On the Symmetries of Deep Learning Models and their Internal Representations

Anonymous Author(s)
Affiliation
Address
email

Abstract

Symmetry has been a fundamental tool in the exploration of a broad range of complex systems. In machine learning, symmetry has been explored in both models and data. In this paper we seek to connect the symmetries arising from the architecture of a family of models with the symmetries of that family's internal representation of data. We do this by calculating a set of fundamental symmetry groups, which we call the *intertwiner groups* of the model. Each of these arises from a particular nonlinear layer of the model and different nonlinearities result in different symmetry groups. These groups change the weights of a model in such a way that the underlying function that the model represents remains constant but the internal representations of data inside the model may change. We connect intertwiner groups to a model's internal representations of data through a range of experiments that probe similarities between hidden states across models with the same architecture. Our work suggests that the symmetries of a network are propagated into the symmetries in that network's representation of data, providing us with a better understanding of how architecture affects the learning and prediction process. Finally, we speculate that for ReLU networks, the intertwiner groups may provide a justification for the common practice of concentrating model interpretability exploration on the activation basis in hidden layers rather than arbitrary linear combinations thereof.

1 Introduction

Symmetry provides an important path to understanding across a range of disciplines. This principle is well-established in mathematics and physics, where it has been a fundamental tool (e.g., Noether's Theorem [Noe18]). Symmetry has also been brought to bear on deep learning problems from a number of directions. There is, for example, a rich research thread that studies symmetries in data types that can be used to inform model architectures. The most famous examples of this are standard convolutional neural networks which encode the translation invariance of many types of semantic content in natural images into a network's architecture. In this paper, we focus on connections between two other types of symmetry associated with deep learning models: the symmetries in the learnable parameters of the model and the symmetries across different models' internal representation of the same data.

The first of these directions of research starts with the observation that in modern neural networks there exist models with different weights that behave identically on all possible input. We show that at least some of these equivalent models arise because of symmetries intrinsic to the nonlinearities of the network. We call these groups of symmetries, each of which is attached to a particular type of nonlinear layer σ of dimension n, the *intertwiner groups* G_{σ_n} of the model. These intertwiner groups come with a natural action on network weights for which the realization map to function

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

space of [JGH18] is invariant (proposition 3.4). As such they provide a unifying framework in which to discuss well-known weight space symmetries such as permutation of neurons [Bre+19] and scale invariance properties of ReLU and batch-norm [IS15; NH10].

Next, we tie our intertwiner groups to the symmetries between different model's internal representations of the same data. We do this through a range of experiments that we describe below; each builds on a significant recent advance in the field.

Neural stitching with intertwiner groups: The work of [BNB21; Csi+21; LV15] demonstrated that one can take two trained neural networks, say A and B, with the same architecture but trained from different randomly initialized weights, and connect the early layers of network A to the later layers of network B with a simple "stitching" layer and achieve negligible loss in prediction accuracy. This was taken as evidence of the similarity of strong model's representations of data. Though the original experiments use a fully connected linear layer to stitch, we provide theoretical evidence in theorem 4.2 that much less is needed. Indeed, we show that the intertwiner group (which has far fewer parameters in general) is the minimal viable stitching layer to preserve accuracy. We conduct experiments stitching networks at ReLU activation layers with the stitching layer restricted to elements of the group G_{ReLU} showing in fig. 1 that one can stitch CNNs on CIFAR10 [Kri09] with only elements of G_{ReLU} incurring less than $\approx 10\%$ accuracy penalty at most activation layers. This is surprisingly close to the losses found when one allows for a much more expressive linear layer to be used to stitch two networks together. However, we see that there remains a significant gap between the stitching accuracies obtained using G_{ReLU} and fully connected linear layers; this provides independent confirmation of earlier findings that neurons of independently trained networks are not simply permutations of each other [Li+15; Wan+18]. It is also consistent with observed phenomena such as distributed representations in hidden features [GBC16, §15.4] and perhaps also polysemantic neurons [Ola+20].

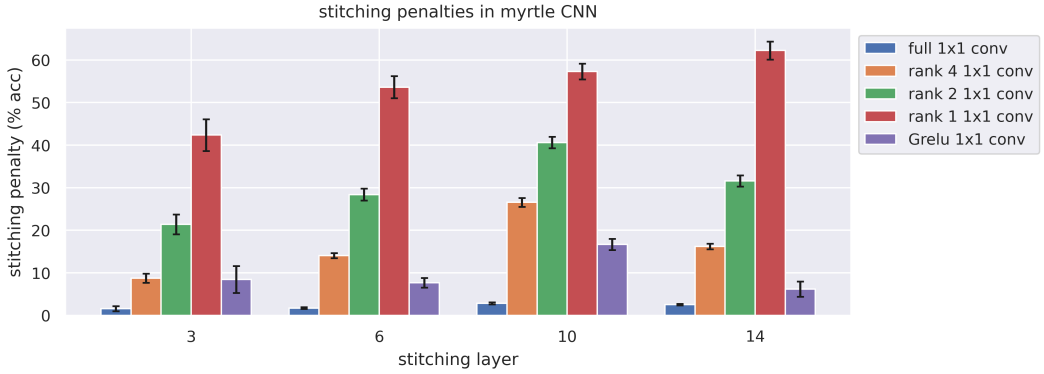
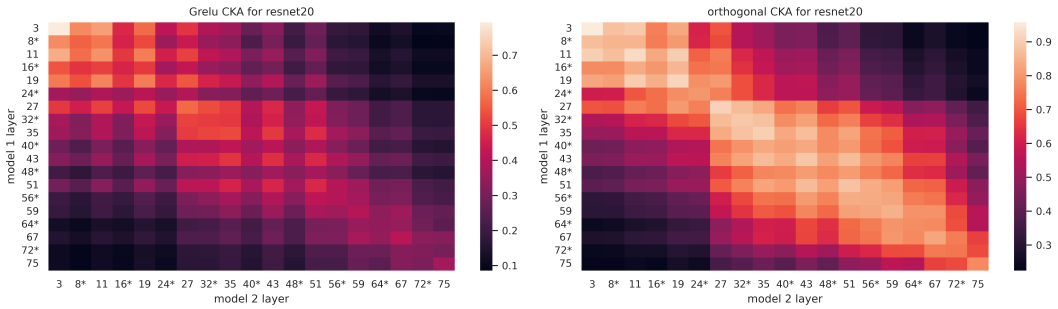


Figure 1: Full, reduced rank, and G_{ReLU} 1-by-1 convolution stitching penalties (4.3) for Myrtle CNNs [Pag18] on CIFAR10. Confidence intervals correspond to evaluating stitching penalties for 32 pairs of independently initialized models. Confidence intervals were taken over 5 independent runs. The accuracy of the independently initialized models was 91.3 ± 0.2 %.

Representation dissimilarity measures for G_{ReLU}: In section 5 we present two statistical dissimilarity measures for ReLU-activated hidden features in different networks, say A and B, that are maximized when the hidden features differ by a transformation from G_{ReLU} (that is, permutations and independent scaling of variables). These measures are analogous to the measures Centered Kernel Alignment (CKA)[1] [Kor+19] and Singular Vector Canonical Correlation Analysis [Rag+17], which are invariant to orthogonal transformations, and are maximized when both hidden features differ only up to orthogonal rotation and isotropic scaling. We compare and contrast these measures with their orthogonal counterparts, as well as with stitching experiment results.

Impact of activation functions on interpretability: An intriguing finding from [Bau+17, §3.2] was that individual neurons are more interpretable than random linear combinations of neurons. Our results on intertwiner groups (theorem 3.3) predict that this is a particular feature of ReLU networks. Indeed, we show in section 6 that in the absence of ReLU activations interpretability does

[1] with linear or RBF kernel.



(a) (b)

Figure 2: G_{ReLU}-CKA and orthogonal CKA for ResNet20s trained on CIFAR10, described further in section 5. Results averaged over 16 pairs of independently initialized models (for more details on how we sample models we refer to appendix D.1). Layers marked with '*' occur inside residual blocks (remark 3.6).

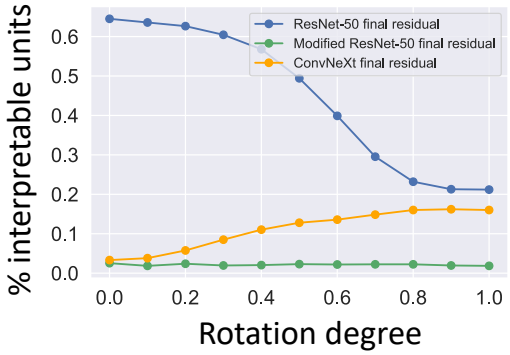


Figure 3: Fraction of network dissection interpretable units under rotations of the representation basis for a ResNet-50, as well as a modified ResNet-50 and ConvNeXt model (both without an activation function on the residual output). We note that the models without residual activation functions also have far fewer concept covering units for a given basis. Section 6 contains details and further discussion.

not decrease when one moves from individual neurons to linear combinations of neurons. This result suggests that intertwiner groups provide a theoretical justification for the explainable AI community's focus on individual neuron activations, rather than linear combinations thereof [Erh+09; Na+19; Yos+15; ZF14; Zho+15], but that this justification is only valid for layers with certain activation functions.

Taken together, our experiments provide evidence that a network's symmetries (realized through intertwiner groups) propagate down to symmetries of a model's internal representation of data. Since understanding how different models process the same data is a fundamental goal in fields such as explainable AI and the safety of deep learning systems, we hope that our results will provide an additional lens with which to grapple with these problems.

2 Related Work

The research on symmetries of neural networks is extensive, especially when one considers that symmetry considerations are a frequently used tool even when they are not the primary focus of a given paper, hence we aim to provide a representative sample knowing it will be incomplete. [Bre+19; FB17; GBC16; Yi+19] study the effect of weight space symmetries on the loss landscape. On the

other hand, [BMC15; GBC16; Kun+21; Men+19] study the effect of weight space symmetries on training dynamics, while [GBC16; RK20] show that weight space symmetries pose an obstruction to model identifiability.

Neural stitching was introduced as a means of comparing learned representations between networks in [LV15]. In [BNB21] it was shown to have intriguing connections with the "Anna Karenina " (high performance models share similar internal representations of data) and "more is better" (stitching later layers of a weak model to early layers of a model trained with more data/parameters/epochs can improve performance) phenomena. [Csi+21] considered constrained stitching layers by restricting the rank of the stitching matrix or by introducing an ℓ_1 sparsity penalty — our methods are distinct in that we explicitly optimize over the intertwiner group for ReLU nonlinearities (permutations and scalings). Both [BNB21; Csi+21] compare their stitching results with statistical dissimilarity measures such as CKA. One of our dissimilarity measures in section 5 is a close relative of the permutation Procrustes distance introduced in [Wil+21], and also closely related to the orthogonal Procrustes distance considered in [DDS21a]. The other is a an instance of of CKA [Kor+19] in which the kernel is taken to be $\max\{x_1 \cdot y_1, \ldots, x_d \cdot y_d\}$.

Approaches to deep learning interpretability often assume that the activation basis is special [Erh+09; Na+19; Yos+15; ZF14; Zho+15]. Studying individual neurons rather than linear combinations of neurons significantly reduces the complexity of low-level interpretability approaches to the understanding of neural networks [Elh+21]. In tension with this, many different projections of hidden layer activations appear to be semantically coherent [Sze+14]. However, Bau et al. found evidence that the representations closer to the coordinate basis align more with human concepts than random unitary transformations.

3 The Symmetries of Nonlinearities

Let $\mathrm{Mat}_{n_1,n_0}(\mathbb{R})$ be the algebra of all $n_0 \times n_1$ real matrices and $GL_n(\mathbb{R})$ be the group of all invertible $n \times n$ matrices. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous function. For any $n \in \mathbb{N}$, we can build a *nonlinearity* σ_n from $\mathbb{R}^n$ to $\mathbb{R}^n$ by applying σ coordinatewise, i.e., $\sigma_n(x_1, \ldots, x_n) = (\sigma(x_1), \ldots, \sigma(x_n))$. Fix some $k > 1$ and for each $1 \leq i < k$ let $\ell_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ be the composition of an affine layer and a nonlinear layer, so that $\ell_i(x) := \sigma_{n_i}(W_i x + b_i)$, and let $\ell_k(x) := W_k x + b_k$. Here $W_i \in \mathrm{Mat}_{n_i,n_{i-1}}(\mathbb{R})$ and $b_i \in \mathbb{R}^{n_i}$ are the weights and bias of layer i respectively. We define $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_k}$ to be the neural network $f = \ell_k \circ \cdots \circ \ell_1$. For each $1 \leq i < k-1$ we can then decompose f as $f = f_{>i} \circ f_{\leq i}$ where

$$f_{\leq i} = \ell_i \circ \cdots \circ \ell_1 \quad \text{and} \quad f_{>i} = \ell_k \circ \cdots \circ \ell_{i+1}.$$

We define

$$W := (W_i, b_i \,|\, i = 1, \ldots, k) \quad \text{and} \quad \mathcal{W} := \prod_{i=1}^{k} (\mathrm{Mat}_{n_i,n_{i-1}}(\mathbb{R}) \times \mathbb{R}^{n_i})$$

where the former is the collection of all weights of f and the latter is the space of all possible weights for a given architecture. When we want to emphasize the dependence of f on weights W, we write $f(-, W)$ (and similarly $f_{\leq i}(-, W), f_{>i}(-, W)$).

One of the topics this work will consider is vector space bases for f's hidden spaces $\mathbb{R}^{n_i}$, for $1 \leq i \leq k-1$. We will investigate the legitimacy of analyzing features $f_{\leq i}(D)$ for dataset $D \subset \mathbb{R}^{n_0}$ with respect to the *activation basis* for $\mathbb{R}^{n_i}$ which is simply the usual coordinate basis, $e_1, \ldots, e_{n_i}$ where $e_j = [\delta_{j\ell}]_{\ell=1}^{n_i}$ is naturally parameterized by individual neuron activations. Note that $\mathbb{R}^{n_i}$ has an infinite number of other possible bases that could be chosen.

3.1 Intertwiner Groups

For any $0 \leq i < k$, elements of $GL_{n_i}(\mathbb{R})$ can be applied to the hidden activation space $\mathbb{R}^{n_i}$ both before and after the nonlinear layer σ_{n_i}. We define

$$G_{\sigma_{n_i}} := \{A \in GL_{n_i}(\mathbb{R}) \,|\, \text{ there exists a } B \in GL_{n_i}(\mathbb{R}) \text{ such that } \sigma_{n_i} \circ A = B \circ \sigma_{n_i}\}.$$

Informally, we can understand $G_{\sigma_{n_i}}$ to be the set of all invertible linear transformations whose action on $\mathbb{R}^{n_i}$ prior to the nonlinear layer σ_{n_i} has an equivalent invertible transformation after σ_{n_i}. This

Activation	G_{σ_n}	$\phi_\sigma(A)$
$\sigma(x) = x$ (identity)	$GL_n(\mathbb{R})$	A
$\sigma(x) = \frac{e^x}{1+e^x}$	Σ_n	A
$\sigma(x) = \mathrm{ReLU}(x)$	Matrices PD, where D has positive entries	A
$\sigma(x) = \mathrm{LeakyReLU}(x)$	Same as ReLU as long as negative slope $\neq 1$	A
$\sigma(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ (RBF)	Matrices PD, where D has entries in $\{\pm 1\}$	$\mathrm{abs}(A)$
$\sigma(x) = x^d$ (polynomial)	Matrices PD, where D has non-zero entries	$A^{\odot d}$

Table 1: Explicit descriptions of G_{σ_n} and ϕ_σ for six different activations. Here $P \in \Sigma_n$ is a permutation matrix, D is a diagonal matrix, abs denotes the operation of taking the absolute value of every entry of A, and $A^{\odot d}$ denotes the process of taking the dth power of every entry of A.

is a generalization of the common procedure of understanding a function by understanding those operators that commute with it. For any $A \in GL_{n_i}(\mathbb{R})$, we can write $\sigma(A)$ for the $n_i \times n_i$ matrix formed by applying σ to all entries in A.

Lemma 3.1. *Suppose $\sigma(I_n)$ is invertible and for each $A \in GL_n(\mathbb{R})$ define $\phi_\sigma(A) = \sigma(A)\sigma(I_n)^{-1}$. Then G_{σ_n} is a group, $\phi_\sigma : G_{\sigma_n} \to GL_n(\mathbb{R})$ is a homomorphism and $\sigma_n \circ A = \phi_\sigma(A) \circ \sigma_n$.*

We defer all proofs to appendix E. We include concrete examples of σ for small n_i there as well.

Definition 3.2. When the hypotheses of lemma 3.1 are satisfied (namely, $\sigma(I_n)$ is invertible) we call G_{σ_n} the **intertwiner group of the activation** σ_n. We denote the image of the homomorphism ϕ_σ as $\phi_\sigma(G_{\sigma_n})$.

The intertwiner group G_{σ_n} and ϕ_σ are concretely described for a range of activations in table 1 — the last two examples motivate the generality of definition 3.2. Note also that in both of those cases $A \mapsto \phi_\sigma(A)$ is *not* a homomorphism on all of $GL_n(\mathbb{R})$, but *is* a homomorphism when restricted to the appropriate subgroup G_{σ_n}. While a substantial part of table 1 can be found scattered in prior work, our calculations in appendix E.2 deal with the different cases of table 1 in a uniform way, by what amounts to an algorithm that compute G_{σ_n} and ϕ_σ given in terms of elementary properties of any (reasonable) activation function σ.[2] As design of activation functions remains an active industry (for example [Elh+22]), our techniques for computing G_{σ_n} could be useful in future studies of network symmetries.

The following theorem shows that the activation basis is intimately related to the intertwiner group of ReLU: G_{ReLU} admits a natural group-theoretic characterization in terms of the rays spanned by the activation basis, and dually the rays spanned by the activation basis can be recovered from G_{ReLU}. While both its statement and proof are elementary, our interest in this theorem lies in the question of whether it could *potentially* provide theoretical justification for focusing model interpretation studies on individual activations. We investigate this question further in section 6.

Theorem 3.3. *The group G_{ReLU_n} is precisely the stabilizer of the set of rays $\{\mathbb{R}_{\geq 0} e_i \subset \mathbb{R}^n | i = 1, \ldots, n\}$. Moreover if $\mathbb{R}_{\geq 0} v_1, \ldots, \mathbb{R}_{\geq 0} v_N \subseteq \mathbb{R}^n$ is a finite set of rays stabilized by G_{ReLU}, then for each $v_i = [v_{i1}, \ldots, v_{in}]^T$, it must be that $v_{ij} = 0$ for all but one $j \in \{1, \ldots, n\}$. Equivalently up to multiplication by a positive scalar every v_i is of the form $\pm e_j$ for some j.*

3.2 Weight space symmetries

The intertwiner group is also a natural way to describe the weight space symmetries of a neural network. We denote by $\mathcal{F} \subseteq C(\mathbb{R}^{n_0}, \mathbb{R}^{n_k})$ the space of continuous functions that can be described by a network with the same architecture as f. As described in [JGH18] there is a **realization map** $\Phi : \mathcal{W} \to \mathcal{F}$ mapping weights $W \in \mathcal{W}$ to the associated function $f \in \mathcal{F}$. Φ arises because there are generally multiple sets of weights that yield the same function. We will show that Φ is invariant with respect to an action of the intertwiner groups on $\mathcal{W}$ so that intertwiner groups form a set of "built-in" weight space symmetries of f. This result, which encompasses phenomena including

[2] We defer further discussion of and references to this prior work to appendix E.2.

permutation symmetries of hidden neurons, is well known in many particular cases (e.g., [GBC16, §8.2.2], [Bre+19, §3], [FB17, §2], [Men+19, §3], [RK20, §3, A]). From Proposition 3.4 we can also derive corollaries regarding symmetries of the loss landscape — these are included in appendix E.6.

Proposition 3.4. *Suppose $A_i \in G_{\sigma_{n_i}}$ for $1 \leq i \leq k-1$, and let*

$$W' = (A_1 W_1, A_1 b_1, A_2 W_2 \phi_\sigma(A_1^{-1}), A_2 b_2, \ldots, W_k \phi_\sigma(A_{k-1}^{-1}), b_k)$$

Then, as functions, for each m

$$f_{\leq m}(x, W') = \phi_\sigma(A_m) \circ f_{\leq m}(x, W) \text{ and } f_{>m}(x, W') = f_{>m}(x, W) \circ \phi_\sigma(A_m)^{-1}, \tag{3.5}$$

In particular, $f(x, W') = f(x, W)$ for all $x \in \mathbb{R}^{n_0}$. Equivalently, we have $\Phi(W') = \Phi(W) \in \mathcal{F}$.

Remark 3.6. We will show in appendix E that the statement of this theorem must be modified if the architecture of f contains residual connections. By placing suitable restrictions on the matrices A_i[3] we can recover a form of eq. (3.5) provided m occurs at the end of a residual block. However, there doesn't seem to be a way to obtain such an identity when m occurs *inside* a residual block; we see empirical evidence consistent with this point in figs. 2, 6 and 9 below.

4 Intertwining group symmetries and model stitching

In this section we provide evidence that some of the differences between distinct model's internal representations can be explained in terms of symmetries encoded by intertwiner groups. We do this using the stitching framework from [BNB21; Csi+21]. We start by reviewing the concept of network stitching.

Suppose $f, \tilde{f}$ are two networks as in section 3, with weights $W, \tilde{W}$ respectively. For any $1 \leq l \leq k-1$ we may form a **stitched network** $S(f, \tilde{f}, l, \varphi) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_k}$ defined in the notation of section 3 by $S(f, \tilde{f}, l, \varphi) = \tilde{f}_{>l} \circ \varphi \circ f_{\leq l}$ – here $\varphi : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ is a **stitching layer**. In a typical stitching experiment one freezes the weights of networks f and $\tilde{f}$ respectively where f and $\tilde{f}$ were trained from different initializations, constrains φ to some simple function class $\mathcal{S}$ (e.g., affine maps in [BNB21]), and trains $S(f, \tilde{f}, l, \varphi)$ to optimize φ alone. The final validation accuracy $\operatorname{Acc} S(f, \tilde{f}, l, \varphi)$ of $S(f, \tilde{f}, l, \varphi)$ is then considered a measure of similarity (or lack therof) of the internal representations of f and $\tilde{f}$ in $\mathbb{R}^{n_l}$ — in this framework the situation

$$\operatorname{Acc} S(f, \tilde{f}, l, \varphi) \approx \operatorname{Acc} f, \operatorname{Acc} \tilde{f} \tag{4.1}$$

corresponds to high similarity since the hidden representations of model f and $\tilde{f}$ could be related by a transformation $\mathcal{S}$.

Recall that even though the networks $f(W)$ and $f(W')$ may be *equal as functions*, their hidden representations need not be the same (an example of this is given in appendix C). Our next result shows that in the case where f and $\tilde{f}$ do only differ up to an element of $G_{\sigma_{n_l}}$, eq. (4.1) is achievable even when the stitching function class $\mathcal{S}$ is restricted down to elements of $G_{\sigma_{n_l}}$ or $\phi_\sigma(G_{\sigma_{n_l}})$ (see definition 3.2).

Theorem 4.2. *Suppose $\tilde{W} = (A_1 W_1, A_1 b_1, A_2 W_2 \phi_\sigma(A_1^{-1}), A_2 b_2, \ldots, W_k \phi_\sigma(A_k^{-1}), b_k)$ where $A_i \in G_{\sigma_{n_i}}$ for all i. Then eq. (4.1) is achievable with equality* if *the stitching function class $\mathcal{S}$ containing φ contains $\phi_\sigma(G_{\sigma_{n_l}})$.*

Motivated by theorem 4.2, we attempt to stitch various networks at ReLU activation layers using the group G_{ReLU} described in Figure 1. Every matrix $A \in G_{\mathrm{ReLU}}$ can be written as PD, where P is a permutation matrix and D is diagonal with positive diagonal entries — hence optimization over G_{ReLU} requires optimizing over permutation matrices. We use the well-known convex relaxation of permutation matrices to doubly stochastic matrices. See Section D.2 in the Appendix for details related to our optimization procedure.

Figure 1 gives the difference between the average test error of Myrtle CNN [Pag18] networks [4] f and $\tilde{f}$ and the network $S(f, \tilde{f}, l, \varphi)$,

$$\operatorname{Avg}(\operatorname{Acc}(f), \operatorname{Acc}(\tilde{f})) - \operatorname{Acc}(S(f, \tilde{f}, l, \varphi)) \tag{4.3}$$

[3] Namely, that $A_l = A_m$ if layers l and m are joined by a sequence of residual connections.

[4] This is a simple 5-layer CNN, with no residual connections; we describe its architecture and training hyperparameters in further detail in appendix D.

where $S(f, \tilde{f}, l, \varphi)$ was stitched together at layer l via a stitching transformations φ that was either optimized over all affine transformations, reduced rank affine transformations as in [Csi+21] or transformations restricted to G_{ReLU}. We consider only the ReLU activation layers, as these are the only layers where the theory of section 3 applies, and we only act on the channel tensor dimension — in practice, this is accomplished by means of 1-by-1 convolution operations. In particular, with G_{ReLU} we are *only permuting and scaling channels*. Lower values indicate that the stitching layer was sufficient to translate between the internal representation of f at layer l and the internal representation of $\tilde{f}$.

We find that when we are allowed to learn a stitching layer over arbitrary affine transformations of channels, we can nearly achieve the accuracy of the original models. When we are only allow to optimize over G_{ReLU} there is an appreciable increase in test error difference (consistent with findings in [Csi+21; Li+15; Wan+18] discussed in section 2, and also consistent with observations that hidden features of neural networks exhibit distributed representations and polysemanticism [Ola+20]). Nonetheless, that $S(f, \tilde{f}, l, \varphi)$ is able get within less than 10% of the accuracy of f and $\tilde{f}$ in all but one layer suggests that elements of G_{ReLU} can account for a substantial amount of the variation in the internal representations of independently trained networks. We include the reduced rank transformations as the dimension of their parameter spaces is greater than that of G_{ReLU}, and yet they incur significantly higher stitching penalties. If n_l is the number of channels, we have $\dim G_{\mathrm{ReLU}_{n_l}} = n_l$ whereas the dimension of rank r transformations is $2n_l \cdot r - r^2$ (hence greater than $\dim G_{\mathrm{ReLU}_{n_l}}$ even for $r = 1$).[5] Finally, the stitching penalties incurred when using any layer other than 1-by-1 convolution with a rank 1 matrix all follow similar trends, in the specific case of the Myrtle CNNs: they increase up to the third activation layer, then decrease at the final activation layer.

Further stitching results on the ResNet20 architecture can be found in appendix D.3. This includes results where we modify the architecture to have LeakyReLU activation functions, vary the negative slope of the LeakyReLU, and find similar stitching penalties up to but not including a slope of 1, providing further evidence of the importance of the intertwiner group.

5 Dissimilarity measures for the intertwiner group of ReLU

Stitching penalties can be viewed as task oriented measures of hidden feature dissimilarity. From a different perspective, we can consider raw statistical measures of hidden feature dissimilarity. In the design of measures of dissimilarity, a crucial choice is the group of transformations under which the dissimilarity measure is invariant. For example, Centered Kernel Alignment (CKA) [Kor+19] with either a dot product or radial basis function kernel is invariant with respect to orthogonal transformation and isotropic scaling. We ask for a statistical dissimilarity metric μ on datasets $X, Y \in \mathbb{R}^{N \times d}$ with the properties that (0) $0 \leq \mu(X, Y) \leq 1$, (i) (G_{ReLU}-*Invariance*) If $A, B \in G_{\mathrm{ReLU}_d}$ and $v, w \in \mathbb{R}^d$ then $\mu(XA + \mathbf{1}v^T, YB + \mathbf{1}w^T) = \mu(X, Y)$, and (ii) (*Alignment Property*) $\mu(X, Y) = 1$ if $(*)$ $Y = XA + \mathbf{1}v^T$ for some $A \in G_{\mathrm{ReLU}_d}, v \in \mathbb{R}^d$. To motivate this question, we note that given such a metric μ, one can detect if X and Y do *not* differ by an element of G_{ReLU} by checking if $\mu(X, Y) < 1$. Our basic tool for ensuring (i) is the next lemma.

Lemma 5.1. *Suppose $\mu(XA, YB) = \mu(X, Y)$ if A, B are* either *positive diagonal matrices or permutation matrices. Then, (i) holds.*

In effect, this allows us to divide the columns of X and Y by their norms to achieve invariance to the action of positive diagonal matrices and then apply dissimilarity measures for the permutation group such as those presented in [Wil+21]. Ensuring (ii) seems to require case-by-case analysis to determine an appropriate normalization constant.

Definition 5.2 (G_{ReLU}-Procrustes)**.** Let $D_X = \mathrm{diag}(|X_{[:,i]}|)$ and $D_Y = \mathrm{diag}(|Y_{[:,i]}|)$. Assuming these are invertible, let $\tilde{X} = XD_X^{-1}$ and $\tilde{Y} = YD_Y^{-1}$. Let δ be the permutation Procrustes distance between $\tilde{X}, \tilde{Y}$, defined by

$$\delta := \min_{P \in \Sigma_d} |\tilde{X} - \tilde{Y}P|$$

[5] A valid concern is that the preceding analysis underestimates the size of G_{ReLU}, since it ignores a large discrete factor: $G_{\mathrm{ReLU}_{n_l}}$ has $n_l!$ connected components. In appendix E.7 we include a comparison of the size of $G_{\mathrm{ReLU}_{n_l}}$ with that of reduced rank transformations using ϵ-nets, obtaining the same conclusion that even rank 1 transformations are larger.

(as pointed out in [Wil+21] this can be computed via the linear sum assignment problem). Then the G_{ReLU}**-Procrustes measure** is

$$\mu_{\text{Procrustes}}(X, Y) := 1 - \frac{\delta}{2\sqrt{d}}.$$

The factor of $2\sqrt{d}$ ensures this lies in $[0, 1]$, and equals 1 if (and only if) the condition $*$ of (ii) holds.

	layer 3	layer 6	layer 10	layer 14
G_{ReLU}	0.6208 ± 0.008	0.5106 ± 0.005	0.4432 ± 0.004	0.4899 ± 0.002
Orthogonal	0.7724 ± 0.028	0.5743 ± 0.040	0.5087 ± 0.016	0.5825 ± 0.019

Table 2: $\mu_{\text{Procrustes}}$ and orthogonal Procrustes similarities for Myrtle CNNs trained on CIFAR10. Confidence intervals correspond to evaluating similarities for 32 pairs of independently initialized models. For more details on how we sample models we refer to appendix D.1.

We apply G_{ReLU}-Procrustes and orthogonal Procrustes similarities to 4 different hidden representations from Myrtle CNNs in table 2 and many more layers of ResNet20s in fig. 9, all trained on CIFAR10 [Kri09]. In keeping with the discussion of section 4, we only consider permutations or orthogonal transformations of channels (for details on how this is implemented we refer to appendix D.4). We see that distinct representations register less similarity in terms of G_{ReLU}-Procrustes than they do in terms of orthogonal Procrustes. This makes sense as similarity up to G_{ReLU}-transformation requires a greater degree of absolute similarity between representations than is required of similarity up to orthogonal transformation (the latter being a higher-dimensional group containing all of the permutations in G_{ReLU}). Otherwise patterns in G_{ReLU}-Procrustes similaritiy largely follow those of orthogonal Procrustes, with similarity between representations decreasing as one progresses through the network, only to increase again in the last layer. This correlates with the stitching penalties of fig. 1, which increase with depth only to decrease in the last layer.

Definition 5.3 (G_{ReLU}-CKA)**.** Assume that X and Y are data matrices that have been centered by subtracting means of rows: $X \leftarrow X - \frac{1}{d}\mathbf{1}\mathbf{1}^T X$ and $Y \leftarrow Y - \frac{1}{d}\mathbf{1}\mathbf{1}^T Y$. Let $\tilde{X} = XD_X^{-1}$ and $\tilde{Y} = YD_Y^{-1}$. Let $\tilde{x}_1, \ldots, \tilde{x}_N$ be the rows of $\tilde{X}$, and similarly for $\tilde{Y}$. Form the matrices $K, L \in \mathbb{R}_{\geq 0}^{N \times N}$ defined by $K_{ij} = \max(\tilde{x}_i \odot \tilde{x}_j)$ and $L_{ij} = \max(\tilde{y}_i \odot \tilde{y}_j)$ where $\odot$ is the Hadamard product. Then the G_{ReLU}-CKA for X and Y is defined as:

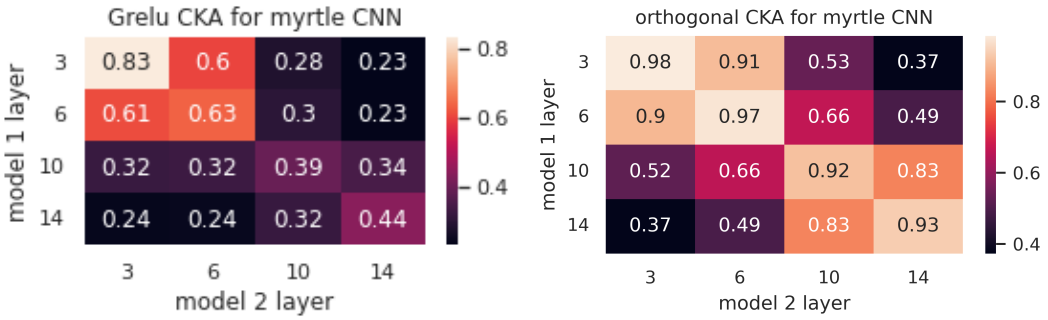
$$\mu_{\text{CKA}}(X, Y) := \frac{\mathrm{HSIC}_1(K, L)}{\sqrt{\mathrm{HSIC}_1(K, K)}\sqrt{\mathrm{HSIC}_1(L, L)}}. \tag{5.4}$$

where HSIC_1 is the unbiased form of the Hilbert-Schmidt independence criterion of [NRK21, eq. 3].

Symmetry of the $\max$ function ensures (i), the Cauchy-Schwarz inequality ensures $\mu_{\text{CKA}}(X, Y) \in [0, 1]$, and we claim that $\mu_{\text{CKA}}(X, Y) = 1$ if the condition $*$ of (ii) is met. We do not claim 'if and only if', however we point out the following in lemma E.44: if A is a matrix such that $\max(Ax_1 \odot Ax_2) = \max(x_1 \odot x_2)$ for all $x_1, x_2 \in \mathbb{R}^d$, then A is of the form PD where P is a permutation matrix and D is diagonal with diagonal entries in $\{\pm 1\}$. In fact, μ_{CKA} is simply an instance of CKA for a the "$\max$ kernel."

Lemma 5.5. *The function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ defined by $k(x, y) = \max(x \odot y)$ is a positive semi-definite kernel.*

As with CKA [Kor+19], this metric makes sense even if X, Y are datasets in $\mathbb{R}^d, \mathbb{R}^{d'}$ respectively with $d \neq d'$. Results for a pair of Myrtle CNNs independently trained on CIFAR10, as well as standard orthogonal CKA for comparison, are shown in fig. 4. Analogous results for ResNet20s are shown in fig. 2. We find that G_{ReLU}-CKA respects basic trends found in their orthogonal counterparts: model layers at the same depth are more similar, early layers are highly similar, and the metric surfaces the block structure of the ResNet in fig. 2: layers inside residual blocks are less similar than those at residual connections. One notable difference for G_{ReLU}-CKA in fig. 2 is that the similarity difference between early and later layers in the orthogonal CKA (discussed for ResNets in [Rag+21]) shown in (b) is less pronounced in (a), and in fact later layers are found to be less similar between runs. We found similar results for stitching in fig. 1.



(a) (b)

Figure 4: G_{ReLU}-CKA and orthogonal CKA for two independently initialized Myrtle CNNs trained on CIFAR10. Results averaged over 16 pairs of independently initialized models (for more details on how we sample models we refer to appendix D.1).

6 Interpretability of the coordinate basis

In this section we explore the confluence of model interpretability and intertwiner symmetries using *network dissection* from [Bau+17]. Network dissection measures alignment between the individual neurons of a hidden layer and single, pre-defined concepts (see appendix F.1 for the methodology). We use an experiment from [Bau+17] to compare the axis-aligned interpretability of hidden activation layers with and without an activation function. Bau et al. compares the interpretability of individual neurons, measured via network dissection, with that of random orthogonal rotations of neurons. We likewise rotate the hidden layer representations and then measure their interpretability. Using the methodology from [Dia05], we define a random orthogonal transform Q drawn uniformly from $SO(n)$ by using Gram-Schmidt to orthonormalize the normally-distributed $QR = A \in \mathbb{R}^{n^2}$. Like in [Bau+17], we also consider smaller rotations $Q^\alpha \in SO(n)$ where $0 \leq \alpha \leq 1$, where α is chosen to form a minimal geodesic rotating from I to Q. [Bau+17] found that the number of interpretable units decreased away from the activation basis as α increased for layer5 of an AlexNet.

We compare three models trained on ImageNet: a normal ResNet-50, a modified ResNet-50 where we remove the ReLU on the residual outputs (training details in appendix F.3), and a ConvNeXt [Liu+22] analog of the ResNet-50, which also does not have an activation function before the final residual output. We give results in fig. 3, and provide sample unit detection outputs and full concept labels for the figures in appendix F.2. As was shown in [Bau+17], interpretability decreases as we rotate away from the axis for the normal ResNet-50 in appendix F.3. On the other hand, with no activation function and therefore no canonical basis, neuron interpretability does not drop with rotation for the modified ResNet-50 and the ConvNeXt. Interestingly, while the number of interpretable units remains constant for the residual output of the modified ResNet-50, for the ConvNeXt model it actually *increases*. We find similar results, where the number of interpretable units increase with rotation, for the convolutional layer inside the residual block for the modified ResNet-50 in fig. 15.

7 Limitations

Our theoretical analysis does not account for standard regularization techniques that are known to have symmetry-breaking effects (for example weight decay reduces scaling symmetry). More generally, we do not account for any implicit regularization of our training algorithms. As illustrated in figs. 1 and 6, stitching with intertwiner groups appears to have significantly more architecture-dependent behaviour than stitching with arbitrary affine transformations (however, since different architectures have different symmetries this is to be expected). Our empirical tests of the dissimilarity measures in section 5 are limited to what [Kor+19] terms "sanity tests"; in particular we did not perform the specificity, sensitivity and quality tests of [DDS21a].

8 Conclusion

In this paper we describe groups of symmetries that arise from the nonlinear layers of a neural network. We calculate these symmetry groups for a number of different types of nonlinearities, and explore their fundamental properties and connection to weight space symmetries. Next, we provide evidence that these symmetries induce symmetries in a network's internal representation of the data that it processes, showing that previous work on the internal representations of neural networks can be naturally adapted to incorporate awareness of the intertwiner groups that we identify. Finally, in the special case where the network in question has ReLU nonlinearities, we find experimental evidence that intertwiner groups justify the special place of the activation basis within interpretable AI research.

References

[Bau+17] David Bau et al. "Network Dissection: Quantifying Interpretability of Deep Visual Representations". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 3319–3327.

[BMC15] Vijay Badrinarayanan, Bamdev Mishra, and R. Cipolla. "Understanding Symmetries in Deep Networks". In: *ArXiv* (2015).

[BNB21] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. "Revisiting Model Stitching to Compare Neural Representations". In: *NeurIPS*. 2021.

[Bre+19] Johanni Brea et al. "Weight-Space Symmetry in Deep Networks Gives Rise to Permutation Saddles, Connected by Equal-Loss Valleys across the Loss Landscape". In: (July 5, 2019). arXiv: 1907.02911 [cs, stat]. URL: http://arxiv.org/abs/1907.02911 (visited on 01/12/2022).

[Csi+21] Adrián Csiszárik et al. "Similarity and Matching of Neural Network Representations". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021. URL: https://openreview.net/forum?id=aedFIIRRfXr.

[DDS21a] Frances Ding, Jean-Stanislas Denain, and J. Steinhardt. "Grounding Representation Similarity with Statistical Testing". In: *ArXiv* (2021).

[DDS21b] Frances Ding, Jean-Stanislas Denain, and Jacob Steinhardt. "Grounding Representation Similarity Through Statistical Testing". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021. URL: https://openreview.net/forum?id=_kwj6V53ZqB.

[Den+09] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[Dia05] Persi Diaconis. "What is a random matrix". In: *Notices of the AMS* 52.11 (2005), pp. 1348–1349.

[Elh+21] N Elhage et al. *A mathematical framework for transformer circuits*. 2021.

[Elh+22] Nelson Elhage et al. "Softmax Linear Units". In: *Transformer Circuits Thread* (2022).

[Erh+09] Dumitru Erhan et al. "Visualizing higher-layer features of a deep network". In: *University of Montreal* 1341.3 (2009), p. 1.

[FB17] C. Daniel Freeman and Joan Bruna. "Topology and Geometry of Half-Rectified Network Optimization". In: *ArXiv* abs/1611.01540 (2017).

[Fog+13] Fajwel Fogel et al. "Convex Relaxations for Permutation Problems". In: *SIAM J. Matrix Anal. Appl.* 2013.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[HG16] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2016. DOI: 10.48550/ARXIV.1606.08415. URL: https://arxiv.org/abs/1606.08415.

[IS15] S. Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *ICML* (2015).

[JGH18] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural tangent kernel: convergence and generalization in neural networks (invited paper)". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (2018).

[Kor+19] Simon Kornblith et al. "Similarity of Neural Network Representations Revisited". In: *ICML* (2019).

[Kri09] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[KTB19] J. Kileel, Matthew Trager, and Joan Bruna. "On the Expressive Power of Deep Polynomial Neural Networks". In: *NeurIPS*. 2019.

[Kun+21] Daniel Kunin et al. *Neural Mechanics: Symmetry and Broken Conservation Laws in Deep Learning Dynamics*. Mar. 29, 2021. arXiv: 2012.04728 [cond-mat, q-bio, stat]. URL: http://arxiv.org/abs/2012.04728 (visited on 01/12/2022).

[Lec+22] Guillaume Leclerc et al. *ffcv*. https://github.com/libffcv/ffcv/. commit 849. 2022.

[Li+15] Yixuan Li et al. "Convergent Learning: Do Different Neural Networks Learn the Same Representations?" In: *FE@NIPS*. 2015.

[Liu+22] Zhuang Liu et al. "A ConvNet for the 2020s". In: *arXiv preprint arXiv:2201.03545* (2022).

[LV15] Karel Lenc and A. Vedaldi. "Understanding Image Representations by Measuring Their Equivariance and Equivalence". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015). DOI: 10.1109/CVPR.2015.7298701.

[LW14] Cong Han Lim and Stephen J. Wright. "Beyond the Birkhoff Polytope: Convex Relaxations for Vector Permutation Problems". In: *NIPS*. 2014.

[Men+18] Gonzalo Mena et al. "Learning Latent Permutations with Gumbel-Sinkhorn Networks". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=Byt3oJ-0W.

[Men+19] Qi Meng et al. "G-SGD: Optimizing ReLU Neural Networks in Its Positively Scale-Invariant Space". In: *ICLR*. 2019.

[MR10] Sébastien Marcel and Yann Rodriguez. "Torchvision the machine-vision package of torch". In: *Proceedings of the 18th ACM international conference on Multimedia*. 2010, pp. 1485–1488.

[Na+19] Seil Na et al. "Discovery of natural language concepts in individual units of cnns". In: *arXiv preprint arXiv:1902.07249* (2019).

[NH10] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *ICML*. 2010.

[Noe18] E Noether. "Invariante Variationsprobleme". In: 1918, pp. 235–257.

[NRK21] Thao Nguyen, Maithra Raghu, and Simon Kornblith. "Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=KJNcAkY8tY4.

[Ola+20] Chris Olah et al. "Zoom in: An introduction to circuits". In: *Distill* 5.3 (2020), e00024–001.

[Pag18] David Page. *How to Train Your ResNet*. Myrtle. Sept. 24, 2018. URL: https://myrtle.ai/learn/how-to-train-your-resnet/ (visited on 05/09/2022).

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[PE20] Sebastian Prillo and Julian Martin Eisenschlos. "SoftSort: A Continuous Relaxation for the argsort Operator". In: *ICML*. 2020.

[Rag+17] M. Raghu et al. "SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability". In: *NIPS*. 2017.

[Rag+21] Maithra Raghu et al. "Do vision transformers see like convolutional neural networks?" In: *Advances in Neural Information Processing Systems* 34 (2021).

[RK20] D. Rolnick and Konrad Paul Kording. "Reverse-Engineering Deep ReLU Networks". In: *ICML*. 2020.

[Rud76] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 1976. 342 pp. ISBN: 978-0-07-085613-4. Google Books: kwqzPAAACAAJ.

[Ser77] Jean-Pierre Serre. *Linear representations of finite groups*. Vol. 42. Springer, 1977.

[Sin64] Richard Sinkhorn. "A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices". In: *The Annals of Mathematical Statistics* 35.2 (1964), pp. 876–879. DOI: 10.1214/aoms/1177703591. URL: https://doi.org/10.1214/aoms/1177703591.

[Sze+14] Christian Szegedy et al. "Intriguing properties of neural networks". In: *CoRR* abs/1312.6199 (2014).

[Vir+20] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[Wan+18] Liwei Wang et al. "Towards Understanding Learning Representations: To What Extent Do Different Neural Networks Learn the Same Representation". In: *NeurIPS*. 2018.

[Wig19] Ross Wightman. *PyTorch Image Models*. https://github.com/rwightman/pytorch-image-models. 2019. DOI: 10.5281/zenodo.4414861.

[Wil+21] Alex H. Williams et al. "Generalized Shape Metrics on Neural Representations". In: *NeurIPS*. 2021.

[Yi+19] Mingyang Yi et al. "Positively Scale-Invariant Flatness of ReLU Neural Networks". In: *ArXiv* (2019).

[Yos+15] Jason Yosinski et al. "Understanding Neural Networks Through Deep Visualization". In: *Deep Learning Workshop, International Conference on Machine Learning (ICML)*. 2015.

[ZF14] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[Zho+15] Bolei Zhou et al. "Object Detectors Emerge in Deep Scene CNNs." In: *ICLR*. 2015. URL: http://arxiv.org/abs/1412.6856.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 7.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] This paper is largely focused on the mathematical aspects of deep learning so we do not think there are any immediate negative societal impact to the methods described. From a broader perspective though, we see this work helping to create a more principled groundwork for many explainable AI techniques. We explain why this could have positive societal impacts in Section A.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs can be found in appendix E.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? **[TODO]** We are in the process of making code publicly available.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See appendix D — while we did not keep precise track of CPU/GPU hours, we do specify the hardware used.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See Section G.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Societal Impact

Though deep learning models are in the process of being deployed for safety critical applications, we still have very little understanding of the structure and evolution of their internal representations. In this paper we discuss one aspect of these representations. We hope that by better illuminating the inner workings of these networks, we will be a small part of the larger effort to make deep learning more understandable, reliable, and fair.

B Code availability

We are in the process of releasing code — once available it will be located at https://github.com/TvfBaRoZCKqJ/FjEsAbqqjdeY.

C Examples

We first give an example of two networks with distinct weights which are functionally equivalent. Let f be a 2 layer network with ReLU activations and weight matrices

$$W_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \text{and} \quad W_2 = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

(and biases = 0). Let $\tilde{f}$ be a network with the same architecture, but with weights

$$W_1 = \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad W_2 = \begin{bmatrix} 0 & 3 \\ 1 & 0 \end{bmatrix}.$$

Then one can verify that $\tilde{f}(x) = f(x)$ for all $x \in \mathbb{R}$, but that the weights of f and $\tilde{f}$ differ.

We also work through a small example of ϕ_{σ_n} where $n = 2$. Assume that σ is the ReLU nonlinearity. Then,

$$A = \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix}$$

belongs to G_{σ_2}, and we can compute directly that

$$\mathrm{ReLU} \circ \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \mathrm{ReLU}(x_2) \\ \mathrm{ReLU}(2x_1) \end{bmatrix} = \begin{bmatrix} \mathrm{ReLU}(x_2) \\ 2\,\mathrm{ReLU}(x_1) \end{bmatrix},$$

where in the last equality we used the fact that $\mathrm{ReLU}(ax) = a\,\mathrm{ReLU}(x)$ when a is positive. On the other hand,

$$\begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix} \circ \mathrm{ReLU}(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{ReLU}(x_1) \\ \mathrm{ReLU}(x_2) \end{bmatrix} = \begin{bmatrix} \mathrm{ReLU}(x_2) \\ 2\,\mathrm{ReLU}(x_1) \end{bmatrix}.$$

D Experimental Details

In this section we provide additional experimental results, as well as implementation details for the purposes of reproducibility. All experiments were run on an Nvidia GPUs using PyTorch [Pas+19].

D.0.1 A "sanity test" for intertwiners

We include a simple experiment related to section 3. In this experiment, we take an even simpler Myrtle CNN (with no batch normalization) trained for 50 epochs, fix a *pre-activation layer* l, and apply a transformation A to the weights W_l and biases b_l to obtain AW_l and Ab_l (again, we only act on channels, hence in practice this is implemented by an auxiliary 1-by-1 convolution layer). We consider 2 choices of A:

- A random element of G_{ReLU} where we generate P via a random permutation and we generate D via a diagonal matrix with random lognormal entries (normalized to ensure that the expected value of D is the identity).

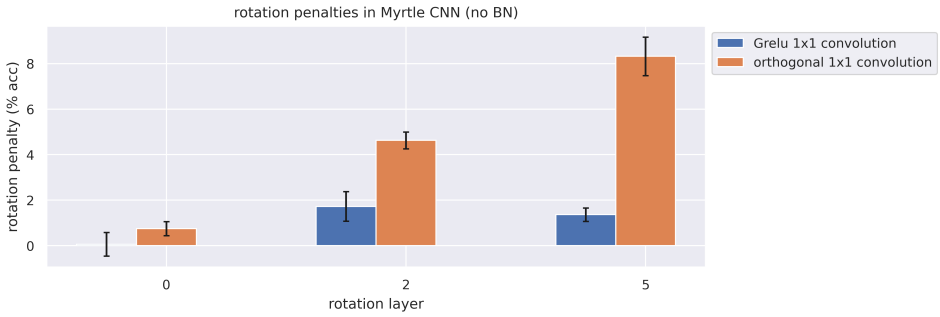


Figure 5: "Rotation penalties" for a simple Myrtle CNN, measured in baseline validation accuracy (before transforming layer l minus validation accuracy after fine-tuning later layers. Baseline accuracy was $\approx 81\%$. Results averaged over 10 independent runs of the experiment.

- A random orthogonal matrix, generated using the QR decomposition as in section 6.

Next, we freeze layers up to and including l and finetune the later layers for another 50 epochs. Based on the theory in section 3, when $A \in G_{\mathrm{ReLU}}$ the network should be able to recover reasonable performance even with the transformed features — for example, by updating W_{l+1} to $W_{l+1}\phi_\sigma(A)^{-1}$. On the other hand, with probability 1 there is no matrix B such that updating W_{l+1} to $W_{l+1}B$ counteracts the effect of an orthogonal rotation A on all possible input. We see that this is indeed the case in fig. 5: transforming by $A \in G_{\mathrm{ReLU}}$ rather a random orthogonal matrix results in significantly higher fine-tuned accuracy.

D.1 Sampling pairs of independently initialized models

We began by training 100 independently initialized models for each of the following architectures:

(i) Myrtle CNN: a simple 5-layer feed-forward CNN with batch normalization.

(ii) ResNet20: a ResNet tailored to the CIFAR10 dataset (numbers of channels are $16, 32, 64$ respectively in the 3 residual blocks).

(iii) ResNet18: an ImageNet-style ResNet adapted to the input size of CIFAR10 — much wider than the above (numbers of channels are $64, 128, 256$ respectively in the 3 residual blocks).

More detailed architecture schematics are included in figs. 22a, 23a and 24a.

All models were trained for 50 epochs using the Adam optimizer with PyTorch's default settings. We use a batch size of 32, initial learning rate 0.001 and 4 evenly spaced learning rate drops with factor 0.5. We augment data with translations of up to 2 pixels (padded as necessary with the mean RGB value for CIFAR10) and left-right flips, and we save the weights with best validation accuracy. Training this many CIFAR10 models on a reasonable budget of time and computing resources was greatly aided by the excellent FFCV library [Lec+22].

In the later stitching and dissimilarity measure experiments, we sample pairs of models from these "zoos" uniformly with replacement (but of course making sure that the two models in the pair are distinct). Thus the cost of training hundreds of models is amortized across many runs of stitching and dissimilarity measurement; this can be also viewed as bootstrap estimation of our experimental quantities of interest using empirical samples from certain distributions of CIFAR10 models.

D.2 Stitching Experiments

For stitching layers, we train for 20 epochs with batch size 32 and learning rate 0.001 (with no drops), however we use vanilla SGD with no momentum (we found the approximate second-order and/or momentum aspects of Adam interacted in complicated ways with the PGD algorithm described in appendix D.2.1 below, even after following some helpful advice from the Internet[6]). Augmentation is described in the previous paragraph.

[6]https://datascience.stackexchange.com/questions/31709/adam-optimizer-for-projected-gradient-descent

We parameterize reduced rank 1-by-1 convolutions as a composition of 2 1-by-1 convolutions, with `in_channels`, `out_channels` = `in_channels`, `rank` and `rank`, `in_channels` respectively. In contrast to [BNB21] we omit both batch norm and bias from stitching layers (to stick closely to the statement of theorem 4.2).

D.2.1 Approximate Optimization over Permutation Matrices

By far the most complicated stitching layer is the one using G_{ReLU}, which we describe here. Recall that G_{ReLU} is equal to the $n \times n$ matrices of the form PD, where $P \in \Sigma_n$ is a permutation matrix and D is a diagonal matrix with positive entries We parameterize D simply as $D = \mathrm{diag}(\lambda_i)$ where $\lambda_1, \ldots, \lambda_{n_l} \in \mathbb{R}_{\geq 0}$ — we preserve non-negativity during training by a projected gradient descent step $D \leftarrow \mathrm{ReLU}(D)$. During stitching layer training, we parameterize P as a doubly stochastic matrix, that is, an element of the Birkhoff polytope

$$\mathcal{B} = \{A = (a_{ij}) \in \mathrm{Mat}_{n_l,n_l}(\mathbb{R}) \,|\, a_{ij} \geq 0 \text{ for all } i, j, \mathbf{1}^T A = \mathbf{1}^T \text{ and } A\mathbf{1} = \mathbf{1}\}$$

— after each gradient descent step we project P back onto $\mathcal{B}$ by the operation $P \leftarrow \mathrm{ReLU}(P)$ followed by $P \leftarrow \mathrm{sink}(P)$, where "sink" denotes Sinkhorn iterations. These consist of T iterations of

$$A \leftarrow A\,\mathrm{diag}(\mathbf{1}^T A)^{-1} \text{ followed by } A \leftarrow \mathrm{diag}(A\mathbf{1})^{-1} A$$

(it is a theorem of Sinkhorn that this sequence converges to a doubly stochastic matrix of the form DAE with D, E positive diagonal matrices [Sin64]). We use $T = 16$ in all experiments (this choice drew on the work of [Men+18]). In addition, we add a regularization term $-\alpha|P|_2$ to the stitching objective, where $\alpha > 0$ is a hyperparameter (the motivation here is that permutation matrices are precisely the elements of $\mathcal{B}$ with maximal ℓ_2-norm). In all our experiments $\alpha = 0.1$.

At evaluation time, we threshold P to an actual permutation matrix via the Hungarian algorithm (specifically its implementation in scipy.optimize.linear_sum_assignment [Vir+20]). This amounts to

$$P_{\text{eval}} = \max_{Q \in \Sigma_{n_l}} \mathrm{tr}(P_{\text{train}} Q^T)$$

As stated above, we train for 20 epochs with batch size 32 and learning rate 0.001 (with no drops), using SGD with momentum 0.9. However, we allow the permutation factor to get a "head start" by keeping D fixed at the identity I for the first 10 epochs. This may not be essential. More generally, *there is no reason to assume any of our stitching hyperparameters (T, α, learning rate, head start etc.) are optimal as we did not perform any thorough hyperparameter sweeps*.

Finally, before evaluating the stitched model on the CIFAR10 validation set, we perform a no-gradient epoch *on the training data* with stitching layer P_{eval}. This is critical as it allows the batch normalization running means and variances in later layers to adapt to the thresholded permutation matrix P_{eval}; observe that if we omitted this step, during evaluation the "batch normalization layers" would not even be performing batch normalization per se, since their running statistics would be computed from features produced by a layer P_{train} no longer in use.

As an aside, we also experimented with the differeniable relaxation of permutation matrices SoftSort [PE20]. Our final results were comparable, however this method took far longer ($> 10\times$) to optimize than the Birkhoff polytope method. It is perhaps of interest that we used SoftSort on permutations far larger than those of [PE20] (e.g., the 512 channels of late layers of our Myrtle CNN). The next section (appendix D.2.2) contains some of our technical findings.

We wish to aknowledge a couple articles, [Fog+13] and [LW14], that provided us with useful backround on optimization over doubly stochastic matrices.

D.2.2 Stitching with SoftSort

We parameterized D simply as $D = \mathrm{diag}(e^{\lambda_i})$ where $\lambda_1, \ldots, \lambda_{n_l} \in \mathbb{R}$. During stitching layer training, we parameterized P using SoftSort [PE20], a continuous relaxation of permutation matrices given by the formula

$$P = \mathrm{SoftSort}(s, \tau) := \mathrm{softmax}\big(-\frac{1}{\tau}(\mathrm{sort}(s)\mathbf{1}^T - \mathbf{1}s^T)\big), \text{ where } s \in \mathbb{R}^{n_l},$$

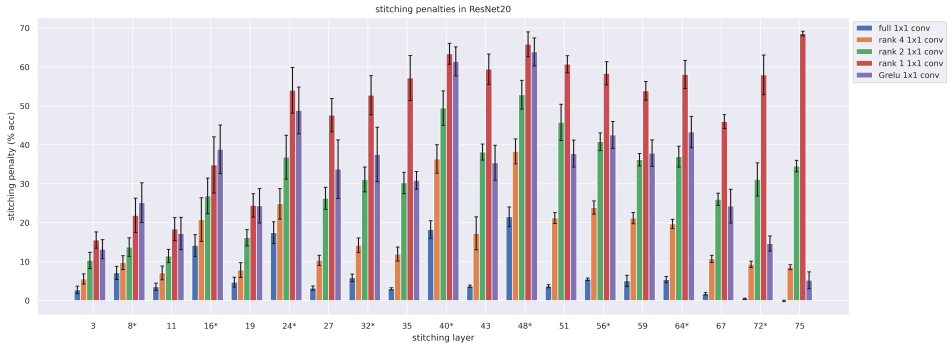


Figure 6: Full/reduced rank and G_{ReLU} 1-by-1 convolution stitching penalties (4.3) for two independently initialized ResNet20s on CIFAR-10. Confidence intervals were taken over 5 independent runs. Accuracy of the independently initialized models was 89.6 $\pm$ 1.2 %. Layers marked with '*' occur inside residual blocks (remark 3.6).

$\mathrm{sort}(s)$ denotes s sorted in descending order, and $\mathrm{softmax}$ is applied over rows. The parameter $\tau > 0$ controls $\mathrm{softmax}$ temperature, and we were only able to obtain reasonable results when tuning it according to $\tau \approx 1/n_l$. At validation time, we threshold P to an actual permutation matrix by applying $\arg\max$ over rows as in [PE20].

D.3 Stitching and G_{ReLU}-dissimilarity measures for ResNets

Here we include further results for ResNet20 and ResNet18 architectures. Figure 6 and fig. 7 include results for full 1-by-1 convolution, reduced randk 1-by-1 convolution and G_{ReLU} 1-by-1 convolutions stitching in the ResNet20 and ResNet18 architectures respectively. Note that in general, layers inside residual blocks incur higher penalties, consisent with remark 3.6. This holds even in the full 1-by-1 convolution case, a finding that to the best of our knowledge is new.

In the case of ResNet20 we also observe that the relative ranking of the different stitching constraints tends to change inside of residual blocks: whereas G_{ReLU} stitching consistently outperforms rank 1 (and sometimes rank 2) stitching outside residual blocks, it consistently underperforms all strategies inside residual blocks. Lastly, we remark that the ResNet20 is significantly narrower than the Myrtle CNN (channels are 16, 32, 64 vs. 64, 128, 256, see figs. 23a and 24a), and hence the low-rank transformations account for a larger *proportion* of the available total rank (for example, in early layers of the ResNet20 rank 4 is $0.25 \cdot \mathrm{fullrank}$ whereas in the early layers of the Myrtle CNN rank 4 is $0.0625 \cdot \mathrm{fullrank}$). Heuristically, in the narrower network low-rank transformations may suffice to align for a larger fraction of the principal components of hidden features.

We also observe generally lower stitching penalties in the ResNet18 with the exception of the penultimate inside-a-residual-block layer — we do not have a satisfactory explanation for random chance performance at that layer. We also remark that while the penalties in fig. 6 are significantly higher than those in fig. 1, especially in later layers, we also saw significant dissimilarity in fig. 2 (a), especially in later layers.

We also modify the ResNet20 to use the $\mathrm{LeakyReLU}$ activation function and train models with different negative slopes s. Accuracy for two independently trained models at different $\mathrm{LeakyReLU}$ given in table 3. We perform G_{ReLU} stitching in fig. 6. Note that for a negative slope $s = 1$, the activation function is the identity. We find the results difficult to interpret due to the significant decrease in CIFAR-10 accuracy for larger s. With this being said, unlike for $s << 1$, we note that the stitching penalties for $s = 1$ (and to a lesser extent, $s = 0.9$) are mostly constant throughout the layers of the network. This is most prominent for the final two ResNet20 layers (72 and 75), where the stitching penalty for models with small $\mathrm{LeakyReLU}$ slopes is the lowest.

Figure 9 contains G_{ReLU} and orthogonal Procrustes dissimilarities for the ResNet20. The 2 measures seem qualitatively quite similar in this case. For the most part the same applies to the ResNet18 in fig. 10, with the exception of layer 70 (penultimate inside-a-residual-block layer), where we see high G_{ReLU} *similarity*, in conflict with both fig. 7 and fig. 11 below.

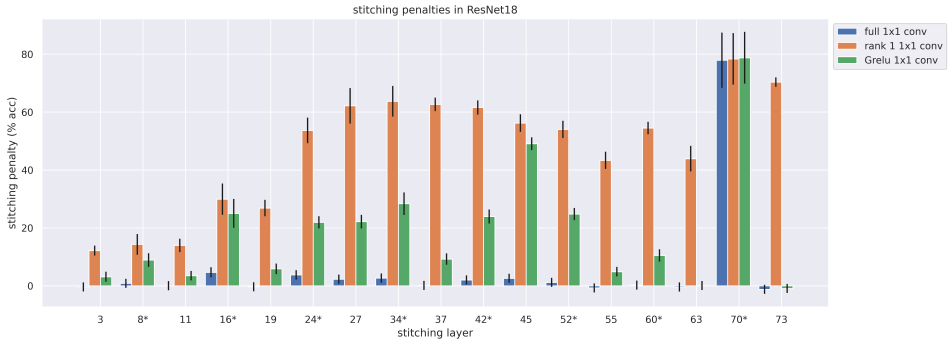


Figure 7: Full/reduced rank and G_{ReLU} 1-by-1 convolution stitching penalties (4.3) for two independently initialized ResNet18s on CIFAR-10. Confidence intervals were taken over 5 independent runs. Accuracy of the independently initialized models was 91.0 ± 1.6 %. Layers marked with '*' occur inside residual blocks (remark 3.6). Rank 2 and 4 cases are omitted only since those experiments did not return in time for submission (they could easily be included in a revision).

Table 3: ResNet20 with LeakyReLU CIFAR-10 accuracy

	LeakyReLU SLOPE						
	1e−4	1e−3	1e−2	0.1	0.5	0.9	1.0
% acc.	89.3 ± 0.2	89.4 ± 0.2	89.2 ± 0.2	89.4 ± 0.1	86.6 ± 0.1	73.0 ± 0.2	41.8 ± 0.1

We include G_{ReLU} and orthogonal CKA dissimilarities for the wider ResNet18 in fig. 11. For the most part the qualitative remarks on fig. 2 apply here as well — note also the extreme dissimilarity in layer 70 (in both G_{ReLU} and orthogonal cases) consistent with fig. 7.

D.4 Implementing dissimilarity measures

As mentioned in section 5, we aim to capture invariants to permuting and scaling channels, but not spatial coordinates. This requires some care; practically speaking it means we cannot simply flatten feature vectors.

In all cases we compute our measures over the entire CIFAR10 validation set. In particular, we do not require batched computations as in [NRK21].

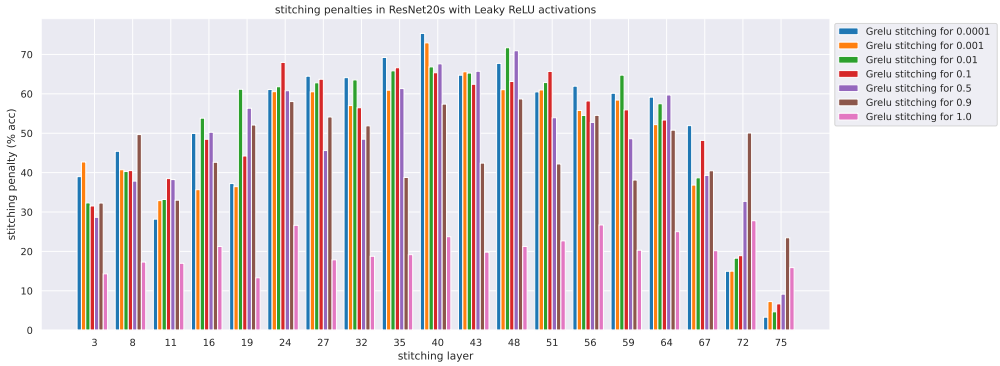


Figure 8: Stitching penalties (4.3) for independently initialized ResNet20s on CIFAR-10, where respective ResNet20 models are trained with LeakyReLU activation functions with different slopes. Accuracy of the independently initialized models with different LeakyReLU slopes is given in table 3.

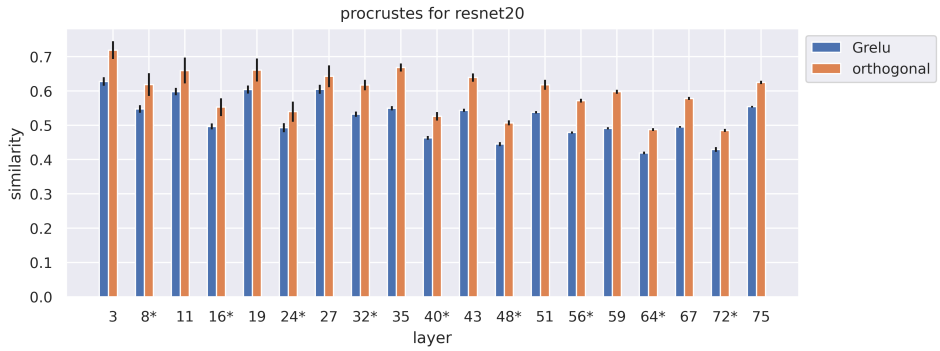


Figure 9: G_{ReLU} and orthogonal Procrustes dissimilarities for two independently initialized ResNet20s trained on CIFAR10. Layers marked with '*' occur inside residual blocks (remark 3.6). Confidence intervals correspond to evaluating similarities for 32 pairs of independently initialized models.

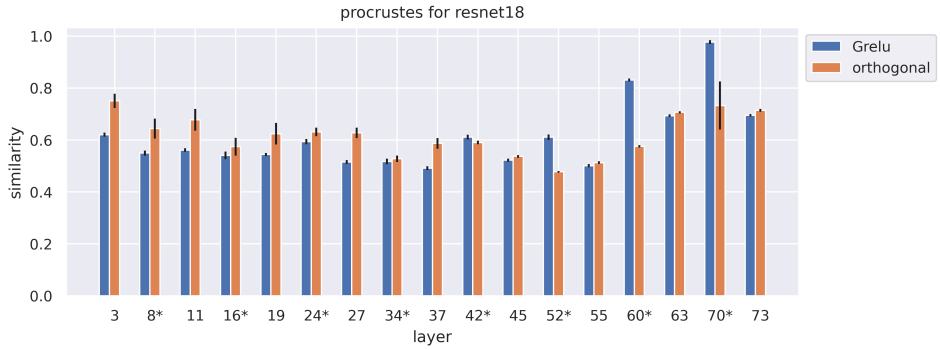


Figure 10: G_{ReLU} and orthogonal Procrustes dissimilarities for two independently initialized ResNet18s trained on CIFAR10. Layers marked with '*' occur inside residual blocks (remark 3.6). Confidence intervals correspond to evaluating similarities for 32 pairs of independently initialized models.

D.4.1 Procrustes

As in [DDS21b; Wil+21]

$$\min_{P \in \Sigma_d} |\tilde{X} - \tilde{Y}P| = \sqrt{\min_{P \in \Sigma_d} |\tilde{X} - \tilde{Y}P|^2}$$

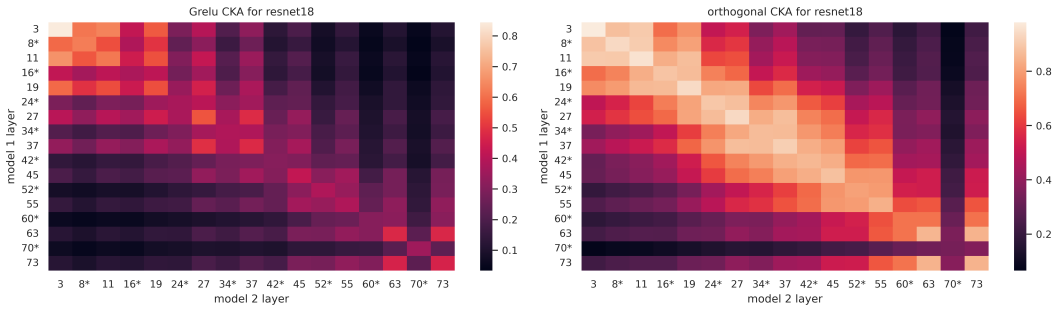
so it suffices to consider minimizing the Frobenius norm-squared, and expanding as

$$|\tilde{X} - \tilde{Y}P|^2 = |\tilde{X}|^2 + |\tilde{Y}|^2 - 2\,\mathrm{tr}(\tilde{X}^T\tilde{Y}P)$$

we see that this is equivalent to *maximizing* $\mathrm{tr}(\tilde{X}^T\tilde{Y}P)$. In our case X, Y have shape (N, C, H, W) where N is the size of the entire CIFAR10 validation set and C, H, W are the channels, height, and width at the given hidden layer respectively. We want P to be a $C \times C$ permutation matrix. Hence for $\tilde{X}^T\tilde{Y}$ we compute the tensor dot product

$$(\tilde{X}^T\tilde{Y})_{c,c'} = \sum_{n,h,w} \tilde{X}_{n,c,h,w}\tilde{Y}_{n,c',h,w} \tag{D.1}$$

The same method is used for orthogonal Procrustes, where instead of scipy.optimize.linear_sum_assignment we use the nuclear norm of eq. (D.1) as in [DDS21a].



(a) (b)

Figure 11: G_{ReLU}-CKA and orthogonal CKA for two independently initialized ResNet18s trained on CIFAR10. Layers marked with '*' occur inside residual blocks (remark 3.6). Results averaged over 16 pairs of independently initialized models.

D.4.2 CKA

In this case for a set of hidden features X of shape (N, C, H, W) as above, we first subtract the mean over all but the channel dimension:

$$X_{n,c,h,w} \leftarrow X_{n,c,h,w} - \frac{1}{NHW} \sum_{n',h',w'} X_{n',c,h',w'}$$

and divide by the norms over all but the channel dimension:[7]

$$X_{n,c,h,w} \leftarrow \frac{X_{n,c,h,w}}{\sqrt{\sum_{n',h',w'} X^2_{n',c,h',w'}}}.$$

Next, we compute a tensor dot product of X with itself *over spatial dimensions*, to obtain the shape (N, N, C) tensor

$$J_{m,n,c} := \sum_{h,w} X_{n,c,h,w} X_{n,c,h,w}$$

and finally we apply $\max$ over the channel dimension to get

$$K_{m,n} = \max_c J_{m,n,c}.$$

Remark D.2. It could be interesting to refrain from applying a dot product over spatial dimensions, and thus measure not only similarity between hidden features of different images, but similarity between hidden features of different images *at certain locations*. However, the memory requirements would have been far beyond our computational limits.

E Proofs

E.1 A proof of lemma 3.1, plus some abstractions thereof

Proof of lemma 3.1. Since by definition $G_{\sigma_n} \subseteq GL_n(\mathbb{R})$, to prove G_{σ_n} is a subgroup it suffices to show that if $A_1, A_2 \in G_{\sigma_n}$ then $A_1 A_2^{-1} \in G_{\sigma_n}$. By hypotheses, there are matrices $B_1, B_2 \in GL_n(\mathbb{R})$ so that

$$\sigma_n \circ A_1 = B_1 \circ \sigma_n \tag{E.1}$$

$$\text{and} \quad \sigma_n \circ A_2 = B_2 \circ \sigma_n. \tag{E.2}$$

Applying A_2^{-1} on the right hand side of eq. (E.1) gives

$$\sigma_n \circ (A_1 A_2^{-1}) = B_1 \circ \sigma_n \circ (A_2^{-1}). \tag{E.3}$$

[7] In retrospect, it would arguably make more sense to use standard deviation rather than ℓ_2 norm; however, for us the choice is irrelevant in the end since the 2 choices differ by a factor of $\sqrt{NHW}$ which gets cancelled in eq. (5.4).

On the other hand, applying A_2^{-1} on the right hand side of eq. (E.2) gives $\sigma_n = B_2 \circ \sigma_n \circ (A_2^{-1})$ and hence

$$\sigma_n \circ (A_2^{-1}) = B_2^{-1} \circ \sigma_n. \tag{E.4}$$

Combining eqs. (E.3) and (E.4) we obtain

$$\sigma_n \circ (A_1 A_2^{-1}) = B_1 B_2^{-1} \circ \sigma_n \tag{E.5}$$

and hence G_{σ_n} is a subgroup. Next, we solve

$$\sigma_n \circ A = B \circ \sigma_n$$

for B in terms of A by evaluating both sides at $e_1, \ldots, e_n \in \mathbb{R}^n$ (standard basis vectors). Letting $A[:, j]$ denote the j-th column of A we obtain

$$\sigma_n(A[:, j]) = B\sigma_n(e_j), \text{ for } j = 1, \ldots, n$$

and stacking these columns to obtain the full $n \times n$ matrix yields

$$\sigma(A) = B\sigma(I)$$

where $I \in GL_n(\mathbb{R})$ is the identity matrix and $\sigma(A)$ denotes σ applied to the coordinates of A (similarly for $\sigma(I)$). As $\sigma(I)$ is invertible by hypotheses, this implies $B = \sigma(A)\sigma(I)^{-1} =: \phi_\sigma(A)$, and finally substituting $B_i = \phi_\sigma(A_i)$ for $i = 1, 2$ in eq. (E.5) shows that

$$\sigma_n \circ (A_1 A_2^{-1}) = \phi_\sigma(A_1)\phi_\sigma(A_2)^{-1} \circ \sigma_n$$

while at the same time

$$\sigma_n \circ (A_1 A_2^{-1}) = \phi_\sigma(A_1 A_2^{-1}) \circ \sigma_n$$

so that $\phi_\sigma(A_1 A_2^{-1}) \circ \sigma_n = \phi_\sigma(A_1)\phi_\sigma(A_2)^{-1} \circ \sigma_n$. Using the invertibility of $\sigma(I)$ one more time, we conclude

$$\phi_\sigma(A_1 A_2^{-1}) = \phi_\sigma(A_1)\phi_\sigma(A_2)^{-1},$$

which implies ϕ_σ is a homomorphism. □

Remark E.6 (for the mathematically inclined reader). Here is a more abstract definition of G_{σ_n} that makes lemma 3.1 appear more natural: let X be a topological space with a continuous (left) action of a topological group G. There is a natural (right) action of G on $C(X, \mathbb{R})$ by precomposition ($(f, g) \mapsto f \circ g$). For any subspace $V \subseteq C(X, \mathbb{R})$ define

$$G_V := \{g \in G \,|\, V \cdot g \subseteq V\}$$

(that is, the elements of G stabilize V *as a subspace*, but not necessarily pointwise — one can show this is always a subgroup of G). Then, for every such subspace V, the group G_V acts linearly on V, and if we have a basis $f_1, \ldots, f_n \in V$, we can obtain a matrix representation of G in $GL_n(\mathbb{R})$. To obtain the special case in lemma 3.1, we take $X = \mathbb{R}^n$, $G = GL_n(\mathbb{R})$ with the usual action, and V to be the subspace spanned by the functions $f_i(x_1, \ldots, x_n) = \sigma(x_i)$. The condition that $\sigma(I)$ is invertible is equivalent to the condition that the column space of the matrix $(f_i(e_j)$ is n-dimensional, which in turn implies V is n-dimensional.

We end this section with a lemma that allows for easy verification that $\sigma(I)$ is invertible. We used this on all the activation functions considered in table 1.

Lemma E.7. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be any function and let $I \in GL_n(\mathbb{R})$ be the identity matrix. Then $\sigma(I)$ is invertible provided*

$$\sigma(1) \neq \sigma(0) \text{ and } \sigma(1) \neq -(n-1)\sigma(0). \tag{E.8}$$

Proof. Let $N = \mathbf{1}\mathbf{1}^T - I$. Then

$$\sigma(I) = \sigma(1)I + \sigma(0)N.$$

Note that the eigenvalues of $\mathbf{1}\mathbf{1}^T$ are n (corresponding to eigenvector $\mathbf{1}$) and 0's (corresponding to the orthogonal complement of $\mathbf{1}$). For any linear operator $A \in \mathrm{Mat}_n(\mathbb{R})$ with eigenvector/eigenvalue pair (v, λ), v is easily seen to be an eigenvector of $A - I$ with eigenvalue $\lambda - 1$. Hence the eigenvalues of N are $n - 1$ and -1, and it follows that the eigenvalues of $\sigma(I)$ are

$$\sigma(1) + \sigma(0)(n-1), \sigma(1) - \sigma(0), \ldots, \sigma(1) - \sigma(0)$$

which are all non-zero if and only if eq. (E.8) holds. □

Remark E.9. In particular eq. (E.8) holds when $\sigma(1) = 1, \sigma(0) = 0$ (which holds for example when $\sigma(x) = \mathrm{ReLU}(x)$ or $\sigma(x) = x^d$). In this situation, $\sigma(I) = I$ and $\phi_\sigma(A) = \sigma(A)$ (coordinatewise application of σ). For example, if $\sigma \geq 0$ is non-negative, then $\phi_\sigma(A) = \sigma(A)$ has non-negative entries.

E.2 Calculating intertwiner groups (for table 1)

We begin with two lemmas: the first puts a "lower bound" on G_{σ_n} and the second is a "differential form" of the definition of the intertwiner group from section 3. Together, these two results effectively allow us to reduce calculation of intertwiner groups to the $n = 1$ case.

Lemma E.10 (cf. [GBC16, §8.2.2], [Bre+19, §3])**.** *G_{σ_n} always contains the permutation matrices Σ_n, and ϕ_σ restricts to the identity on Σ_n.*

Proof. If $A \in \Sigma_n$ is a permutation matrix, so that $Ae_i = e_{\pi(i)}$ where π is a permutation of $\{1, \dots, n\}$ then for any $x \in \mathbb{R}^n$ we observe

$$A\sigma(x) = A(\sum_i \sigma(x_i)e_i) = \sum_i \sigma(x_i)Ae_i = \sum_i \sigma(x_i)e_{\pi(i)}$$

which is exactly σ applied coordinatewise to $\sum_i x_i e_{\pi(i)} = Ax$. □

Corollary E.11. *If $A \in GL_n(\mathbb{R}), P \in \Sigma_n$, and $AP \in G_{\sigma_n}$ or $PA \in G_{\sigma_n}$, then $A \in G_{\sigma_n}$.*

Proof. If $B = AP \in G_{\sigma_n}$, then $A = BP^{-1}$, where $B \in G_{\sigma_n}$ by hypothesis and $P \in G_{\sigma_n}$ by lemma E.10. The result follows as G_{σ_n} is a group (lemma 3.1) and hence closed under multiplication. The other case is similar. □

Lemma E.12. *Suppose $A, B \in GL_n(\mathbb{R})$ and $\sigma_n \circ A = B \circ \sigma_n$. Suppose $x = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ and assume σ is differentiable at $x_1, \dots, x_n$ as well as*

$$(Ax)_i = \sum_j a_{ij}x_j, \quad \text{for } i = 1, \dots, n.$$

Then,

$$\operatorname{diag}(\sigma'((Ax)_i)|i = 1, \dots, n)A = B \operatorname{diag}(\sigma'(x_i)|i = 1, \dots, n)$$

(here $\operatorname{diag} : \mathbb{R}^n \to \operatorname{Mat}_{n\times n}(\mathbb{R})$ takes a vector to a diagonal matrix). Explicitly, for each $i, j \in \{1, \dots, n\}$

$$\sigma'(\sum_k a_{ik}x_k)a_{ij} = b_{ij}\sigma'(x_j). \tag{E.13}$$

Proof. By the chain rule [Rud76, Thm. 9.15], and since the differential of a matrix is itself,

$$d\sigma_n|_{Ax}A = Bd\sigma_n|_x.$$

Finally, by the definition of σ_n

$$\frac{\partial(\sigma_n(x))_i}{\partial x_j} = \frac{\partial\sigma(x_i)}{\partial x_j} = \begin{cases} \sigma'(x_j) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

□

Theorem E.14. *Suppose σ is non-constant, non-linear, and differentiable on a dense open set with finite complement.*[8] *Then,*

(i) Every $A \in G_{\sigma_n}$ is of the form PD, where $P \in \Sigma_n$ and D is diagonal.

(ii) For a diagonal $D = \operatorname{diag}(\lambda_1, \dots, \lambda_n) \in G_{\sigma_n}$, we have $\lambda_i \in G_{\sigma_1}$ for $i = 1, \dots, n$ and

$$\phi_\sigma(\operatorname{diag}(\lambda_1, \dots, \lambda_n)) = \operatorname{diag}(\phi_\sigma(\lambda_1), \dots, \phi_\sigma(\lambda_1))$$

where we make a slight abuse of notation: on the right hand side ϕ_σ is the homomorphism $G_{\sigma_1} \to GL_1(\mathbb{R})$.

In particular, ϕ_σ is determined by lemma E.10 and its behavior for $n = 1$.

[8] The differentiability assumption is probably not necessary, however it holds in all of the examples we consider and allows us to safely use lemma E.12.

Proof. For any $A \in G_{\mathrm{ReLU}}$ we observe that the differentiability hypotheses of lemma E.12 holds for $x \in U$ where U is a dense open set with measure-0 complement. Indeed, if $t_1, \ldots, t_M \in \mathbb{R}$ are the points where σ fails to be differentiable, we can take U to be the complement of the hyperplane arangement given by

$$(\bigcup_{ij}\{x \in \mathbb{R}^n \mid x_i = t_j\}) \cup (\bigcup_{ij}\{x \in \mathbb{R}^n \mid (Ax)_i \in= t_j\}) \subseteq \mathbb{R}^n$$

Fix a row i — the matrix A is invertible by hypotheses, and so there must be some j such that $a_{ij} \neq 0$ (otherwise the i-th row of A is 0). For any $x \in U$, we have by lemma E.12

$$\sigma'(\sum_k a_{ik}x_k)a_{ij} = b_{ij}\sigma'(x_j) \tag{E.15}$$

and we claim that this cannot hold unless $a_{ik} = 0$ for $k \neq j$. First, there is a $(x_1, \ldots, x_n) \in U$ such that $\sigma'(x_j) \neq 0$ (otherwise σ would be constant). Next, fixing x_j at a value with $\sigma'(x_j) \neq 0$ and rearranging eq. (E.15) we have

$$\frac{\sigma'(a_{ij}x_j + \sum_{k\neq j} a_{ik}x_k)}{\sigma'(x_j)}a_{ij} = b_{ij} = \text{ constant.} \tag{E.16}$$

By hypothesis σ is non-linear and so σ' is non-constant — hence if there were some $a_{ik} \neq 0$ for $k \neq j$, the left hand side of eq. (E.16) would be non-constant.

We have shown each row of A has at most one non-0 entry a_{ij} and that $a_{ij} \neq 0$. For A to be invertible, it must be that these non-0 entries land in distinct *columns*. This is exactly the form described in item (i).

Next, we note that for any ij (without assuming $a_{ij} \neq 0$) eqs. (E.15) and (E.16) tell us

$$a_{ij} = 0 \implies b_{ij} = 0,$$

and hence if $D = \mathrm{diag}(\lambda_1, \ldots, \lambda_n) \in G_{\sigma_n}$ and $\sigma_n \circ D = E \circ \sigma_n$ (i.e. $E = \phi_\sigma(D)$), it must be that $E = \mathrm{diag}(\mu_1, \ldots, \mu_n)$ for some $\mu_1, \ldots, \mu_n \in \mathbb{R}$. Now the equation $\sigma_n \circ D = E \circ \sigma_n$ is equivalent to

$$\sigma(\lambda_i x_i) = \beta_i \sigma(x_i) \text{ for } i = 1, \ldots, n$$

which in turn is equivalent to $\lambda_i \in G_{\sigma_1}$ and $\beta_i = \phi_\sigma(\lambda_i)$ for $i = 1, \ldots, n$, proving item (ii). □

In light of theorem E.14, to fill in the table of table 1 it will suffice to deal with the $n = 1$ cases, which we do below.

Calculation of G_{ReLU}. We remark that this is just the "positive homogeneous" property of ReLU, which is quite well known (cf. [GBC16, §8.2.2], [FB17, §2], [Kun+21, §3], [Men+19, §3], [RK20, §3, A], [Yi+19, §2-3]). Using remark E.9 if $a \in G_{\sigma_1}$

$$\max\{0, ax\} = \max\{0, a\}\max\{0, x\}.$$

If $a < 0$ then setting $x = -1$ results in $a = 0$, a contradiction. So $a > 0$ and $\max\{0, ax\} = a\max\{0, x\}$, showing $\phi_\sigma(a) = a$. □

Modifications for LeakyReLU. By definition, for $0 < s \ll 1$.

$$\mathrm{LeakyReLU}(x, a) := \begin{cases} sx & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

which we may simplify to $\mathrm{LeakyReLU}(x) = sx + (1 - s)\,\mathrm{ReLU}(x)$. Suppose now that

$$\begin{aligned} \mathrm{LeakyReLU}(ax) &= b\,\mathrm{LeakyReLU}(x), \text{ or using our simplification} \\ sax + (1 - s)\,\mathrm{ReLU}(ax) &= b(sx + (1 - s)\,\mathrm{ReLU}(x)). \end{aligned} \tag{E.17}$$

If $a < 0$, we may choose $x = -1$ to obtain

$$-a = -sa - (1 - s)a = -bs \text{ and } x = 1 \text{ to obtain}$$

$$sa = b$$

showing that $a = as^2$, which is impossible when $0 < s \ll 1$. So it must be $a > 0$, and then evaluating eq. (E.17) at $x = 1$ gives $a = b$. □

The sigmoid case: $\sigma(x) = 1/(1+e^x)$. We will leverage of a useful fact about the sigmoid function:

$$\sigma'(x) \text{ is a smooth probability distribution function on } \mathbb{R}, \text{ with } \sigma'(x) > 0 \text{ for all } x \in \mathbb{R}. \qquad (*)$$

If $\sigma(ax) = b\sigma(x)$, differentiating with respect to x gives

$$\sigma'(ax)a = b\sigma'(x). \tag{E.18}$$

Using eq. (*) and the fact that to probability distribution functions are proportional if and only if they are equal, we get $\sigma'(ax) = \sigma'(x)$. Then integrating from $-\infty$ to x tells us $\sigma(ax)/a = \sigma(x)$; setting $x = 0$ we see $\frac{1}{2a} = \frac{1}{2}$, hence $a = 1$.

To show $\phi_\sigma = \mathrm{id}$, backtracking to eq. (E.18) and setting $x = 0$ shows $b = a$. □

The Gaussian RBF case: $\sigma(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. We make use of several properties of this $\sigma(x)$:

(i) For any $a > 0$ the function $\sigma(ax)$ is a probability distribution function with mean 0 and variance $\frac{1}{a^2}$,and with $\sigma(ax) > 0$ for all $x \in \mathbb{R}$ and

(ii) σ is an even function ($\sigma(-x) = \sigma(x)$).

Now if $\sigma(ax) = b\sigma(x)$, then the pdfs $\sigma(ax)$ and $\sigma(x)$ are proportional hence equal by item (i). Therefore they have the same means and variances — since these are $0, \frac{1}{a^2}$ and $0, 1$ respectively we conclude $a = \pm 1$.

Finally, we explain why $\phi_\sigma(A) = \mathrm{abs}(A)$ (entrywise absolute value). Differentiating with respect to x gives

$$\sigma'(ax)a = b\sigma'(x). \tag{E.19}$$

This implies $b = 1$ when $a = 1$. On the other hand differentiating item (ii) tells us $-\sigma'(-x) = \sigma'(x)$, so when $a = -1$

$$b\sigma'(x) = -\sigma'(-x) = \sigma'(x)$$

and hence $b = 1$. □

The polynomial case: $\sigma(x) = x^d$. We remark that the description given in table 1 is implicit in [KTB19]. By theorem E.14 we only need to describe $\phi_\sigma : G_{\sigma_1} \to GL_1(\mathbb{R})$; for any $a \neq 0$

$$(ax)^d = a^d x^d$$

and this shows $G_{\sigma_1} = \mathbb{R} \setminus \{0\}$ and $\phi_\sigma(a) = a^d$. □

E.2.1 Gaussian error linear units (GeLUs)

Introduced and first studied in [HG16], these are defined as $\mathrm{GeLU}(x) = x\Phi(x)$ where Φ is the standard normal cummulative distribution function:

$$\Phi(x) = \int_{-\infty}^{x} \frac{e^{-\frac{t^2}{2}}}{\sqrt{2\pi}}\, dt.$$

By inspecting plots in fig. 12, we see that GeLU and ReLU are globally quite similar (they converge as $|x| \to \infty$) but that they differ when x within a few standard normal standard deviations of 0. One can show that $G_{\mathrm{GeLU}_n} = \Sigma_n$: indeed, by theorem E.14 it suffices to show that the only $\lambda \in \mathbb{R} \setminus \{0\}$ such that $\mathrm{GeLU}(\lambda x) = \phi(\lambda)\,\mathrm{GeLU}(x)$ for all x (where ϕ is some non-zero function of λ) is $\lambda = 1$. Expanding, we see that

$$\lambda x \Phi(\lambda x) = \phi(\lambda) x \Phi(x), \tag{E.20}$$

and rearranging this becomes

$$\frac{\Phi(\lambda x)}{\Phi(x)} = \frac{\phi(\lambda)}{\lambda} =: c, \tag{E.21}$$

that is, the right hand side is constant as a function of x. Then $\Phi(\lambda x) = c\Phi(x)$, and since Φ is positive it must be c is too. Moreover it must be $\lambda > 0$, as otherwise $\Phi(\lambda x)$ is monotonically decreasing while $\Phi(x)$ is increasing. Finally, letting $x \to \infty$ we see that $c = 1$, and from there we conclude $\lambda = 1$ by an argument similar to the use of item (i) in the Gaussian RBF case.

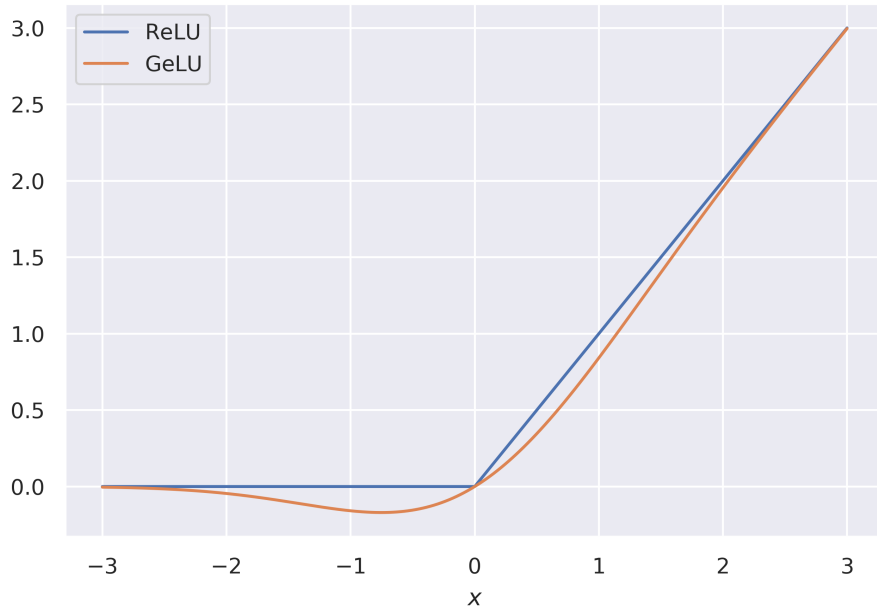


Figure 12: The ReLU and GeLU functions.

Despite the above calculation, it seems natural to ask how far G_{GeLU} is from G_{ReLU}, in other words how badly GeLU fails to be positive homogeneous. One measure of this is obtained by letting $X \sim \mathcal{N}(0,1)$ be a standard normal variable and computing the root-mean-square error

$$\xi(\lambda) := \sqrt{E[|\mathrm{GeLU}(\lambda X) - \lambda\,\mathrm{GeLU}(X)|^2]} \tag{E.22}$$

as a function of $\lambda > 0$, where the expectation is over X. Here our choice of a standard normal X is motivated by the same reasoning as discissed in [HG16], namely that activation inputs are roughly standard normal, especially in the presence of batch normalization. Evaluating eq. (E.22) doesn't seem particularly tractible analytically, but it does simplify to

$$\xi(\lambda) = \lambda\sqrt{E[|x(\Phi(\lambda x) - \Phi(x))|^2]}. \tag{E.23}$$

In fig. 13 we estimate $\xi(\lambda)$ by sampling X and replacing the expectation with the corresponding average. Evidently, as $\lambda \to \infty$ the function $\xi(\lambda)$ becomes linear: since $\Phi(x) \to \mathbf{1}_{x\geq 0}$ as $\lambda \to \infty$ (here $\mathbf{1}_{x\geq 0}$ is the indicator of $x > 0$, also known as the Heaviside or unit-step function), the asymptotic slope is $\sqrt{E[|x(\mathbf{1}_{x\geq 0} - \Phi(x))|^2]} \approx 0.127$.

E.3 Proof of theorem 3.3

Proof. The explicit description of $G(\mathrm{ReLU}, n)$ in table 1 is enough to show $G(\mathrm{ReLU}, n)$ *stabilizes* $\{\mathbb{R}_{\geq 0}e_i | i = 1, \ldots, n\}$. Indeed, any $A \in G(\mathrm{ReLU}, n)$ may be written as PD where $D = \mathrm{diag}(a_i)$ for some $a_i > 0$ and P is a permutation matrix associated to a permutation π. It suffices to show that P and $\mathrm{diag}(a_i)$ each preserves $\{\mathbb{R}_{\geq 0}e_i | i = 1, \ldots, n\}$. First,

$$\mathrm{diag}(a_i)\{\mathbb{R}_{\geq 0}e_i | i = 1, \ldots, n\} = \{\mathbb{R}_{\geq 0}a_i e_i | i = 1, \ldots, n\} = \{\mathbb{R}_{\geq 0}e_i | i = 1, \ldots, n\}$$

since the $a_i > 0$ so scaling by a_i preserves the ray $\mathbb{R}_{\geq 0}e_i$. Second,

$$P\{\mathbb{R}_{\geq 0}e_i | i = 1, \ldots, n\} = \{\mathbb{R}_{\geq 0}Pe_i | i = 1, \ldots, n\} = \{\mathbb{R}_{\geq 0}e_{\pi(i)} | i = 1, \ldots, n\} = \{\mathbb{R}_{\geq 0}e_i | i = 1, \ldots, n\}$$

(the last equality is due to the fact that we only consider the set of rays, not the ordered tuple of rays).

Conversely, if $A \in GL(n)$ stabilizes $\{\mathbb{R}_{\geq 0}e_i | i = 1, \ldots, n\}$, then in particular for each j $Ae_j \in \mathbb{R}_{\geq 0}e_i$ for some i and as A is invertible, setting $j = \pi(i)$ yields a permutation of $\{1, \ldots, n\}$.

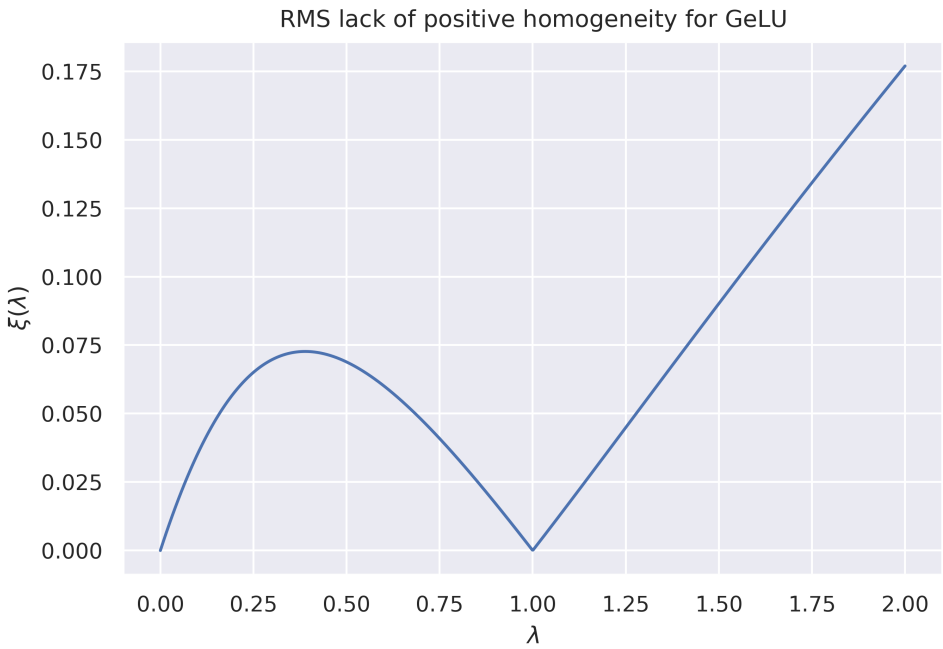


Figure 13: Root-mean-square lack of positive homogeneity for the GeLU function, estimated using 10^5 samples of X.

Moreover,since $Ae_j \neq 0$ (A is invertible) there must be some $a_j > 0$ so that $Ae_j = a_j e_i$. One can now verify that $A = PD$ where P is the permutation matrix associated to π and $D = \mathrm{diag}(a_j)$ which matches the description of $G(\mathrm{ReLU}, n)$ from table 1.

For the "moreover," we prove the contrapositive, namely that if $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$ has at least 2 non-0 coordinates v_i, v_j, then the G_{ReLU}-orbit of the ray $\mathbb{R}_{\geq 0} v$ that v generates cannot be finite. Indeed, suppose $t \geq 0$ and let $D = \mathrm{diag}(1, \ldots, 1, t, 1, \ldots, 1)$ (t in the ith position). The $n \times 2$ matrix $(v|Dv)$ has a 2×2 minor

$$\begin{pmatrix} v_i & tv_i \\ v_j & v_j \end{pmatrix} \text{ with deteriminant } (1-t)v_i v_j \neq 0 \text{ as long as } t \neq 1. \tag{E.24}$$

Thus v and Dv are linearly independent, and hence define distinct rays, for all $t \neq 1$. It follows that any set of rays stabilized by $G(\mathrm{ReLU}, n)$ that contains $\mathbb{R}_{\geq 0} v$ is uncountable. □

E.4 Proof of proposition 3.4

To give a rigorous proof we use induction on the depth k; since this to some extent obscures the main point, we briefly outline an informal proof: consider a composition of 2 layers of the network f with weights W':

$$\sigma(A_{i+1} W_{i+1} \phi_\sigma(A_i)^{-1} \sigma(A_i W_i \phi_\sigma(A_{i-1})^{-1} x + A_i b_i) + A_{i+1} b_{i+1}). \tag{E.25}$$

Using the defining properties of G_σ and ϕ_σ, we can extract A_i like

$$\sigma(A_i W_i \phi_\sigma(A_{i-1})^{-1} x + A_i b_i) = \phi_\sigma(A_i) \sigma(W_i \phi_\sigma(A_{i-1})^{-1} x + b_i). \tag{E.26}$$

The resulting copy of $\phi_\sigma(A_i)$ on the right hand side of eq. (E.26) is cancelled by the copy of $\phi_\sigma(A_i)^{-1}$ right-multiplying W_{i+1} in eq. (E.25), so that eq. (E.25) reduces to

$$\sigma(A_{i+1} W_{i+1} \sigma(W_i \phi_\sigma(A_{i-1})^{-1} x + b_i) + A_{i+1} b_{i+1}).$$

In this way, in between any two layers $\ell_{i+1} \circ \ell_i$ the A_i in $A_i W_i \phi_\sigma(A_{i-1})^{-1}$ and the $\phi_\sigma(A_i)^{-1}$ in $A_{i+1} W_{i+1} \phi_\sigma(A_i)^{-1}$ *cancel*. However, the factors $\phi_\sigma(A_m)$ and $\phi_\sigma(A_m)^{-1}$ appear on endpoints of the truncated networks $f_{\leq m}, f_{>m}$ and so they are not cancelled.

To keep track of W, W' while using the notation from section 3, we write

$$\ell_i(x, W) = \sigma(W_i x + b_i) \text{ and } \ell_i(x, W') = \sigma(W'_i x + b'_i) \text{ for } i < k$$

and so on.

Proof. By induction on k, the depth of the network. The case $k = 1$ is trivial, since there $W' = W$ and there is nothing to prove. For $k > 1$ we consider 2 cases:

Case $m = 1$: In this case let $V = (W_2, b_2, \ldots, W_k, b_k)$ and $V' = (A_2 W_2, A_2 b_2, A_3 W_3 \phi_\sigma(A_2)^{-1}, A_3 b_3, \ldots, W_k \phi_\sigma(A_{k-1}^{-1}), b_k)$, and let

$$g(x, V) = \ell_{k-1}(x, V) \circ \cdots \circ \ell_1(x, V) \text{ and } g(x, V') = \ell_{k-1}(x, V') \circ \cdots \circ \ell_1(x, V')$$

where $\ell_i(x, V) = \sigma(W_{i+1} x + b_{i+1})$ and similarly for V'. In other words, the weights V, V' and function g represent the architecture obtained by removing the earliest layer of f. Then, $f_{\leq 1}(x, W) = \sigma(W_1 x + b_1)$ and on the other hand

$$f_{\leq 1}(x, W') = \sigma(A_1 W_1 x + A_1 b_1) = \sigma(A_1(W_1 x + b_1)).$$

Using the identity $\sigma(A_1 z) = \phi_\sigma(A_1)\sigma(z)$ for any $z \in \mathbb{R}^{n_1}$, we obtain

$$f_{\leq 1}(x, W') = \phi_\sigma(A_1)\sigma(W_1 x + b_1) = \phi_\sigma(A_1)\sigma(W_1 x + b_1).$$

This shows $f_{\leq 1}(x, W') = \phi_\sigma(A_1) \circ f_{\leq 1}(x, W)$. Next, $f_{>1}(x, W) = g(x, V)$ but *because* $V'_1 = A_2 W_2$ whereas $W'_2 = A_2 W_2 \phi_\sigma(A_1)^{-1}$

$$f_{>1}(x, W') = g(x, V') \circ \phi_\sigma(A_1)^{-1}.$$

By induction on k, we may assume $g(x, V) = g(x, V')$ and it follows that $f_{>1}(x, W') = f_{>1}(x, W) \circ \phi_\sigma(A_1)^{-1}$

Case $m > 1$: Defining V, V' and g as above, we observe that

$$\begin{aligned} &f_{\leq i}(x, W) = g_{\leq i-1}(\sigma(W_1 x + b_1), V) \text{ and} \\ &f_{\leq i}(x, W') = g_{\leq i-1}(\phi_\sigma(A_1)^{-1}\sigma(A_1 W_1 x + A_1 b_1), V') = g_{\leq i-1}(\sigma(W_1 x + b_1), V'). \end{aligned} \tag{E.27}$$

By induction on k we may assume $g_{\leq i-1}(x, V') = \phi_\sigma(A_i) g_{\leq i-1}(x, V)$ and so

$$f_{\leq i}(x, W') = \phi_\sigma(A_i) g_{\leq i-1}(\sigma(W_1 x + b_1), V) = \phi_\sigma(A_i) f_{\leq i}(x, W).$$

Finally, $f_{>i}(x, W) = g_{>i-1}(x, V)$ and $f_{>i}(x, W') = g_{>i-1}(x, V')$ and we may assume by induction on k that $g_{>i-1}(x, V') = g_{>i-1}(x, V) \circ \phi_\sigma(A_i)^{-1}$, hence $f_{>i}(x, W') = f_{>i}(x, W) \circ \phi_\sigma(A_i)^{-1}$. □

E.5 Proof of theorem 4.2

Proof. Observe that by proposition 3.4, $\tilde{f}_{>l} = f_{>l} \circ \phi_\sigma(A_l^{-1})$. Hence

$$S(f, \tilde{f}, l, \varphi) = \tilde{f}_{>l} \circ \varphi \circ f_{\leq i} = f_{>l} \circ \phi_\sigma(A_l^{-1}) \circ \varphi \circ f_{\leq l}.$$

If $\mathcal{S}$ contains $\phi_\sigma(G_{\sigma_{n_l}})$ we may choose $\varphi = \phi_\sigma(A_l)$ to achieve $S(f, \tilde{f}, l, \varphi) = f$ as functions. Similarly, $\tilde{f}_{\leq l} = \phi_\sigma(A_l) \circ f_{\leq l}$ so if $\mathcal{S}$ contains $\phi_\sigma(G_{\sigma_{n_l}})$ we may choose $\varphi = \phi_\sigma(A_l^{-1})$ to achieve $S(f, \tilde{f}, l, \varphi) = \tilde{f}$ as functions. In either case eq. (4.1) holds. □

E.6 Symmetries of the loss landscape

Given that the intertwiner group describes a large set of symmetries of a network, it is not surprising that it also provides a way of understanding the relationship between equivalent networks. Proposition 3.4 has an interpretation in terms of the loss landscape of model architecture. For any layer n_i in f, the action of $G_{\sigma_{n_i}}$ on the weight space $\mathcal{W}$, translates to the obvious group action on the loss landscape.

Corollary E.28. *For any $1 \leq i \leq k$, the group $G_{\sigma_{n_i}}$ acts on $\mathcal{W}$ and for any test set $D_t \subset X \times Y$, model loss on D_t is invariant with respect to this action. More precisely, if $\ell(\Phi(W), D)$ is the loss of $\Phi(W)$ on test set D_t, then for any $g \in G_{\sigma_{n_i}}$, $\ell(\Phi(W), D) = \ell(\Phi(gW), D)$.*

E.7 Comparing capacities of stitching layers via discretization

Let $\mathrm{Mat}_{n\times n}(\mathbb{R})$ denote the space of $n\times n$ matrices. For $r=1,\dots,n$ let $\mathrm{Mat}^r_{n\times n}(\mathbb{R})\subseteq \mathrm{Mat}_{n\times n}(\mathbb{R})$ denote the rank r matrices, and let G_{ReLU_n} be as described in table 1. Suppose that each real dimension of $\mathrm{Mat}_{n\times n}(\mathbb{R})$ is replaced by a discrete grid $N(M,\epsilon)=\{-M+i\epsilon \mid i=0,\dots,\lfloor\frac{2M}{\epsilon}\rfloor-1\}$ — here ϵ could represent the limits of numerical precision in a floating point number system, and M could represent the maximum numerical magnitude. The size of each such grid is $\lfloor\frac{2M}{\epsilon}\rfloor$, and so the number of points in the resulting mesh grid $N(M,\epsilon)^{n^2}\subset \mathrm{Mat}_{n\times n}(\mathbb{R})$ is $\lfloor\frac{2M}{\epsilon}\rfloor^{n^2}$. We now estimate the number of points of G_{ReLU_n} and $\mathrm{Mat}^r_{n\times n}(\mathbb{R})$ in such a mesh grid.

G_{ReLU_n} is a disjoint union of $n!$ irreducible components, corresponding to the $n!$ possible permutations P in table 1. Each of these components is n-dimensional, corresponding to the fact that the factor D in table 1 is an arbitrary positive diagonal matrix. Hence we obtain

$$|G_{\mathrm{ReLU}_n}\cap N(M,\epsilon)^{n^2}|\approx n!\cdot(\frac{M}{\epsilon})^n. \tag{E.29}$$

On the other hand, any matrix $A\in \mathrm{Mat}^r_{n\times n}(\mathbb{R})$ can be written as $A=UV$ where $U\in \mathrm{Mat}_{n\times r}(\mathbb{R})$ and $V\in \mathrm{Mat}_{r\times n}(\mathbb{R})$. These U and V are not unique: given any invertible $r\times r$ matrix $W\in GL_r(\mathbb{R})$, we have $A=(UW)(W^{-1}V)$. From this we obtain the approximation[9]

$$|\mathrm{Mat}^r_{n\times n}(\mathbb{R})\cap N(M,\epsilon)^{n^2}|\approx \frac{|\mathrm{Mat}_{n\times r}(\mathbb{R})\cap N(M,\epsilon)^{nr}|\cdot|\mathrm{Mat}_{r\times n}(\mathbb{R})\cap N(M,\epsilon)^{rn}|}{|GL_r(\mathbb{R})\cap N(M,\epsilon)^{r^2}|} \tag{E.30}$$

$$\approx \frac{(\frac{2M}{\epsilon})^{nr}(\frac{2M}{\epsilon})^{rn}}{(\frac{2M}{\epsilon})^{r^2}} \tag{E.31}$$

$$\approx (\frac{2M}{\epsilon})^{2nr-r^2}. \tag{E.32}$$

It follows that

$$\log|G_{\mathrm{ReLU}_n}\cap N(M,\epsilon)^{n^2}|-\log|\mathrm{Mat}^r_{n\times n}(\mathbb{R})\cap N(M,\epsilon)^{n^2}| \tag{E.33}$$

$$=\log(n!)+n\log(\frac{M}{\epsilon})-(2nr-r^2)\log(\frac{M}{\epsilon}+\log 2). \tag{E.34}$$

Ignoring the term $(2nr-r^2)\log 2$, which is independent of M,ϵ, we get the approximation

$$\log(n!)+n\log(\frac{M}{\epsilon})-(2nr-r^2)\log(\frac{M}{\epsilon}+\log 2)\approx \log(n!)-((2r-1)n-r^2)\log(\frac{M}{\epsilon}). \tag{E.35}$$

Next, we make the coarse approximation

$$\log(n!)=\sum_{k=1}^{n}\log k\approx\int_1^n \log x\,dx=n\log n-n; \tag{E.36}$$

with this approximation the expression of eq. (E.35) is approximated as

$$\log(n!)-((2r-1)n-r^2)\log(\frac{M}{\epsilon})\approx n\log n-n-((2r-1)n-r^2)\log(\frac{M}{\epsilon}). \tag{E.37}$$

From this we conclude that as long as

1. $r\geq 1$ (we actually already assumed this when defining $\mathrm{Mat}^r_{n\times n}(\mathbb{R})$) and
2. $\frac{M}{\epsilon}\gg n$, which roughly says that the number of grid points per dimension is greater than the number of rows (equivalently columns) in $\mathrm{Mat}_{n\times n}(\mathbb{R})$,

[9] Here we ignore a significant subtlety: whether or not the multiplication map $\mathrm{Mat}_{n\times r}(\mathbb{R})\times \mathrm{Mat}_{r\times n}(\mathbb{R})\to \mathrm{Mat}_{n\times n}(\mathbb{R})$ induces a map $N(M,\epsilon)^{nr}\times N(M,\epsilon)^{rn}\to N(M,\epsilon)^{n^2}$ (with our naive setup it probably doesn't) and moreover whether the fibers of this map, which in the non-discretized case are generically isomorphic to $GL_r(\mathbb{R})$, have intersection with $N(M,\epsilon)^{nr}\times N(M,\epsilon)^{rn}$ of the expected size. We do not expect that these technical details will impact the takeaway of this analysis.

$$n\log n - n - ((2r-1)n - r^2)\log(\frac{M}{\epsilon}) \leq n\log n - n - (n-1)\log(\frac{M}{\epsilon}) \text{ using item 1} \tag{E.38}$$

$$= (n-1)(\log n - \log(\frac{M}{\epsilon})) + \log n - n \tag{E.39}$$

$$< (n-1)(\log n - \log(\frac{M}{\epsilon})) \text{ for } n > 1 \tag{E.40}$$

$$< 0 \text{ using item 2.} \tag{E.41}$$

The upshot is that our approximations and items 1 and 2 imply

$$\log|G_{\mathrm{ReLU}_n} \cap N(M,\epsilon)^{n^2}| - \log|\mathrm{Mat}^r_{n\times n}(\mathbb{R}) \cap N(M,\epsilon)^{n^2}| < 0, \text{ and hence} \tag{E.42}$$

$$|G_{\mathrm{ReLU}_n} \cap N(M,\epsilon)^{n^2}| < |\mathrm{Mat}^r_{n\times n}(\mathbb{R}) \cap N(M,\epsilon)^{n^2}|. \tag{E.43}$$

E.8 Calculations related to dissimilarity measures (for section 5)

Proof of lemma 5.5. We must show that for any $x_1, \ldots, x_r \in \mathbb{R}^d$ and any $c_1, \ldots, c_r \in \mathbb{R}$ that

$$\sum_{i,j} c_i c_j \max(x_i \odot x_j) \geq 0.$$

We use the elementary "max $\geq$ mean" inequality:

$$\max(x_i \odot x_j) \geq \frac{1}{d}\sum_k x_{ik}x_{jk} = \frac{1}{d} x_i \cdot x_j$$

where on the right hand side "$\cdot$" denotes dot product. This implies

$$\sum_{i,j} c_i c_j \max(x_i \odot x_j) \geq \frac{1}{d}\sum_{i,j} c_i c_j (x_i \cdot x_j) \geq 0$$

where the last inequality follows from the fact that the dot product is a kernel function. □

The last string of inequalities also shows $\sum_{i,j} c_i c_j \max(x_i \odot x_j) > 0$ when $x_1, \ldots, x_r$ are linearly independent, but does not directly imply the converse. It would be interesting to know conditions on $x_1, \ldots, x_r$ that imply positive *definiteness* of the matrix $(\max(x_i \odot x_j))$.

Proof of lemma 5.1. By table 1 any $A \in G_{\mathrm{ReLU}}$ can be factored as $A = PD$ with P a permutation matrix and D a positive diagonal matrix, and we can obtain a similar factorization $B = QE$. Then

$$\mu(XA, YB) = \mu(XPD, YQE) = \mu(XP, YQ) = \mu(X, Y)$$

where the second equality uses the hypothesis that μ is invariant to right multiplication by positive diagonal matrices, and the third uses the hypothesis that μ is invariant to right multiplication by permutation matrices. □

Lemma E.44. *Suppose A is a matrix such that $\max(Ax_1 \odot Ax_2) = \max(Ax_1 \odot Ax_2)$ for all $x_1, x_2 \in \mathbb{R}^d$. Then, A is of the form PD where P is a permutation matrix and D is diagonal with diagonal entries in $\{\pm 1\}$.*

Proof. We only need the special case where $x = y$: observe that

$$\max(x \odot x) = \max\{x_1^2, \ldots, x_d^2\} = (\max\{|x_1|, \ldots, |x_d|\})^2 = |x|_\infty^2$$

This means that if $\max(Ax_1 \odot Ax_2) = \max(Ax_1 \odot Ax_2)$, then A preserves the ℓ_∞ norm on $\mathbb{R}^d$, hence in particular preserves the unit hypercube in $\mathbb{R}^d$, and it is known that symmetries of the hypercube have the form PD where P is a permutation matrix and D is diagonal with diagonal entries in $\{\pm 1\}$ (see for example [Ser77, §5.9]). □

E.9 Intertwiners and more general architecture features (justification of remark 3.6)

Here we briefly discuss how ubiquitous architecture features like batch normalization and residual connections interact with intertwiner groups. For simplicity in this section we only consider $\sigma = \mathrm{ReLU}$.

E.9.1 Batch normalization

A batch normalization layer that takes as input $X \in \mathbb{R}^{b \cdot n}$ (where b is the batch size and n is the dimension of the layer) and returns

$$\tilde{X} \operatorname{diag}(\tilde{X}^T \tilde{X})^{-1} \operatorname{diag}(\gamma) + \beta \text{ where } \tilde{X} = X - \mathbf{1}\mathbf{1}^T X$$

and where $\gamma, \beta \in \mathbb{R}^n$ are the "gain" and "bias" parameters of the batch normalization layer, is *invariant* under independent scaling of coordinates, that is transformations of the form $X \leftarrow XD$ where D is an $n \times n$ positive diagonal matrix (see e.g. [BMC15]). Hence a k-layer ReLU MLP as in section 3 enhanced with batch normalization (*pre*-activation, as is standard) is invariant under the action of the slightly larger group $\prod_{l=1}^{k-1}(\mathbb{R}^{n_l}_{>0} \rtimes G_{\mathrm{ReLU}_{n_l}})$, where the action is given by[10]

$$\begin{aligned}
&(c_1, A_1, \ldots, c_{k-1}, A_{k-1}) \cdot (W_1, \gamma_1, \beta_1, \ldots, W_{k-1}, \gamma_{k-1}, \beta_{k-1}, W_k, b_k) \\
&= (A_1 W_1, \pi(A_1) \operatorname{diag} c_1 \gamma_1, \pi(A_1) \operatorname{diag} c_1 \beta_1, \\
&A_2 W_2 (\pi(A_1) \operatorname{diag} c_1)^{-1}, \pi(A_2) \operatorname{diag} c_2 \gamma_2, \pi(A_2) \operatorname{diag} c_2 \beta_2, \\
&\ldots, A_{k-1} W_{k-1} (\pi(A_{k-2}) \operatorname{diag} c_{k-2})^{-1}, \pi(A_{k-1}) \operatorname{diag} c_{k-1} \gamma_{k-1}, \pi(A_{k-1}) \operatorname{diag} c_{k-1} \beta_{k-1}, \\
&W_k (\pi(A_{k-1}) \operatorname{diag} c_{k-1})^{-1}, b_k).
\end{aligned} \tag{E.45}$$

Here,

- $c_l \in \mathbb{R}^{n_l}_{>0}$ and $A_l \in G_{\mathrm{ReLU}_{n_l}}$, for all $l = 1, \ldots, k-1$
- $\pi : G_{\mathrm{ReLU}_{n_l}} \to \Sigma_{n_l}$ is the homomorphism setting the positive entries to 1.

The key point is that we get another factor of $\mathbb{R}^{n_l}_{>0}$ at each layer. We also note that the space of matrices of the form $\pi(A_l) \operatorname{diag} c_l$ is, incidentally, exactly $G_{\mathrm{ReLU}_{n_l}}$, and that using eq. (E.45) one can generalize proposition 3.4 and theorem 4.2 to the case of networks with batch normalization.

E.9.2 Residual connections

We expand on remark 3.6 and explain what exactly transpires with residual connections below.

Suppose we have a k-layer MLP as in section 3 (again for simplicity with $\sigma = \mathrm{ReLU}$), together with residual connections between a set of layers $R = \{r_1, \ldots, r_m\} \subseteq \{2, \cdots, k-1\}$[11]:

$$\begin{tikzcd}
\mathbb{R}^{n_0} \arrow[r, "\sigma W_1"] & \cdots \arrow[r, "\sigma W_{r_{i-1}}"] & \mathbb{R}^{n_{r_{i-1}}} \arrow[r, "\sigma W_{r_{i-1}+1}"] \arrow[rr, bend right, "\mathrm{id}"'] & \cdots \arrow[r, "W_{r_i}"] & \mathbb{R}^{n_{r_i}} \arrow[r, "\sigma"] & \cdots \mathbb{R}^{n_{k-1}} \arrow[r, "W_k"] & \mathbb{R}^{n_{L+1}}
\end{tikzcd} \tag{E.46}$$

(for legibility biases b_l are suppressed). In addition we assume that the depth of each residual block is some fixed, that is $r_i - r_{i-1} = b =$ constant for all i.

First, we claim that a $(A_1, \ldots A_{k-1}) \in \prod_{l=1}^{L} G_{\mathrm{ReLU}_{n_l}}$ stabilizes the function f *if* (not claiming if and only if) $A_{r_i} = A_{r_j}$ for all $r_i, r_j \in R$. To see this suppose $g_i(x, W)$, $i = 1, \ldots, m$ are the depth b feedforward networks of the residual blocks, so that the $f_{\leq r_i}$ is given by

$$\begin{aligned}
f_{\leq r_i}(x, W) &= f_{\leq r_{i-1}}(x, W) + g_i(f_{\leq r_{i-1}}(x, W), W) \text{ where} \\
g_i(z, W) &= \sigma(W_{r_i+b} \sigma(\cdots \sigma(W_{r_{i-1}+1} z) \cdots)).
\end{aligned} \tag{E.47}$$

Assuming by induction on i that proposition 3.4 applies *at the residual connections in* R we note that with weights W' eq. (E.47) turns into

$$\begin{aligned}
f_{\leq r_i}(x, W') &= f_{\leq r_{i-1}}(x, W') + g_i(f_{\leq r_{i-1}}(x, W'), W') \\
&= A_{r_{i-1}} f_{\leq r_{i-1}}(x, W) + g_i(A_{r_{i-1}} f_{\leq r_{i-1}}(x, W), W').
\end{aligned} \tag{E.48}$$

[10] As is best practice we omit the biases on the linear layers ℓ_l for $l < k$, since they would be redundant now that we have biases on batch norm layers.

[11] In particular we assume there is at least one linear layer before the first outgoing/after the last incoming residual connection, as occurs in e.g. ResNets.

Note that proposition 3.4 applies directly to the g_i, so we may compute $g_i(z, W') = A_{r_i} g_i(A_{r_{i-1}}^{-1} z, W)$. Hence

$$\begin{aligned} f_{\leq r_i}(x, W') &= A_{r_{i-1}} f_{\leq r_{i-1}}(x, W) + A_{r_i} g_i(A_{r_{i-1}}^{-1} A_{r_{i-1}} f_{\leq r_{i-1}}(x, W), W) \\ &= A_{r_{i-1}} f_{\leq r_{i-1}}(x, W) + A_{r_i} g_i(f_{\leq r_{i-1}}(x, W), W). \end{aligned} \tag{E.49}$$

and we see that the only way $f_{\leq r_i}(x, W') = B f_{\leq r_i}(x, W)$ for some matrix B is if $A_{r_{i-1}} = A_{r_i}$, proving our claim. This also shows that if A_r denotes the common value of the A_{r_i} for $r_i \in R$, we have $f_{\leq r_i}(x, W') = A_r f_{\leq r_i}(x, W)$ for all i. It is also true that $f_{>r_i}(x, W') = f_{>r_i}(A_r^{-1} x, W)$: observe that

$$f_{>r_i}(x, W) = f_{>r_{i+1}}(x + g_{i+1}(x, W), W). \tag{E.50}$$

By *descending* induction on k, we may assume $f_{r_{i+1}}(x, W') = f_{r_{i+1}}(A_r^{-1} x, W)$, and as above $g_{i+1}(z, W') = A_r g_{i+1}(A_r^{-1} z, W)$, so that with weights W' eq. (E.50) becomes

$$\begin{aligned} f_{>r_i}(x, W') &= f_{>r_{i+1}}(x + g_{i+1}(x, W'), W') \\ &= f_{>r_{i+1}}(A_r^{-1}(x + A_r g_{i+1}(A_r^{-1} x, W)), W) \\ &= f_{>r_{i+1}}(A_r^{-1} x + g_{i+1}(A_r^{-1} x, W), W) = f_{>r_i}(A_r^{-1} x, W). \end{aligned} \tag{E.51}$$

as claimed.

Finally, we describe how stitching fails inside a residual block. Suppose we use weights W_l for $l \leq r_i + j$ where $0 < j < b$) (recall $b =$ depth of our basic block) and weights W_l' for $l > r_i + j$. The resulting stitched network is (attempting to use indentation to increase legibility)

$$\begin{aligned} & f_{r_{i+1}}(\\ & \quad f_{\leq r_i}(x, W) + g_{>j}(\\ & \qquad\qquad \varphi g_{\leq j}(f_{\leq r_i}(x, W), W), \\ & \qquad\qquad W'), \\ & \quad W'). \end{aligned} \tag{E.52}$$

By proposition 3.4 $g_{>j}(z, W') = A_r g_{>j}(A_{r_i+j}^{-1} z, W)$, and we have shown $f_{>r_{i+1}}(z, W') = f_{>r_{i+1}}(A_r^{-1} z, W)$. Combining these facts eq. (E.52) becomes

$$\begin{aligned} & f_{r_{i+1}}(\\ & \quad A_r^{-1} f_{\leq r_i}(x, W) + A_r^{-1} A_r g_{>j}(\\ & \qquad\qquad A_{r_i+j}^{-1} \varphi g_{\leq j}(f_{\leq r_i}(x, W), W), \\ & \qquad\qquad W), \\ & \quad W). \end{aligned} \tag{E.53}$$

Even after cancelling to remove the $A_r^{-1} A_r$ and in the ideal case where $\varphi = A_{r_i+j}$, we are left with an extra factor of A_r^{-1} left multiplying $f_{\leq r_i}(x, W)$:

$$\begin{aligned} & f_{r_{i+1}}(\\ & \quad A_r^{-1} f_{\leq r_i}(x, W) + g_{>j}(\\ & \qquad\qquad g_{\leq j}(f_{\leq r_i}(x, W), W), \\ & \qquad\qquad W), \\ & \quad W). \end{aligned} \tag{E.54}$$

F Network dissection details

Here we include some supplementary results and experiments for examining coordinate basis interpretability with network dissection [Bau+17].

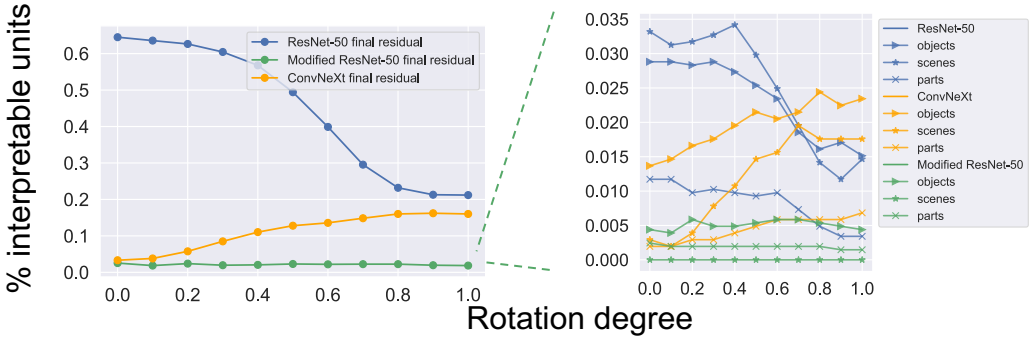


Figure 14: Supplement for the same network dissection experiment for the ResNet-50, modified ResNet-50, and ConvNeXt models in fig. 3, highlighting the categories of interpretable units for each model and basis on the right. The y-axis for the plot on the right is distinct concepts.

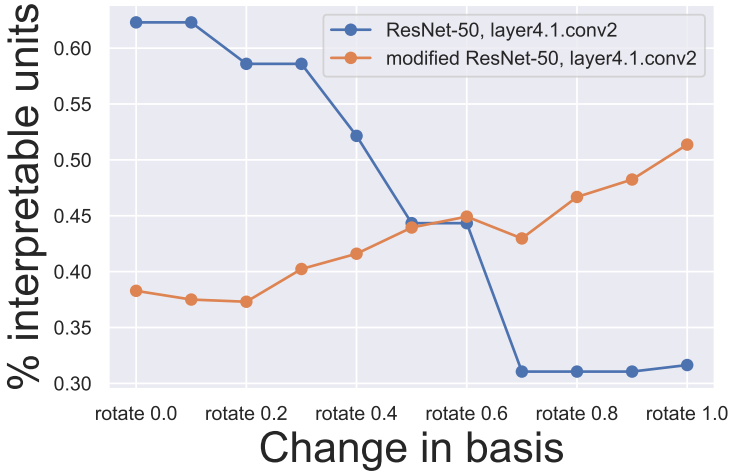


Figure 15: Fraction of network dissection interpretable units under rotations of the representation basis for a ResNet-50 and a modified ResNet-50 without an activation function on the residual output.

F.1 Network dissection methodology

The Broden concept dataset, compiled by Bau et al., contains pixel-level annotations for hierarchical concepts including colors, textures, objects, and scenes. For every channel activation, network dissection assesses the binary segmentation performance with every visual concept from Broden. The method first computes the channel activation for every Broden image. The distribution of the activations for the channel is used to binarize the activation (where we threshold by the top 0.5% of all activations for the channel) to define a segmentation mask for the channel activation which is interpolated to the size of the input image. If the Intersection over Union (IoU) of the activation segmentation mask and a concept mask is high enough (namely where IoU > 0.04), network dissection labels the activation an interpretable detector for the concept.

F.2 Additional experiments

fig. 14 breaks down the categories of interpretable units for the models and rotations examined in fig. 3. The number of interpretable units tends to be dominated by the object and scene concept detectors for the ResNet-50 and the ConvNeXt models.

We also perform an analogous network dissection experiment to section 6 within the residual blocks for the normal and modified ResNet-50 in fig. 15. Like the ConvNeXt model in fig. 3 we find that, surprisingly, the percentage of interpretable units actually tends to increase away from the activation basis.

Per-concept breakdowns produced by network dissection for three different rotation powers in the experiment in fig. 3 are given for the ResNet-50 in fig. 16, the modified ResNet-50 (without a ReLU activation function on the residual output) in fig. 18, and the ConvNeXt in in fig. 20. We also include the units with the highest concept intersection over union scores for the the three representative rotations for the ResNet-50 in fig. 17, the modified ResNet-50 (without a ReLU activation function on the residual output) in fig. 19, and the ConvNeXt in in fig. 21

F.3 Model training details

We train a ResNet-50 without ReLU (or any activation function) on the residual blocks in PyTorch using [Lec+22] on ImageNet [Den+09]. We train with SGD with momentum for 88 epochs with a cyclic learning rate rate of 1.7, label smoothing of 0.1, a batch size of 512, and weight decay of 10^{-4}. The model achieves 76.1% top-1 accuracy. We use pretrained weights for the ResNet-50 (unmodified) and ConvNeXt models from [MR10] and [Wig19] respectively.

G Dataset Details

CIFAR10: CIFAR10 is covered by the MIT License (MIT). We use canonical train/test splits (imported using torchvision).

Broden: the code used to generate the dataset is covered by the MIT license.

ImageNet: ImageNet is covered by CC-BY 4.0. We use canonical train/test splits.

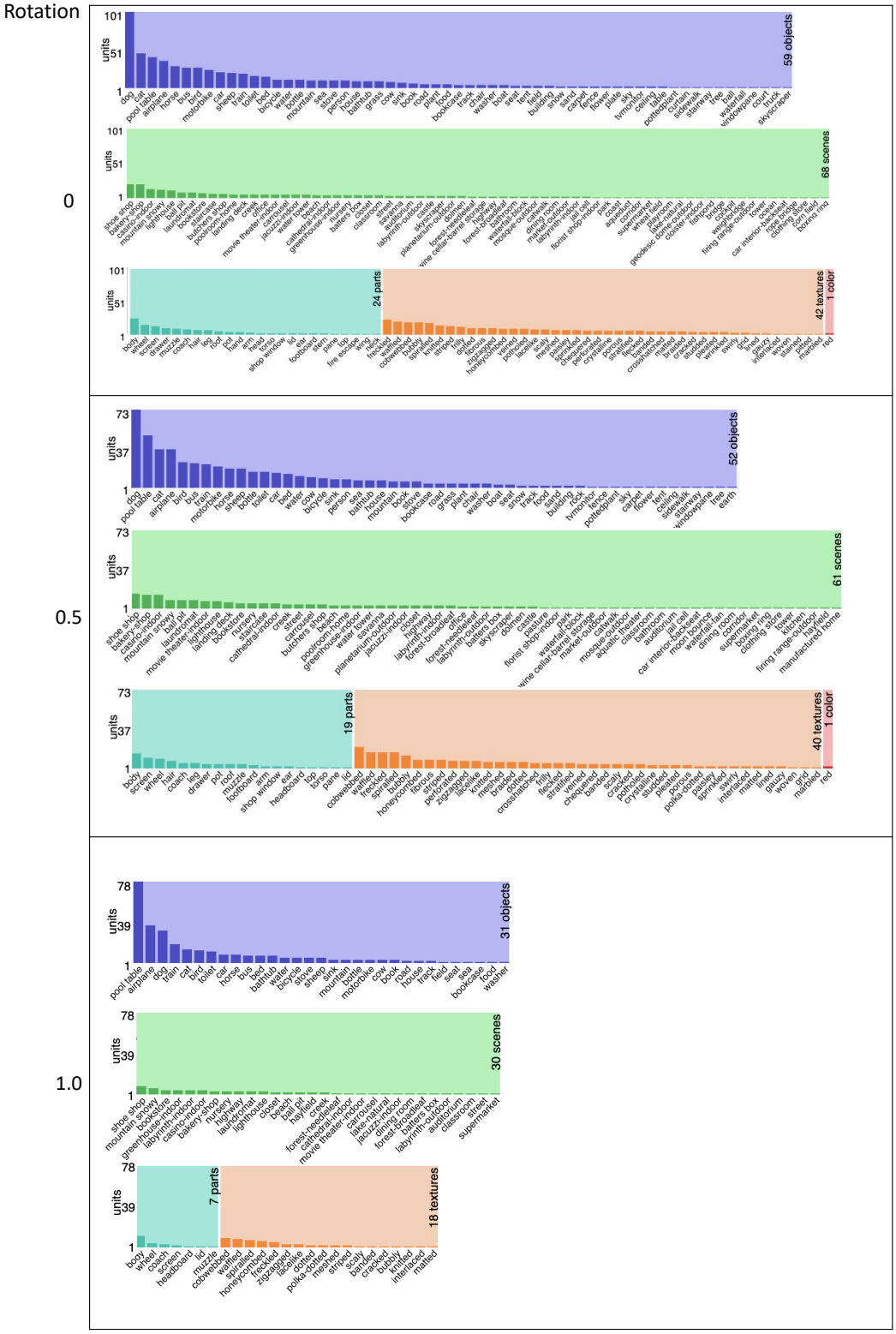


Figure 16: Network dissection bar graph of categories of unique concepts at three different rotation powers for the ResNet-50 model in fig. 3.

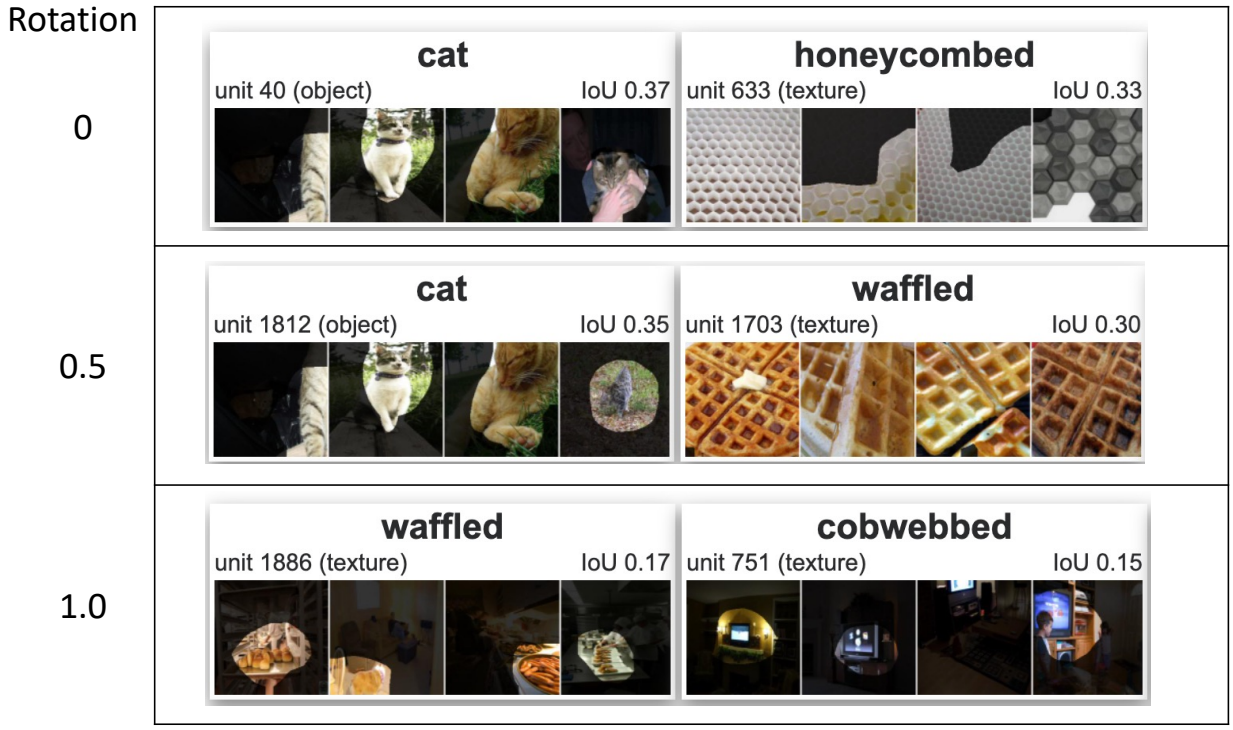


Figure 17: Top two highest scoring units for network dissection at three different rotation powers for the ResNet-50 model in fig. 3.

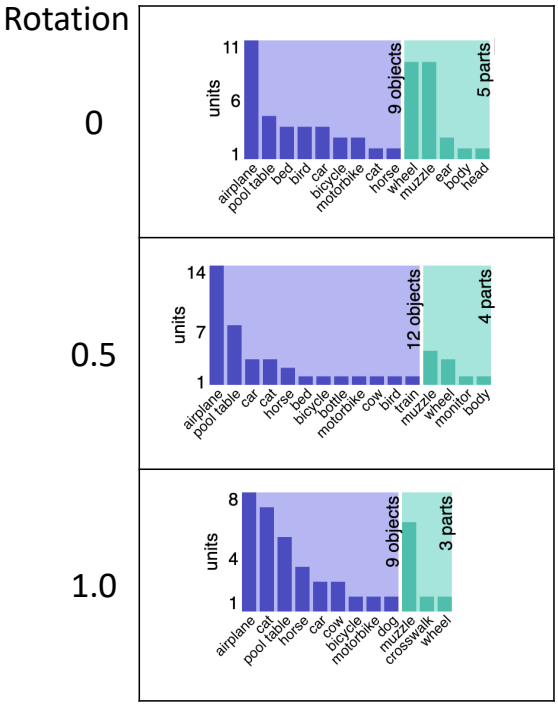


Figure 18: Network dissection bar graph of categories of unique concepts at three different rotation powers for the modified ResNet-50 model (without a ReLU activation function on the residual output) in fig. 3.

Figure 19: Top two highest scoring units for network dissection at three different rotation powers for the modified ResNet-50 model (without a ReLU activation function on the residual output) in fig. 3.

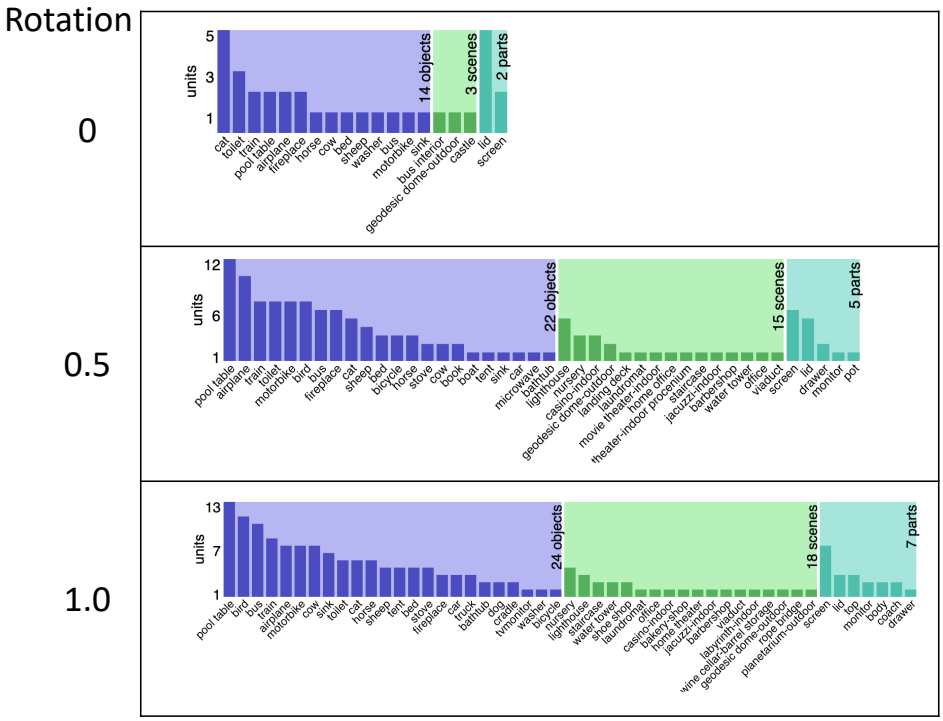


Figure 20: Network dissection bar graph of categories of unique concepts at three different rotation powers for the ConvNeXt model in fig. 3.

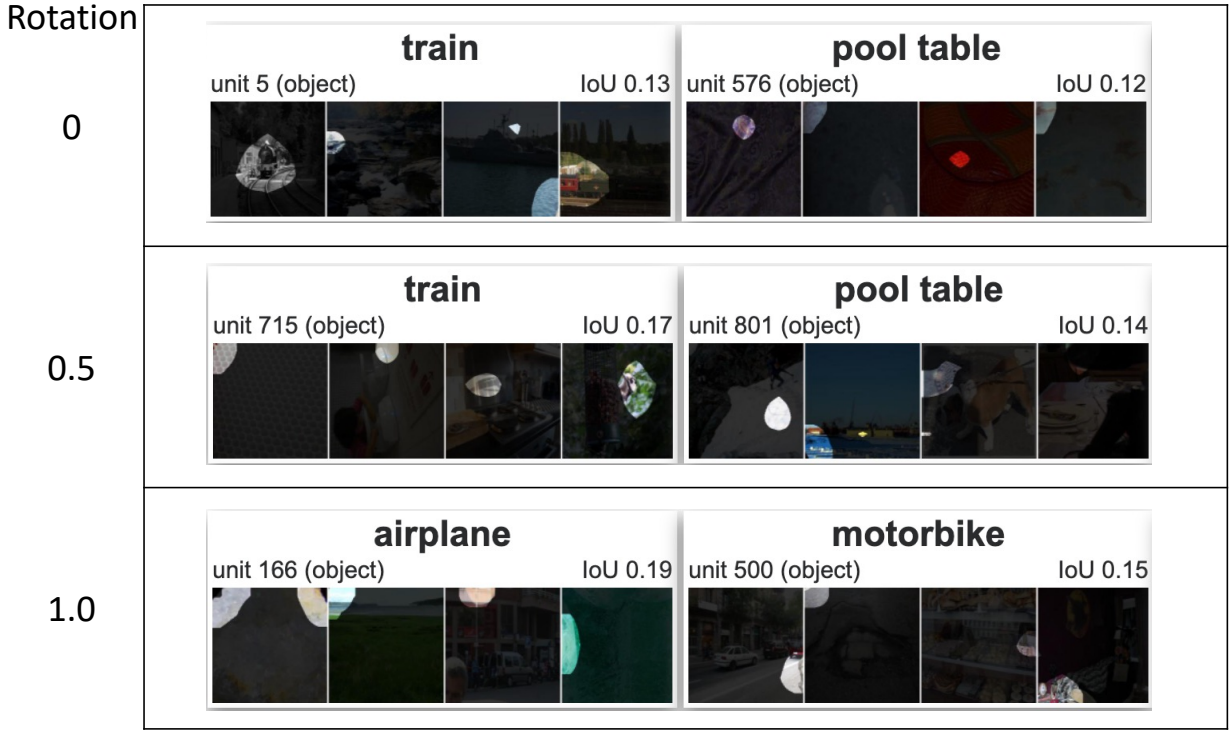


Figure 21: Top two highest scoring units for network dissection at three different rotation powers for the ConvNeXt model in fig. 3.

Layer (type:depth-idx)	Output Shape	Param #
Sequential	--	--
├─Conv2d: 1-1	[1, 64, 32, 32]	1,792
├─BatchNorm2d: 1-2	[1, 64, 32, 32]	128
├─ReLU: 1-3	[1, 64, 32, 32]	--
├─Conv2d: 1-4	[1, 128, 32, 32]	73,856
├─BatchNorm2d: 1-5	[1, 128, 32, 32]	256
├─ReLU: 1-6	[1, 128, 32, 32]	--
├─MaxPool2d: 1-7	[1, 128, 16, 16]	--
├─Conv2d: 1-8	[1, 256, 16, 16]	295,168
├─BatchNorm2d: 1-9	[1, 256, 16, 16]	512
├─ReLU: 1-10	[1, 256, 16, 16]	--
├─MaxPool2d: 1-11	[1, 256, 8, 8]	--
├─Conv2d: 1-12	[1, 512, 8, 8]	1,180,160
├─BatchNorm2d: 1-13	[1, 512, 8, 8]	1,024
├─ReLU: 1-14	[1, 512, 8, 8]	--
├─MaxPool2d: 1-15	[1, 512, 4, 4]	--
├─MaxPool2d: 1-16	[1, 512, 1, 1]	--
├─Flatten: 1-17	[1, 512]	--
├─Linear: 1-18	[1, 10]	5,130

Total params: 1,558,026
Trainable params: 1,558,026
Non-trainable params: 0
Total mult-adds (M): 228.56

Input size (MB): 0.01
Forward/backward pass size (MB): 4.72
Params size (MB): 6.23
Estimated Total Size (MB): 10.96

(a) Myrtle CNN architecture, summary courtesy of torchinfo.

Layer (type:depth-idx)	Output Shape	Param #
Sequential	--	--
├─Conv2d: 1-1	[1, 64, 32, 32]	1,792
├─ReLU: 1-2	[1, 64, 32, 32]	--
├─Conv2d: 1-3	[1, 128, 32, 32]	73,728
├─ReLU: 1-4	[1, 128, 32, 32]	--
├─MaxPool2d: 1-5	[1, 128, 16, 16]	--
├─Conv2d: 1-6	[1, 256, 16, 16]	294,912
├─ReLU: 1-7	[1, 256, 16, 16]	--
├─MaxPool2d: 1-8	[1, 256, 8, 8]	--
├─Conv2d: 1-9	[1, 512, 8, 8]	1,179,648
├─ReLU: 1-10	[1, 512, 8, 8]	--
├─MaxPool2d: 1-11	[1, 512, 4, 4]	--
├─MaxPool2d: 1-12	[1, 512, 1, 1]	--
├─Flatten: 1-13	[1, 512]	--
├─Linear: 1-14	[1, 10]	5,120

Total params: 1,555,200
Trainable params: 1,555,200
Non-trainable params: 0
Total mult-adds (M): 228.33

Input size (MB): 0.01
Forward/backward pass size (MB): 2.36
Params size (MB): 6.22
Estimated Total Size (MB): 8.59

(b) Myrtle CNN architecture without batch norm (only used in appendix D.0.1), summary courtesy of torchinfo.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ResNet                                   --                        --
├─Conv2d: 1-1                            [1, 16, 32, 32]           432
├─BatchNorm2d: 1-2                       [1, 16, 32, 32]           32
├─ReLU: 1-3                              [1, 16, 32, 32]           --
├─Sequential: 1-4                        [1, 64, 8, 8]             --
│    └─BasicBlock: 2-1                   [1, 16, 32, 32]           4,672
│    └─BasicBlock: 2-2                   [1, 16, 32, 32]           4,672
│    └─BasicBlock: 2-3                   [1, 16, 32, 32]           4,672
│    └─BasicBlock: 2-4                   [1, 32, 16, 16]           13,952
│    └─BasicBlock: 2-5                   [1, 32, 16, 16]           18,560
│    └─BasicBlock: 2-6                   [1, 32, 16, 16]           18,560
│    └─BasicBlock: 2-7                   [1, 64, 8, 8]             55,552
│    └─BasicBlock: 2-8                   [1, 64, 8, 8]             73,984
│    └─BasicBlock: 2-9                   [1, 64, 8, 8]             73,984
├─Linear: 1-5                            [1, 10]                   650
==========================================================================================
Total params: 269,722
Trainable params: 269,722
Non-trainable params: 0
Total mult-adds (M): 40.55
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 3.01
Params size (MB): 1.08
Estimated Total Size (MB): 4.11
==========================================================================================
```

(a) Our ResNet20 architecture, summary courtesy of torchinfo.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
BasicBlock                               --                        --
├─Conv2d: 1-1                            [1, 16, 32, 32]           2,304
├─BatchNorm2d: 1-2                       [1, 16, 32, 32]           32
├─ReLU: 1-3                              [1, 16, 32, 32]           --
├─Conv2d: 1-4                            [1, 16, 32, 32]           2,304
├─BatchNorm2d: 1-5                       [1, 16, 32, 32]           32
├─Sequential: 1-6                        [1, 16, 32, 32]           --
├─ReLU: 1-7                              [1, 16, 32, 32]           --
==========================================================================================
Total params: 4,672
Trainable params: 4,672
Non-trainable params: 0
Total mult-adds (M): 4.72
==========================================================================================
Input size (MB): 0.07
Forward/backward pass size (MB): 0.52
Params size (MB): 0.02
Estimated Total Size (MB): 0.61
==========================================================================================
```

(b) Internals of the 1st BasicBlock (the sequential contains the residual connection).

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ResNetImageNet                           --                        --
├─Conv2d: 1-1                            [1, 64, 32, 32]           1,728
├─BatchNorm2d: 1-2                       [1, 64, 32, 32]           128
├─ReLU: 1-3                              [1, 64, 32, 32]           --
├─Sequential: 1-4                        [1, 512, 4, 4]            --
│    └─BasicBlock: 2-1                   [1, 64, 32, 32]           73,984
│    └─BasicBlock: 2-2                   [1, 64, 32, 32]           73,984
│    └─BasicBlock: 2-3                   [1, 128, 16, 16]          230,144
│    └─BasicBlock: 2-4                   [1, 128, 16, 16]          295,424
│    └─BasicBlock: 2-5                   [1, 256, 8, 8]            919,040
│    └─BasicBlock: 2-6                   [1, 256, 8, 8]            1,180,672
│    └─BasicBlock: 2-7                   [1, 512, 4, 4]            3,673,088
│    └─BasicBlock: 2-8                   [1, 512, 4, 4]            4,720,640
├─Linear: 1-5                            [1, 10]                   5,130
==========================================================================================
Total params: 11,173,962
Trainable params: 11,173,962
Non-trainable params: 0
Total mult-adds (M): 555.43
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 9.83
Params size (MB): 44.70
Estimated Total Size (MB): 54.54
==========================================================================================
```

(a) Our ResNet18 architecture, summary courtesy of torchinfo.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
BasicBlock                               --                        --
├─Conv2d: 1-1                            [1, 64, 32, 32]           36,864
├─BatchNorm2d: 1-2                       [1, 64, 32, 32]           128
├─ReLU: 1-3                              [1, 64, 32, 32]           --
├─Conv2d: 1-4                            [1, 64, 32, 32]           36,864
├─BatchNorm2d: 1-5                       [1, 64, 32, 32]           128
├─Sequential: 1-6                        [1, 64, 32, 32]           --
├─ReLU: 1-7                              [1, 64, 32, 32]           --
==========================================================================================
Total params: 73,984
Trainable params: 73,984
Non-trainable params: 0
Total mult-adds (M): 75.50
==========================================================================================
Input size (MB): 0.26
Forward/backward pass size (MB): 2.10
Params size (MB): 0.30
Estimated Total Size (MB): 2.66
==========================================================================================
```

(b) Internals of the 1st BasicBlock (the sequential contains the residual connection).