

CONVERGENT LEARNING: DO DIFFERENT NEURAL NETWORKS LEARN THE SAME REPRESENTATIONS?

Yixuan Li¹*, Jason Yosinski¹*, Jeff Clune², Hod Lipson³, & John Hopcroft¹

¹Cornell University

²University of Wyoming

³Columbia University

{yli,yosinski,jeh}@cs.cornell.edu

jeffclune@uwyo.edu, hod.lipson@columbia.edu

ABSTRACT

Recent successes in training large, deep neural networks have prompted active investigation into the representations learned on their intermediate layers. Such research is difficult because it requires making sense of non-linear computations performed by millions of learned parameters, but valuable because it increases our ability to understand current models and training algorithms and thus create improved versions of them. In this paper we investigate the extent to which neural networks exhibit what we call *convergent learning*, which is when the representations learned by multiple nets converge to a set of features which are either individually similar between networks or where subsets of features span similar low-dimensional spaces. We propose a specific method of probing representations: training multiple networks and then comparing and contrasting their individual, learned representations at the level of neurons or groups of neurons. We begin research into this question by introducing three techniques to approximately align different neural networks on a feature or subspace level: a bipartite matching approach that makes one-to-one assignments between neurons, a sparse prediction and clustering approach that finds one-to-many mappings, and a spectral clustering approach that finds many-to-many mappings. This initial investigation reveals a few interesting, previously unknown properties of neural networks, and we argue that future research into the question of convergent learning will yield many more. The insights described here include (1) that some features are learned reliably in multiple networks, yet other features are not consistently learned; (2) that units learn to span low-dimensional subspaces and, while these subspaces are common to multiple networks, the specific basis vectors learned are not; (3) that the representation codes show evidence of being a mix between a local (single unit) code and slightly, but not fully, distributed codes across multiple units; (4) that the average activation values of neurons vary considerably within a network, yet the mean activation values across different networks converge to an almost identical distribution.¹

1 INTRODUCTION

Many recent studies have focused on understanding deep neural networks from both a theoretical perspective (Arora et al., 2014; Neyshabur & Panigrahy, 2013; Montavon et al., 2011; Paul & Venkatasubramanian, 2014; Goodfellow et al., 2014) and from an empirical perspective (Erhan et al., 2009; Eigen et al., 2013; Szegedy et al., 2013; Simonyan et al., 2013; Zeiler & Fergus, 2014; Nguyen et al., 2014; Yosinski et al., 2014; Mahendran & Vedaldi, 2014; Yosinski et al., 2015; Zhou et al., 2014). In this paper we continue this trajectory toward attaining a deeper understanding of neural net training by proposing a new approach. We begin by noting that modern deep neural networks (DNNs) exhibit an interesting phenomenon: networks trained starting at different random initializations frequently converge to solutions with similar performance (see Dauphin et al. (2014) and Section 2 below). Such similar performance by different networks raises the question of to

*Authors contributed equally.

¹A preliminary version of this work was presented at the NIPS 2015 Feature Extraction workshop.

what extent the learned internal representations differ: Do the networks learn radically different sets of features that happen to perform similarly, or do they exhibit *convergent learning*, meaning that their learned feature representations are largely the same? This paper makes a first attempt at asking and answering these questions. Any improved understanding of what neural networks learn should improve our ability to design better architectures, learning algorithms, and hyperparameters, ultimately enabling more capable models. For instance, distributed data-parallel neural network training is more complicated than distributed data-parallel training of convex models because periodic direct averaging of model parameters is not an effective strategy: perhaps solving a neuron correspondence problem before averaging would mitigate the need for constant synchronization. As another example, if networks converge to diverse solutions, then perhaps additional performance improvements are possible via training multiple models and then using model compilation techniques to realize the resulting ensemble in a single model.

In this paper, we investigate the similarities and differences between the representations learned by neural networks with the same architecture trained from different random initializations. We employ an architecture derived from AlexNet (Krizhevsky et al., 2012) and train multiple networks on the ImageNet dataset (Deng et al., 2009) (details in Section 2). We then compare the representations learned across different networks. We demonstrate the effectiveness of this method by both visually and quantitatively showing that the features learned by some neuron clusters in one network can be quite similar to those learned by neuron clusters in an independently trained neural network. Our specific contributions are asking and shedding light on the following questions:

1. By defining a measure of similarity between units² in different neural networks, can we come up with a permutation for the units of one network to bring it into a one-to-one alignment with the units of another network trained on the same task? Is this matching or alignment close, because features learned by one network are learned nearly identically somewhere on the same layer of the second network, or is the approach ill-fated, because the representations of each network are unique? (Answer: a core representation is shared, but some rare features are learned in one network but not another; see Section 3).
2. Are the above one-to-one alignment results robust with respect to different measures of neuron similarity? (Answer: yes, under both linear correlation and estimated mutual information metrics; see Section 3.2).
3. To the extent that an accurate one-to-one neuron alignment is not possible, is it simply because one network’s representation space is a rotated version³ of another’s? If so, can we find and characterize these rotations? (Answers: by learning a sparse weight LASSO model to predict one representation from only a few units of the other, we can see that the transform from one space to the other can be possibly decoupled into transforms between small subspaces; see Section 4).
4. Can we further cluster groups of neurons from one network with a similar group from another network? (Answer: yes. To approximately match clusters, we adopt a spectral clustering algorithm that enables many-to-many mappings to be found between networks. See Section S.3).
5. For two neurons detecting similar patterns, are the activation statistics similar as well? (Answer: mostly, but with some differences; see Section S.4).

2 EXPERIMENTAL SETUP

All networks in this study follow the basic architecture laid out by Krizhevsky et al. (2012), with parameters learned in five convolutional layers (conv1 – conv5) followed by three fully connected layers (fc6 – fc8). The structure is modified slightly in two ways. First, Krizhevsky et al. (2012) employed limited connectivity between certain pairs of layers to enable splitting the model across two GPUs.⁴ Here we remove this artificial group structure and allow all channels on each layer to connect to all channels on the preceding layer, as we wish to study only the group structure, if any, that arises naturally, not that which is created by architectural choices. Second, we place the local response normalization layers after the pooling layers following the defaults released with the Caffe

²Note that we use the words “filters”, “channels”, “neurons”, and “units” interchangeably to mean channels for a convolutional layer or individual units in a fully connected layer.

³Or, more generally, a space that is an affine transformation of the first network’s representation space.

⁴In Krizhevsky et al. (2012) the conv2, conv4, and conv5 layers were only connected to half of the preceding layer’s channels.

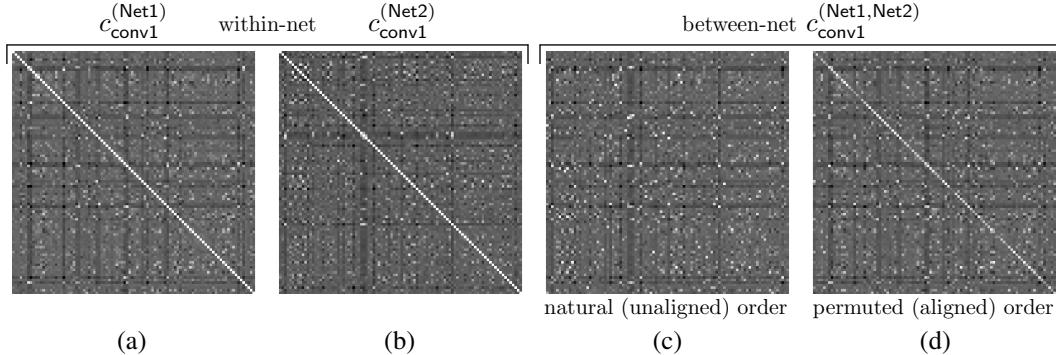


Figure 1: Correlation matrices for the conv1 layer, displayed as images with minimum value at black and maximum at white. **(a,b)** Within-net correlation matrices for Net1 and Net2, respectively. **(c)** Between-net correlation for Net1 vs. Net2. **(d)** Between-net correlation for Net1 vs. a version of Net2 that has been permuted to approximate Net1’s feature order. The partially white diagonal of this final matrix shows the extent to which the alignment is successful; see Figure 3 for a plot of the values along this diagonal and further discussion.

framework, which does not significantly impact performance (Jia et al., 2014). Networks are trained using Caffe on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 dataset (Deng et al., 2009). Further details and the complete code necessary to reproduce these experiments is available at https://github.com/yixuanli/convergent_learning.

We trained four networks in the above manner using four different random initializations. We refer to these as Net1, Net2, Net3, and Net4. The four networks perform very similarly on the validation set, achieving top-1 accuracies of 58.65%, 58.73%, 58.79%, and 58.84%, which are similar to the top-1 performance of 59.3% reported in the original study (Krizhevsky et al., 2012).

We then aggregate certain statistics of the activations within the networks. Given a network Net_n trained in this manner, the scalar random variable $X_{l,i}^{(n)}$ denotes the series of activation values produced over the entire ILSVRC validation dataset by unit i on layer $l \in \{\text{conv1}, \text{conv2}, \text{conv3}, \text{conv4}, \text{conv5}, \text{fc6}, \text{fc7}\}$.⁵ We collect the following statistics by aggregating over the validation set (and in the case of convolutional layers also over spatial positions):

$$\begin{aligned} \text{Mean: } \mu_{l,i}^{(n)} &= \mathbb{E}[X_{l,i}^{(n)}] \\ \text{Standard deviation: } \sigma_{l,i}^{(n)} &= \sqrt{(\mathbb{E}[(X_{l,i}^{(n)} - \mu_{l,i}^{(n)})^2])} \\ \text{Within-net correlation: } c_{l,i,j}^{(n)} &= \mathbb{E}[(X_{l,i}^{(n)} - \mu_{l,i}^{(n)})(X_{l,j}^{(n)} - \mu_{l,j}^{(n)})]/\sigma_{l,i}^{(n)}\sigma_{l,j}^{(n)} \\ \text{Between-net correlation: } c_{l,i,j}^{(n,m)} &= \mathbb{E}[(X_{l,i}^{(n)} - \mu_{l,i}^{(n)})(X_{l,j}^{(m)} - \mu_{l,j}^{(m)})]/\sigma_{l,i}^{(n)}\sigma_{l,j}^{(m)} \end{aligned}$$

Intuitively, we compute the mean and standard deviation of the activation of each unit in the network over the validation set. For convolutional layers, we compute the mean and standard deviation of each channel. The mean and standard deviation for a given network and layer is a vector with length equal to the number of channels (for convolutional layers) or units (for fully connected layers).⁶ The within-net correlation values for each layer can be considered as a symmetric square matrix with side length equal to the number of units in that layer (e.g. a 96×96 matrix for conv1 as in Figure 1a,b). For a pair of networks, the between-net correlation values also form a square matrix, which in this case is not symmetric (Figure 1c,d).

⁵ For the fully connected layers, the random variable $X_{l,i}^{(n)}$ has one specific value for each input image; for the convolutional layers, the value of $X_{l,i}^{(n)}$ takes on different values at each spatial position. In other words, to sample an $X_{l,i}^{(n)}$ for an FC layer, we pick a random image from the validation set; to sample $X_{l,i}^{(n)}$ for a conv layer, we sample a random image and a random position within the conv layer.

⁶ For reference, the number of channels for conv1 to fc8 is given by: $\mathcal{S} = \{96, 256, 384, 384, 256, 4096, 4096, 1000\}$. The corresponding size of the correlation matrix in each layer is: $\{s^2 \mid \forall s \in \mathcal{S}\}$. Furthermore, the spatial extents of each channel in each convolutional layer is given by: $\{\text{conv1} : 55 \times 55, \text{conv2} : 27 \times 27, \text{conv3} : 13 \times 13, \text{conv4} : 13 \times 13, \text{conv5} : 13 \times 13\}$

We use these correlation values as a way of measuring how related the activations of one unit are to another unit, either within the network or between networks. We use correlation to measure similarity because it is independent of the scale of the activations of units. Within-net correlation quantifies the similarity between two neurons in the same network; whereas the between-net correlation matrix quantifies the similarity of two neurons from different neural networks. Note that the units compared are always on the same layer on the network; we do not compare units between different layers. In the Supplementary Information (see Figure S1), we plot the activation values for several example high correlation and low correlation pairs of units from conv1 and conv2 layers; the simplicity of the distribution of values suggests that the correlation measurement is an adequate indicator of the similarity between two neurons. To further confirm this suspicion, we also tested with a full estimate of the mutual information between units and found it to yield similar results to correlation (see Section 3.2).

3 IS THERE A ONE-TO-ONE ALIGNMENT BETWEEN FEATURES LEARNED BY DIFFERENT NEURAL NETWORKS?

We would like to investigate the similarities and differences between multiple training runs of same network architecture. Due to symmetries in the architecture and weight initialization procedures, for any given parameter vector that is found, one could create many equivalent solutions simply by permuting the unit orders within a layer (and permuting the outgoing weights accordingly). Thus, as a first step toward analyzing the similarities and differences between different networks, we ask the following question: if we allow ourselves to permute the units of one network, to what extent can we bring it into alignment with another? To do so requires finding equivalent or nearly-equivalent units across networks, and for this task we adopt the magnitude independent measures of correlation and mutual information. We primarily give results with the simpler, computationally faster correlation measure (Section 3.1), but then confirm the mutual information measure provides qualitatively similar results (Section 3.2).

3.1 ALIGNMENT VIA CORRELATION

As discussed in Section 2, we compute within-net and between-net unit correlations. Figure 1 shows the within-net correlation values computed between units on a network and other units on the same network (panels a,b) as well as the between-net correlations between two different networks (panel c). We find matching units between a pair of networks — here Net1 and Net2 — in two ways. In the first approach, for each unit in Net1, we find the unit in Net2 with maximum correlation to it, which is the max along each row of Figure 1c. This type of assignment is known as a bipartite *semi-matching* in graph theory (Lawler, 1976), and we adopt the same nomenclature here. This procedure can result in multiple units of Net1 being paired with the same unit in Net2. Figure 2 shows the eight highest correlation matched features and eight lowest correlation matched features using the semi-matching approach (corresponding to the leftmost eight and rightmost eight points in Figure 3). To visualize the functionality each unit, we plot the image patch from the validation set that causes the highest activation for that unit. For all the layers shown, the most correlated filters (on the left) reveal that there are nearly perfect counterparts in each network, whereas the low-correlation filters (on the right) reveal that there are many features learned by one network that are unique and thus have no corollary in the other network.

An alternative approach is to find the one-to-one assignment between units in Net1 and Net2 without replacement, such that every unit in each network is paired with a unique unit in the other network. This more common approach is known as bipartite *matching*.⁷ A matching that maximizes the sum of the chosen correlation values may be found efficiently via the Hopcroft-Karp algorithm (Hopcroft & Karp, 1973) after turning the between-net correlation matrix into a weighted bipartite graph. Figure 1c shows an example between-net correlation matrix; the max weighted matching can be thought of as a path through the matrix such that each row and each column are selected exactly once, and the sum of elements along the path is maximized. Once such a path is found, we can permute the units of Net2 to bring it into the best possible alignment with Net1, so that the first channel of Net2 approximately matches (has high correlation with) the first channel of Net1, the second channels of each also approximately match, and so on. The correlation matrix of

⁷Note that the *semi-matching* is “row-wise greedy” and will always have equal or better sum of correlation than the *matching*, which maximizes the same objective but must also satisfy global constraints.

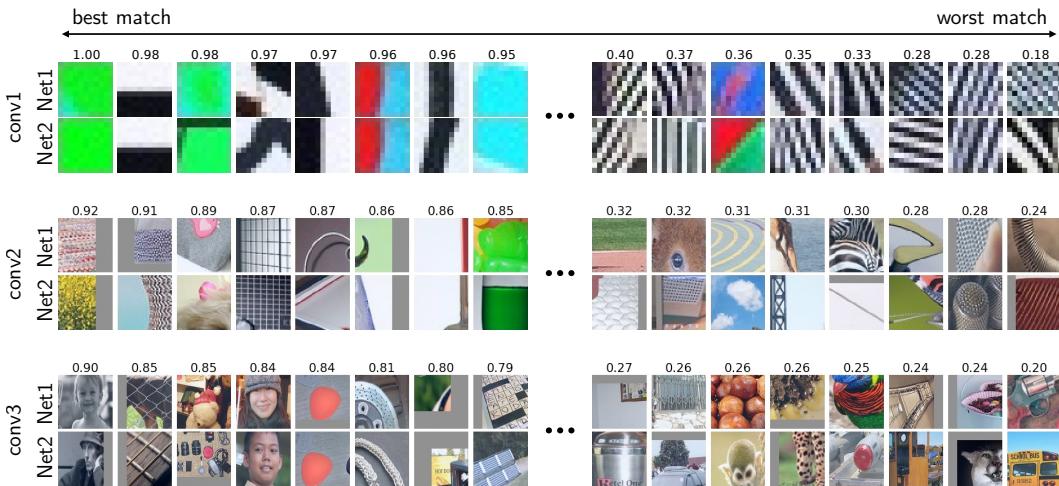


Figure 2: With assignments chosen by semi-matching, the eight best (highest correlation, left) and eight worst (lowest correlation, right) matched features between Net1 and Net2 for the conv1 – conv3 layers. For all layers visualized, (1) the most correlated filters are near perfect matches, showing that many similar features are learned by independently trained neural networks, and (2) the least correlated features show that many features are learned by one network and are not learned by the other network, at least not by a single neuron in the other network. The results for the conv4 and conv5 layers can be found in the Supplementary Material (see Figure S2).

Net1 with the permuted version of Net2 is shown in Figure 1d. Whereas the diagonal of the self correlation matrices are exactly one, the diagonal of the permuted between-net correlation matrix contains values that are generally less than one. Note that the diagonal of the permuted between-net correlation matrix is bright (close to white) in many places, which shows that for many units in Net1 it is possible to find a unique, highly correlated unit in Net2.

Figure 3 shows a comparison of assignments produced by the semi-matching and matching methods for the conv1 layer (Figure S3 shows results for other layers). Insights into the differing representations learned can be gained from both assignment methods. The first conclusion is that for most units, particularly those with the higher semi-matching and matching correlations (Figure 3, left), the semi-matching and matching assignments coincide, revealing that for many units a one-to-one assignment is possible. Both methods reveal that the average correlation for one-to-one alignments varies from layer to layer (Figure 4), with the highest matches in the conv1 and conv5 layers, but worse matches in between. This pattern implies that the path from a relatively matchable conv1 representation to conv5 representation passes through an intermediate middle region where matching is more difficult, suggesting that what is learned by different networks on conv1 and conv2 is more convergent than conv3, conv4 and conv5. This result may be related to previously observed greater complexity in the intermediate layers as measured through the lens of optimization difficulty (Yosinski et al., 2014).

Next, we can see that where the semi-matching and matching differ, the matching is often much worse. One hypothesis for why this occurs is that the two networks learn different numbers of units to span certain subspaces. For example, Net1 might learn a representation that uses six filters to span a subspace of human faces, but Net2 learns to span the same subspace with five filters. With unique matching, five out of the six filters from Net1 may be matched to their nearest counterpart in Net2, but the sixth Net1 unit will be left without a counterpart and will end up paired with an almost unrelated filter.

Finally, with reference to Figure 3 (but similarly observable in Figure S3 for other layers), another salient observation is that the correlation of the semi-matching falls significantly from the best-matched unit (correlations near 1) to the lowest-matched (correlations near 0.3). This indicates that some filters in Net1 can be paired up with filters in Net2 with high correlation, but other filters in Net1 and Net2 are network-specific and have no high-correlation pairing in the alternate network, implying that those filters are rare and not always learned. This holds across the conv1 – conv5 layers.

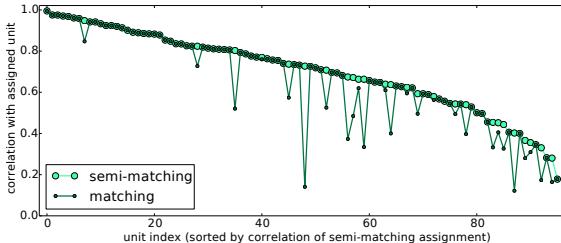


Figure 3: Correlations between paired conv1 units in Net1 and Net2. Pairings are made via semi-matching (light green), which allows the same unit in Net2 to be matched with multiple units in Net1, or matching (dark green), which forces a unique Net2 neuron to be paired with each Net1 neuron. Units are sorted by their semi-matching values. See text for discussion.

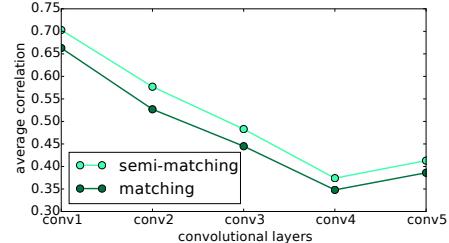


Figure 4: Average correlations between paired conv1 units in Net1 and Net2. Both semi-matching (light green) and matching (dark green) methods suggest that features learned in different networks are most convergent on conv1 and least convergent on conv4.

3.2 ALIGNMENT VIA MUTUAL INFORMATION

Because correlation is a relatively simple mathematical metric that may miss some forms of statistical dependence, we also performed one-to-one alignments of neurons by measuring the *mutual information* between them. Mutual information measures how much knowledge one gains about one variable by knowing the value of another. Formally, the mutual information of the two random variables $X_{l,i}^{(n)}$ and $X_{l,j}^{(m)}$ representing the activation of the i -th neuron in Net n and the j -th neuron in Net m , is defined as:

$$I\left(X_{l,i}^{(n)}; X_{l,j}^{(m)}\right) = \sum_{a \in X_{l,i}^{(n)}} \sum_{b \in X_{l,j}^{(m)}} p(a,b) \log\left(\frac{p(a,b)}{p(a)p(b)}\right),$$

where $p(a,b)$ is the joint probability distribution of $X_{l,i}^{(n)}$ and $X_{l,j}^{(m)}$, and $p(a)$ and $p(b)$ are their marginal probability distributions, respectively. The within-net mutual information matrix and between-net mutual information matrix have the same shapes as their equivalent correlation matrices.

We apply the same matching technique described in Section 3.1 to the between-net mutual information matrix,⁸ and compare the highest and lowest mutual information matches (Figure S4) to those obtained via correlation (Figure 2). The results are qualitatively the same. For example, seven out of eight best matched pairs in the conv1 layer stay the same. These results suggest that correlation is an adequate measurement of the similarity between two neurons, and that switching to a mutual information metric would not qualitatively change the correlation-based conclusions presented above.

4 RELAXING THE ONE-TO-ONE CONSTRAINT TO FIND SPARSE, FEW-TO-ONE MAPPINGS

The preceding section showed that, while some neurons have a one-to-one match in another network, for other neurons no one-to-one match exists (with correlation above some modest threshold). For example, 17% of conv1 neurons in Net1 have no match in Net2 with a correlation above 0.5 (Figure 3); this number rises to 37% for conv2, 63% for conv3, and 92% for conv4, before dropping to 75% for conv5 (see Figure S3).

These numbers indicate that, particularly for intermediate layers, a simple one-to-one mapping is not a sufficient model to predict the activations of some neurons in one network given the activations of neurons in another network (even with the same architecture trained on the same task). That result could either be because the representations are unique (i.e. not convergent), or because the

⁸The mutual information between each pair of neurons is estimated using 1D and 2D histograms of paired activation values over 60,000 random activation samples. We discretize the activation value distribution into percentile bins along each dimension, each of which captures 5% of the marginal distribution mass. We also add a special bin with range $(-\inf, 10^{-6}]$ in order to capture the significant mass around 0.

	Sparse Prediction Loss (after 4,500 iterations)					
	decay 0	decay 10^{-5}	decay 10^{-4}	decay 10^{-3}	decay 10^{-2}	decay 10^{-1}
conv1	0.170	0.169	0.162	0.172	0.484	0.517
conv2	0.372	0.368	0.337	0.392	0.518	0.514
conv3	0.434	0.427	0.383	0.462	0.497	0.496
conv4	0.478	0.470	0.423	0.477	0.489	0.488
conv5	0.484	0.478	0.439	0.436	0.478	0.477

Table 1: Average prediction error for mapping layers with varying L1 penalties (i.e. decay terms). Larger decay parameters enforce stronger sparsity in the learned weight matrix. Notably, on conv1 and conv2, the prediction errors do not rise much compared to the dense (decay = 0) case with the imposition of a sparsity penalty until after an L1 penalty weight of over 10^{-3} is used. This region of roughly constant performance despite increasing sparsity pressure is shown in bold. That such extreme sparsity does not hurt performance implies that each neuron in one network can be predicted by only one or a few neurons in another network. For the conv3, conv4, and conv5 layers, the overall error is higher, so it is difficult to draw any strong conclusions regarding those layers. The high errors could be because of the uniqueness of the learned representations, or the optimization could be learning a suboptimal mapping layer for other reasons.

best possible one-to-one mapping is insufficient to tell the complete story of how one representation is related to another. We can think of a one-to-one mapping as a model that predicts activations in the second network by multiplying the activations of the first by a permutation matrix — a square matrix constrained such that each row and each column contain a single one and the rest zeros. Can we do better if we learn a model without this constraint?

We can relax this one-to-one constraint to various degrees by learning a *mapping layer* with an L1 penalty (known as a LASSO model, (Tibshirani, 1996)), where stronger penalties will lead to sparser (more few-to-one or one-to-one) mappings. This sparsity pressure can be varied from quite strong (encouraging a mostly one-to-one mapping) all the way to zero, which encourages the learned linear model to be dense. More specifically, to predict one layer’s representation from another, we learn a single mapping layer from one to the other (similar to the “stitching layer” in Lenc & Vedaldi (2015)). In the case of convolutional layers, this mapping layer is a convolutional layer with 1×1 kernel size and number of output channels equal to the number of input channels. The mapping layer’s parameters can be considered as a square weight matrix with side length equal to the number of units in the layer; the layer learns to predict any unit in one network via a linear weighted sum of any number of units in the other. The model and resulting square weight matrices are shown in Figure 5. This layer is then trained to minimize the sum of squared prediction errors plus an L1 penalty, the strength of which is varied.⁹

Mapping layers are trained for layers conv1 – conv5. The average squared prediction errors for each are shown in Table 1 for a variety of L1 penalty weights (i.e. different decay values). For the conv1 and conv2 layers, the prediction errors do not rise with the imposition of a sparsity penalty until a penalty greater than 10^{-3} . A sparsity penalty as high as 10^{-3} results in mapping layer models that are nearly as accurate as their dense counterparts, but that contain mostly zero weights. Additional experiments for conv1 revealed that a penalty multiplier of $10^{-2.6}$ provides a good trade-off between sparsity and accuracy, resulting in a model with sparse prediction loss 0.235, and an average of 4.7 units used to predict each unit in the target network (i.e. an average of 4.7 significantly non-zero weights). For conv2 the 10^{-3} multiplier worked well, producing a model with an average of 2.5 non-zero connections per predicted unit. The mapping layers for higher layers (conv3 – conv5) showed poor performance even without regularization, for reasons we do not yet fully understand, so further

⁹Both representations (input and target) are taken after the relu is applied. Before training, each channel is normalized to have mean zero and standard deviation $1/\sqrt{N}$, where N is the number of dimensions of the representation at that layer (e.g. $N = 55 \cdot 55 \cdot 96 = 290400$ for conv1). This normalization has two effects. First, the channels in a layer are all given equal importance, without which the channels with large activation values (see Figure S10) dominate the cost and are predicted well at the expense of less active channels, a solution which provides little information about the less active channels. Second, the representation at any layer for a single image becomes approximately unit length, making the initial cost about the same on all layers and allowing the same learning rate and SGD momentum hyperparameters to be used for all layers. It also makes the effect of specific L1 multipliers approximately commensurate and allows for rough comparison of prediction performance between layers, because the scale is constant.

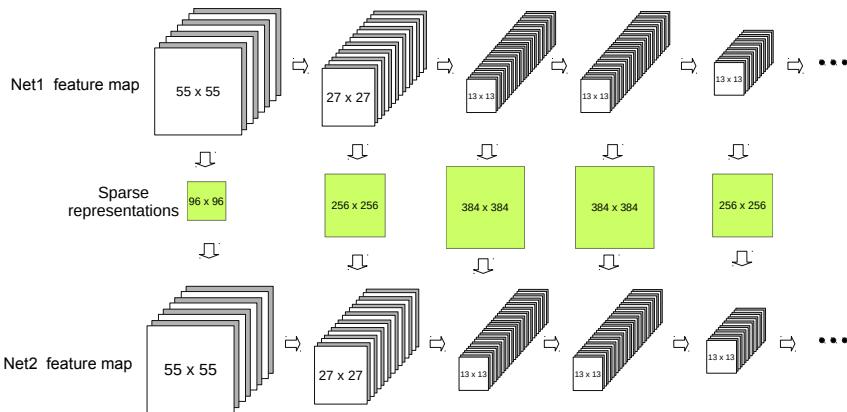


Figure 5: A visualization of the network-to-network sparse “mapping layers” (green squares). The layers are trained independently of each other and with an L1 weight penalty to encourage sparse weights.

results on those layers are not included here. Future investigation with different hyperparameters or different architectures (e.g. multiple hidden layers) could train more powerful predictive models for these layers.

The one-to-one results of Section 3 considered in combination with these results on sparse prediction shed light on the open, long-standing debate about the extent to which learned representations are local vs. distributed : The units that match well one-to-one suggest the presence of a *local* code, where each of these dimensions is important enough, independent enough, or privileged enough in some other way to be relearned by different networks. Units that do not match well one-to-one, but are predicted well by a sparse model, suggest the presence, along those dimensions, of *slightly distributed* codes.

The results could have been otherwise: if all units could accurately be matched one-to-one, we would suspect a local code across the whole layer. On the other hand, if making predictions from one network to another required a dense affine transformation, then we would interpret the code as *fully distributed*, with each unit serving only as a basis vector used to span a large dimensional subspace, whose only requirement was to have large projection onto the subspace (to be useful) and small projection onto other basis vectors (to be orthogonal). The story that actually emerges is that the first two layers use a mix of a local and a slightly distributed code.

Figure 6 shows a visualization of the learned weight matrix for conv1, along with a permuted weight matrix that aligns units from Net2 with the Net1 units that most predict them. We also show two example units in Net2 and, for each, the only three units in Net1 that are needed to predict their activation values.

These sparse prediction results suggest that small groups of units in each network span similar subspaces, but we have not yet identified or visualized the particular subspaces that are spanned. Below we present one approach to do so by using the sparse prediction matrix directly, and in the Supplementary Section S.3 we discuss a related approach using spectral clustering.

For a layer with s channels we begin by creating a $2s \times 2s$ block matrix B by concatenating the blocks $[I, W; WT, I]$ where I is the $s \times s$ identity matrix and W is the learned weight matrix. Then we use Hierarchical Agglomerative Clustering (HAC), as implemented in Scikit-learn (Pedregosa et al., 2011) to recursively cluster individual units into clusters, and those clusters into larger clusters, until all have been joined into one cluster. The HAC algorithm as adapted to this application works in the following way: (1) For all pairs of units, we find the biggest off-diagonal value in B , i.e. the largest prediction weight; (2) We pair those two units together into a cluster and consider it as a single entity for the remainder of the algorithm; (3) We start again from step 2 using the same process (still looking for the biggest value), but whenever we need to compare unit \leftrightarrow cluster or cluster \leftrightarrow cluster, we use the average unit \leftrightarrow unit weight over all cross-cluster pairs;(4) Eventually the process terminates when there is a single cluster.¹⁰

¹⁰For example, in the conv1 layer with 96 channels, this happens after $96 \cdot 2 - 1 = 191$ steps.

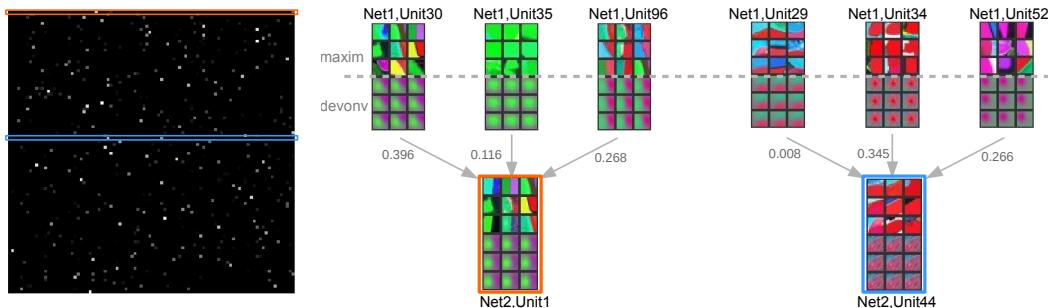


Figure 6: (left) The learned mapping layer from Net1 to Net2 for the conv1 layer. (right) Two example units (bottom) in Net2 — which correspond to the same colored rows in the left weight matrix — together with, for each, the only three units in Net1 that are needed to predict their activation. To fully visualize the functionality each unit, we plot the top 9 image patches from the validation set that causes the highest activation for that unit (“maxim”), as well as its corresponding “deconv” visualization introduced by Zeiler & Fergus (2014). We also show the actual weight associated with each unit in Net1 in the sparse prediction matrix.

The resulting clustering can be interpreted as a tree with units as leaves, clusters as intermediate nodes, and the single final cluster as the root node. In Figure 7 we plot the B matrix with rows and columns permuted together in the order leaves are encountered when traversing the tree, and intermediate nodes are overlaid as lines joining their subordinate units or clusters. For clarity, we color the diagonal pixels (which all have value one) with green or red if the associated unit came from Net1 or Net2, respectively.

Plotted in this way, structure is readily visible: Most parents of leaf clusters (the smallest merges shown as two blue lines of length two covering a 2×2 region) contain one unit from Net1 and one from Net2. These units can be considered most predictive of each other.¹¹ Slightly higher level clusters show small subspaces, comprised of multiple units from each network, where multiple units from one network are useful for predicting activations from the other network (see the example zoomed regions on the right side of Figure 7).

The HAC method employs greedy merges, which could in some cases be suboptimal. In the Supplementary Section S.3 we explore a related method that is less greedy, but operates on the denser correlation matrix instead. Future work investigating or developing other methods for analyzing the structure of the sparse prediction matrices may shed further light on the shared, learned subspaces of independently trained networks.

5 CONCLUSIONS

We have demonstrated a method for quantifying the feature similarity between different, independently trained deep neural networks. We show how insights may be gained by approximately aligning different neural networks on a feature or subspace level by blending three approaches: a bipartite matching that makes one-to-one assignments between neurons, a sparse prediction and clustering approach that finds one-to-many mappings, and a spectral clustering approach that finds many-to-many mappings. Our main findings include:

1. Some features are learned reliably in multiple networks, yet other features are not consistently learned.
2. Units learn to span low-dimensional subspaces and, while these subspaces are common to multiple networks, the specific basis vectors learned are not.
3. The representation codes are a mix between a local (single unit) code and slightly, but not fully, distributed codes across multiple units.
4. The average activation values of neurons vary considerably within a network, yet the mean activation values across different networks converge to an almost identical distribution.

¹¹Note that the upper right corner of B is $W = W_{1 \rightarrow 2}$, the matrix predicting $\text{Net1} \rightarrow \text{Net2}$, and the lower left is just the transpose $W_{1 \rightarrow 2}^T$. The corners could instead be $W_{1 \rightarrow 2}$ and $W_{2 \rightarrow 1}$, respectively.

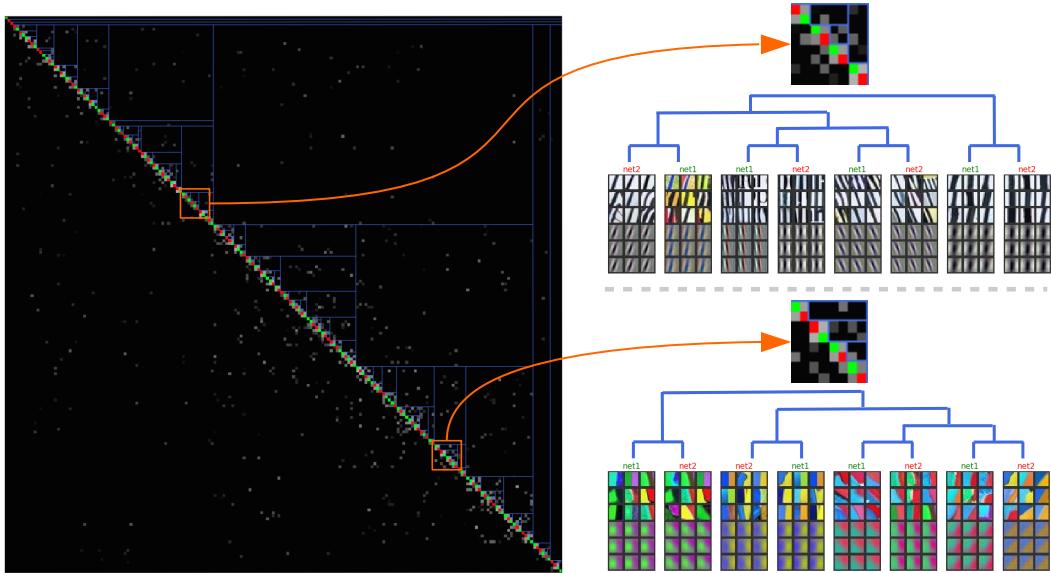


Figure 7: The results of the Hierarchical Agglomerative Clustering (HAC) algorithm described in Section 4 on the conv1 layer. Left: The B matrix permuted by the tree-traversal order of leaf nodes. Pixels on the diagonal are leaf nodes and represent original units of either network (green for Net1 and red for Net2). The brighter the gray pixel is, the larger the weight is in the matrix. See text for a complete interpretation. Right: Two zoomed in regions of the diagonal, showing two different four-dimensional subspaces spanned by four units in each network. The top 9 and bottom 9 images correspond to the maxim and deconv visualizations, respectively. Best viewed digitally with zoom.

6 FUTURE WORK

The findings in this paper open up new future research directions, for example: (1) Model compression. How would removing low-correlation, rare filters affect performance? (2) Optimizing ensemble formation. The results show some features (and subspaces) are shared between independently trained DNNs, and some are not. This suggests testing how feature correlation among different DNNs in an ensemble affects ensemble performance. For example, the “shared cores” of multiple networks could be deduplicated, but the unique features in the tails of their feature sets could be kept. (3) Similarly, one could (a) post-hoc assemble ensembles with greater diversity, or even (b) directly encourage ensemble feature diversity during training. (4) Certain visualization techniques, e.g., deconv (Zeiler & Fergus, 2014), DeepVis (Yosinski et al., 2015), have revealed neurons with multiple functions (e.g. detectors that fire for wheels and faces). The proposed matching methods could reveal more about why these arise. Are these units consistently learned because they are helpful or are they just noisy, imperfect features found in local optima? (5) Model combination: can multiple models be combined by concatenating their features, deleting those with high overlap, and then fine-tuning? (6) Apply the analysis to networks with different architectures — for example, networks with different numbers of layers or different layer sizes — or networks trained on different subsets of the training data. (7) Study the correlations of features in the same network, but across training iterations, which could show whether some features are trained early and not changed much later, versus perhaps others being changed in the later stages of fine-tuning. This could lead to complementary insights on learning dynamics to those reported by Erhan et al. (2009). (8) Study whether particular regularization or optimization strategies (e.g., dropout, ordered dropout, path SGD, etc.) increase or decrease the convergent properties of the representations to facilitate different goals (more convergent would be better for data-parallel training, and less convergent would be better for ensemble formation and compilation).

ACKNOWLEDGMENTS

The authors are grateful to the NASA Space Technology Research Fellowship (JY) for funding, Wendy Shang for conversations and initial ideas, Yoshua Bengio for modeling suggestions, and Anh Nguyen and Kilian Weinberger for helpful comments and edits. This work was supported in part by US Army Research Office W911NF-14-1-0477, NSF grant 1527232. Jeff Clune was

supported by an NSF CAREER award (CAREER: 1453549) and a hardware donation from the NVIDIA Corporation.

REFERENCES

- Arora, Sanjeev, Bhaskara, Aditya, Ge, Rong, and Ma, Tengyu. Provable bounds for learning some deep representations. In *ICML*, pp. 584–592, 2014.
- Dauphin, Yann, Pascanu, Razvan, Gülcabay, Çaglar, Cho, Kyunghyun, Ganguli, Surya, and Bengio, Yoshua. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572, 2014. URL <http://arxiv.org/abs/1406.2572>.
- Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.
- Eigen, David, Rolfe, Jason, Fergus, Rob, and LeCun, Yann. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847*, 2013.
- Erhan, Dumitru, Manzagol, Pierre-Antoine, Bengio, Yoshua, Bengio, Samy, and Vincent, Pascal. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *International Conference on artificial intelligence and statistics*, pp. 153–160, 2009.
- Goodfellow, Ian J, Shlens, Jonathon, and Szegedy, Christian. Explaining and Harnessing Adversarial Examples. *ArXiv e-prints*, December 2014.
- Hopcroft, John E and Karp, Richard M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.
- Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoff. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.
- Lawler, Eugene L. *Combinatorial optimization: networks and matroids*. Courier Corporation, 1976.
- Lenc, Karel and Vedaldi, Andrea. Understanding image representations by measuring their equivariance and equivalence. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- Mahendran, Aravindh and Vedaldi, Andrea. Understanding deep image representations by inverting them. *arXiv preprint arXiv:1412.0035*, 2014.
- Montavon, Grégoire, Braun, Mikio L, and Müller, Klaus-Robert. Kernel analysis of deep networks. *The Journal of Machine Learning Research*, 12:2563–2581, 2011.
- Neyshabur, Behnam and Panigrahy, Rina. Sparse matrix factorization. *arXiv preprint arXiv:1311.3315*, 2013.
- Nguyen, Anh, Yosinski, Jason, and Clune, Jeff. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2014.
- Paul, Arnab and Venkatasubramanian, Suresh. Why does deep learning work?-a perspective from group theory. *arXiv preprint arXiv:1412.6621*, 2014.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- Simonyan, Karen, Vedaldi, Andrea, and Zisserman, Andrew. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034, presented at ICLR Workshop 2014*, 2013.

- Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian J., and Fergus, Rob. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.
- Tibshirani, Robert. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.
- Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3320–3328. Curran Associates, Inc., December 2014.
- Yosinski, Jason, Clune, Jeff, Nguyen, Anh, Fuchs, Thomas, and Lipson, Hod. Understanding neural networks through deep visualization. In *Deep Learning Workshop, International Conference on Machine Learning (ICML)*, 2015.
- Zeiler, Matthew D and Fergus, Rob. Visualizing and understanding convolutional networks. In *ECCV*, pp. 818–833, 2014.
- Zhou, Bolei, Khosla, Aditya, Lapedriza, Àgata, Oliva, Aude, and Torralba, Antonio. Object detectors emerge in deep scene cnns. In *International Conference on Learning Representations (ICLR)*, volume abs/1412.6856, 2014. URL <http://arxiv.org/abs/1412.6856>.

SUPPLEMENTARY INFORMATION

S.1 ACTIVATION VALUES: HIGH CORRELATION VS. LOW CORRELATION

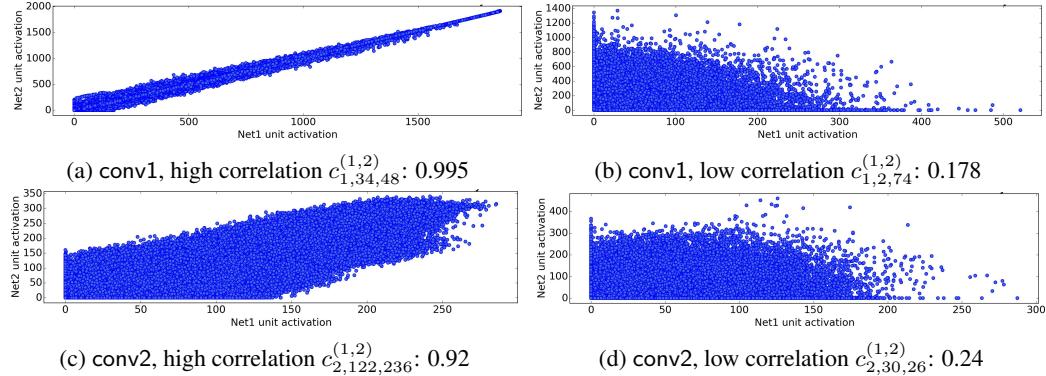


Figure S1: Activation values are computed across 5,000 randomly sampled images and all spatial positions (55×55 and 27×27 for layers conv1 and conv2, respectively). The joint distributions appear simple enough to suggest that a correlation measure is sufficient to find matching units between networks.

S.2 ADDITIONAL MATCHING RESULTS

Figure S3 shows additional results of comparison of assignments produced by semi-matching and matching methods in conv2 – conv5. For each unit, both the semi-matching and matching are found, then the units are sorted in order of decreasing semi-matching value and both correlation values are plotted.

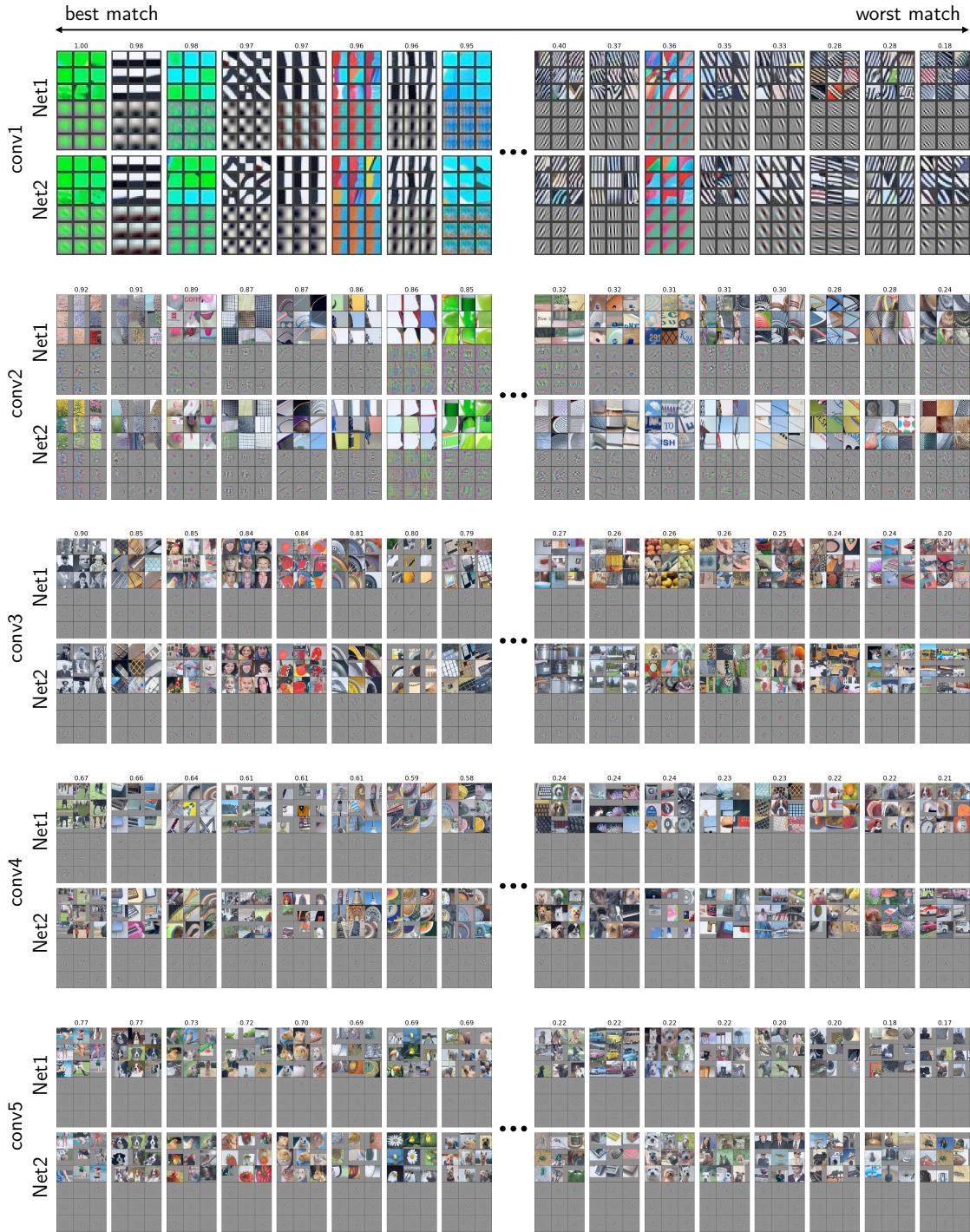


Figure S2: With assignments chosen by semi-matching, the eight best (highest correlation, left) and eight worst (lowest correlation, right) matched features between Net1 and Net2 for the conv1 through conv5 layers. To visualize the functionality each unit, we plot the nine image patches (in a three by three block) from the validation set that causes the highest activation for that unit and directly beneath that block show the “deconv” visualization of each of the nine images. Best view with significant zoom in.

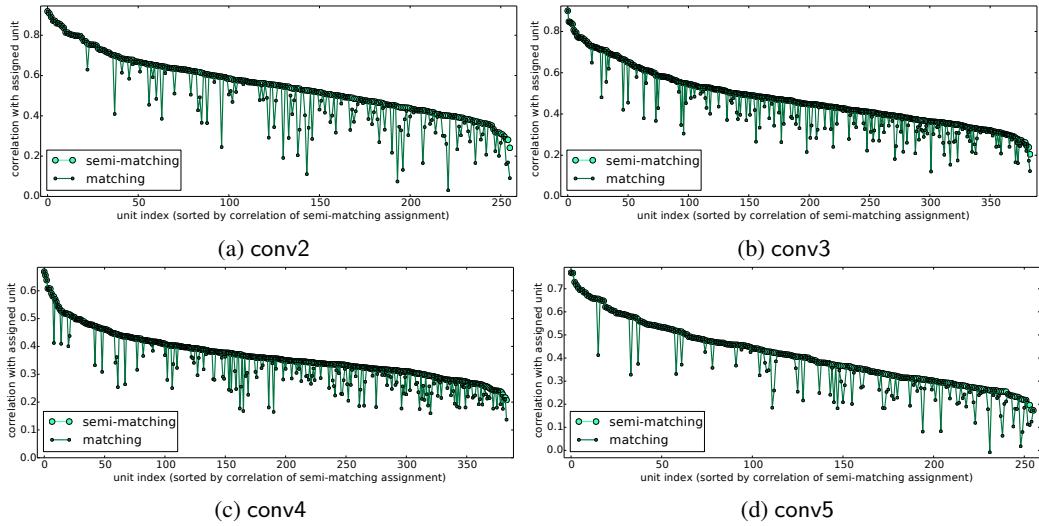


Figure S3: Correlations between units in conv2 - conv5 layers of Net1 and their paired units in Net2, where pairings are made via semi-matching (large light green circles) or matching (small dark green dots).

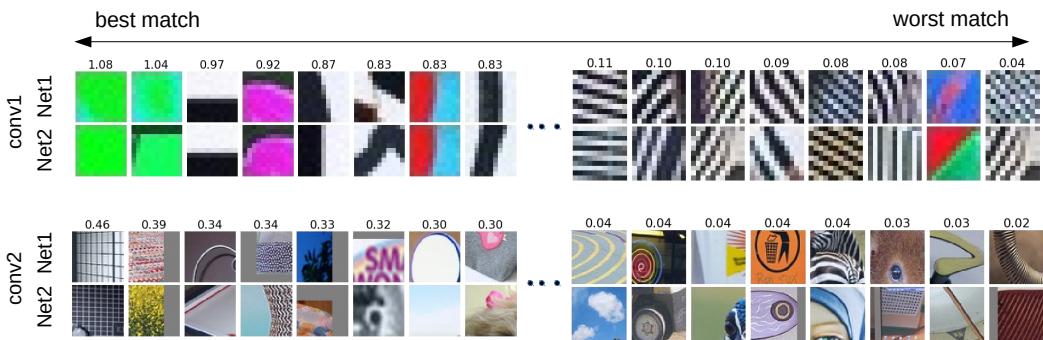


Figure S4: The eight best (highest mutual information, left) and eight worst (lowest mutual information, right) features in the semi-matching between Net1 and Net2 for the conv1 and conv2 layers.

S.3 DOES RELAXING THE ONE-TO-ONE CONSTRAINT TO FIND MANY-TO-MANY GROUPINGS REVEAL MORE SIMILARITIES BETWEEN WHAT DIFFERENT NETWORKS LEARN?

Since the preceding sections have shown that neurons may not necessarily correspond via a globally consistent one-to-one matching between networks, we now seek to find many-to-many matchings between networks using a spectral clustering approach.

S.3.1 NEURON SIMILARITY GRAPHS

We define three types of similarity graphs based on the correlation matrices obtained above.

Single-net neuron similarity graphs. Given a fully trained DNN X and a specified layer l , we first construct the *single-net neuron similarity graph* $G_{X,l} = (V, E)$. Each vertex v_p in this graph represents a unit p in layer l . Two vertices are connected by an edge of weight c_{pq} if the correlation value c_{pq} in the self-correlation matrix $\text{corr}(X_l, X_l)$ between unit p and unit q is greater than a certain threshold τ .

Between-net neuron similarity graphs. Given a pair of fully trained DNNs X and Y , the *between-net neuron similarity graph* $G_{XY,l} = (V, E)$ can be constructed in a similar manner. Note that $G_{XY,l}$ is a bipartite graph and contains twice as many vertices as that in $G_{X,l}$ since it incorporates units from both networks. Two vertices are connected by an edge of weight c_{pq} if the correlation value c_{pq} in the between-net correlation matrix $\text{corr}(X_l, Y_l)$ between unit p in X and unit q in Y is greater than a certain threshold τ .

Combined similarity graphs. The problem of matching neurons in different networks can now be reformulated as finding a partition in the combined neuron similarity graphs $G_{X+Y,l} = G_{X,l} + G_{Y,l} + G_{XY,l}$, such that the edges between different groups have very low weights and the edges within a group have relatively high weights.

S.3.2 SPECTRAL CLUSTERING AND METRICS FOR NEURON CLUSTERS

We define three types of similarity graphs based on the correlation matrices obtained above (see Section S.3.1 for definition). Define $W_l \in \mathbb{R}^{2S_l \times 2S_l}$ to be the combined correlation matrices between two DNNs, X and Y in layer l , where w_{jk} is the entry at j th row and k th column of that matrix. S_l is the number of channels (units) in layer l . W_l is given by

$$W_l = \begin{bmatrix} \text{corr}(X_l, X_l) & \text{corr}(X_l, Y_l) \\ \text{corr}(X_l, Y_l)^\top & \text{corr}(Y_l, Y_l) \end{bmatrix}.$$

The unnormalized Laplacian matrix is defined as $L_l = D_l - W_l$, where the degree matrix D_l is the diagonal matrix with the degrees $d_j = \sum_{k=1}^{2S_l} w_{jk}$. The unnormalized Laplacian matrix and its eigenvalues and eigenvectors can be used to effectively embed points in a lower-dimension representation without losing too information about spatial relationships. If neuron clusters can be identified, then the Laplacian L_l is approximately block-diagonal, with each block corresponding a cluster. Assuming there are k clusters in the graph, spectral clustering would then take the first k eigenvectors¹², $\mathcal{U} \in \mathbb{R}^{2S_l \times k}$, corresponding to the k smallest eigenvalues, and partition neurons in the eigenspace with the k -means algorithm.

S.3.3 SPECTRAL CLUSTERING RESULTS

We use Net1 and Net2 as an example for showing the results of matching neurons between DNNs. Figure S5 shows the permuted combined correlation matrix after apply the spectral clustering algorithm for conv1 – conv5. Figure S6 displays 12 neuron clusters with high between-net similarity measurement in conv1 layer, and Figure S7 displays the top 8 neuron clusters with highest between-net similarity measurement in conv2 layer. The matching results imply that there exists many-to-many correspondence of the feature maps between two fully trained networks with different random initializations, and the number of neurons learning the same feature can be different between networks. For example, the four units of $\{89, 90, 134, 226\}$ in Net1 and three units of $\{2, 39, 83\}$ in Net2 are learning the features about green objects.

¹²In practice, when the number of clusters is unknown, the best value of k to choose is where the eigenvalue shows a relatively abrupt change.

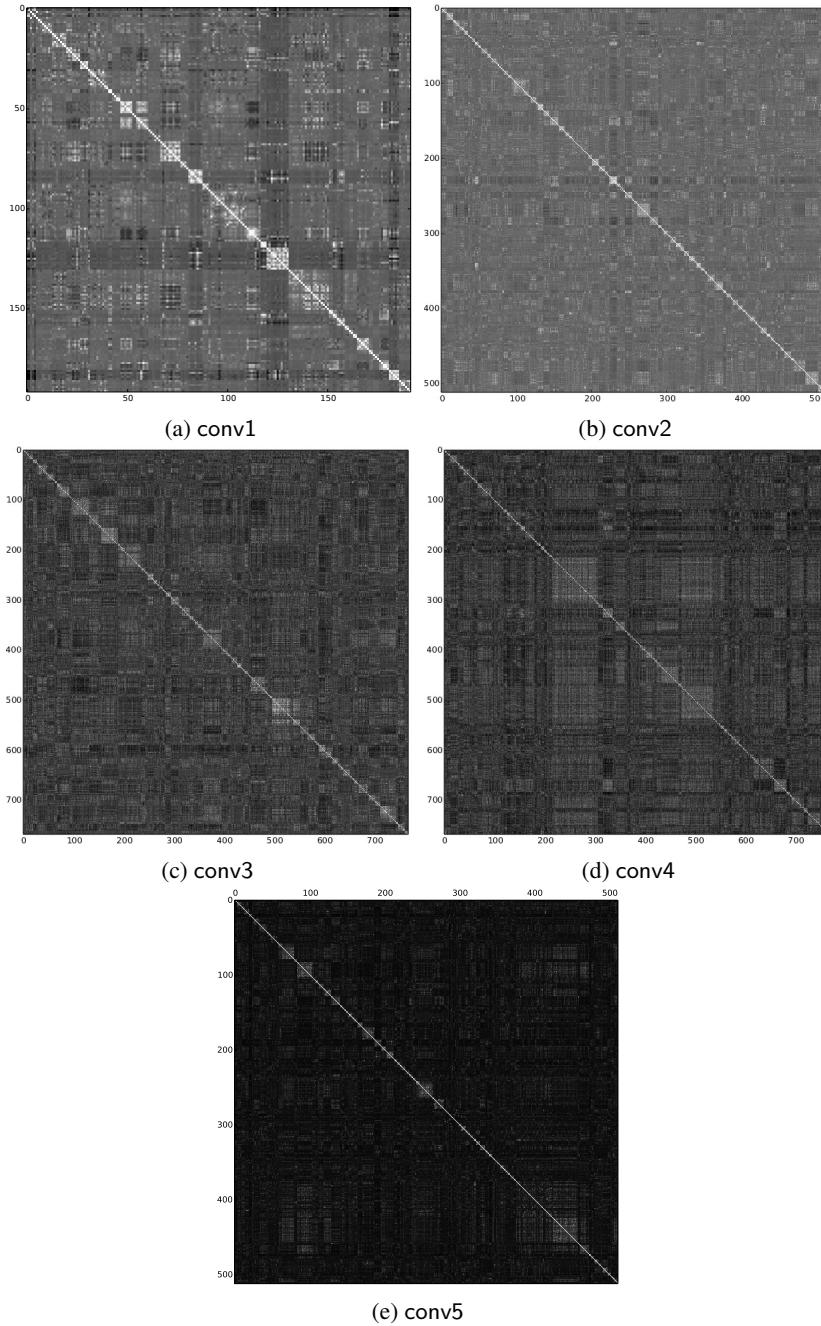


Figure S5: The permuted combined correlation matrix after apply spectral clustering method (conv1 – conv5). The diagonal block structure represents the groups of neurons that are clustered together. The value of k adopted for these five layers are: $\{40, 100, 100, 100, 100\}$, which is consistent with the parameter setting for other experiments in this paper.

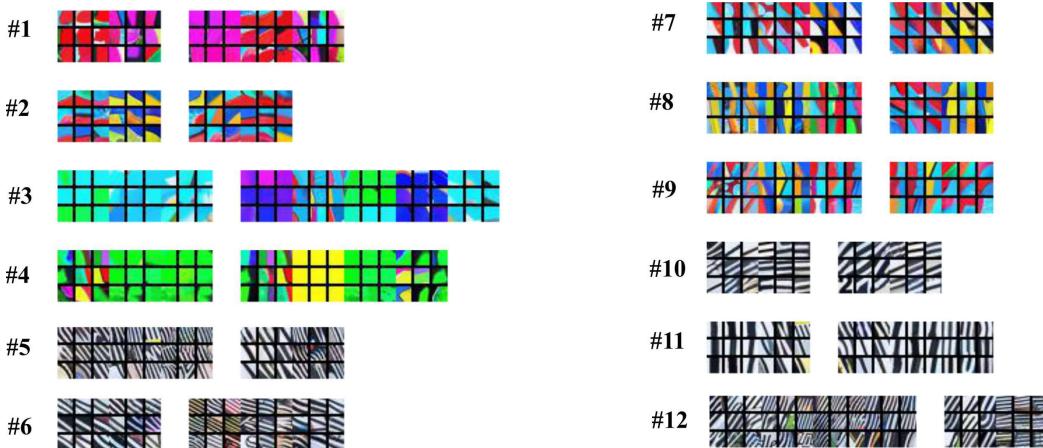


Figure S6: The neuron matchings between two DNNs (Net1 and Net2) in conv1 layer. Here we display the 12 neuron clusters with relatively high between-net similarity measurement. Each labeled half-row corresponds to one cluster, where the filter visualizations for neurons from Net1 and Net2 are separated by white space slot. The matching results imply that there exists many-to-many correspondence of the feature maps between two fully trained networks with different random initializations. For instance, in cluster #6, neurons from Net1 and Net2 are both learning 135° diagonal edges; and neurons in cluster #10 and #12 are learning 45° diagonal edges.

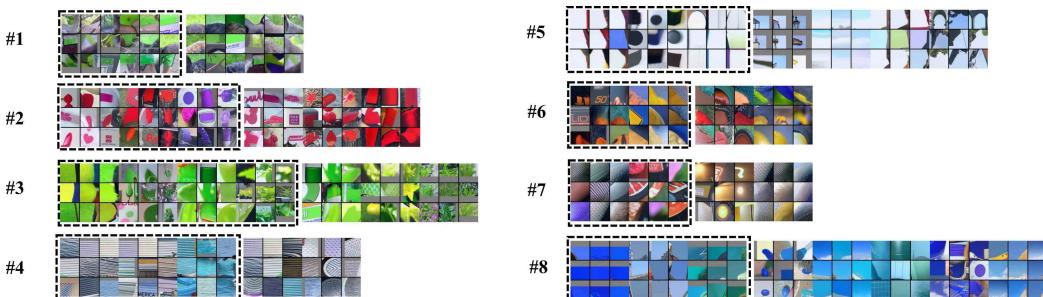


Figure S7: The neuron matchings between two DNNs: Net1 and Net2. Each of the 3×3 block displays the top 9 image patches that cause the highest activations to each neuron. Each labeled half-row corresponds to one cluster, where the filter visualizations with dashed boxes represent neurons from Net1 and those without are from Net2. For example, there are 7 neurons learning similar features in cluster #3, where the left four neurons are in Net1 and the right three are from Net2. Best viewed in electronic form with zoom.

We also computed and visualized the matching neurons in other layers as well. The results of conv1 are shown in Figure S6. As noted earlier, the conv1 layer tends to learn more general features like Gabor filters (edge detectors) and blobs of color. Our approach finds many matching Gabor filters (e.g., clusters #5, #6, #10, #11 and #12), and also some matching color blobs (e.g., clusters #1, #3 and #4).

S.3.4 HIERARCHICAL SPECTRAL CLUSTERING RESULTS

Due to the stochastic effects of randomly initializing centroids in k -means clustering, some of the initial clusters contains more neurons than others. To get more fine-grained cluster structure, we recurrently apply k -means clustering on any clusters with size $> 2\alpha \cdot \mathcal{S}_l$, where α is a tunable parameter for adjusting the maximum size of the leaf clusters. Figure S8 shows the partial hierarchical structure of neuron matchings in the conv2 layer. The cluster at the root of the tree is a first-level cluster that contains many similar units from both DNNs. Here we adopt $\alpha = 0.025$ for the conv2 layer, resulting in a hierarchical neuron cluster tree structure with leaf clusters containing less than 6 neurons from each network. The bold box of each subcluster contains neurons from Net1 and the remaining neurons are from Net2. For example, in subcluster #3, which shows conv2 features, units

$\{62, 137, 148\}$ from Net1 learned similar features as units $\{33, 64, 230\}$ from Net2, namely, red and magenta objects.

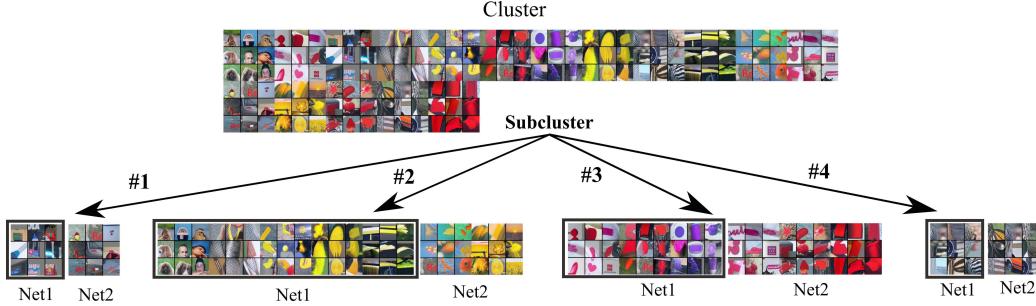


Figure S8: The hierarchical structure of neuron matchings between two DNNs: Net1 and Net2 (conv2 layer). The initial clusters are obtained using spectral clustering with the number of clusters $k = 100$ and threshold $\tau = 0.2$.

S.3.5 METRICS FOR NEURON CLUSTERS

Here we introduce two metrics for quantifying the similarity among neurons grouped together after applying the clustering algorithm above.

$$\text{Between-net similarity: } \text{Sim}_{X_l \rightarrow Y_l} = \left(\sum_{p=1}^{S_l} \sum_{q=1}^{S_l} \text{corr}(X_l, Y_l)_{pq} \right) / S_l^2$$

$$\text{Within-net similarity: } \text{Sim}_{X_l, Y_l} = (\text{Sim}_{X_l \rightarrow X_l} + \text{Sim}_{Y_l \rightarrow Y_l}) / 2$$

We further performed experiments in quantifying the similarity among neurons that are clustered together. Figure S9 shows the between-net and within-net similarity measurement for conv1 – conv5. The value of k for initial clustering is set to be 40 for conv1 layer and 100 for all the other layers. In our experiments, the number of final clusters obtained after further hierarchical branching is $\{43, 113, 130, 155, 131\}$. The tail in those curves with value 0 is due to the non-existence of between-net similarity for the clusters containing neurons from only one of the two DNNs. To better capture the distribution of non-zero similarity values, we leave out the tail after 100 in the plot for conv3 - conv5 layers.

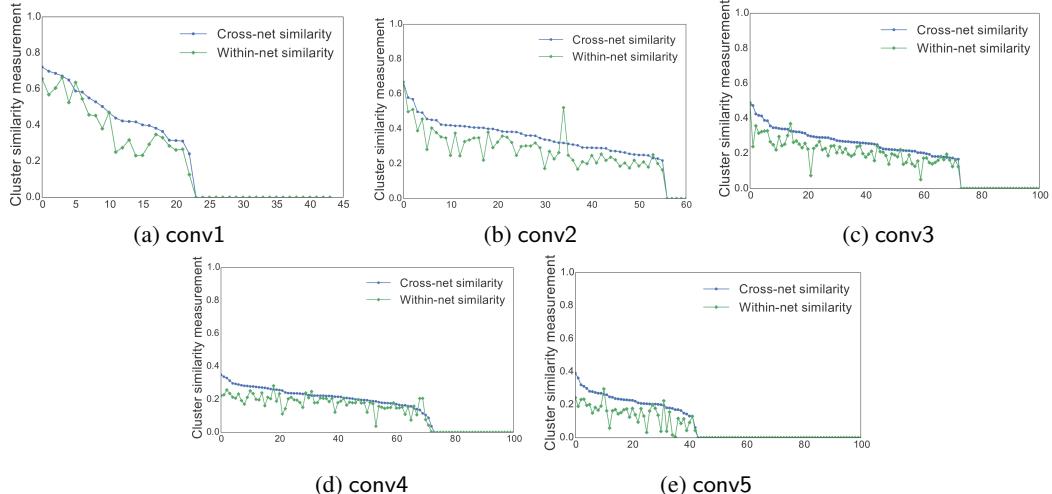


Figure S9: The distribution of between-net and within-net similarity measurement after clustering neurons (conv1 – conv5). The x-axis represents obtained clusters, which is reshuffled according to the sorted between-net similarity value.

S.4 COMPARING AVERAGE NEURAL ACTIVATIONS WITHIN AND BETWEEN NETWORKS

The first layer of networks trained on natural images (here the conv1 layer) tends to learn channels matching patterns similar to Gabor filters (oriented edge filters) and blobs of color. As shown in Figures S10, S11, and S12, there are certain systematic biases in the relative magnitudes of the activations of the different channels of the first layer. Responses of the low frequency filters have much higher magnitude than that of the high frequency filters. This phenomenon is likely a consequence of the $1/f$ power spectrum of natural images in which, on average, low spatial frequencies tend to contain higher energy (because they are more common) than high spatial frequencies.

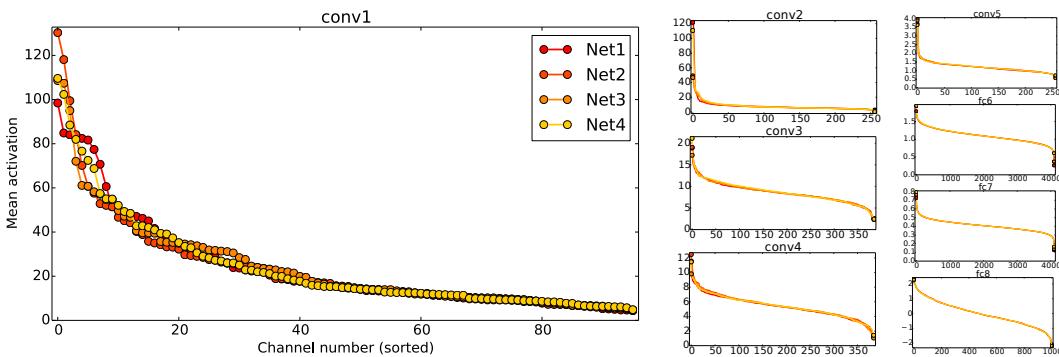


Figure S10: The average activation values of each unit on all layers of Net1 – Net4. A couple salient effects are observable. First, in a given network, average activations vary widely within each layer. While most activations fall within a relatively narrow band (the middle of each plot), a rare few highly active units have one or two orders of magnitude higher average output than the least active. Second, the overall distribution of activation values is similar across networks. However, also note that the max single activation does vary across networks in some cases, e.g. on the conv2 layer by a factor of two between networks. For clarity, on layers other than conv1 circle markers are shown only at the line endpoints.

In Figure S10 we show the mean activations for each unit of four networks, plotted in sorted order from highest to lowest. First and most saliently, we see a pattern of widely varying mean activation values across units, with a gap between the most active and least active units of one or two orders of magnitude (depending on the layer). Second, we observe a rough overall correspondence in the spectrum of activations between the networks. However, the correspondence is not perfect: although much of the spectrum matches well, the most active filters converged to solutions of somewhat different magnitudes. For example, the average activation value of the filter on conv2 with the highest average activation varies between 49 to 120 over the four networks; the range for conv1 was 98 to 130.¹³ This effect is more interesting considering that all filters were learned with constant weight decay, which pushes all individual filter weights and biases (and thus subsequent activations) toward zero with the same force.

Figure S11 shows the conv1 units with the highest and lowest activations for each of the four networks. As mentioned earlier (and as expected), filters for lower spatial frequencies have higher average activation, and vice versa. What is surprising is the relative lack of ordering between the four networks. For example, the top two most active filters in Net1 respond to constant color regions of black or light blue, whereas none of the top eight filters in Net2 or Net3 respond to such patterns. One might have thought that whatever influence from the dataset caused the largest filters to be black and blue in the first network would have caused similar constant color patches to dominate the other networks, but we did not observe such consistency. Similar differences exist when observing the learned edge filters: in Net1 and Net4 the most active edge filter is horizontal; in Net2 and Net3 it is vertical. The right half of Figure S11 depicts the least active filters. The same lack of alignment arises, but here the activation values are more tightly packed, so the exact ordering is less meaningful. Figure S12 shows the even more widely varying activations from conv2.

¹³Recall that the units use rectified linear activation functions, so the activation magnitude is unbounded. The max activation over all channels and all spatial positions of the first layer is often over 2000.

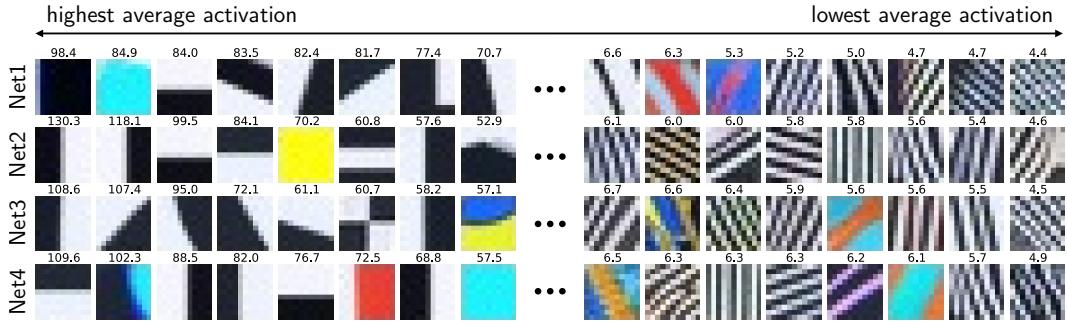


Figure S11: The most active (left) to least active (right) conv1 filters from Net1 – Net4, with average activation values printed above each filter. The most active filters generally respond to low spatial frequencies, and the least active filtered to high spatial frequencies, but the lack of alignment is interesting (see text).

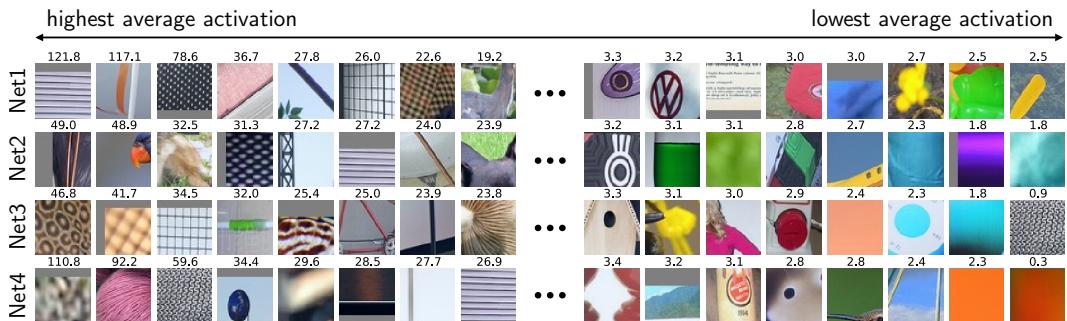


Figure S12: Most active (left) to least active (right) conv2 filters as in Figure S11. Compared to the conv1 filters, here the separation and misalignment between the top filters is even larger. For example, the top unit responding to horizontal lines in Net1 has average activation of 121.8, whereas similar units in Net2 and Net4 average 27.2 and 26.9, respectively. The unit does not appear in the top eight units of Net3 at all. The least active units seem to respond to rare specific concepts.