Are the proposed similarity metrics also a measure of functional similarity?

by

Manikanta Srikar Yellapragada

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Computer Science Department
New York University
November 2020

Advisor: Professor Kyunghyun Cho

Second Reader: Professor Joan Bruna

Acknowledgements

I would like to wholeheartedly thank Professor Kyunghyun Cho for advising me on this thesis. Your valuable suggestions have been instrumental in making this thesis a reality. I would like to thank Professor Joan Bruna for agreeing to be the second reader.

I could not have arrived at writing this thesis without the guidance of Cinjon Resnick and Roberta Raileanu. You have been great discussion partners, and I've learned a lot from you.

Finally, I would like to thank my parents for their everlasting support and encouragement.

Abstract

A recent body of work attempts to understand the behavior and training dynamics of neural networks by analyzing intermediate representations and designing metrics to define the similarity between those representations. We observe that the representations of the last layer could be thought of as the functional output of the model up to that point. In this work, we investigate if the similarity between these representations can be considered a stand-in for the similarity of the networks' output functions. This can have an impact for many downstream tasks, but we specifically analyze it in the context of transfer learning. Consequently, we perform a series of experiments to understand the relationship between the representational similarity and the functional similarity of neural networks. We show in two ways that the leading metric for representational similarity, CKA, does not bear a strict relationship with functional similarity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work addresses a question on similarity metrics for neural networks [1–3]. Namely, does the degree of similarity amongst the representations of two networks tell us anything about how similar the functions defined by those networks are? If the representation is taken from layer M of an N layer network, then the function is the composition of that representation and the following $N - M$ layers resulting in an output distribution.

This investigation matters because we have developed strong metrics for representational similarity such as CCA [4], SVCCA [1], PWCCA [2], or CKA [3]. Yet, we have little ability to probe the functional similarity besides training multiple networks on a task and then evaluating the output distributions with respect to some divergence metric such as Jensen-Shannon (JSD) or Kullback-Liebler (KL).

On the other hand, it would be useful to have a way of inferring the functional similarity of two networks from their representations (i.e., features). This would allow us to estimate the functional similarity of two networks on some downstream task without ever actually having to train (or evaluate) them on that task. With this tool, we could, for example, expedite transfer learning in real-world scenarios by grouping networks according to their (feature) similarity and then testing our transfer learning approaches with only one network from each group. If networks with similar representations are guaranteed to have similar outputs, then they must also have similar capabilities on downstream tasks.

We perform a series of experiments to answer the above question. We begin by examining how learned representations cluster by training an SVM to classify among various representation sets. To assess whether feature similarity correlates with functional similarity, we train networks with different architectures and initializations on a wide array of tasks. This allows us to empirically conclude that similarity of representations (measured using CKA) and similarity of functions (as measured by JSD or the Spearman's rank correlation coefficient) are not strictly related. We

further devise a setup whereby we produce networks with high representational similarity but entirely different outputs. The fact that we can do this without any other assumptions validates the empirical study.

Overall, our novel contributions are the following:

- We empirically show that representational similarity is not suggestive of functional similarity across a wide range of domains, including both supervised and reinforcement learning tasks.
- We show that we can create two networks with very high representational similarity but opposing functions.

Chapter 2

Background

In this section, we briefly review different methods of measuring similarity between neural network activation vectors. Let $X \in R^{n \times p1}$ denote a matrix of activations of p_1 neurons for n examples, and $Y \in R^{n \times p2}$ denote a matrix of activations of p_2 neurons for the same n examples. We assume that these matrices are centered and $p_1 \leq p_2$. Q_X and Q_Y represent any orthonormal basis for the columns of X and Y, which can be computed by QR decomposition.

2.1 Canonical correlation analysis (CCA)

Canonical correlation analysis (CCA) [4] is a statistical technique for inferring the relationship between two sets of random variables, X and Y. CCA finds two sets of basis vectors (one for X and one for Y) with the maximum correlation between the projections of X and Y onto the basis vectors. If we consider two layers of a neural network as the two sets of random variables, CCA can be used to find the similarity between the layers. As stated in [2], CCA is invariant to invertible affine transforms, which makes it suitable for neural networks, where representations at each layer go through an affine transform. It also enables comparisons between different networks and layers of different output dimensions, where the ordinary dot product between the representations cannot be computed.

In our experiments, we use the mean squared CCA correlation R^2_{CCA}:

$$R^2_{CCA} = \frac{\sum_{i=1}^{p_1} \rho_i^2}{p1} = \frac{\left\| Q_Y^T Q_X \right\|_F^2}{p_1}$$

where $\|\cdot\|_F$ is the Frobenius norm.

2.2 SVCCA

Despite these advantages, CCA is sensitive to perturbations when the condition number of X or Y is large [5]. To improve robustness, Raghu et al. [1] introduce the singular vector canonical correlation analysis (SVCCA). SVCCA first performs a singular value decomposition on X and Y, and then applies CCA to the top m singular vectors. Raghu et al. [1] use SVCCA to investigate the learning dynamics of a neural network and show that networks converge bottom-up, with initial layers converging to their final representations before later layers. They also compare the similarity of representations across different random initializations of the network's weights.

2.3 Projection-weighted CCA

Inspired by SVCCA, Morcos et al. [2] develop projection weighted canonical correlation analysis (PWCCA) to provide new insights into the representational similarity of CNNs. They investigate whether representational similarity is predictive of generalization by studying two types of networks, generalizing networks (i.e., trained on correct labels) and memorizing networks (i.e., trained on randomized labels(. They show that generalizing networks converge to more similar representations than memorizing networks. They also show that networks with identical topology but different learning rates converge to a small set of diverse solutions.

2.4 Centered Kernel Alignment (CKA)

As described in [3], CCA is invariant to invertible linear transformations. If a similarity index is invariant to an invertible linear transformation, it yields the same result for any representation of width greater than or equal to the dataset size. However, PWCCA is not invariant to orthogonal transformations. Invariance to orthogonal transformation implies invariance to permutation, which is necessary when symmetry is present in neural networks.

These limitations led to the adoption of centered kernel alignment (CKA) as a similarity metric for comparing neural network representations. Cortes et al. [6] introduced it as an approach for learning kernels based on the notion of centered alignment. The key insight is that one can first measure the similarity between every pair of examples in each representation separately, and then compare the similarity structures. Kornblith et al. [3] show that if the inner product is used to measure similarity, the similarity between representational similarity matrices reduces to an intuitive notion of pairwise feature similarity.

In all our experiments, we use linear CKA, which is computed as:

$$CKA(X, Y) = \frac{\left\|Y^T X\right\|_F^2}{\left\|X^T X\right\|_F \left\|Y^T Y\right\|_F}$$

Linear CKA is related to CCA as :

$$R_{CCA}^2 = CKA(Q_X Q_X^T, Q_Y Q_Y^T)\sqrt{\frac{p_2}{p_1}}$$

Chapter 3

Experiments

3.1 Biases in representation space

A good representation should capture information about the input data and be invariant with respect to the model that generated the representation. To investigate the learned representations, we perform a similar experiment as Torralba and Efros [7] using the feature representations from the neural net to verify whether an SVM trained on those representations would be able to tell apart which dataset and model did the representation come from. We train Resnet 110 [8] for classification on two datasets: Cifar100 [9] and Imagenet32. Imagenet32 is the downsampled version of Imagenet [10], where each image is of size 32x32. We choose Imagenet32 to have features of the same dimension. We train networks with two seeds on each of the datasets. Using the four trained networks, and two untrained networks (one for each seed), we generate the feature representations on three test datasets: Cifar100, Imagenet32, and Pascal VOC [11]. We use Pytorch lightning [12] to train the networks.

We use the entire test dataset of Pascal VOC and 5800 randomly chosen images from Cifar100 and Imagenet's test sets. We generate features at the last convolution layer. Each feature was of dimension 1024x2x2, which we flatten to obtain a 4096 dimension vector. We normalize the features to zero mean and unit variance. Using this setup, we obtain a total of 18 representation sets (3 datasets x 6 algorithms). We use an SVM [13] to classify these representations into 18 classes. Linear SVM was used, which handles multiclass classification according to a one-vs-the-rest scheme.

The confusion matrix of SVM would give us insights on the kind of mistakes it made during classification. It is important because these mistakes would help us know if the model or dataset has a larger role in cluster formation. Confusion matrix for the SVM classification is in Figure 3.1. The confusion is frequently over the test dataset used (i.e., Cifar vs. Imagenet vs. VOC).

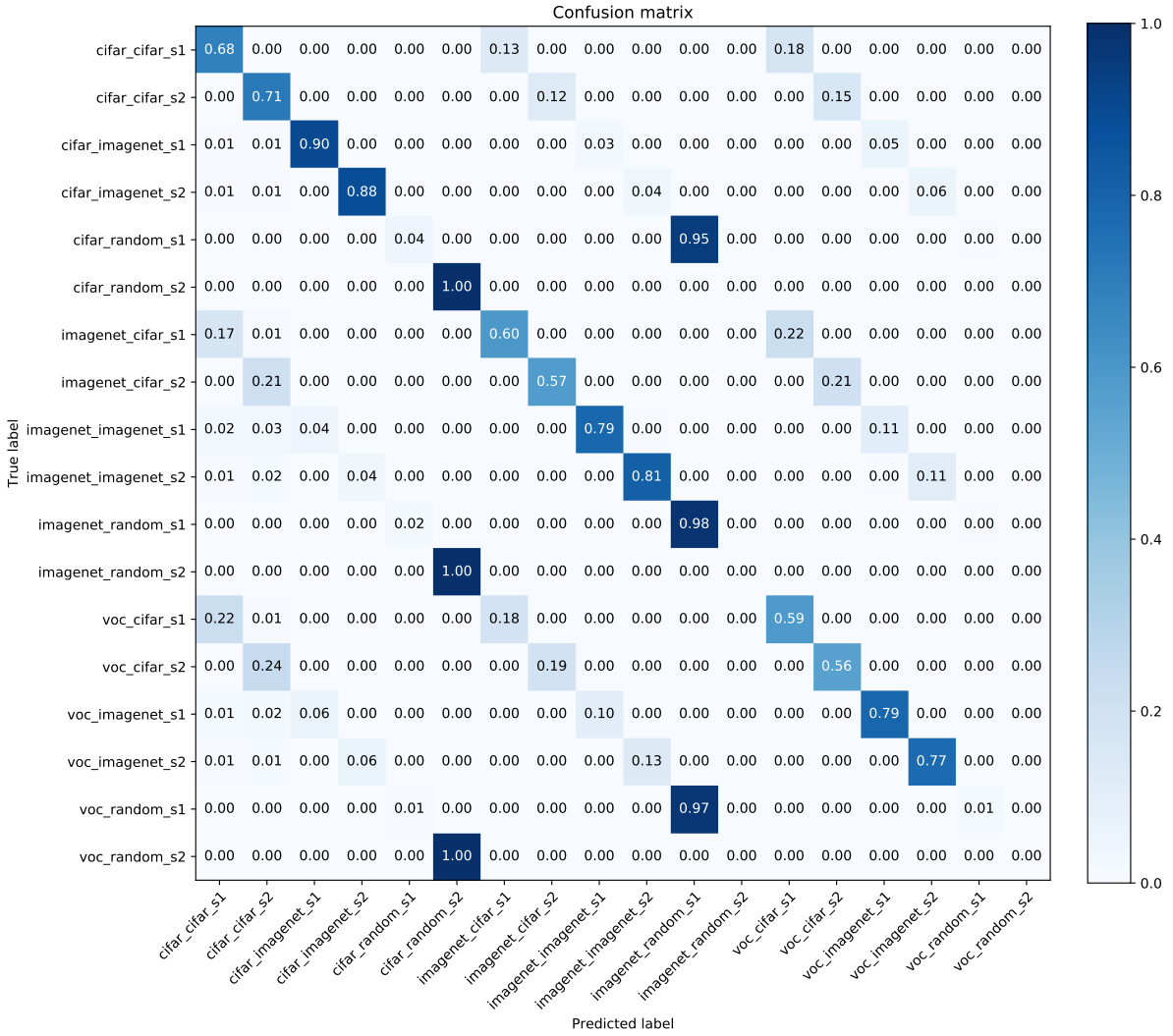
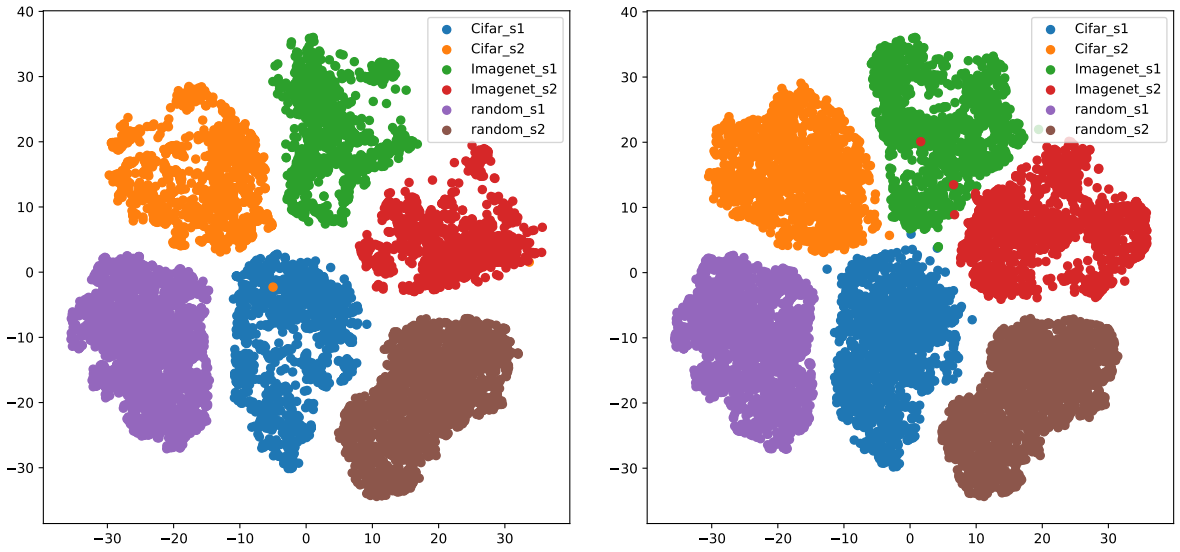


FIGURE 3.1: We train Resnets with two seeds on Imagenet and Cifar, and test on Imagenet, Cifar, and Pascal VOC. We generate feature representations on three datasets for these two seeds and untrained networks. We classify those into 18 classes with an SVM. Cifar_Imagenet_s1 denotes Imagenet representations from a network trained on Cifar. The confusion matrix seems to indicate that the SVM is confused frequently over the test dataset used but never over the model.

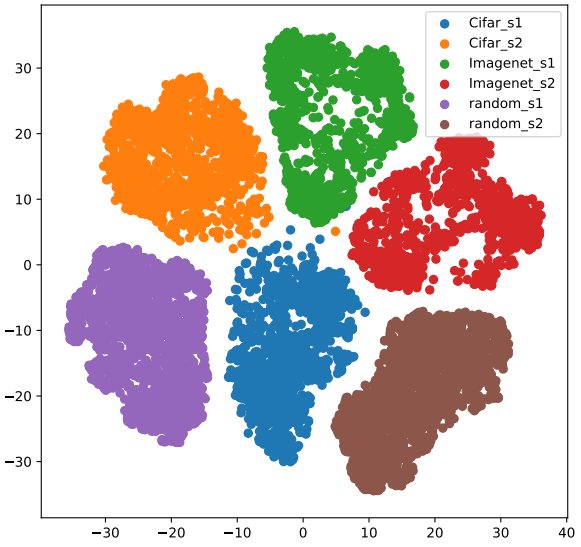
We see that training seed-1 on Cifar gets 70% of the images from Cifar and confuses 15% each with Imagenet and VOC, which is true for both seeds of Cifar. For Imagenet, it does 80% on its dataset, with most of the confusion going to its performance on VOC. On Cifar, it does 90% with the confusion going to VOC. However, on VOC, it does only 80% with confusion split evenly among Cifar and Imagenet. The SVM is never confused about the model, and the confusion does not happen over the seed. The SVM always classifies the random models for seed-1 as being from Imagenet for seed-1. Furthermore, for seed-2, it is always classified as being from Cifar for seed-2.

We also use t-SNE to reduce the feature vectors to 2-dimensions and plot them for each of the test datasets. From the t-SNE plots in Figure 3.2, we observe that there are clusters based on



(a) Representations from Cifar

(b) Representations from Imagenet



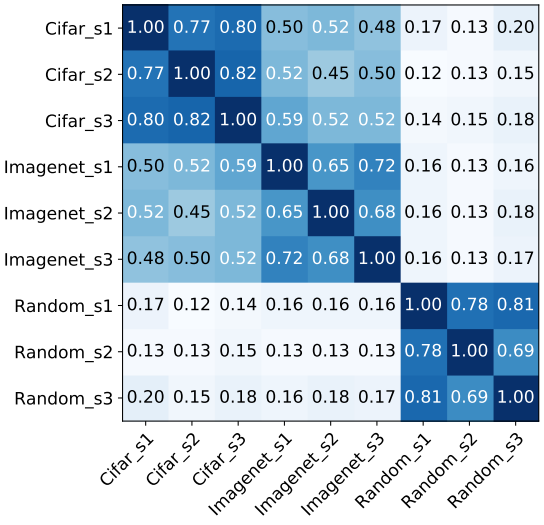
(c) Representations from VOC

FIGURE 3.2: TSNE of features on each of the test datasets. We can see that that there are clusters based on the model (i.e., CNN used to generate the features), and across datasets, the cluster positions remain same.

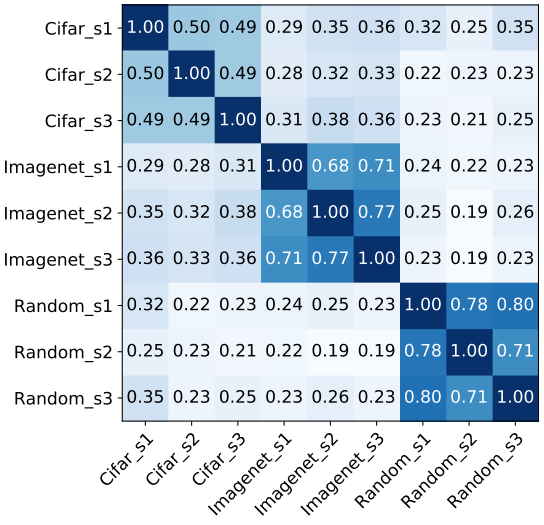
the model (i.e., CNN used to generate features), and across datasets, the cluster positions stay the same. This cluster formation supports the results from SVM that model, and not the dataset has a more significant role in the creation of clusters in representation space.

3.2 Similarity of Network Representations

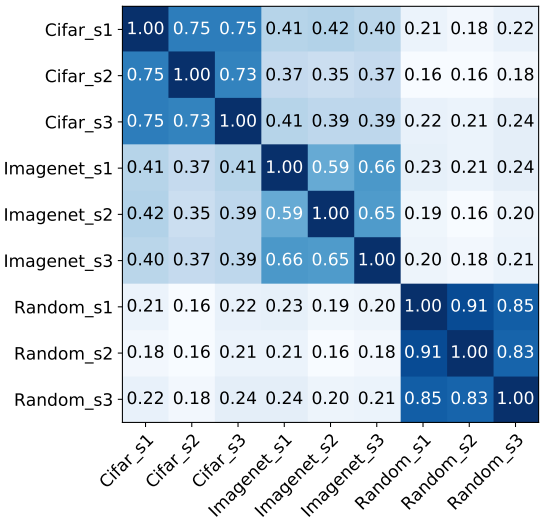
A good representation should capture information about the input data while being invariant with respect to the model used to generate that representation. To investigate whether this is the case for standard training of neural networks, we analyze the similarity between the learned representations of models trained on different datasets.



(a) Tested on Cifar100



(b) Tested on Imagenet32



(c) Tested on Pascal VOC

FIGURE 3.3: CKA similarity of pairwise model representations, computed on test data from Cifar100, Imagenet32, and Pascal VOC (from left to right). The same networks are used to obtain representations on each of the three datasets. Cifar_s1 denotes a network trained on Cifar100 with seed 1, while random_s1 denotes an untrained network initialized with seed 1.

We train Resnet18 [8] on Cifar100 [9] and Imagenet32. Imagenet32 is the downsampled version of Imagenet [10], where each image is of size 32x32. We choose Imagenet32 to have features of the same dimension. For each dataset, we train three networks with different initializations (i.e., seeds). Then we use the trained networks to generate representations of unseen samples from Cifar100, Imagenet32, and Pascal VOC. We use the entire test dataset of Pascal VOC and 5800 randomly chosen images from Cifar100's and Imagenet32's test sets. Here the representations are defined as the features after the last convolutional layer. Each feature has dimension 1024x2x2, which is flattened to obtain a 4096-dimensional vector. We also generate features from three untrained networks with different initializations. Using this setup, we obtain a total of 27 representation sets (9 models x 3 datasets). We then use linear CKA between each pair of models' representations keeping the test dataset constant. Figure 3.3 shows the results with linear CKA.

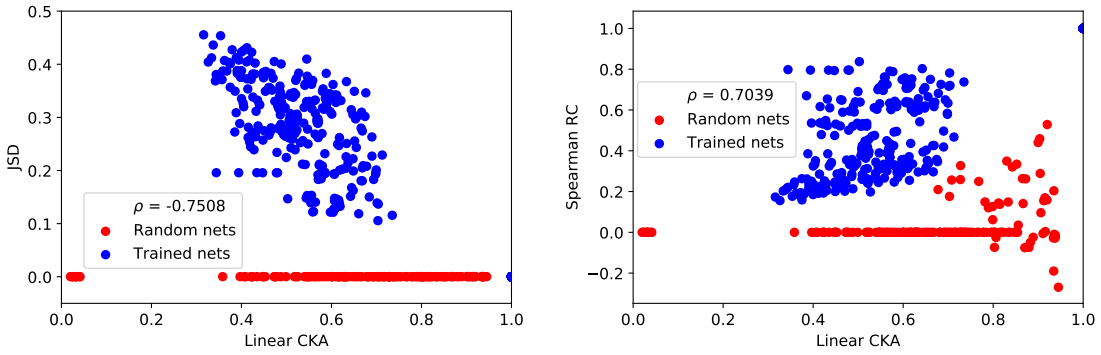
Each (i, j) entry from Figure 3.3 is the CKA between the representation of the network on row i and that of the network on column j. The subplots show the similarities of the corresponding feature vectors computed on the test data from Cifar, Imagenet32, and VOC (from left to right). The same networks are used to obtain representations on each of the three datasets. Note that models trained on the same dataset have a high CKA similarity, even when tested on different datasets. This suggests that the learned representations are biased with respect to the model used to generate them, which is not always desirable.

3.3 Feature Similarity vs. Output Similarity

One motivation for measuring feature similarity between different networks is to infer their functional similarity. A reasonable assumption is that a pair of models with 0.75 CKA has more similar outputs (and thus their performance on a task will be more similar) to each other than a pair of networks with 0.5 CKA. However, to the extent of our knowledge, this hypothesis has not been thoroughly tested.

In this section, we investigate this question by analyzing the relationship between CKA and output similarity. We use the Jensen-Shannon divergence (JSD) and Spearman rank coefficient to measure the similarity between the probability distributions outputted by two networks. We choose JSD over other similarity metrics such as Kullback-Leibler divergence (KL) because JSD is symmetric.

To compute the JSD of two probability distributions, the output distributions should have the same dimension. In particular, the models should have the same number of output classes. Hence, we train the networks only on the Cifar100 dataset. The architectures used for training were VGG16, VGG11, VGG13 [14], Resnet18, Resnet34 [8], Mobilenet, and Mobilenetv2 [15].



(a) CKA vs. JSD

(b) CKA vs. Spearman rank coefficient

FIGURE 3.4: CKA similarity vs. JSD and Spearman rank coefficient for networks trained on Cifar100 dataset. Blue points correspond to pairs of fully trained models, while red points correspond to pairs of random networks. ρ is the correlation between X and Y of the blue points.

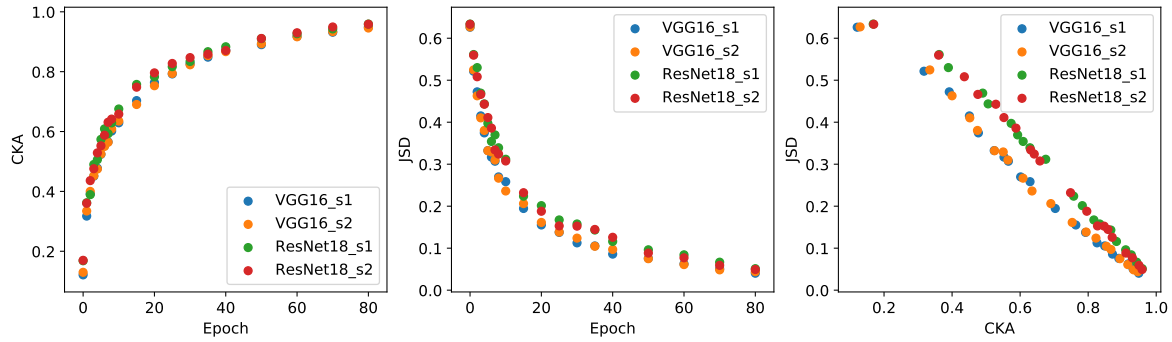
Figure 3.4 shows the relationship between CKA and JSD, and CKA and Spearman rank coefficient for pairwise model representations and outputs on the Cifar100 dataset. The blue points correspond to pairs of fully trained models, while the red points correspond to pairs of random networks. Note that there is a vaguely inverse relationship between CKA similarity and JSD. However, there are many points with the same CKA and different JSD, or with different CKA and the same JSD. The points in red also contradict the inverse relationship as all of them have a JSD close to 0. These points are the untrained networks that predict nearly uniform probability for all classes. This suggests that there is some correlation between the similarity of two networks in representation space and their functional (or output) similarity, but this relationship is not very strong and can be easily violated.

3.4 Mapping the Full Similarity Domain

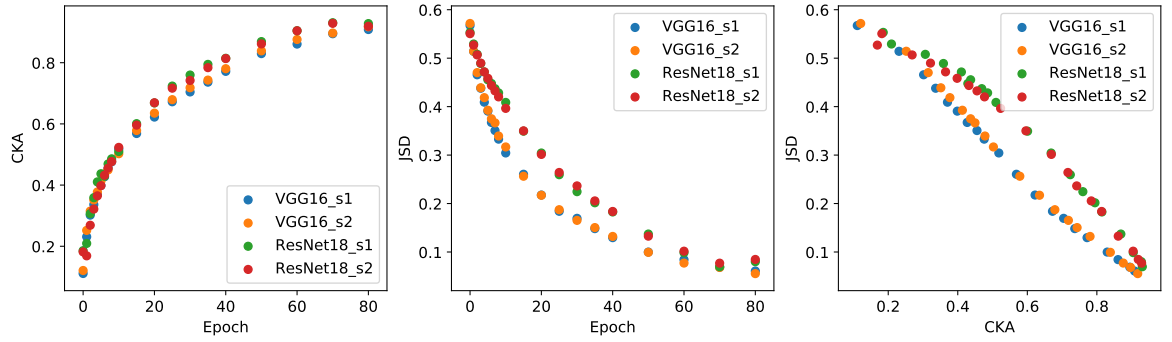
Note that Figure 3.4 does not contain any points with $CKA < 0.3$ or $CKA > 0.8$. However, it is important to understand how JSD varies with CKA for the entire range of CKA similarities (i.e., from 0 to 1). One way to generate pairs of networks that cover the full similarity domain is by comparing a fully trained network with different checkpoints of that network during training. This experiment will provide insight on whether during training, a network's representation and output become progressively more similar to its final ones.

We train Resnet18 and VGG16 on Cifar100, with two seeds for each architecture. During training, since the rate at which a network becomes similar to its final version depends on the learning rate, we used three different learning rates (0.01, 0.001, 0.0001). We generate features at epochs [0,1,2,...10, 15, 20, .. 40, 50, 60, ... 90, 100], and compute the CKA and JSD between the last checkpoint and each of the previous ones.

Figure 3.5 shows the scatter plots for variation of similarity with epoch, JSD with epoch, and the relationship between similarity and JSD of a model during training with the fully trained version of itself. The plots show that models during training converge to the fully trained version in both representation space and outputs, and there is a clear inverse relationship between the similarity of representations and output similarity. With a higher learning rate, we can see that the models converge faster to their final representation and output, which can be explained by larger gradient steps taken during training. We observe a linear relationship between CKA and output similarity, but the slope seems to be dependent on the architecture.



(a) Learning rate = 0.01



(b) Learning rate = 0.001

FIGURE 3.5: From left to right: Variation of similarity with epoch, JSD with epoch and relationship between similarity and jsd of a model during training with the fully trained version of itself. We train the networks on Cifar-100. VGG16_s1 denotes VGG_16 trained with seed=1. There is a linear relationship between similarity and JSD, and the models converge to their final form both in CKA and JSD.

3.5 Relation in Reinforcement learning

While the inverse relationship between CKA and JSD holds between network checkpoints and their final version, the relationship is not as strict across different networks (as we saw in Section 3.2). In this section, we investigate whether the relationship discovered in Section 3.2 for classification tasks also holds for reinforcement learning (RL) tasks. In RL with discrete action spaces, the output of the networks is the probability distribution over all possible actions. For

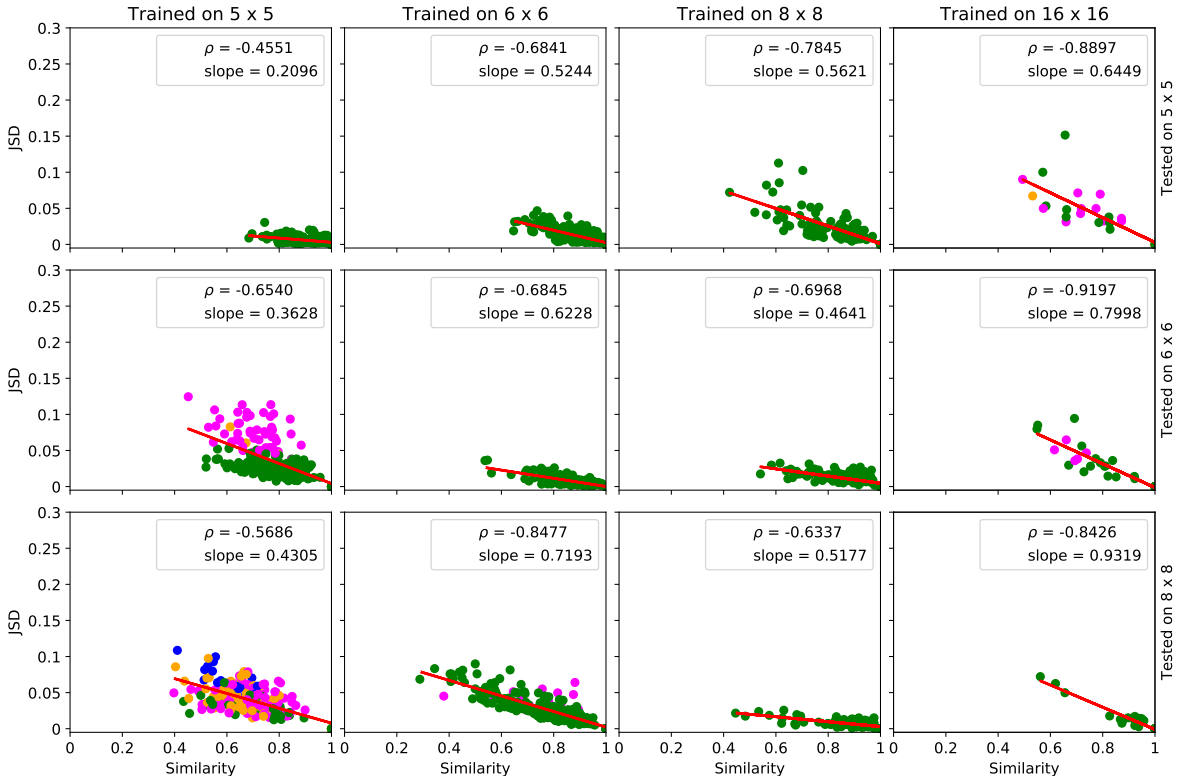
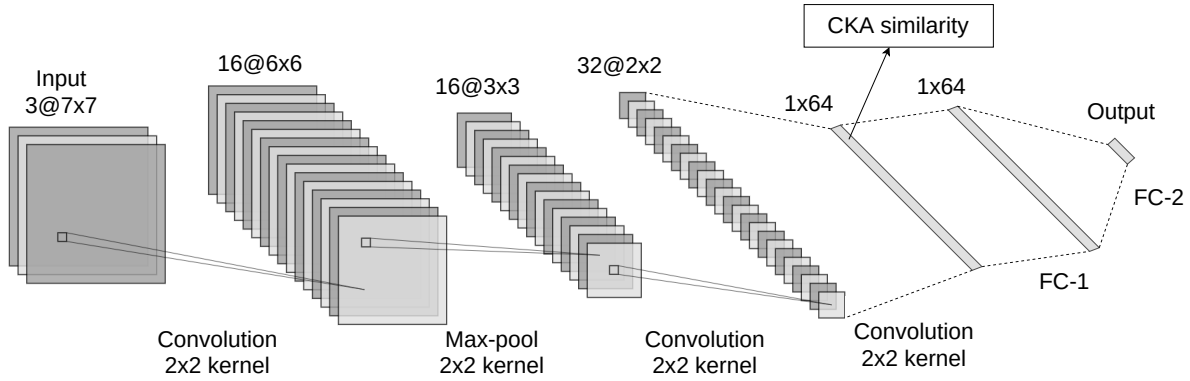


FIGURE 3.6: Relation between CKA of features and JSD of action probabilities for agents trained on MiniGrid. Slope is the slope of the best fit line (red line), and ρ is the Spearman correlation coefficient between x and y. We can see that there is a linear relationship, but the slope of the best fit line is not constant across the plots. Plots to the right have fewer points because we ignore the agents that follow sub-optimal paths to traverse the grid. Green: Both agents are optimal, Magenta: Optimal vs mediocre, Blue: Optimal vs poor, Orange: Everything else.

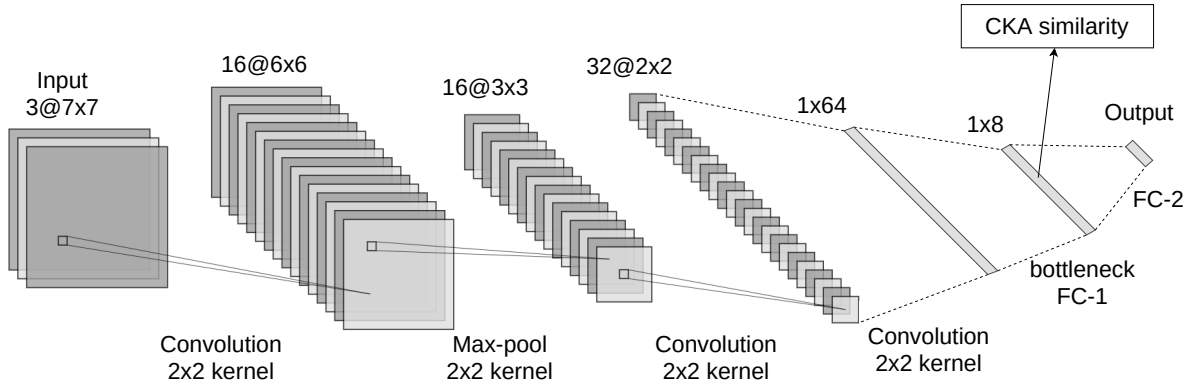
our experiments, we use the Empty environment from MiniGrid [16]. The agent starts at the top-left tile, and the goal is to reach the bottom-right tile of the grid. We train agents on four grid sizes: 5, 6, 8, and 16. To generate intermediate features and action probabilities, we use grids of size 5, 6, 8. Unlike supervised learning, RL does not have a test set on which we can compare the networks. To compute the similarity between a pair of networks (in this case agent policies), we gather rollouts using the two corresponding policies and use the rollout states as inputs. We use the proximal policy optimization (PPO) [17] algorithm to train the agents.

Figure 3.6 shows CKA-JSD scatter plots for agents trained on MiniGrid. The columns correspond to different grid sizes used during training (5x5, 6x6, 8x8, and 16x16 from left to right). The rows correspond to grid sizes used during testing (5x5, 6x6, and 8x8, from top to bottom). Each point is a CKA and JSD comparison between two agents. We use the color scheme below to differentiate between optimal ($R > 0.8$), sub-optimal ($0.8 > R > 0.2$) and poor ($R < 0.2$) agents where R is the average total reward obtained by the agent on the grid on which the agent was trained.

We also plot the line of best fit for each of the scatter plots and calculate the correlation coefficient r. Similar to Figure 3.4, one can see that many points violate the inverse linear relationship. Note that the slope of the best fit lines is not constant across the plots.



(a) Original implementation



(b) Bottleneck implementation

FIGURE 3.7: Network architecture of original implementation (a) vs bottleneck implementation (b) for agents trained on MiniGrid. Bottleneck was introduced to reduce the capacity of the network between the features used to measure CKA similarities and the output probabilities.

3.5.1 FC layer with bottleneck

One potential explanation for the difference in best-fit slopes is that the fully-connected (FC) layer (on top of the feature vectors used to measure CKA) has enough capacity to change the relationship between CKA and JSD. In other words, the features are similar up to the last convolutional layer, but the outputs (i.e., probability distributions) diverge due to the large capacity of the penultimate and final FC layers. To verify this hypothesis, we introduce a bottleneck layer that reduces the number of filters and computes the CKA similarity after the bottleneck. The bottleneck is followed by the final FC layer, which outputs the probability distribution over classes or actions.

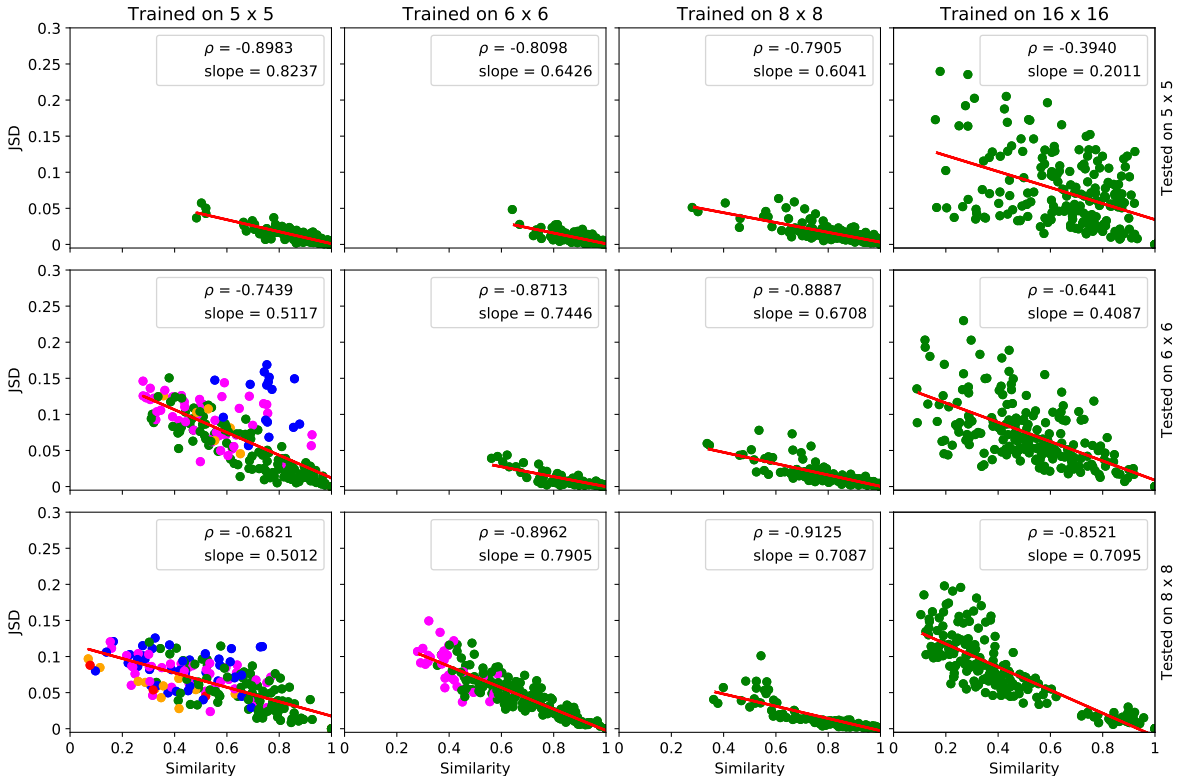


FIGURE 3.8: Relation between CKA of features and JSD of action probabilities for agents trained with the bottleneck. The dimension of the bottleneck layer is 8. Slope is the slope of the best fit line (red line), and ρ is the Spearman correlation coefficient between x and y. The best fit lines have more consistent slopes across plots, relative to earlier experiments. However, we can still observe many points with the same JSD and different CKA, and vice versa. Green: Both agents are optimal, Magenta: Optimal vs mediocre, Blue: Optimal vs poor, Orange: Everything else

Experiment	Spearman correlation	
	CKA vs. JSD	CKA vs. Spearman correlation
Cifar : trained nets	-0.75	0.703
MiniGrid (average)	-0.721	0.58
MiniGrid with bottleneck	-0.781	0.602

TABLE 3.1: Correlation coefficient of CKA vs JSD and CKA vs Spearman RC from Figures 3.4, 3.6 and 3.8.

The scatter plot of CKA similarity vs. JSD can be found in Figure 3.8. We use a bottleneck that downsamples the features from 64 to 8 dimensions. We compute CKA for the downsampled features. The best fit lines have more consistent slopes across plots in comparison to earlier experiments. However, we can still observe many points with the same JSD, different CKA, and vice versa. This result suggests that FC capacity might not be the reason for the inconsistent relationship between feature and output similarity.

We compare the scatter plots from Figures 3.4, 3.6, and 3.7 in Table 3.1. We estimate the goodness of a scatter plot by measuring the Spearman correlation between X and Y coordinates

of the points. Ideally, we expect the correlation between CKA and JSD to be 1, and between CKA and Spearman correlation to be -1. We can see that using the bottleneck improves the correlation, but the relationship from the plots is still inconsistent. For the Minigrid experiments, we report the correlation averaged across all train and test grid sizes.

Chapter 4

Existance Proof

The question we have been addressing is whether high feature similarity is correlated with high output similarity. Up to now, we have seen results that mostly suggested a positive, albeit loose, correlation. However, we show in this section that we can design a pair of networks with high feature similarity and low or variable output similarity. To achieve this, we train two networks on the Cats vs. Dogs dataset, Net_1, and Net_2. The networks use a pretrained Resnet18 (on ImageNet32) as the base architecture, which was then frozen for this task. The input to both networks is a three-channel image of size 224x224, and the output is a feature vector of size 512. The features obtained from the base are masked, setting the majority of the activations to 0.

However, the two networks are masked in different ways. We first set a ratio, d_{ratio}, which is the number of unmasked nodes over the total number of nodes. Net_1 has active nodes for the first N dimensions of the feature vector, while Net_2 has active nodes for the last $512 - N$ dimensions of the feature vector. The ratio of active nodes over the total number of nodes of the feature vector, d_{ratio}, is varied. We train Net_1 with true labels and Net_2 with flipped labels. See figure 4.1 for the network architecture.

The representational similarity is computed using CKA and CCA between the feature vectors before the masking layer. Since the two networks have the same base (which is never updated), their feature similarity is high. However, Net_2 is trained with inverted labels, so its outputs will be very different from those of Net_1. JSD and rank coefficient is used to measure the functional similarity of the two networks. The results can be seen in Table 4.1. Note that the output similarity can be controlled by adjusting d_{ratio}.

This experiment shows that we can easily construct an example in which the similarity between the representations of two networks is very high, while the similarity of their outputs is very low. This observation supports our claim that current feature similarity metrics are not always an good estimate of functional similarity.

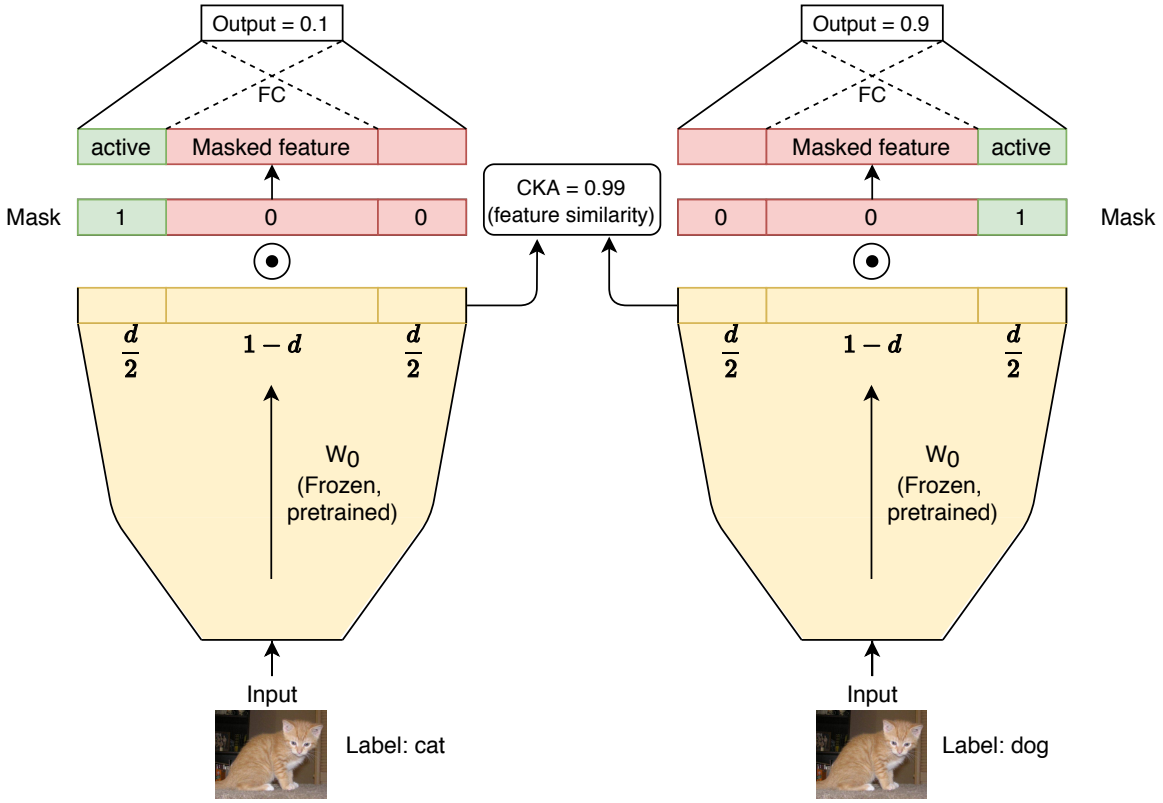


FIGURE 4.1: Architecture of the setup used to train networks with a high feature similarity and a wide range of output similarities. d is a hyperparameter that controls the number of active nodes after the masking layer. We train Net_1 with true labels and Net_2 with inverted labels. Since the networks' outputs are different, we expect them to have a low feature similarity. However, the CKA (and the CCA) similarity between the networks' representations is high since their bottom layers have the same weights.

d_{ratio}	Similarity		JSD	Rank coefficient	Validation accuracy
	CKA	CCA			
0	0.99	0.99	0.59	-0.95	97.7
0.25	0.99	0.99	0.57	-0.93	97.1
0.5	0.99	0.99	0.53	-0.91	96.4
0.8	0.99	0.99	0.41	-0.81	93.5
0.9	0.99	0.99	0.29	-0.70	89.5

TABLE 4.1: Relation between feature similarity and output similarity for our setup with masking. We can see that CKA and CCA estimate a very high similarity between the layers, whereas JSD and rank coefficient between the outputs are entirely different.

Chapter 5

Conclusion

Designing a feature similarity metric that fully captures the functional similarity of the neural networks is a challenging problem. In this paper, we show that the proposed similarity metrics lack this property. We also show how to easily design pairs of neural networks with high representational similarities and a wide range of output similarities (including very low similarities). This design can be used for evaluating similarity metrics of network representations and their correlation with output similarity. One promising direction for future work is the use of gradients for capturing both representational and functional similarity.

Appendix

We show the scatter plots from Figure 3.6 and 3.8 with the output similarity computed by Spearman rank correlation instead of JSD.

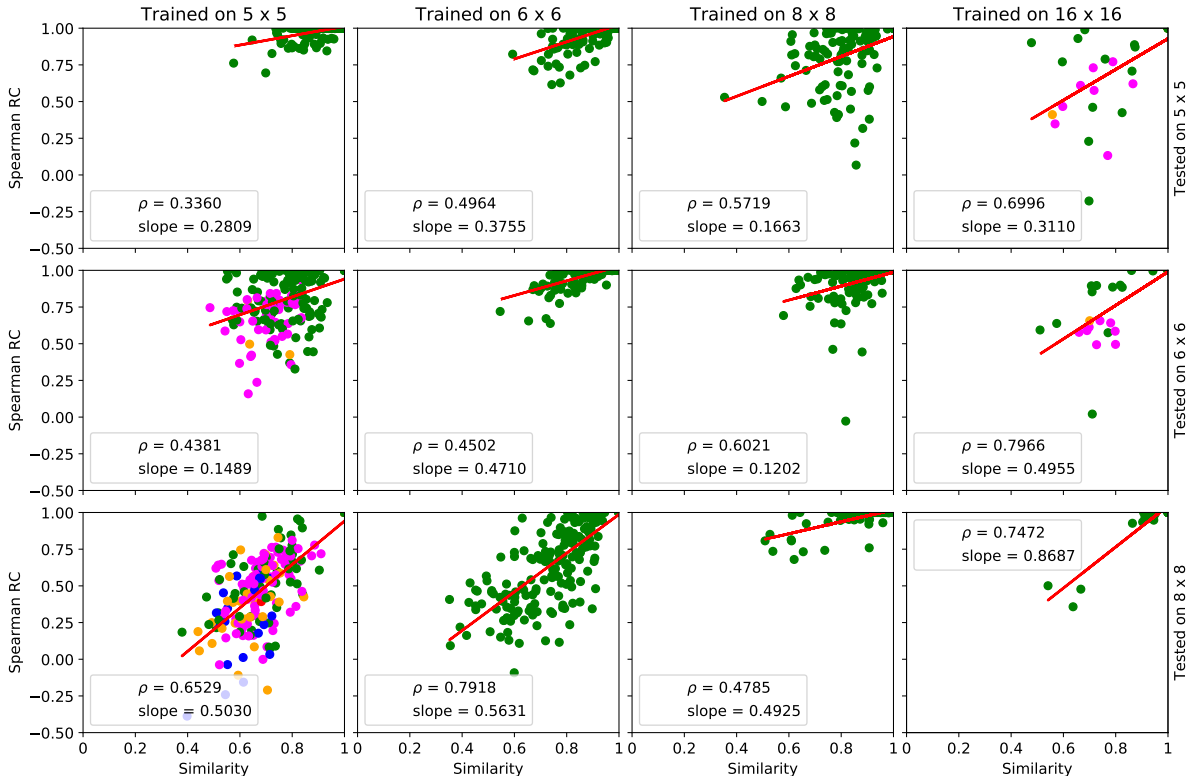


FIGURE 1: Relation between CKA and the Spearman Rank coefficient of action probability distributions for agents trained on MiniGrid. Slope is the slope of the best fit line (red line), and ρ is the Spearman correlation between X and Y. We can see that there is a linear relationship, but the slope of the best fit line varies across the plots, just like in Figure 3.6. Green: Both agents are optimal, Magenta: Optimal vs mediocre, Blue: Optimal vs poor, Orange: Everything else

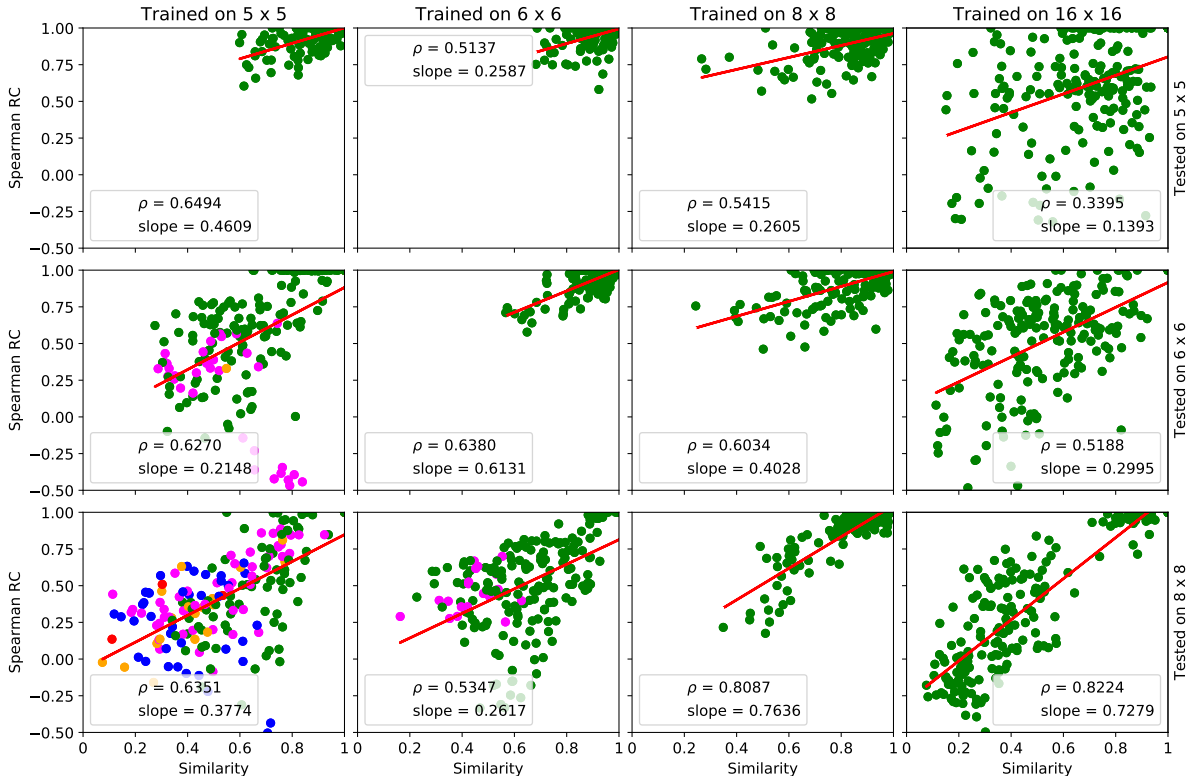


FIGURE 2: Relation between CKA of features and Spearman Rank coerrelation of action probabilities for agents trained with the bottleneck. The dimension of the bottleneck layer is 8. Slope is the slope of the best fit line (red line), and r is the Spearman correlation coefficient between x and y. Just like Figure 3.8 We can still observe many points with the same rank coefficient, different CKA and vice versa.

Bibliography

[1] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pages 6076–6085, 2017.

[2] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems*, pages 5727–5736, 2018.

[3] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

[4] Harold Hotelling. Relations between two sets of variates. In *Breakthroughs in statistics*, pages 162–190. Springer, 1992.

[5] Gene H Golub and Hongyuan Zha. The canonical correlations of matrix pairs and their numerical computation. In *Linear algebra for signal processing*, pages 27–49. Springer, 1995.

[6] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *Journal of Machine Learning Research*, 13(Mar):795–828, 2012.

[7] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE, 2011.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[11] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[12] W.A. et al. Falcon. Pytorch lightning. https://github.com/PytorchLightning/pytorch-lightning, 2019.

[13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[16] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.