set of possible inputs (Davari et al., 2023) (see also Appendix Fig. 6). To eliminate dependence on external probing data, it would be desirable to compare NNs using only their parameters and computation graphs.

There are two primary obstacles to this objective. First, many model architectures and components are in common use, including multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), and transformers, with or without skip connections, batch and layer normalization, and max pooling (LeCun et al., 2015; Dosovitskiy et al., 2021). There is no obvious correspondence between weights across different architectures. Second, even for identical architectures, it is not straightforward to obtain a 1:1 weight mapping between models, as neurons in a hidden layer can be reordered without changing the network outputs (Godfrey et al., 2022). Permutation symmetry makes the problem of optimal weight matching NP-hard (Ainsworth et al., 2023). The central challenge is to construct a representation in which otherwise incommensurable neural networks become directly comparable.

In this work, we overcome these obstacles by mapping heterogeneous NNs into a shared input-to-output operator space derived from their trained states. We term this mapping the *chain weight matrix*. Our work extends the chained normalization operator of Wang et al. (2022), which is invariant to neuron permutation, to a range of architectures. We compare chain weight matrices using the Ahmad RV coefficient (Ahmad, 2019), yielding a data-independent method for quantifying NN similarity, which we term the *weight-based Ahmad RV coefficient* (wARV). We show that wARV robustly groups models together by training task across diverse architectures and different random states.

Our contributions are summarized as follows:



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.

- We introduce a common comparison space for neural networks of distinct architectures, constructed from a model's parameters and computation graph, and invariant to key weight-space reparameterizations.
- On top of this representation, we define wARV, a bias-corrected, data-independent similarity measure with an associated statistical significance test.
- We perform a systematic evaluation of wARV across image classification tasks, model architectures, standard and adversarial training, training epochs, random initializations, and training batch orderings. We show that wARV tracks with functional similarity, and that the task trained is the primary source of variation in wARV values.

## 2 Related Work

**Data-Dependent Similarity.** Current approaches to NN similarity compare model outputs or internal representations on a chosen set of inputs, corresponding to functional and representational similarity, respectively (Klabunde et al., 2025). Despite the valuable insights provided by data-dependent similarity metrics, there remains a lack of consensus on interpreting estimations from different metrics. For example, Davari et al. (2023) characterized in detail the sensitivity of centered kernel alignment (CKA) (Kornblith et al., 2019) to data transformations that do not lead to functional changes for neural networks. These findings motivate a complementary perspective based on model weights.

**Weight-Based Similarity.** Comparing NNs in parameter space is challenging, and research on weight similarity remains relatively limited and fragmented. Wang et al. (2022) explored weight similarity by proposing a chained normalization operator invariant to hidden neuron permutation. A related line of work tries to learn a "hyper-representation" of "model zoos" via an auxiliary representation model such as a sequential autoencoder (Schürholt et al., 2024). More recently, Min & Wang (2025) compared layer weight matrices in large language models via the distribution of cosine similarity, suggesting patterns of layer redundancy and modularity. These studies reflect increasing interest in exploiting model weights to understand the behavior and characteristics of NNs.

**Weight Space Symmetries.** A core challenge in comparing NNs is parameter non-identifiability – different parameter values can define the same underlying function. For example, permuting hidden neurons together with their incoming and outgoing weights alters weight matrices and internal representations while preserving the model function. Similarity measures intended to reflect learned functions should therefore account for such symmetry. However, which additional transformations should leave similarity unchanged remains a matter of debate (Kornblith et al., 2019; Klabunde et al., 2025; Min & Wang, 2025). Godfrey et al. (2022) formalize weight space symmetry as "intertwiner groups" – linear transforms that can be pushed through the activation function up to another linear transform, including hidden neuron permutation and isotropic scaling for ReLU activations. These works highlight the need for explicit invariance design when comparing NNs.

## 3 Methodology

### 3.1 Problem Formulation

Consider two NN models $f_i : \mathbb{R}^{p_i} \to \mathbb{R}^q$, $i \in \{1, 2\}$, where $p_i$ and $q$ represent input and output dimension, respectively. Let the weights of $f_i$ be parameterized as $\boldsymbol{\theta}_i \in \mathbb{R}^{\nu_i}$. $f_1$ and $f_2$ may differ in classification task, architecture, training method, and initialization. Given that $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are incommensurable due to weight space symmetry and possible architectural differences, we formulate a mapping $\boldsymbol{\theta}_i \to \boldsymbol{M}_i \in \mathbb{R}^{q \times p_i}$, under which comparisons can be made.

### 3.2 Generating Mappings

Consider a feedforward NN with weight matrices $\boldsymbol{W}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$, where $d_\ell$ represents layer dimension, $\ell \in \{1, \cdots, L\}$. The chain product of each layer's weight matrix in order, omitting nonlinear activations and bias terms, yields the *chain weight matrix*:

$$\boldsymbol{M} = \boldsymbol{W}_L \boldsymbol{W}_{L-1} \cdots \boldsymbol{W}_1 \in \mathbb{R}^{d_L \times d_0} \tag{1}$$

Note that $\boldsymbol{M}^\top \boldsymbol{M}$ is the chain normalization operator of Wang et al. (2022).

We incorporate architectural variations into the framework of chain weight matrices by representing each component as an effective linear operator and composing them according to the computation graph. We represent convolutions as sparse block Toeplitz matrices (Hansen, 2002), batch normalization (Ioffe & Szegedy, 2015) as channel-wise scaling by its stored statistics, and max pooling (LeCun et al., 1998) as a summation. We approximate scaled dot product attention (Vaswani et al., 2017) by combining a uniform, input-independent attention matrix with the value and output projections. For activation functions for which pre-activation statistics can be readily obtained, such as when directly following batch normalization, we derive their least-squares affine slopes under Gaussian pre-activations. We represent residual blocks as $\boldsymbol{R}_j = \widehat{\boldsymbol{F}}_j + \boldsymbol{S}_j$, where $\widehat{\boldsymbol{F}}_j$ and $\boldsymbol{S}_j$ correspond to scaled main and shortcut (i.e., identity or downsampling) operators respectively. Details for these and additional architectural features are provided in Appendix Section B.

**Intertwiner group invariance.** Godfrey et al. (2022) formalize the concept of the intertwiner group $G_\sigma$ as invertible linear transformations that can be pushed through a non-linear activation via another linear transform, i.e., $\sigma \circ \boldsymbol{A} = \phi(\boldsymbol{A}) \circ \sigma$ for $\boldsymbol{A} \in G_\sigma$. In particular, their Proposition 3.4 formalizes function-preserving reparameterization as

$$\boldsymbol{W}' = (\boldsymbol{A}_1 \boldsymbol{W}_1, \cdots, \boldsymbol{A}_k \boldsymbol{W}_k \phi_\sigma(\boldsymbol{A}_{k-1}^{-1}), \cdots, \boldsymbol{W}_L \phi_\sigma(\boldsymbol{A}_{L-1}^{-1}))$$

The chain weight matrix after such reparameterization is

$$\boldsymbol{M}' = \boldsymbol{W}_L \phi_\sigma(\boldsymbol{A}_{L-1}^{-1}) \boldsymbol{A}_{L-1} \boldsymbol{W}_{L-1} \phi_\sigma(\boldsymbol{A}_{L-2}^{-1}) \cdots \boldsymbol{A}_1 \boldsymbol{W}_1$$

Therefore, $\phi(\boldsymbol{A}^{-1}) = \boldsymbol{A}^{-1} \implies \boldsymbol{M}' = \boldsymbol{M}$, i.e., the chain weight matrix is invariant to the intertwiner group. This is satisfied for ReLU, leaky ReLU, and GELU activations (see proof in Appendix Section C).

### 3.3 Comparing Mappings

The RV coefficient, also known as linear CKA (Kornblith et al., 2019), is a generalization of the squared Pearson coefficient that compares samples measured on two sets of variables (Escoufier, 1973). Let $\boldsymbol{X}_i \in \mathbb{R}^{n \times d_i},\ i \in \{1, 2\}$, denote matrices containing $d_i$ features taken on a shared set of $n$ observations. Take the centering matrix $\boldsymbol{C} = \boldsymbol{I}_n - \frac{1}{n}\mathbf{1}_\mathbf{n}\mathbf{1}_\mathbf{n}^\top$, and let $\tilde{\boldsymbol{X}}_i = \boldsymbol{C}\boldsymbol{X}_i$.

Ahmad (2019) provides an unbiased estimator of $\|\boldsymbol{\Sigma}_{ij}\|_F^2$, where $\|\cdot\|_F$ indicates the Frobenius norm and $\boldsymbol{\Sigma}_{ij}$ is the sample covariance or cross-covariance between the sets of variables in $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$. This estimator is used to compute Ahmad's RV coefficient (ARV):

$$\widehat{\|\boldsymbol{\Sigma}_{ij}\|_F^2} = \frac{1}{n-3}\left[\frac{\operatorname{tr}(\boldsymbol{G}_i\boldsymbol{G}_j)}{n} + \frac{\operatorname{tr}(\boldsymbol{G}_i)\operatorname{tr}(\boldsymbol{G}_j)}{n(n-1)(n-2)} - \frac{\operatorname{tr}(\boldsymbol{G}_i \odot \boldsymbol{G}_j)}{(n-2)}\right] \tag{2}$$

$$\operatorname{ARV}(\boldsymbol{X}_1, \boldsymbol{X}_2) = \frac{\widehat{\|\boldsymbol{\Sigma}_{12}\|_F^2}}{\sqrt{\widehat{\|\boldsymbol{\Sigma}_{11}\|_F^2}\,\widehat{\|\boldsymbol{\Sigma}_{22}\|_F^2}}} \tag{3}$$

where $\boldsymbol{G}_i = \tilde{\boldsymbol{X}}_i\tilde{\boldsymbol{X}}_i^\top$ represents Gram matrices formed from the column-centered inputs, $\operatorname{tr}(\cdot)$ indicates the trace, and $\odot$ is the element-wise Hadamard product.

ARV has values in $[-1, 1]$ and an expected value of 0 under the null hypothesis of orthogonality between centered columns of the input matrices. ARV is invariant to both isotropic scaling and orthogonal transformation, eliminating the possibility that the comparison is affected by class ordering. Hyodo et al. (2026) show that under the null hypothesis, ARV converges in distribution to a sum of i.i.d. chi-squared distributions, weighted by the eigenvalues of $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$. We use characteristic functions to implement a one-tailed significance test for the hypotheses $H_0 : \operatorname{ARV} = 0$ vs. $H_1 : \operatorname{ARV} > 0$. Details can be found in Appendix Section A.

### 3.4 wARV

We propose comparing NNs $f_1$, $f_2$ by first mapping their parameters to chain weight matrices as described above, $\boldsymbol{\theta}_i \rightarrow \boldsymbol{M}_i$, then evaluating the similarity of these matrices using ARV. We term this result the weight-based Ahmad RV coefficient:

$$\text{wARV}(f_1, f_2) = \text{ARV}(\boldsymbol{M}_1, \boldsymbol{M}_2) \tag{4}$$

This construction requires the NNs to have the same output dimension, but it poses no constraints on the input or hidden dimensions.

## 4 Experiments

We evaluate wARV on its ability to identify expected training-related structure across NNs and its reflection of model functionality. To do so, we conduct a systematic comparison of image classification models across all combinations of six factors: training task, model architecture, training method, training progress, initialization, and training batch ordering (Fig. 1a).

We train models from scratch on one of five tasks: MNIST, EMNIST Digits, CIFAR10, and two ImageNet tasks corresponding to the CIFAR10 classes (C10-ImgNet, a subset of CINIC10 (Darlow et al., 2018)) or to distinct categories (Not-C10). Each task set consists of 50,000 training images and 10,000 testing images, equally divided among 10 classes, and with images scaled to $32 \times 32$ pixels in RGB color. We consider five architecture families – MLP, Bottleneck MLP (BMLP) (Bachmann et al., 2023), LeNet (LeCun et al., 1998), ResNet (He et al., 2016), and Vision Transformer (ViT) (Dosovitskiy et al., 2021) – each with approximately one-million- and eleven-million-parameter variants. As a point of reference, the 11M-parameter ResNet model is equivalent to ResNet-18

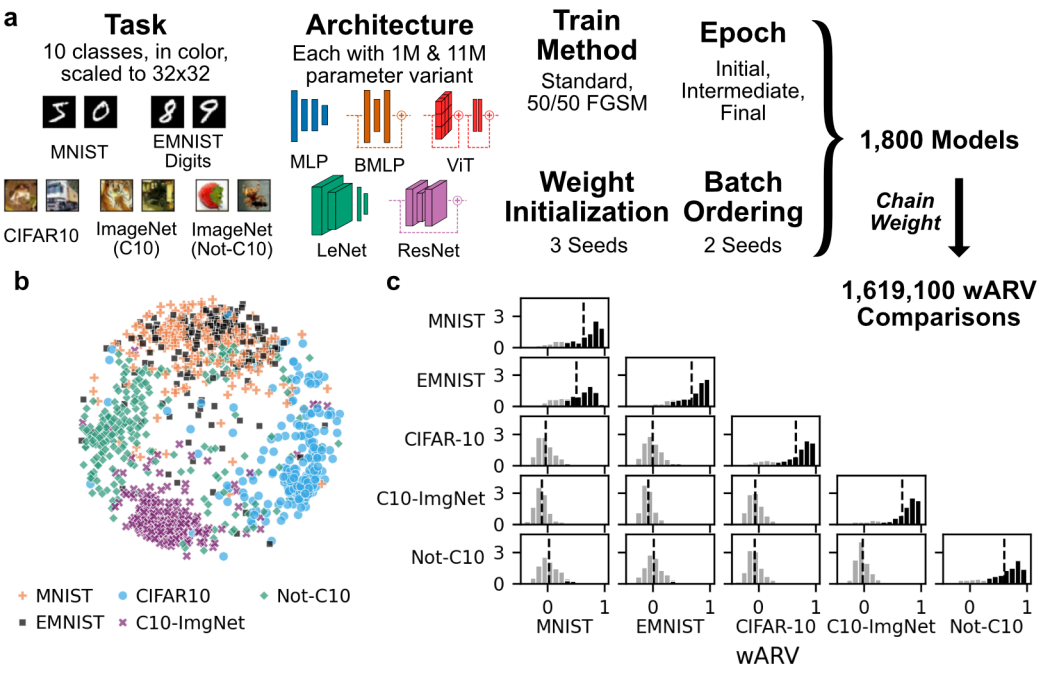


Figure 1: **wARV differentiates models by training task.** (a) Experimental design diagram. (b) Multidimensional scaling embedding, with points colored by training task and distances between points reflecting wARV values ($N = 1{,}200$ trained models, embedded via 719,400 wARV comparisons). (c) Probability density function histograms of wARV values comparing trained models by task learned. Architecture, training method, and training stage are held constant across comparisons. Gray bars: $\text{wARV} = 0$ with $P \geq 0.05$. Black bars: $P < 0.05$. Dashed line indicates the mean. $N = 600$ (along diagonal), 1,440 (off-diagonals).

adapted for CIFAR10-like problems. We train the NNs with either clean samples or a 50/50 mixture of clean and FGSM-perturbed adversarial samples Goodfellow et al. (2015).

For each configuration of task, architecture, and training method, we train from three different weight initialization seeds, and for each initialization we train using two different training sample batch orders. We track training progress by monitoring the loss on the unseen test set; specifically, we estimate the time constant ($\tau$, for $ne^{-t/\tau}$, where $n$ is the starting loss and $t$ is the training epoch) of the test loss trajectory and terminate training when test loss has not improved for $2\tau$ epochs. This ensures comparable convergence across models that train at significantly different rates. After termination, we retain the checkpoints from initialization, the best test loss, and an intermediate point corresponding to $3\tau$ epochs (95% convergence).

Training all combinations of these factors results in 1,800 models (including initializations), for which 1,619,100 pairwise comparisons can be made. For each pair, we computed wARV and the corresponding p-value against the null hypothesis. As a reference standard, we also computed the cosine similarity of logit values for a probing set of 1,000 samples, equally divided among the five training sets.

### 4.1 Task is the dominant source of variation in wARV measurements

An unsupervised multidimensional scaling (MDS) embedding of 719,400 wARV comparisons between the 1,200 trained models (excluding initializations) reveals visually apparent clustering according to training task (Fig. 1b). Holding non-random factors constant and plotting comparisons between pairs of tasks, wARV values tend towards 1.0 and significance at $p < 0.05$ for models trained on the same task; for models trained on distinct tasks, wARV values tend towards 0.0 and non-significance (Fig. 1c). wARV identifies MNIST and EMNIST digits as similar tasks, while distinguishing CIFAR10 and C10-ImgNet – which, though related, represent a significant distributional shift (Darlow et al., 2018).

We confirm the observed ability of wARV to differentiate models by training task via PERMANOVA (Anderson, 2001) (Table 1). Note that the untrained initializations are excluded from this analysis to prevent duplicates from breaking independence requirements. The identity of the trained task explains over six times more variation in wARV than architecture, suggesting that wARV is substantially more sensitive to what a model has learned than to how that knowledge is architecturally represented. Training progress (epoch) and training method also account for small but significant variation, while random initialization and batch ordering are not significant at the $p < 0.01$ level.

Compared to cosine similarity of logits, these explainable factors account for less total variation of wARV values than of cosine similarity values (Table 1). Nevertheless, the relative effect sizes match, confirming that wARV is capturing functionally significant variation without the use of probing data. Furthermore, wARV appears to suppress nuisance variation arising from stochastic training factors, suggesting the chain weight representation can filter out run-specific variability while preserving training-related structure. While a proportion of the observed wARV variance remains unexplained by the given factors, loss of some information is expected, as generating the chain weight matrix involves linear approximations of nonlinear components and extensive compression of NN weights.

Table 1: PERMANOVA results for similarity values of trained models

| | **wARV** | | **Cos Similarity** | |
|---|---|---|---|---|
| **Source** | $\mathbf{R^2}$ | **p** | $\mathbf{R^2}$ | **p** |
| Task | 0.423 | 0.001 | 0.617 | 0.001 |
| Architecture | 0.066 | 0.001 | 0.111 | 0.001 |
| Train Method | 0.002 | 0.001 | 0.001 | 0.001 |
| Epoch | 0.002 | 0.002 | 0.004 | 0.001 |
| Batch Ordering | 0.001 | 0.047 | 0.0002 | 0.451 |
| Weight Initialization | 0.001 | 0.193 | 0.001 | 0.004 |
| *Residual* | 0.505 | | 0.265 | |

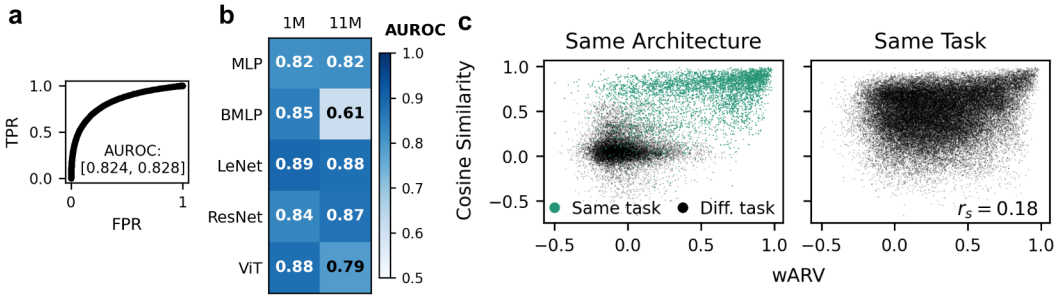


Figure 2: **wARV identifies task-specific variation across model architectures.** (a) Receiver operating characteristic (ROC) curve showing false positive vs. true positive rates for using wARV to identify if models of different architectures were trained on the same task ($N = 162{,}000$). 95% confidence interval is indicated. (b) Areas under the ROC curve (AUROC) for cross-architectural comparisons between the models of each indicated architecture and models with different architectures ($N = 32{,}400$ each). (c) Scatter plots comparing wARV to mean cosine similarity of logits, based on a 1000-item probing set containing data from all five tasks ($N = 17{,}700$ left, $N = 50{,}100$ right). Spearman correlation is indicated in the lower right for comparisons between models of any architecture trained on the same task.

### 4.2 wARV identifies training-specific variation across architectures

A notable characteristic of wARV is its cross-architectural compatibility. wARV readily detects if models of different architectures were trained on the same task, evidenced by an area under the receiver operating characteristic curve (AUROC) of 0.83 (Fig. 2a). Note that here and after, we consider MNIST and EMNIST digits as the same task. Given that there exist true architecture-specific differences in learning capacity and feature extraction capability, perfect classification on this experiment is not achievable, so this AUROC value indicates highly non-random discrimination capability.

Investigating further this characteristic of wARV, we next consider cross-architectural task recognition AUROCs for subsets of models. Comparisons involving a LeNet or ResNet model have higher AUROCs than average, while comparisons involving 11M BMLPs tend to have lower discriminative capability (Fig. 2b). We speculate that this stems from a combination of two effects. First, convolutional architectures may truly exhibit more robust training, so that what they learn is more comparable on average to what is learned in independent training runs. Secondly, these results in part reflect limitations of wARV. Specifically, our implementation of the chain weight matrix for ViT disregards the query and key matrices in the self-attention blocks, and the linear approximations made for BMLP residual blocks are less accurate than for ResNet residual blocks. Still, the detection of non-random, data-independent, task-specific similarity in chain weight matrices generated across markedly different architectures is a notable finding.

wARV tracks more strongly with cosine similarity for comparisons between models of the same architecture, but maintains a weak Spearman correlation of 0.18 for comparisons between models trained on the same task, regardless of architecture (Fig. 2c). These points highlight that wARV can provide information about functional similarity in a data-independent way, even for different architectures. Because model similarity lacks an accepted ground truth, wARV comparisons should be interpreted carefully and in conjunction with other methods.

### 4.3 wARV reflects additional characteristics of model training

We investigate the extent to which wARV can identify relevant model differences throughout the course of training. Because our analysis to this point only compares models at ∼95% convergence to final models, the contribution of training epoch to wARV variation has been small (Table 1). Examining training trajectories in more detail reveals a strong correspondence between wARV and training stage. We compute wARV for publicly available BMLP checkpoints pretrained on ImageNet-21k (Bachmann et al., 2023). The model architecture consists of 12 blocks, each with

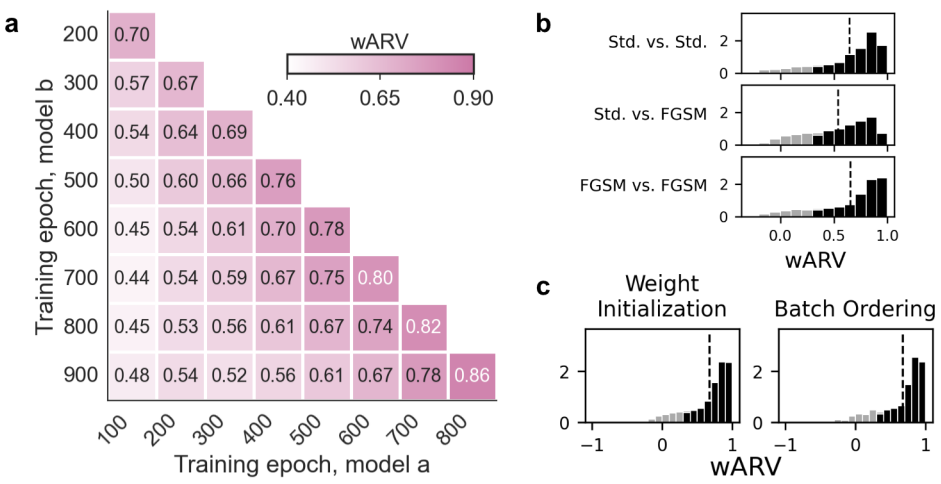


Figure 3: **wARV captures model variations by epoch and training method, while varying minimally across random states.** (a) Heatmap of wARV between publicly available BMLP model checkpoints pretrained on ImageNet-21k. (b) Probability density histograms of wARV values comparing trained models by training method ($N$ = 1,500, 2,100, 1,500). Task, architecture, and training stage are held constant across comparisons. (c) Probability density histograms of wARV values comparing trained models by random states ($N$ = 1,200 left, 400 right). Task, architecture, training method, training stage, and other random states are held constant across comparisons. (b-c) Gray bars: $\text{wARV} = 0$ with $P \geq 0.05$. Black bars: $P < 0.05$. Dashed line indicates the mean.

input dimension 1,024 and an expansion layer of size 4,096. wARV recapitulates expected features of training: nearby checkpoints are more similar to each other than checkpoints further away, and later checkpoints are more similar to each other than earlier ones as training converges (Fig. 3a).

Adversarial training is a popular method for improving robustness by exposing a model to samples with imperceptible perturbations that can confound correct classification (Szegedy et al., 2013; Goodfellow et al., 2015). Research suggests an association between adversarial robustness and minima flatness on the loss landscape (Stutz et al., 2021). However, it remains unclear the extent to which adversarial training results in convergence to a different basin on the loss landscape. Our analysis reveals a small but significant difference in models trained with standard, non-augmented data and FGSM-augmented data (Table 1). wARV identifies models trained using the same method as slightly more similar on average than models trained using different methods (Fig. 3b).

In contrast, comparisons between models differing only in random initialization or training batch ordering tend to produce significant wARV values close to 1.0, suggesting strong similarity (Fig. 3c). Together, these results show that wARV captures subtle variation introduced by adversarial training and training progress, while being less sensitive to the effects of stochastic training factors.

### 4.4 wARV VARIES GRADUALLY WITH INCREMENTAL CHANGES IN TRAINING TASK

Beyond discriminating between models trained on entirely non-overlapping datasets, wARV can also detect partial changes in what models have learned. We train LeNet and MLP architectures on various 10-class problems consisting of ImageNet32 data. We start with a dataset matching the classes of CIFAR10, then gradually replace one CIFAR10 class at a time with a new class of images (fruit, spiders, etc.) until reaching a dataset with entirely distinct classes ("Not-C10"). We train 10 models of each architecture from scratch on each of these 11 datasets. wARV varies in accordance with the incremental transition in the task learned; however, similarity transitions gradually in LeNets, while declining rapidly towards 0 for the MLPs (Fig. 4a). This result provides evidence that training is more robust in convolutional architectures than in MLPs, and further demonstrates the strong ability of wARV to differentiate models by what they have learned.

By contrast, wARV does not vary substantially with gradual changes to model architecture. We train models from scratch on CIFAR10, incrementally toggling the skip connections to transition between

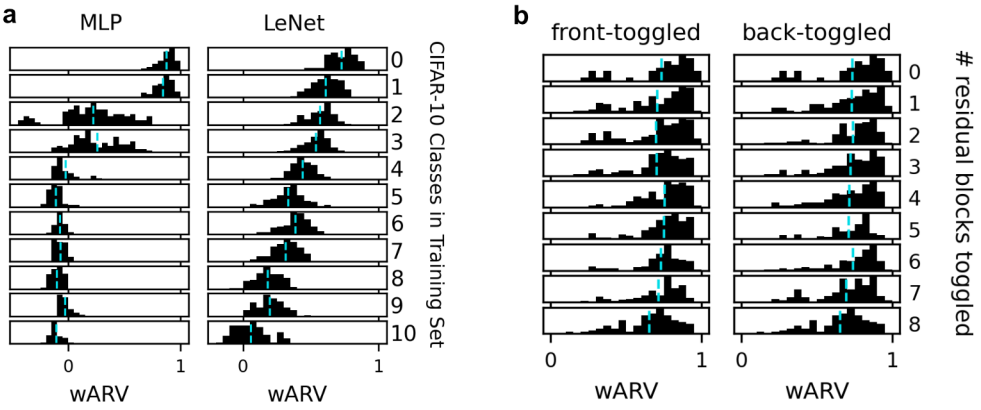


Figure 4: **wARV highlights gradual changes in model similarity with shifting tasks and architectures.** (a) Histograms of wARV values comparing 10-class models trained on ImageNet32 data with varying proportions of CIFAR10 and not-CIFAR10 classes (ranging from 0:10 to 10:0). Comparisons are made relative to the models trained entirely on not-CIFAR10 data ($N = 45$ for top panels, $N = 100$ elsewhere). (b) Histograms of wARV values comparing PlainNet18 (same as ResNet18 but without residual connections) with hybrid PlainNet-ResNet models having gradually more residual connections toggled on in forward or backward directions. $k = 0$ represents comparison between PlainNet models. $k = 8$ represents comparisons between PlainNet18 and ResNet18 models ($N = 45$ for $k = 0$, $N = 100$ elsewhere). (a-b) Dashed blue lines indicate the mean.

ResNet18 and PlainNet18 architectures. wARV largely remains the same as residual blocks are substituted for non-residual modules, dropping only for the most dissimilar architectures and even then, only slightly (Fig. 4b). As before, it is important to acknowledge that this result may reflect both true differences in what the models have learned, as well as accuracy limitations of wARV.

## 5 Discussion

We have shown that collapsing a model's weights and computation graph into a permutation-invariant input-to-output comparison space – the chain weight matrix – preserves substantial information about what a model has learned. Ahmad's bias-corrected version of the RV coefficient enables significance testing to determine whether chain weight matrices contain related information. Together, this framework provides a data-free metric of neural network similarity.

The chain weight matrix is obtained by formulating coarse linear approximations of the transformations encoded by a neural network. Its sensitivity to the task learned and relative insensitivity to architectural variation are consistent with previous findings showing neural network functionality to be a substantially linear function of the parameters (Jacot et al., 2018; Hassid et al., 2022). Our aim has not been to provide guarantees about the optimality of the present approximations for any individual model component, but instead to show in general that making such approximations preserves comparable training-relevant information across diverse architectures. Rather than a fixed method, wARV is best described as a framework for data-free model comparison. Signal strength and interpretation depend on the level of detail and assumptions made in generating the approximations.

For example, our present implementation for attention layers involves replacing the data-dependent attention matrix by a constant matrix. This excludes the query and key matrix information; nevertheless, strong task-specific discrimination remains when making comparisons to non-attention-based architectures. In the case of vision transformers learning a simple image classification task, we interpret this as evidence that self-attention may not be considerably contributing to model functionality, outside of passing information between patches in a fairly uniform way.

This is one of several research questions to which wARV may be applicable. wARV seems to reflect meaningful geometry of the optimization trajectory, suggesting it could offer a data-independent notion of distance traveled during training. In a similar vein, because of its invariance to intertwiner

group transformations, wARV may provide a useful tool for exploring linear mode connectivity (Frankle et al., 2020) and the single basin conjecture of Entezari et al. (2022). In particular, the tendency of networks trained from different initializations to arrive at similar chain weight matrices lends credence to the hypothesis that SGD solutions converge to a linearly connected basin, up to permutation.

Deep learning models are known to exploit shortcut patterns in training data (Geirhos et al., 2020). Several approaches have been proposed to mitigate this by incorporating an interpretability constraint during training (Ross et al., 2017; Viviano et al., 2021; Simpson et al., 2019). These are collectively known as Right for the Right Reasons methods, as the model should base its decisions on task-relevant features rather than spurious correlations. Although these methods achieve high accuracy on challenge datasets, they incur a loss on standard test accuracy (Santos & Zanchettin, 2023), indicating that enforcing the right reasoning patterns comes at the cost of knowledge about the original data distribution. A similar robustness-accuracy trade-off arises in adversarial training (Tsipras et al., 2018). More recently, machine unlearning has emerged as a related direction (Maini et al., 2024), aiming to preserve model accuracy on a *retain set* while erasing knowledge tied to a *forget set*.

Right for the Right Reasons, adversarial training, and unlearning all require modifying a model's behavior on a problematic subset of data while preserving its behavior on the original distribution. wARV provides a natural constraint for these settings; by penalizing divergence in wARV between the model being optimized and its original counterpart, one can encourage the model to solve the data challenge while retaining its overall behavior. Concretely, this amounts to including a term of the form $1 - \text{wARV}(\theta_{\text{new}}, \theta_{\text{orig}})$ in the optimization objective. This is conceptually similar to the Representation Misdirection for Unlearning method (Li et al., 2024), which regularizes intermediate representations to remain close to those of the original model. The chain weight matrix of wARV encodes the full input-to-output transformation, offering a more global, data-independent measure of behavioral change instead of anchoring similarity to a particular layer's activations.

## 6 Conclusion

We introduce the weight-based Ahmad RV coefficient (wARV), a data-free framework for comparing neural networks. By collapsing a model's parameters and computation graph into a shared comparison space, wARV can differentiate models trained across a range of image classification tasks, and it can make reliable comparisons across MLP, BMLP, LeNet, ResNet, and ViT architectures. wARV can be used as a binary probe for model similarity through the provided significance test, but it also identifies gradual variation throughout the optimization trajectory and incremental changes in the composition of training data. This demonstrates the utility of wARV as a data-independent similarity metric and suggests applications for future work on linear mode connectivity, interpretability, and unlearning.

### AI use statement

Generative AI was used to assist with coding. We have manually verified all code and take responsibility for its correctness.

### Reproducibility statement

In an effort to facilitate reproducibility, we describe the methodology in detail in Section 3 and Appendix Sections A-B. We describe the experiments in detail in the introduction to Section 4 and Appendix Section E. Jupyter Notebooks for fully reproducing the experiments and figures can be accessed at [to be provided following de-anonymization] or by contacting the authors.

## References

M. Rauf Ahmad. A significance test of the RV coefficient in high dimensions. *Computational Statistics & Data Analysis*, 131:116–130, 2019.

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023.

Marti J Anderson. A new method for non-parametric multivariate analysis of variance. *Austral ecology*, 26(1):32–46, 2001.

Gregor Bachmann, Sotiris Anagnostidis, and Thomas Hofmann. Scaling MLPs: A tale of inductive bias. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Luke Nicholas Darlow, Elliot J. Crowley, Antreas Antoniou, and Amos J. Storkey. CINIC-10 is not imagenet or CIFAR-10. *CoRR*, abs/1810.03505, 2018.

MohammadReza Davari, Stefan Horoi, Amine Natik, Guillaume Lajoie, Guy Wolf, and Eugene Belilovsky. Reliability of CKA as a similarity measure in deep learning. In *The Eleventh International Conference on Learning Representations*, 2023.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *International Conference on Learning Representations*, 2022.

Yves Escoufier. Le traitement des variables vectorielles. *Biometrics*, 29:751, 1973.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International conference on machine learning*, pp. 3259–3269. PMLR, 2020.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Charles Godfrey, Davis Brown, Tegan Emerson, and Henry Kvinge. On the symmetries of deep learning models and their internal representations. In *Advances in Neural Information Processing Systems*, volume 35, pp. 11893–11905, 2022.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations*, 2015.

Per Christian Hansen. Deconvolution and regularization with toeplitz matrices. *Numerical algorithms*, 29(4):323–378, 2002.

Michael Hassid, Hao Peng, Daniel Rotem, Jungo Kasai, Ivan Montero, Noah A. Smith, and Roy Schwartz. How much does attention actually attend? questioning the importance of attention in pretrained transformers. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1403–1416, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

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.

Masashi Hyodo, Takahiro Nishiyama, and Shoichi Narita. Low dimensional factor model-based tests for assessing vector correlation in high-dimensional settings. *Journal of Multivariate Analysis*, 213:105588, 2026.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Max Klabunde, Tobias Schumacher, Markus Strohmaier, and Florian Lemmerich. Similarity of neural network models: A survey of functional and representational measures. *ACM Comput. Surv.*, 57(9), May 2025. ISSN 0360-0300.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529. PMLR, 09–15 Jun 2019.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D Li, Ann-Kathrin Dombrowski, Shashwat Goel, Long Phan, et al. The wmdp benchmark: Measuring and reducing malicious use with unlearning. *arXiv preprint arXiv:2403.03218*, 2024.

Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C Lipton, and J Zico Kolter. Tofu: A task of fictitious unlearning for LLMs. *arXiv preprint arXiv:2401.06121*, 2024.

Zeping Min and Xinshang Wang. DOCS: Quantifying weight similarity for deeper insights into large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pp. 2662–2670. AAAI Press, 2017. ISBN 9780999241103.

Flávio Arthur Oliveira Santos and Cleber Zanchettin. Exploring image classification robustness and interpretability with right for the right reasons data augmentation. In *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 4149–4158. IEEE, 2023.

Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43947–43966. PMLR, 21–27 Jul 2024.

Becks Simpson, Francis Dutil, Yoshua Bengio, and Joseph Paul Cohen. Gradmask: Reduce overfitting by regularizing saliency. *arXiv preprint arXiv:1904.07478*, 2019.

David Stutz, Matthias Hein, and Bernt Schiele. Relating Adversarially Robust Generalization to Flat Minima . In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7787–7797, Los Alamitos, CA, USA, October 2021. IEEE Computer Society.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2815–2823. IEEE, 2019.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Joseph D. Viviano, Becks Simpson, Francis Dutil, Yoshua Bengio, and Joseph Paul Cohen. Saliency is a possible red herring when diagnosing poor generalization. In *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net, 2021.

Guangcong Wang, Guangrun Wang, Wenqi Liang, and Jianhuang Lai. Understanding weight similarity of neural networks via chain normalization rule and hypothesis-training-testing. *arXiv preprint arXiv:2208.04369*, 2022.

# A ARV SIGNIFICANCE TESTING

Hyodo et al. (2026) showed that the distribution of ARV$(\boldsymbol{X}_1, \boldsymbol{X}_2)$ under the null ($\mathbb{E}$(ARV) $= 0$ for orthogonal columns) is affected by the spectra of the Gram matrices $\boldsymbol{S}_1 = \tilde{\boldsymbol{X}}_1\tilde{\boldsymbol{X}}_1^\top$ and $\boldsymbol{S}_2 = \tilde{\boldsymbol{X}}_2\tilde{\boldsymbol{X}}_2^\top$. Specifically, asymptotic normality holds under a weak spike structure

$$\frac{\lambda_{i,1}^2}{\sum_k \lambda_{i,k}^2} = o(1), d_i \to \infty$$

where $\lambda_{i,k}$ denote the eigenvalues of $\boldsymbol{S}_i$, with $\lambda_{i,1}$ being the largest. They show that the null distribution of ARV converges to a sum of i.i.d. chi-squared ($\chi_1^2$) distributions weighted by the magnitudes of $\lambda_{1,k}$ and $\lambda_{2,k}$, so that when one or a few eigenvalues are substantially larger in magnitude than the others, asymptotic normality breaks down.

$$n \cdot \text{ARV} + \frac{\sum \lambda_{1,k} \sum \lambda_{2,k}}{\sqrt{\sum \lambda_{1,k}^2 \sum \lambda_{2,k}^2}} \xrightarrow{\mathcal{D}} \frac{\sum_i \sum_j (\lambda_{1,i}\lambda_{2,j}\chi_{1,(i,j)}^2)}{\sqrt{\sum \lambda_{1,k}^2 \sum \lambda_{2,k}^2}}, \quad n, d_1, d_2 \to \infty \tag{5}$$

Note that although in the asymptotic limit the standard deviation of ARV scales like $\mathcal{O}(n^{-1})$ – corresponding to the leading coefficient of equation 5 – we find empirically that providing an additional order of accuracy, such that the standard deviation scales like $\mathcal{O}((n^2 - 3n)^{-1/2})$, affords an improved fit for $n < 20$ (Fig. 5d).

Hyodo et al. suggest using Monte Carlo methods to estimate the quantile of ARV under the null. We alternatively propose that for modestly sized problems (say, $\min(n, d_i) = \mathcal{O}(10^3)$ non-zero eigenvalues for each of $\boldsymbol{S}_i$), it is computationally reasonable to compute the CDF of the null distribution using its characteristic function and the complete set of non-zero eigenvalues.

The characteristic function of the chi-squared distribution with one degree of freedom is $\phi_{\chi_1^2}(t) = (1 - 2it)^{-1/2}$. For the weighted sum of i.i.d. $\chi_1^2$, $D = \sum_{j=1}^k \gamma_j \chi_{1,j}^2$, the characteristic function is $\phi_D(t) = \prod_{j=1}^k (1 - 2i\gamma_j t)^{-1/2}$. Then applying the Gil-Pelaez inversion formula, the CDF of $D$ is

$$F_D(x) = \frac{1}{2} - \frac{1}{\pi} \int_0^\infty \text{Im}\Big[\frac{\mathrm{e}^{-itx}\phi_D(t)}{t}\Big]\, dt$$

This provides a one-tailed significance test for $H_0$ : ARV $= 0$ vs. $H_1$ : ARV $> 0$. Let each $\gamma_j$ of $\phi_D(t)$ be the $j$-th element of the Kronecker product

$$\frac{(\lambda_{1,1}, \lambda_{1,2}, ...)^T \otimes (\lambda_{2,1}, \lambda_{2,2}, ...)^T}{\sqrt{\sum \lambda_{1,k}^2 \sum \lambda_{2,k}^2}}$$

Then the probability of selecting a value at least as great as ARV under $H_0$ is

$$p = 1 - F_D\left[(n^2 - 3n)^{1/2}\text{ARV} + \frac{\sum \lambda_{1,k} \sum \lambda_{2,k}}{\sqrt{\sum \lambda_{1,k}^2 \sum \lambda_{2,k}^2}}\right] \tag{6}$$

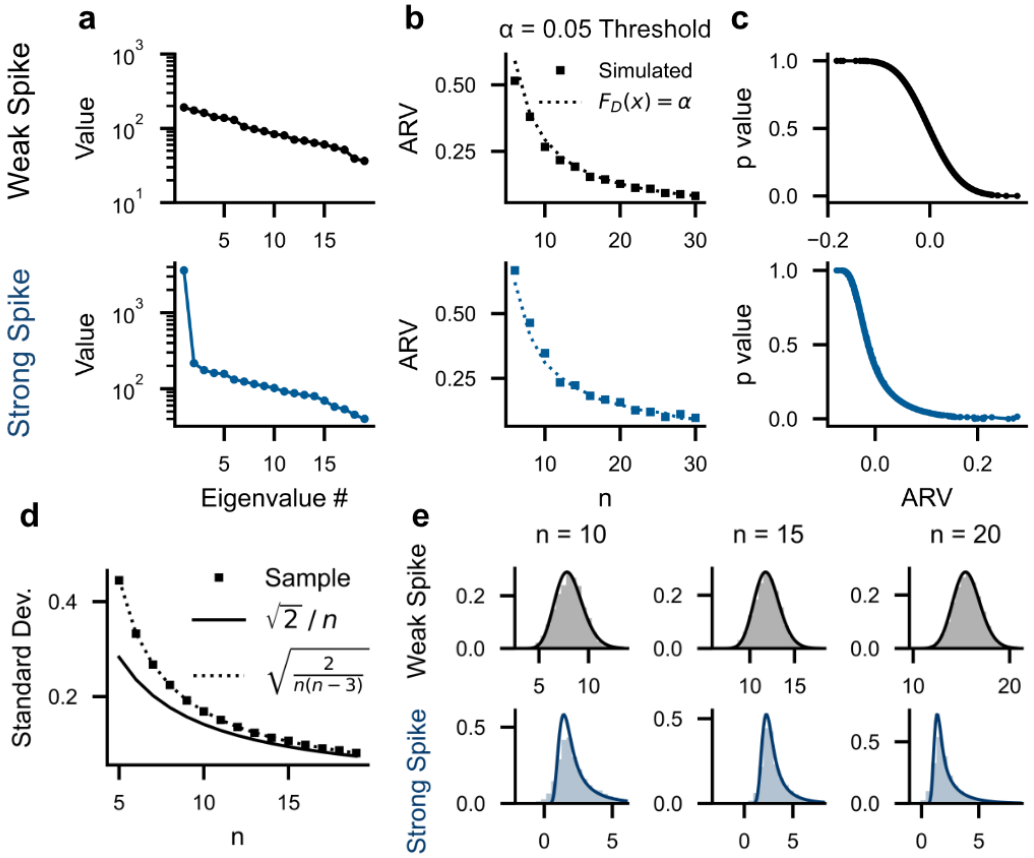


Figure 5: **The null distribution of ARV depends on the spectra of its inputs.** (a) Examples of eigenvalues under the weak spike condition (top) and strong spiking (bottom). (b) Comparison of ARV threshold predicted by equation 6 at $\alpha = 0.05$ significance level vs. 95th percentile ARV value among $N = 1{,}000$ comparisons of randomly generated matrices of size $n \times 100$, for each given $n$. (c) Distributions of p values for $N = 1{,}000$ comparisons between randomly generated matrices of size $30 \times 100$. (d) Standard deviations of ARV calculated for randomly generated matrices of size $n \times 100$ ($N = 10{,}000$ comparisons for each $n$). (e) Probability density function plots showing the theoretical distribution of null-distributed ARV values given by the Gil-Pelaez PDF integral, compared to ARV values for randomly generated matrices of size $n \times 100$ ($N = 10{,}000$ matrices each). (a-e) Random matrices were generated by imposing predefined eigenvalues via SVD on a starting matrix of random standard normal entries.

# B COMPUTING CHAIN WEIGHT MATRICES

Computing the chain weight matrix involves representing a neural network as a sequence of approximately linear transformations. Here, we detail how we treat commonly used components of deep learning architectures.

$$\boldsymbol{M} = \boldsymbol{W}_L \boldsymbol{W}_{L-1} \cdots \boldsymbol{W}_1 \in \mathbb{R}^{d_L \times d_0}$$

## B.1 LINEAR LAYER

A linear layer encodes the affine transformation $\boldsymbol{x} \in \mathbb{R}^{d_x} \mapsto \boldsymbol{Z}\boldsymbol{x} + \boldsymbol{b} \in \mathbb{R}^{d_y}$, where $\boldsymbol{Z} \in \mathbb{R}^{d_y \times d_x}$ represents the learned weights and $\boldsymbol{b} \in \mathbb{R}^{d_y}$ represents the learned bias. The linear component of this transformation is the weight matrix: $\boldsymbol{W}_{\text{Linear}} = \boldsymbol{Z}$.

## B.2 2D CONVOLUTION

A 2D Convolution layer encodes the affine transformation $\boldsymbol{X} \in \mathbb{R}^{c_i \times x_h \times x_w} \mapsto \boldsymbol{\mathcal{K}} \circ \boldsymbol{X} + \boldsymbol{b} \in \mathbb{R}^{c_o \times y_h \times y_w}$, where $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{c_o \times c_i \times k_h \times k_w}$ represents the learned filters, $\boldsymbol{b} \in \mathbb{R}^{c_o}$ is the learned bias, and $\circ$ represents the 2D cross-correlation operator with padding $(p_h, p_w)$, stride $(s_h, s_w)$, and dilation $(1, 1)$, such that $y_h = \left\lfloor \frac{x_h + 2p_h - k_h}{s_h} + 1 \right\rfloor$ and $y_w = \left\lfloor \frac{x_w + 2p_w - k_w}{s_w} + 1 \right\rfloor$.

$\boldsymbol{\mathcal{K}} \circ \boldsymbol{X}$ can alternatively be represented as $\boldsymbol{K}\boldsymbol{x}$, where $\boldsymbol{x} \in \mathbb{R}^{c_i x_h x_w}$ corresponds to $\boldsymbol{X}$ flattened in row-major order, and $\boldsymbol{K} \in \mathbb{R}^{c_o y_h y_w \times c_i x_h x_w}$ is a sparse block Toeplitz matrix (Hansen, 2002). Each element $k_{n,c,h,w}$ of $\boldsymbol{\mathcal{K}}$ is mapped to $\boldsymbol{K}$ at zero-based indices $(\alpha, \beta)$ for all $i \in [0, y_h),\ j \in [0, y_w)$ according to the rules

$$\alpha = n y_h y_w + i y_w + j$$

$$\beta = c x_h x_w + (h - p_h + s_h i) x_w + (w - p_w + s_w j)$$

subject to $0 \leq (h - p_h + s_h i) < x_h$ and $0 \leq (w - p_w + s_w j) < x_w$. Then $\boldsymbol{W}_{\text{Conv2D}} = \boldsymbol{K}$.

## B.3 MULTI-HEAD ATTENTION

A multi-head attention layer encodes the transformation

$$\boldsymbol{X} \in \mathbb{R}^{n \times d_x} \mapsto \text{concat}_i \left[ \text{softmax}\left( \frac{(\boldsymbol{X}\boldsymbol{W}_{Q,i})(\boldsymbol{X}\boldsymbol{W}_{K,i})^\top}{\sqrt{d_a}} \right) (\boldsymbol{X}\boldsymbol{W}_{V,i}) \right] \boldsymbol{W}_O \in \mathbb{R}^{n \times d_x}$$

where $\boldsymbol{W}_{Q,i},\ \boldsymbol{W}_{K,i},\ \boldsymbol{W}_{V,i} \in \mathbb{R}^{d_x \times d_a}$ represent the query, key, and value projections for head $i \in [0, h)$ of attention, $\boldsymbol{W}_O \in \mathbb{R}^{h d_a \times d_x}$ represents the output projection, and softmax is performed row-wise. The attention matrix (the softmax term) is input-dependent, so we make the approximation that each patch attends uniformly to each other patch:

$$\boldsymbol{X} \mapsto \text{concat}_i \left( \frac{1}{n} \boldsymbol{J}_n \boldsymbol{X} \boldsymbol{W}_{V,i} \right) \boldsymbol{W}_O = \frac{1}{n} [(\text{concat}_i(\boldsymbol{W}_{V,i}) \boldsymbol{W}_O)^\top \otimes \boldsymbol{J}_n] \boldsymbol{x}$$

where $\boldsymbol{J}_n \in \mathbb{R}^{n \times n}$ has all elements equal to 1, $\otimes$ indicates the Kronecker product, and $\boldsymbol{x} \in \mathbb{R}^{n d_x}$ corresponds to $\boldsymbol{X}$ flattened in row-major order. Therefore, the transformation is approximated as $\boldsymbol{A}\boldsymbol{x}$ where $\boldsymbol{A} \in \mathbb{R}^{n d_x \times n d_x}$ is a block matrix with each block as $\frac{1}{n}(\text{concat}_i(\boldsymbol{W}_{V,i}) \boldsymbol{W}_O)^\top \in \mathbb{R}^{d_x \times d_x}$. Then $\boldsymbol{W}_{\text{MultiheadAttention}} = \boldsymbol{A}$.

## B.4 BATCH NORMALIZATION

Batch normalization performs channel-wise normalization and affine scaling. For 2D batch normalization, the transformation is $\boldsymbol{X} \in \mathbb{R}^{d_c \times d_h \times d_w} \mapsto \frac{\boldsymbol{X} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \varepsilon}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta} \in \mathbb{R}^{d_c \times d_h \times d_w}$, where $\boldsymbol{\mu},\ \boldsymbol{\sigma}^2 \in \mathbb{R}^{d_c \times 1 \times 1}$ are the channel-wise running mean and variance of $\boldsymbol{X}$ computed during training, $\boldsymbol{\gamma},\ \boldsymbol{\beta} \in \mathbb{R}^{d_c \times 1 \times 1}$ are the learned affine parameters, $\varepsilon$ is a small constant for numerical stability, and $\odot$ is the element-wise product.

The linear component of the transformation applied to each channel is $\gamma_c / \sqrt{\sigma_c^2 + \varepsilon}$. If $\boldsymbol{x} \in \mathbb{R}^{d_c d_h d_w}$ corresponds to $\boldsymbol{X}$ flattened in row-major order, the linear component can be represented $\boldsymbol{B}\boldsymbol{x}$, where $\boldsymbol{B} \in \mathbb{R}^{d_c d_h d_w \times d_c d_h d_w} = \text{diag}(\boldsymbol{\gamma} / \sqrt{\boldsymbol{\sigma}^2 + \varepsilon}) \otimes \boldsymbol{I}_{(d_h d_w)}$, where $\text{diag}(\cdot)$ indicates a 1D vector placed along the diagonal of a 2D sparse matrix, $\otimes$ is the Kronecker product, and $\boldsymbol{I}$ is the identity matrix. Then $\boldsymbol{W}_{\text{BatchNorm2D}} = \boldsymbol{B}$.

### B.5 Layer Normalization

Layer normalization performs a similar operation as batch normalization (see above), except the transformation is typically applied along the final axis instead of per-channel, and statistics are computed at inference time rather than using precomputed statistics. Because the variance scaling term is inherently input-dependent, we suggest obtaining an estimate – either per axis or pooled – via exponential averaging throughout training for use in computing the chain weight matrix downstream.

### B.6 Non-Linear Activations

We suggest omitting the effects of non-linear activations unless the pre-activations can be reliably expressed as Gaussian distributed with known mean and variance – for example, immediately downstream from batch or layer normalization. We are interested in the linear component, $a(x_i) = \text{Cov}(x_i, f(x_i)) \;/\; \text{Var}(x_i)$, of the least squares approximation to the activation, $f(x_i) \approx a(x_i)x_i + b(x_i)$. In cases where inputs can reasonably be expressed as $x_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, it can be shown that

$$\text{Cov}(x_i, \text{ReLU}(x_i)) = \sigma_i^2 \Phi\Big(\frac{\mu_i}{\sigma_i}\Big)$$

$$\text{Cov}(x_i, \text{GELU}(x_i)) = \sigma_i^2 \Phi\Big(\frac{\mu_i}{\sqrt{1+\sigma_i^2}}\Big) + \frac{\mu_i \sigma_i^2}{(1+\sigma_i^2)^{3/2}} \phi\Big(\frac{\mu_i}{\sqrt{1+\sigma_i^2}}\Big)$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal PDF and CDF, respectively. The chain weight representation for a non-linear activation layer can thus be constructed by filling the diagonal of a sparse matrix with the appropriate scaling approximation, $a(x_i)$, for each $x_i$.

### B.7 Skip Connections

We represent residual blocks as $\boldsymbol{R}_j = \widehat{\boldsymbol{F}}_j + \boldsymbol{S}_j$, where $\widehat{\boldsymbol{F}}_j$ and $\boldsymbol{S}_j$ correspond to the main and shortcut (i.e., identity or downsampling) operators, as constructed using the methods above.

### B.8 2D Pooling

2D Max Pooling is applied to each channel individually and involves a convolution in which the maximum value is selected from each local window. We assume that on average each input feature contributes equally to its respective output, so this transformation can be represented much like a 2D Convolution, except this is a depthwise convolution with each channel processed independently. Let $\boldsymbol{X} \in \mathbb{R}^{c_i \times x_h \times x_w}$ be the input representation, $\boldsymbol{X}_1 \in \mathbb{R}^{1 \times x_h \times x_w}$ be the first channel of $\boldsymbol{X}$, $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{1 \times 1 \times k_h \times k_w}$ be the averaging kernel having all values $\frac{1}{k_h k_w}$, and $\boldsymbol{\mathcal{M}} \circ \boldsymbol{X}_1 \in \mathbb{R}^{1 \times y_h \times y_w}$ be the 2D cross-correlation as above. Furthermore, let $\boldsymbol{M} \in \mathbb{R}^{y_h y_w \times x_h x_w}$ be the matrix corresponding to $\boldsymbol{\mathcal{M}}$ that computes the operation on a single flattened channel. Then we represent $\boldsymbol{W}_{\text{MaxPool2D}}$ as the $c_i y_h y_w \times c_i x_h x_w$ sparse matrix consisting of blocks $\boldsymbol{M}$ repeated along the diagonal $c_i$ times.

ResNet architectures also use 2D Adaptive Average Pooling before the final Linear layer to average each channel to a single value. This is an exactly linear transformation, which can be represented similarly using an averaging kernel, $\boldsymbol{\mathcal{A}}$, of size $1 \times 1 \times x_h \times x_w$ and 2D cross-correlation with 0 padding. Then $\boldsymbol{W}_{\text{AdaptiveAvgPool2D}}$ is the $c_i \times c_i x_h x_w$ sparse matrix consisting of blocks $\boldsymbol{A}$ repeated along the diagonal $c_i$ times, where $\boldsymbol{A}$ is the matrix corresponding to $\boldsymbol{\mathcal{A}}$ that computes the operation in flattened space.

# C Intertwiner Group Invariance for Selected Activation Functions

Following lemma 3.1 in Godfrey et al. (2022), $\phi_\sigma(\boldsymbol{A}) = \sigma(\boldsymbol{A})\sigma(\boldsymbol{I})^{-1}$,

$$\phi_\sigma(\boldsymbol{A}) = \boldsymbol{A} \iff \sigma(\boldsymbol{A}) = \boldsymbol{A}\sigma(\boldsymbol{I})$$

For ReLU and leaky ReLU, $\sigma(0) = 0$ and $\sigma(1) = 1$, this simplifies to

$$\phi_\sigma(\boldsymbol{A}) = \boldsymbol{A} \iff \sigma(\boldsymbol{A}) = \boldsymbol{A}$$

Under this condition, and that $\phi_\sigma$ is a homomorphism (lemma 3.1),

$$\phi_\sigma(\boldsymbol{A}^{-1}) = \phi_\sigma(\boldsymbol{A})^{-1} = \boldsymbol{A}^{-1}$$

GELU does not feature the same isotropic scaling invariance as the aforementioned two activation funtions. Section E.2.1 in Godfrey et al. (2022) clarifies that therefore, $G_{\text{GELU}} = \Sigma$ is the group of permutation matrices, since $\text{GELU}(\Sigma) = \text{GELU}(1)\Sigma$ and $\text{GELU}(\boldsymbol{I}) = \text{GELU}(1)\boldsymbol{I}$,

$$\phi_{\text{GELU}}(\Sigma) = \text{GELU}(\Sigma)\text{GELU}(\boldsymbol{I})^{-1} = (\text{GELU}(1)\Sigma)\left(\frac{1}{\text{GELU}(1)}\boldsymbol{I}\right) = \Sigma$$

Therefore, for ReLU, leaky ReLU, and GeLU, $\phi(\boldsymbol{A}^{-1}) = \boldsymbol{A}^{-1}$ holds for all $\boldsymbol{A} \in G_\sigma$.

## D SUPPLEMENTARY FIGURES

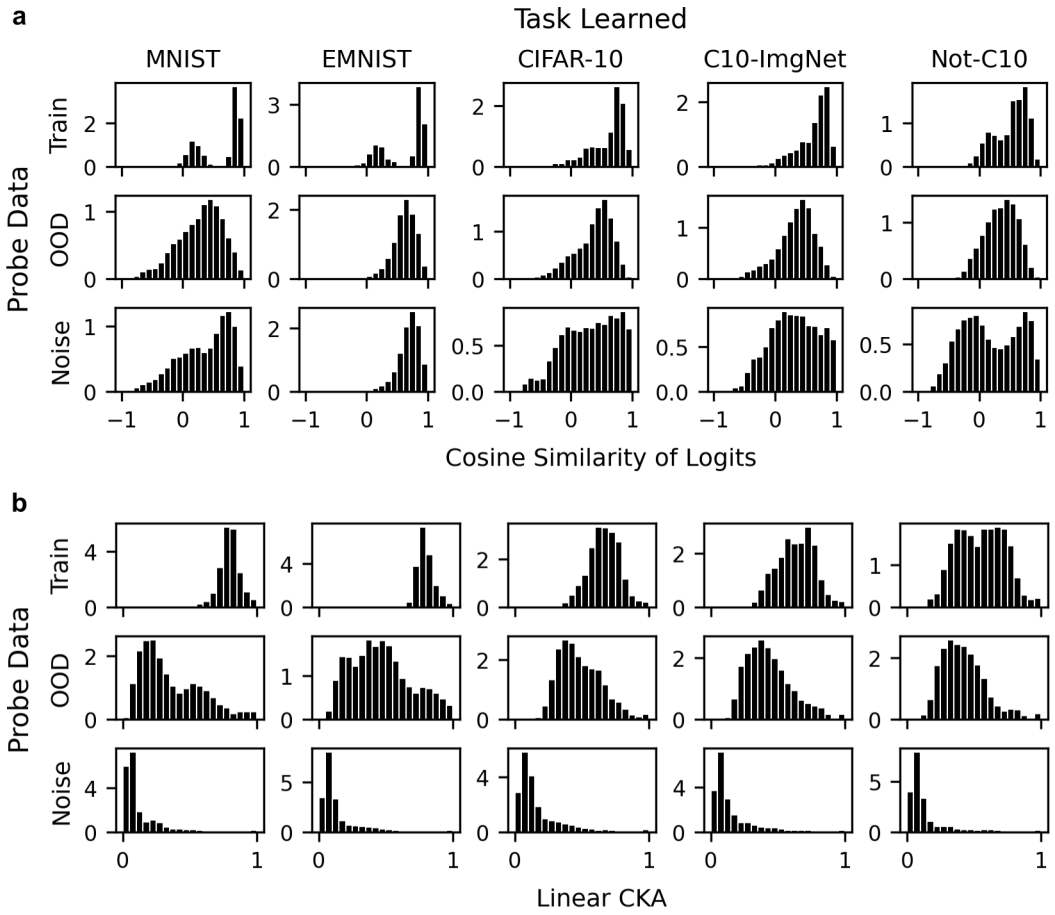


Figure 6: **Data-based similarity metrics are affected by the choice of probing data.** Distributions of (a) cosine similarity of logits and (b) linear CKA of representations for pairs of models trained on the same task and probed with 200 samples of training data, out-of-distribution (OOD) data, or random noise. OOD data is CIFAR10 for MNIST/EMNIST-trained models and MNIST for other models. $N = 14{,}400$ each panel.

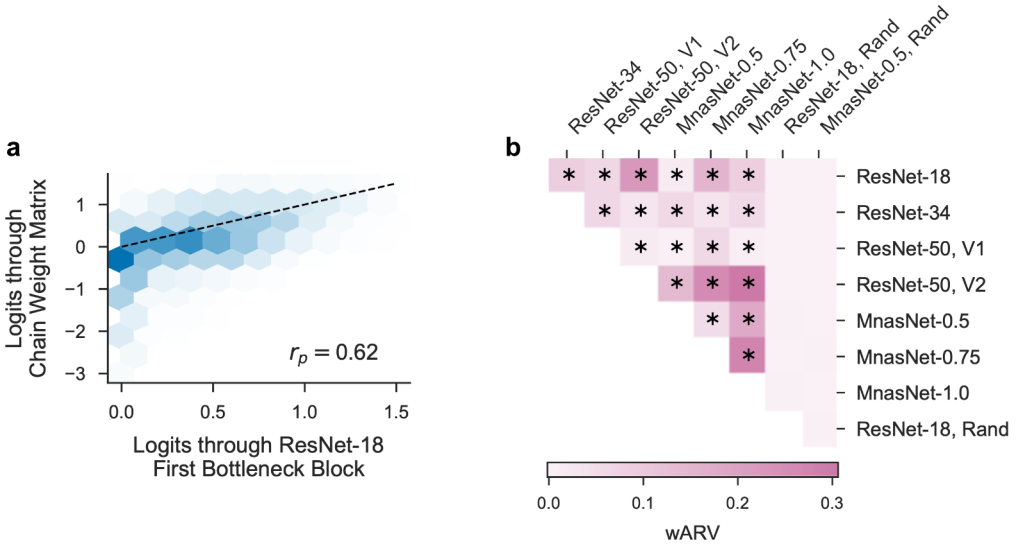


Figure 7: **wARV detects functional similarity for convolutional models trained on ImageNet-1k.** (a) Hexbin plot showing correspondence between logits through the first bottleneck block and the chain weight linear approximation made for that block ($N = 1.27 \times 10^7$ logits). (b) wARV values for comparisons between publicly available ResNet and MnasNet (Tan et al., 2019) models of various sizes trained on ImageNet-1k or untrained with random initializations. ($*P < 0.05$ by our significance test).

## E TRAINING AND ARCHITECTURE SPECIFICATIONS

All training was run with 200 images per batch, using cross entropy loss and the Adam optimizer with initial learning rate of $10^{-3}$. Adversarial training was performed by attacking half of the samples of each batch using FGSM with $\varepsilon = 8/255$.

**MLP 1M:**

- Linear(in: 3072, out: 320)
- GELU
- Linear(in: 320, out: 64)
- GELU
- Linear(in: 64, out: 10)

**MLP 11M:**

- Linear(in: 3072, out: 3072)
- GELU
- Linear(in: 3072, out: 512)
- GELU
- Linear(in: 512, out: 256)
- GELU
- Linear(in: 256, out: 10)

**BMLP 1M:**

- Linear(in: 3072, out: 192)
- 2 x BottleneckBlock. For each:
  - BatchNorm1D(features: 192)
  - Linear(in: 192, out: 576)
  - GELU
  - Linear(in: 576, out: 192)
  - Skip connection to original input from start of block.
- Linear(in: 192, out: 10)

**BMLP 11M:**

- Linear(in: 3072, out: 512)
- 6 x BottleneckBlock. For each:
  - BatchNorm1D(features: 512)
  - Linear(in: 512, out: 1536)
  - GELU
  - Linear(in: 1536, out: 512)
  - Skip connection to original input from start of block.
- Linear(in: 512, out: 10)

**LeNet 1M:**

- Conv2d(in: 3, out: 64, kernel: 3, stride: 1, pad: 1)
- BatchNorm2d(channels: 64)
- GELU
- MaxPool2d(kernel: 4, stride: 4)
- Flatten
- Linear(in: 4096, out: 256)
- GELU
- Linear(in: 256, out: 10)

**LeNet 11M:**

- Conv2d(in: 3, out: 768, kernel: 3, stride: 1, pad: 1)
- BatchNorm2d(channels: 768)
- GELU
- MaxPool2d(kernel: 2, stride: 2)
- Conv2d(in: 768, out: 512, kernel: 3, stride: 1, pad: 1)
- BatchNorm2d(channels: 512)
- GELU
- MaxPool2d(kernel: 2, stride: 2)
- Conv2d(in: 512, out: 256, kernel: 3, stride: 1, pad: 1)
- BatchNorm2d(channels: 256)
- GELU
- MaxPool2d(kernel: 2, stride: 2)
- Flatten
- Linear(in: 4096, out: 1536)
- GELU
- Linear(in: 1536, out: 256)
- GELU
- Linear(in: 256, out: 10)

**ResNet 1M:**

- Conv2d(in: 3, out: 96, kernel: 3, stride: 1, pad: 1)
- BatchNorm2d(channels: 96)
- GELU
- ResidualBlock(bin: 96, bout: 192)
- ResidualBlock(bin: 192, bout: 192)
- AdaptiveAvgPool2d(out: (1, 1))
- Flatten
- Linear(in: 192, out: 10)

- For ResidualBlock details, see ResNet 11M below.

**ResNet 11M:**

- Conv2d(in: 3, out: 64, kernel: 3, stride: 1, pad: 1)
- BatchNorm2d(channels: 64)
- GELU
- ResidualBlock(bin: 64, bout: 64)
- ResidualBlock(bin: 64, bout: 64)
- ResidualBlock(bin: 64, bout: 128)
- ResidualBlock(bin: 128, bout: 128)
- ResidualBlock(bin: 128, bout: 256)
- ResidualBlock(bin: 256, bout: 256)
- ResidualBlock(bin: 256, bout: 512)
- ResidualBlock(bin: 512, bout: 512)
- AdaptiveAvgPool2d(out: (1, 1))
- Flatten
- Linear(in: 512, out: 10)

- For each ResidualBlock:
  - Conv2d(in: bin, out: bout, kernel: 3, stride: 1 if bin=bout or 2 otherwise, pad: 1)
  - BatchNorm2d(channels: bout)
  - GELU
  - Conv2d(in: bout, out: bout, kernel: 3, stride: 1, pad: 1)
  - BatchNorm2d(channels: bout)
  - Skip connection with:
    - Original input from start of block if bin=bout.
    - Otherwise, original input transformed through: Conv2d(in: bin, out: bout, kernel: 1, stride: 2, pad: 0), followed by BatchNorm2d(channels: bout).
  - GELU

**ViT 1M:**

- Conv2d(in: 3, out: 128, kernel: 4, stride: 4, pad: 0)
- Prepend classification token: Parameter(shape: 1, 1, 128)
- Add position embedding: Parameter(shape: 1, 65, 128)
- 8 x TransformerBlock. For each:
  - LayerNorm(shape: 128)
  - MultiheadAttention(inner: 128, heads: 8)
  - Skip connection to original input from start of block.
  - LayerNorm(shape: 128)
  - Linear(in: 128, out: 256)
  - GELU
  - Linear(in: 256, out: 128)
  - Skip connection to output after first skip connection of block.
- Linear(in: 128, out: 10)

**ViT 11M:**

- Conv2d(in: 3, out: 384, kernel: 4, stride: 4, pad: 0)
- Prepend classification token: Parameter(shape: 1, 1, 128)
- Add position embedding: Parameter(shape: 1, 65, 128)
- 9 x TransformerBlock. For each:
  - LayerNorm(shape: 384)
  - MultiheadAttention(inner: 384, heads: 12)
  - Skip connection to original input from start of block.
  - LayerNorm(shape: 384)
  - Linear(in: 384, out: 768)
  - GELU
  - Linear(in: 768, out: 384)
  - Skip connection to output after first skip connection of block.
- Linear(in: 384, out: 10)