Self-Supervised Representation Learning on Neural Network Weights for Model Characteristic Prediction

Konstantin Schürholt
konstantin.schuerholt@unisg.ch
AIML Lab, School of Computer Science
University of St.Gallen

Dimche Kostadinov
dimche.kostadinov@unisg.ch
AIML Lab, School of Computer Science
University of St.Gallen

Damian Borth
damian.borth@unisg.ch
AIML Lab, School of Computer Science
University of St.Gallen

Abstract

Self-Supervised Learning (SSL) has been shown to learn useful and information-preserving representations. Neural Networks (NNs) are widely applied, yet their weight space is still not fully understood. Therefore, we propose to use SSL to learn *neural representations* of the weights of populations of NNs. To that end, we introduce domain specific data augmentations and an adapted attention architecture. Our empirical evaluation demonstrates that self-supervised representation learning in this domain is able to recover diverse NN model characteristics. Further, we show that the proposed learned representations outperform prior work for predicting hyper-parameters, test accuracy, and generalization gap as well as transfer to out-of-distribution settings. Code and datasets are publicly available[1].

1 Introduction

This work investigates populations of Neural Network (NN) models and aims to learn representations of them using Self-Supervised Learning. Within NN populations, not all model training is successful, i.e., some overfit and others generalize. This may be due to the non-convexity of the loss surface during optimization (**?**), the high dimensionality of the optimization space, or the sensitivity to hyperparameters (**?**), which causes models to converge to different regions in weight space. What is still not yet fully understood, is how different regions in weight space are related to model characteristics.

Previous work has made progress investigating characteristics of NN models, e.g by visualizing learned features (**??**). Another line of work compares the activations of pairs of NN models (**???**). Both approaches rely on the expressiveness of the data, and are, in the latter case, limited to two models at a time. Other approaches predict model properties, such as accuracy, generalization gap, or hyperparameters from the margin distribution (**??**), graph topology features (**?**) or eigenvalue decomposition of the weight matrices (**?**). In a similar direction, other publications propose to investigate populations of models and to predict properties directly from their weights or weight statistics in a supervised way (**??**). However, these manually designed features may not fully capture the latent model characteristics embedded in the weight space.

Therefore, our goal is to learn task-agnostic representations from populations of NN models able to reveal such characteristics. Self-Supervised Learning (SSL) is able to reveal latent structure in

[1] https://github.com/HSG-AIML/NeurIPS_2021-Weight_Space_Learning

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

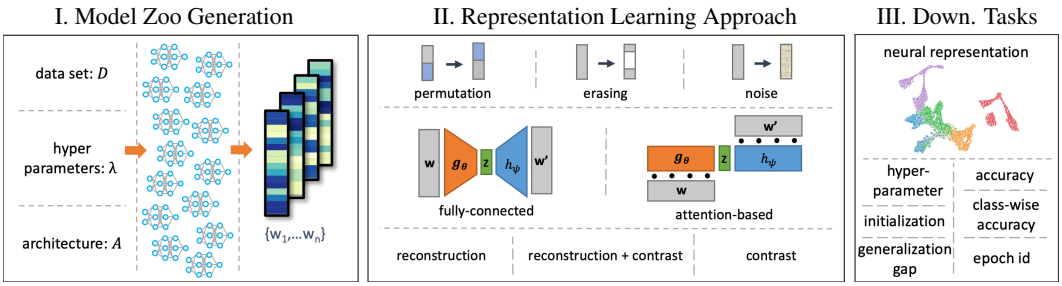


Figure 1: An overview of the proposed self-supervised representation learning approach. **I.** Populations of trained NNs form model zoos; each model is transformed in a vectorized form of its weights. **II.** Neural representations are learned from the model zoos using different augmentations, architectures, and Self-Supervised Learning tasks. **III.** Neural representations are evaluated on downstream tasks which predict model characteristics.

complex data without the need of labels, e.g., by compressing and reconstructing data (**??**). Recently, a specific approach to SSL called contrastive learning has gained popularity (**????**). Contrastive learning leverages inherent symmetries and equivariances in the data, allows to encode inductive biases and thus structure the learned representations.

In this paper, we propose a novel approach to apply SSL to learn representations of the weights of NN populations. We learn representations using reconstruction, contrast, and a combination of both. To that end, we adapt a transformer architecture to NN weights. Further, we propose three novel data augmentations for the domain of NN weights. We introduce structure preserving permutations of NN weights as augmentations, which make use of the structural symmetries within the NN weights that we find necessary for learning generalizing representations. We also adapt erasing (**?**) and noise (**?**) as augmentations for NN weights. We evaluate the learned representations by linear-probing for the generating factors and characteristics of the models. An overview for our learning approach is given in Figure **??**.

To validate our approach, we perform extensive numerical experiments over different populations of trained NN models. We find that *neural representations*[2] can be learned and reveal the characteristics of the model zoo. We show that neural representations have high utility on tasks such as the prediction of hyper-parameters, test accuracy, and generalization gap. This empirically confirms our hypothesis on meaningful structures formed by NN populations. Furthermore, we demonstrate improved performance compared to the state-of-the-art for model characteristic prediction and outlay the advantages in out-of-distribution predictions. For our experiments, we use publicly available NN model zoos and introduce new model zoos. In contrast to (**??**), our zoos contain models with different initialization points and diverse configurations, and include densely sampled model versions during training. Our ablation study confirms that the various factors for generating a population of trained NNs play a vital role in how and which properties are recoverable for trained NNs. In addition, the relation between generating factors and model zoo diversity reveals that seed variation for the trained NNs in the zoos is beneficial and adds another perspective when recovering NNs' properties.

2 Model Zoos and Augmentations

Model Zoo We denote as $\mathcal{D}$ a data set that contains data samples with their corresponding labels. We denote as λ the set of hyper-parameters used for training, (*e.g.*, loss function, optimizer, learning rate, weight initialization, batch-size, epochs). We define as A the specific NN architecture. Training under different prescribed configurations $\{\mathcal{D}, \lambda, A\}$ results in a population of NNs which we refer to as *model zoo*. We convert the weights and biases of all NNs of each model into a vectorized form. In the resulting model zoo $\mathcal{W} = \{\mathbf{w}_1, ..., \mathbf{w}_M\}$, $\mathbf{w}_i$ denotes the flattened vector of dimension N, representing the weights and biases for one trained NN model.

Augmentations. Data augmentation generally helps to learn robust models, both by increasing the number of training samples and preventing overfitting of strong features (**?**). In contrastive learning, augmentations can be used to exploit domain-specific inductive biases, e.g., know symmetries or equivariances (**?**). To the best of our knowledge, augmentations for NN weights do not yet exist.

[2]By *neural representation*, we refer to a learned representation from a population of NNs , i.e., a model zoo.

In order to enable self-supervised representation learning, we propose three methods to augment individual instances of our model zoos.

Neurons in dense layers can change position without changing the overall mapping of the network, if in-going and out-going connections are changed accordingly (**?**). The relation between equivalent versions of the same network translates to permutations of incoming weights with matrix $\mathbf{P}$ and transposed permutation $\mathbf{P}^{\mathrm{T}}$ of the outgoing weights ($\mathbf{P}^{\mathrm{T}}\mathbf{P} = \mathbf{I}$). Considering the output at layer $l+1$, with weight matrices $\mathbf{W}$, biases $\mathbf{b}$, activations $\mathbf{a}$ and activation function σ, we have

$$\mathbf{z}^{l+1} = \mathbf{W}^{l+1}\sigma(\mathbf{W}^{l}\mathbf{a}^{l-1} + \mathbf{b}^{l}) + \mathbf{b}^{l+1} = \hat{\mathbf{W}}^{l+1}\sigma(\hat{\mathbf{W}}^{l}\mathbf{a}^{l-1} + \hat{\mathbf{b}}^{l}) + \mathbf{b}^{l+1}, \tag{1}$$

where $\hat{\mathbf{W}}^{l+1} = \mathbf{W}^{l+1}(\mathbf{P}^{l})^{\mathrm{T}}$, $\hat{\mathbf{W}}^{l} = \mathbf{P}^{l}\mathbf{W}^{l}$ and $\hat{\mathbf{b}}^{l} = \mathbf{P}^{l}\mathbf{b}^{l}$ are the permuted weight matrices and bias vector, respectively. The equivalences hold not only for the forward pass, but also for the backward pass and weight update.[3] The permutation can be extended to kernels of convolution layers. The *permutation augmentation* differs significantly from existing augmentation techniques. As an analogy, flips along the axis of images are similar, but specific instances from the set of possible permutations in the image domain. Each permutable layer with dimension N_l, has $N_l!$ different permutation matrices, and in total there are $\prod_l N_l!$ distinct, but equivalent versions of the same NN. While new data can be created by training new models, the generation is computationally expensive. The permutation augmentation, however, allows to compute valid NN samples at almost no computational cost. Empirically, we found the permutation augmentation crucial for our learning approach.

In computer vision and natural language processing, masking parts of the input has proven to be helpful for generalization (**?**). We adapt the approach of *random erasing* of sections in the vectorized forms of trained NN weights. As in (**?**), we apply the erasing augmentation with a probability p to an area that is randomly chosen with a lower and upper bounds b_{low} and b_{up}. In our experiments, we set $p = 0.5$, $b_{low} = 0.03$, $b_{up} = 0.3$ and erase with zeros. Adding *noise augmentation* is another way of altering the exact values of NN weights without overly affecting their mapping, and has long been used in other domains (**?**).

3 Neural Representation Learning

With this work, we propose to learn representations of structures in weight space formed by populations of NNs. We evaluate the representations by predicting model characteristics. A supervised learning approach has been demonstrated (**??**). (**?**) find that statistics of the weights (mean, var and quintiles) are superior to the weights to predict test accuracy, which we empirically confirm in our results. However, we intend to learn representations of the weights, that contain rich information beyond statistics. 'Labels' for NN models can be obtained relatively simply, yet they can only describe predefined characteristics of a model instance (e.g., accuracy) and so supervised learning may overfit few features, as (**?**) show. Self-supervised approaches, on the other hand, are designed to learn task-agnostic representations, that contain rich and diverse information and are exploitable for multiple downstream tasks (**?**). Below, we present the used architectures and losses for the proposed self-supervised representation learning.

Architectures and Self-Supervised Losses. We apply variations of an encoder-decoder architecture. We denote the encoder as $g_\theta(\mathbf{w}_i)$, its parameters as θ, and the neural representation with dimension L as $\mathbf{z}_i = g_\theta(\mathbf{w}_i)$. We denote the decoder as $h_\psi(\mathbf{z}_i)$, its parameters as ψ, and the reconstructed NN weights as $\hat{\mathbf{w}}_i = h_\psi(\mathbf{z}_i) = h_\psi(g_\theta(\mathbf{w}_i))$. As is common in CL, we apply a projection head $p_\gamma(\mathbf{z}_i)$, with parameters γ, and denote the projected embeddings as $\bar{\mathbf{z}}_i = p_\gamma(\mathbf{z}_i) = p_\gamma(g_\theta(\mathbf{w}_i))$. In all of the architectures, we embed the neural representation $\mathbf{z}_i$ in a low dimensional space, $L < N$. We employ two common SSL strategies: reconstruction and contrastive learning. Autoencoders (AEs) with a reconstruction loss are commonly used to learn low-dimensional representations (**?**). As AEs aim to minimize the reconstruction error, the representations attempt to fully encode samples. Further, contrastive learning is an elegant method to leverage inductive biases of symmetries and inductive biases (**?**).

Reconstruction (ED): For reconstruction, we minimize the MSE $\mathcal{L}_{MSE} = \frac{1}{M}\sum_{i=1}^{M} \|\mathbf{w}_i - h_\psi(g_\theta(\mathbf{w}_i))\|_2^2$. We denote the encoder-decoder with a reconstruction loss as ED.

[3] Details, formal statements and proofs can be found in Appendix A.

Figure 2: Trained NN weights are the input sequence to a transformer. **Left**. Each element in the sequence represents the weight that connects two neurons at two different layers. **Right**. Each element in the sequence represents the set of weights related to one neuron.

Figure 3: Multi-head attention-based encoder. **Left**. Regular sequence-to-sequence translation, each element of the output sequence is used. **Right**. An additional *compression token* is added to the sequence. From the output sequence, only the compression token is taken.

Contrast (E_c): For contrastive learning, we use the common NT_Xent loss (**?**) as $\mathcal{L}_c$. For a batch of M_B model weights, each sample is randomly augmented twice to form the two *views* i and j. With the cosine similarity $\mathrm{sim}(\bar{\mathbf{z}}_i, \bar{\mathbf{z}}_j) = \bar{\mathbf{z}}_i^T \bar{\mathbf{z}}_j / ||\bar{\mathbf{z}}_i|| ||\bar{\mathbf{z}}_j||$, the loss is given as

$$\mathcal{L}_c = \sum_{(i,j)} -\log \frac{\exp(\mathrm{sim}(\bar{\mathbf{z}}_i, \bar{\mathbf{z}}_j)/T}{\sum_{k=1}^{2M_B} \mathbb{I}_{k \neq i} \exp(\mathrm{sim}(\bar{\mathbf{z}}_i, \bar{\mathbf{z}}_j)/T}, \tag{2}$$

where $\mathbb{I}_{k \neq i}$ is 1 if $k \neq i$ and 0 otherwise, and T is the temperature parameter. We denote the encoder with a contrastive loss as E_c.

Reconstruction + Contrast (E_cD): Further we combine reconstruction and contrast via $\mathcal{L} = \beta \mathcal{L}_{MSE} + (1-\beta)\mathcal{L}_c$ in order to achieve good quality compression via reconstruction and well-structured representations via the contrast. We denote this architecture with its loss as $\mathrm{E}_c\mathrm{D}$.

Reconstruction + Positive Contrast (E_{c+}D): In contrastive learning, many methods prevented mode collapse by using negative samples. The combined loss contains a reconstruction term $\mathcal{L}_{MSE}$, which can be seen as a regularizer that prevents mode collapse. Therefore, we also experiment with replacing $\mathcal{L}_c$ in our loss with a modified contrastive term without negative samples:

$$\mathcal{L}_{c+} = \sum_i -\log \left(\exp(\mathrm{sim}(\bar{\mathbf{z}}_i^j, \bar{\mathbf{z}}_i^k))/T \right) = \sum_i -\mathrm{sim}(\bar{\mathbf{z}}_i^j, \bar{\mathbf{z}}_i^k) + \log(T). \tag{3}$$

(**??**) explore similar simplifications without reconstruction. We denote the encoder-decoder with the loss $\mathcal{L} = \beta \mathcal{L}_{MSE} + (1-\beta)\mathcal{L}_{c+}$ as $\mathrm{E}_{c+}\mathrm{D}$.

Attention Module. Our encoder and decoder pairs are symmetrical and of the same type. As there is no intuition on good inductive biases in the weight space, we apply fully connected feed-forward networks (FFN) as baselines. Further, wee use multi-head self-attention modules (Att) (**?**) as an architecture with very little inductive bias. In the multi-head self-attention module, we apply learned position encodings to preserve structural information (**?**). The explicit combination of value and position makes attention modules ideal candidates to resolve the permutation symmetries of NN weight spaces. We propose two methods to encode the weights into a sequence (Figure **??**). In the first method, we encode each weight as a token in the input sequence. In the second method, we linearly transform the weights of one neuron or kernel and use it as a token. Further, we apply two variants to compress representations in the latent space (Figure **??**). In the first variant, we aggregate the output sequence of the transformer and linearly compress it to a neural representation $\mathbf{z}_i$. In the second variant, similarly to (**??**), we add a learned token to the input sequence that we dub *compression token*. After passing the input sequence trough the transformer, only the compression token from the output sequence is linearly compressed to a neural representation $\mathbf{z}_i$. Without the compression token, the information is distributed across the output sequence. In contrast, the compression token is learned as an effective query to aggregate the relevant information from the other tokens, similar to (**?**). The capacity of the compression token can be an information bottleneck. Its dimensionality is directly tied to the dimension of the value tokens and so its capacity affects the overall memory consumption.

	TETRIS-SEED						
	Rec.	Eph	Acc	$F1_{C0}$	$F1_{C1}$	$F1_{C2}$	$F1_{C3}$
E_c	-	96.7	**90.8**	67.7	72.0	**74.4**	85.8
ED	**96.1**	88.3	68.9	47.8	57.2	33.0	58.1
E_cD	84.1	**97.0**	90.2	**70.7**	**75.9**	69.4	**86.6**
$E_{c+}D$	95.9	94.0	69.9	48.9	58.1	32.5	58.8

Table 1: Ablation results over self-supervised learning losses. All models implemented with attention-based reference architecture. All values are given in %.

	TETRIS-SEED						
	Rec.	Eph	Acc	$F1_{C0}$	$F1_{C1}$	$F1_{C2}$	$F1_{C3}$
FF	0.0	80.0	85.3	57.3	53.9	64.5	80.1
Att_W	6.8	95.3	71.1	47.9	69.0	49.9	61.3
Att_{W+t}	74.1	95.4	88.6	65.6	69.8	**69.9**	**86.8**
Att_N	**89.4**	**97**.1	88.4	**71.4**	**80.8**	69.1	82.3
Att_{N+t}	84.1	97.0	**90.2**	70.7	75.9	69.4	86.6

Table 2: Ablation results in % under E_cD setup. We use feed-forward FF and attention-based variants with weight and neuron encoding Att_W and Att_N each with $+t$ and without compress. token.

Downstream Tasks. We use linear probing (**?**) as a proxy to evaluate the utility of the learned neural representations. As downstream tasks we use accuracy prediction (Acc), generalization gap (GGap), epoch prediction (Eph) as proxy to model versioning, F1-Score prediction ($F1_C$), learning rate (LR), ℓ_2-regularization (ℓ_2-reg), dropout (Drop) and training data fraction (TF). Using such targets, we solve a regression problem and measure the R^2 score (**?**). We also evaluate for hyper-parameters prediction tasks, like the activation function (Act), optimizer (Opt), initialization method (Init). Here, we train a linear perceptron by minimizing a cross entropy loss (**?**) and measure the prediction accuracy.

4 Empirical Evaluation

4.1 Model Zoos

Publicly Available Model Zoos. (**?**) introduced model zoos of CNNs with 4970 parameters trained on MNIST (**?**), Fashion-MNIST (**?**), CIFAR10 (**?**) and SVHN (**?**), and made them available under CC BY 4.0. We refer to these as `MNIST-HYP`, `FASHION-HYP`, `CIFAR10-HYP` and `SVHN-HYP`. We categorize these zoos as *large* due to their number of parameters. In their model zoo creation, the CNN architecture and seed were fixed, while the activation function, initialization method, optimizer, learning rate, ℓ_2 regularization, dropout and the train data fraction were varied between the models.

	TETRIS-SEED							
	-	P	E	N	P,E	P,N	E,N	P,E,N
ED	88.1	**95.5**	89.8	88.8	**96.1**	95.6	89.7	**96.1**
E_cD	49.2	**72.9**	67.3	59.4	**84.1**	81.9	65.4	**84.1**
$E_{c+}D$	86.3	**95.6**	88.5	87.6	**95.9**	95.8	89.0	**96.0**

Table 3: Ablation results for different augmentations and representation learning tasks. We use: permutation (P), erasing (E) and noise (N) augmentation and report test-split reconstruction R^2 scores in %.

Our Model Zoos. We hypothesize that using only one fixed seed may limit the variation in characteristics of a zoo. To address that, we train zoos where we also vary the seed and perform an ablation study below. As a toy example to test architectures, SSL tasks and augmentations, we first create a 4x4 grey-scaled image data set that we call *tetris* by using four tetris shapes.

We introduce two zoos, which we call `TETRIS-SEED` and `TETRIS-HYP`, which we group under *small*. Both zoos contain FFN with two layers and have a total number of 100 learnable parameters. In the `TETRIS-SEED` zoo, we fix all hyper-parameters and vary only the seed to cover a broad range of the weight space. The `TETRIS-SEED` zoo contains 1000 models that are trained for 75 epochs. To enrich the diversity of the models, the `TETRIS-HYP` zoo contains FFNs, which vary in activation function [`tanh`, `relu`], the initialization method [`uniform`, `normal`, `kaiming normal`, `kaiming uniform`, `xavier normal`, `xavier uniform`] and the learning rate [$1e$-3, $1e$-4, $1e$-5]. In addition, each combination is trained with seeds 1-100. Out of the 3600 models in total, we have successfully trained 2900 for 75 epochs - the remainders crashed and are disregarded. Similarly to `TETRIS-SEED`, we further create zoos of CNN models with 2464 parameters, each using the MNIST and Fashion-MNIST data sets, called `MNIST-SEED` and `FASHION-SEED` and grouped them as *medium*. To maximize the coverage of the weight space, we again initialize models with seeds 1-1000. [4]

[4]Full details on the generation of the zoos can be found in the Appendix Section C

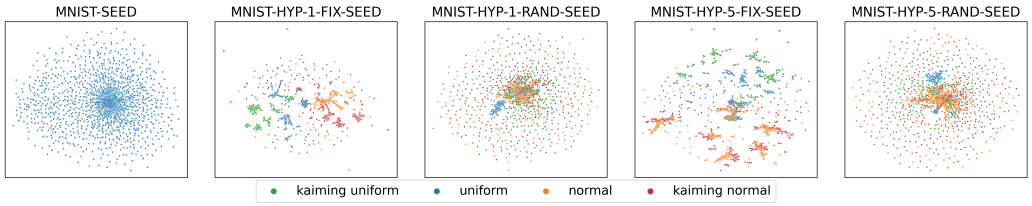


Figure 4: UMAP dimensionality reduction for NN model zoos created with different generating factors. Colors represent initialization methods. Compare numerical results in Table **??**.

	MNIST-SEED	MNIST-HYP-1-FIX-SEED	MNIST-HYP-1-RAND-SEED	MNIST-HYP-5-FIX-SEED	MNIST-HYP-5-RAND-SEED
VAR	.234	.155	.152	.091	.092
VAR_c	.234	.101	.164	.094	.100
R_{tr}	73.4	91.3	80.0	81.3	65.6
R_{tes}	59.0	72.9	37.2	70.0	38.6

	W	s(W)	E_{c+}D	W	s(W)	E_cD	W	s(W)	E_{c+}D	W	s(W)	E_cD	W	s(W)	E_cD
EPH	84.5	**97.7**	97.3	07.2	06.2	**11.6**	-34	00.5	**14.8**	-2.7	7.4	**10.2**	-14	**09.6**	07.2
ACC	91.3	98.7	**98.9**	73.6	72.9	**89.5**	-06	60.1	**85.2**	57.7	75.1	**79.4**	-13	74.5	**82.5**
GGAP	56.9	66.2	**66.7**	55.3	42.1	**61.9**	-36	31.1	**57.7**	37.8	36.7	**57.7**	-3.2	46.1	**60.5**
INIT	–	–	–	**87.7**	63.3	75.4	38.3	56.1	**86.8**	**80.6**	54.5	51.4	35.5	**47.2**	40.4

Table 4: R^2 score in %. Results on the impact of the generating factors for the model zoos. **Top:** VAR is the variance of the weights. VAR_c denotes the mean of the variances for groups of samples with shared initialization method and activation function. R_{tr} and R_{tes} are the reconstruction R^2 of train and test split on a reference E_{c+}D architecture trained for 500 epochs. **Bottom:** Downstream task performance compared to baselines predicting epochs, accuracy, generalization gap and initialization.

Model Zoo Generating Factors. Prior work discusses the impact of random seeds on properties of model zoos. While (**?**) use multiple random seeds for the same hyper-parameter configuration, (**?**) explicitly argue against that to prevent information leakage between samples. To disentangle the generating factors (seeds and hyper-parameters) and model properties, we have created five zoos with approximately the same number of CNN models trained on MNIST [**??**]. `MNIST-SEED` varies only the random seed (1-1000), `MNIST-HYP-1-FIX-SEED` varies the hyper-parameters with one fixed seed per configuration (similarly to (**?**)). To decouple the hyper-parameter configuration from one specific seed, `MNIST-HYP-1-RAND-SEED` draws 1 random seeds for each hyper-parameter configuration. To investigate the influence of repeated configurations, in `MNIST-HYP-5-FIX-SEED` and `MNIST-HYP-5-RAND-SEED` for each hyper-parameter configurations we add 5 models with different seeds, either 5 fixed seeds or randomly drawn seeds.

4.2 Training and Testing Setup

Architectures. We evaluate our approach with different types of architectures, including E_c, ED, E_cD and E_{c+}D as detailed in Section **??**. The encoder E and decoders D in the FFN baseline are symmetrical 10 [FC-ReLU]-layers each and linearly reduce dimensionality for the input to the latent space $\mathbf{z}$. Considering the attention-based encoder and decoder, on the `TETRIS-SEED` and `TETRIS-HYP` zoos, we used 2 attention blocks with 1 attention head each, token dimensions of 128 and FC layers in the attention module of dimension 512. On the larger zoos, we use up to 4 attention heads in 4 attention blocks, token dimensions of up to size 800 and FC layers in the attention module of dimension 1000. For the combined losses, we evaluate different $\beta \in [0.05, 0.85]$.

Neural Representation Learning and Downstream Tasks. We apply the proposed data augmentation methods for representation learning (see Section **??**). We run our representation learning algorithms for up to 2500 epochs, using the adam optimizer (**?**), a learning rate of 1e-4, weight decay of 1e-9, dropout of 0.1 percent and batch-sizes of 500. In all of our experiments, we use 70% of the model zoos for training, 15% for validation and 15% for testing. We use checkpoints of all epochs, but ensure that samples from the same models are either in the train, validation or the test split of the zoo. As quality metric for the neural representation learning, we track the reconstruction R^2 on the

	TETRIS-SEED								TETRIS-HYP						
	W	PCA_l	PCA_c	PCA_r	U_m	s(W)	E_cD		W	PCA_l	PCA_c	PCA_r	U_m	s(W)	E_cD
Eph	55.4	46.9	77.8	93.5	0.00	96.4	**97.0**	Eph	02.8	02.7	0.04	13.1	0.03	16.3	**16.7**
Acc	49.1	23.9	74.7	74.9	0.01	86.9	**90.2**	Acc	11.0	14.0	0.81	73.8	0.7	80.2	**83.7**
Ggap	43.9	28.3	72.9	75.7	0.01	81.0	**81.9**	GGap	12.9	12.9	0.31	76.0	0.3	79.2	**81.6**
F_{C0}	26.6	0.07	52.8	49.1	0.02	62.5	**70.7**	LR	-4.7	-3.2	-1.3	-0.4	-1.4	**53.4**	51.1
F_{C1}	41.4	30.2	48.6	58.6	0.01	63.1	**75.9**	Act	74.7	72.7	45.1	74.2	45.9	73.6	**86.9**
F_{C2}	41.5	0.06	38.3	38.5	0.01	60.9	**69.4**	Init	38.5	36.4	32.7	35.4	33.6	46.6	**47.1**
F_{C3}	53.9	44.6	73.8	72.2	0.00	75.2	**86.6**								

Table 5: R^2 given in %. **Left**. Reconstruction, epoch, accuracy, generalization gap and class-wise F-scores prediction. **Right**. Epoch, accuracy, generalization gap, learning rate, seed, activation function and initialization prediction.

validation split of the zoo and report the reconstruction R^2 on the test split. As a proxy for how much useful information is contained in the neural representation, we evaluate on downstream tasks as described in Section **??**. To ensure numerical stability of the solution to the linear probing, we apply Tikhonov regularization (**?**) with regularization parameter α in the range [$1e$-5, $1e3$] (we choose α by cross-validating over the R^2 score of the validation split of the zoo) and report the R^2 score of the test split of the zoo. To minimize the cross entropy loss for the categorical hyper-parameter prediction, we use the adam optimizer with learning rate of $1e$-4 and weight-decay of $1e$-6. The linear probing is applied to the same train-validation-test splits as it is in our representation learning setup.

Out-of-Distribution Experiments. We follow a setup for out-of-distribution experiments similar to (**?**). We investigate how well the linear probing estimator computed over neural representations generalize to yet unseen data. Therefore, we use the zoos `MNIST-HYP`, `FASHION-HYP`, `CIFAR10-HYP` and `SVHN-HYP`. On each zoo, we apply our self-supervised approach to learn their corresponding neural representations and fit a linear probing estimator to each of them (in-distribution). We then apply both the neural representation mapper and the linear probing estimator of one zoo on the weights of the other zoos (out-of-distribution). The target ranges and distributions vary between the zoos. Linear probe prediction may preserve the relation between predictions, but include a bias. Therefore, we use Kendall's τ coefficient as performance metric, which is a measure of rank correlation. It measures the ordinal association between two measured quantities (**?**).

Baselines, Computing Infrastructure and Run Time. As baseline, we use the model weights (W). In addition, we also consider PCA with linear (PCA_l), cosine (PCA_c), and radial basis kernel (PCA_r), as well as UMAP (U_m) (**?**). We further compare to layer-wise weight-statistics (mean, var, quintiles) s(W) as in (**??**). As computing hardware, we use half of the available resources from NVIDIA DGX2 station with 3.3GHz CPU and 1.5TB RAM memory, that has a total of 16 1.75GHz GPUs, each with 32GB memory. To create one small and medium zoo, it takes 1 to 2 days and 10 to 12 days, respectively. For one experiment over the small zoo it takes around 3 hours to learn the neural representation on a single GPU and evaluate on the downstream tasks. It takes approximately 1 day for the medium zoos and 2 to 3 days for the large scale zoos for the same experiment. The representation learning model size ranges from 225k on the small zoos to 65M parameters on the largest zoos. We use ray.tune (**?**) for hyperparameter optimization and track experiments with W&B (**?**).

4.3 Results

Augmentation Ablation. To evaluate the impact of the proposed augmentations (Section **??**) for our representation learning method, we present an ablation analysis on the `TETRIS-SEED` zoo, in which we measure R^2 for ED, E_cD and $E_{c+}D$[5]. We use 120 permutations, a probability of 0.5 for erasing the weights, and zero-mean noise with standard deviation 0.05 (see Table **??**). We find the permutation augmentation to be necessary for generalization - particularly under higher compression ratios. The additional samples generated with the permutation appear to effectively prevent overfitting of the training set. Without the permutation augmentation, the test performance diverges after few training epochs. Erasing further improves test performance and allows for extended training without

[5]We leave out E_c as it does not use a reconstruction loss

	MNIST-HYP			FASHION-HYP			CIFAR10-HYP			SVHN-HYP		
	W	s(W)	E_cD	W	s(W)	E_cD	W	s(W)	E_cD	W	s(W)	E_cD
EPH	25.8	33.2	**50.0**	26.6	34.6	**51.3**	25.7	30.3	**53.3**	22.8	37.8	**52.6**
ACC	74.7	81.5	**94.9**	70.9	78.5	**96.2**	76.4	82.9	**92.7**	80.5	82.1	**91.1**
GGAP	23.4	24.4	**27.4**	48.1	41.1	**49.0**	37.7	37.4	**40.4**	38.7	42.2	**44.2**
LR	29.3	34.3	**37.1**	33.5	35.6	**42.4**	27.4	32.3	**44.7**	24.5	33.4	**49.1**
ℓ_2-REG	12.5	16.5	**20.1**	11.9	16.3	**25.0**	08.7	13.8	**28.0**	09.0	13.6	**28.0**
DROP	28.5	19.2	**35.8**	26.7	21.3	**38.3**	16.7	16.5	**33.8**	09.0	14.6	**23.3**
TF	03.8	07.8	**15.9**	08.1	08.2	**22.1**	08.4	06.9	**35.4**	03.2	08.8	**21.4**
ACT	88.6	81.1	**88.7**	89.8	82.4	**90.1**	88.3	80.3	**90.0**	86.9	78.8	**87.2**
INIT	**94.6**	72.0	80.6	**95.7**	76.5	86.7	**93.5**	73.3	82.6	**91.0**	73.0	82.8
OPT	**76.7**	65.4	66.4	**79.9**	67.4	73.0	**74.0**	65.5	71.0	**72.5**	68.2	72.3

Table 6: **Top 7 Rows**. R^2 score in % for Eph, Acc, GGap, LR ℓ_2-reg, Drop and TF prediction. **Bottom 3 Rows**. Accuracy score for Act, Init and Opt prediction.

overfitting. The addition of noise yields inconsistent results and is difficult to tune, so, we omit it in our further experiments.

Architecture Ablation. The different architectures are compared in Table **??**. The results show that within the set of used NN architectures for neural representation learning, the attention-based architectures learn considerably faster, yield lower reconstruction error and have the highest performance on the downstream tasks compared to the FFN-based architectures. We attribute this to the attention modules, which are able to reliably capture long-range relations on complex data due the global field of visibility in each layer. While tokenizing each weight individually (Att_W) is able to learn, the computational load is significant, even for a small zoo, due to the large number of tokens in the sequences. The memory load prevents the application of that encoding on larger zoos. We find Att_{N+t} embedding all weights of one neuron (or convolutional kernel) to one token in combination with compression tokens shows the overall best performance and scales to larger architectures. Compression tokens achieve higher performance, which we confirm on larger and more complex datasets. The dedicated token gathers information from all other tokens of the sequence in several attention layers. This appears to enable the neural representation to grasp more relevant information than linearly compressing the entire sequence. On the other hand, compression tokens are only an advantage, if their capacity is high enough, in particular higher than the bottleneck.

Self-Supervised Learning Ablation. In Table **??**, we evaluate the usefulness of the self-supervised learning tasks (Section **??**). The application of purely contrastive loss in E_c, learns very useful representations for the downstream tasks. However, E_c heavily depends on expressive projection heads and by design cannot reconstruct samples to further investigate the representation. Pure reconstruction ED results in embeddings with a low reconstruction loss, but comparably low performance on downstream tasks. Among our losses, the combination of reconstruction with contrastive loss as in E_cD, provides neural representations $\mathbf{z}$ that have the best overall performance. The variation $E_{c+}D$, is closer to ED in general performance, but still outperforms it on the downstream tasks. The addition of a contrastive loss with projection head helps to pronounce distinctiveness, so that the neural representations are good at reconstructing the NN weights, and in revealing properties of the NNs through the encoder. Empirically, we found that on some of the larger zoos, $E_{c+}D$ performed better than E_cD. On these zoos, it appears that $E_{c+}D$ is most suitable for homogeneous zoos without distinct clusters, while E_cD is suitable for zoos with more sub-structures, compare Figure **??** and Table **??**. We therefore applied both learning strategies on all zoos and report the more performant one.

Zoo Generating Factors Ablation. Figure **??** visualizes the weights of the zoos, which contain models of all epochs[6]. In Table **??**, we report numerical properties. Only changing the seeds appears to result in homogeneous development with very high correlation between $s(W)$ and the properties of the samples in the zoo, as previous work already indicated (**?**). Varying the hyper-parameters reduces the correlation. With fixed seeds, we observe clusters of models with shared initialization method and activation function. Quantitatively we obtain lower VAR_c and high predictive value of W for the initialization method. That seems a plausible outcome, given that the architecture and activation

[6]Further visualizations can be found in the Appendix Section C

	MNIST-HYP			FASHION-HYP			SVHN-HYP			CIFAR10-HYP		
	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$
MNIST-HYP	**.36**	.29	**.36**	.21	.14	**.27**	**.26**	.12	.23	-.01	-.04	**.02**
FASHION-HYP	-.02	**.08**	.02	.54	.48	**.56**	.06	**.14**	.01	.07	.10	**.27**
SVHN-HYP	.05	**.15**	-.04	-.02	**.27**	.10	.44	.34	**.45**	-.02	.08	**.10**
CIFAR10-HYP	**.11**	.09	.06	.38	.36	**.39**	.14	.14	**.15**	**.41**	.28	.35

Table 7: Kendall's τ score for the generalization gap (GGap) prediction. We train estimators for each zoo (rows) and evaluate on all zoos (columns). The block diagonal elements contain the in-distribution prediction values. The remaining values are for out-of-distribution prediction.

function determines the shape of the loss surface, while the seed and initialization method decide the starting point. Such clustering appear to facilitate the prediction of categorical characteristics from the weights. We observe similar properties in the zoos of (**?**), see Appendix Section C. Initializing models with random seeds disperses the clusters, compare Figure **??** and Table **??**. While VAR between 1 fixed and 1 random seed is comparable, VAR_c is considerably smaller with fixed seed, the predictive value of W for the initialization methods drops significantly. Random seeds also appear to make both the reconstruction as well as NN property prediction more difficult. The repetition of configurations with five seeds hinders shortcuts in the weight space, make reconstruction and the prediction of characteristics harder. Thus, we conclude that changing only the seeds results in models with very similar evolution during learning. In these zoos, statistics of the weights perform best. In contrast, using one seed shared across models might create shortcuts in the weight space. Zoos that vary both appear to be most diverse and hardest for learning and NN property prediction. Across all zoos with hyperparameter changes, learned neural representations significantly outperform the baselines.

Downstream Tasks. We learn and evaluate our neural representations on 11 different zoos: TETRIS-SEED, TETRIS-HYP, 5 variants of MNIST, MNIST-HYP, FASHION-HYP, CIFAR10-HYP and SVHN-HYP. We compare to multiple baselines and s(W). The results are shown in Tables **??**, **??** and **??**. On all model zoos, neural representations learn useful features for the downstream tasks, which outperform the actual NN weights and biases, all of the baseline dimensionality reduction methods as well as s(W) (**?**). On the TETRIS-SEED, and MNIST-SEED model zoos, s(W) achieves high R^2 scores on all downstream tasks (see Table **??** left). As discussed above, these zoos contain a strong correlation between $s(W)$ and sample properties. Nonetheless, learned neural representations achieve higher R^2 scores on all characteristics, with the exception of MNIST-SEED Eph, where $E_{c+}D$ is competitive to s(W). On TETRIS-HYP, the overall performance of all methods is lower compared to TETRIS-SEED (see Table **??** right), making it the more challenging zoo. Here, too, neural representations have the highest R^2 score on all downstream tasks, except for the LR prediction. On MNIST-HYP, FASHION-HYP, CIFAR10-HYP and SVHN-HYP, neural representations outperform s(W) on all downstream tasks and achieve higher R^2 score compared to W on the prediction of continuous hyper-parameters and activation prediction. On the remaining categorical hyper-parameters initialization method and optimizer, the weight space achieves the highest R^2 scores (see Table **??**). We explain this with the small amount of variation in the model zoo (see Section **??**), which allows to separate these properties in weight space.

Out-of-Distribution Prediction. Table **??** shows the out-of-distribution results for generalization gap prediction[7], which is a very challenging task. The task is rendered more difficult by the fact that many of the samples which have to be ordered by their performance are very close in performance. As the results show, neural representations are able to recover the order of samples by performance to a significant degree. Further, neural representations outperform the baselines in Kendall's τ measure in the majority of the results. This verifies that our approach indeed preserves the distinctive information about trained NNs while compactly relating to their common properties, including the characteristics of the training data. Presumably, learning representations on zoos with multiple different datasets would improve the out-of-distribution capabilities even further.

[7]In Appendix Section D, we also give results for other tasks, including epoch id and test accuracy prediction

5 Related Work

There is ample research evaluating the structures of NNs by visualizing activations, *e.g.*, (**???**), which allow some insights in the patterns of, *e.g.*, the kernels of CNNs. Other research evaluated networks by computing a degree of similarity between networks. (**?**) compared the activations of NNs by a measure of "sameness". (**?**) computed correlations between the activations of different nets. (**?**) tried to match the subspaces of the activation spaces in different networks (**?**), which showed to be unreliable. (**???**) applied correlation metrics to NN activations in order to study the learning dynamics and compare NNs. (**?**) link model properties to characteristics of the loss surface around the minimum solution. In contrast comparing models with similarity metrics, other efforts map models to an absolute representation. (**?**) approximated the space of DNN activations with a convex hull. (**?**) also used activations to approximate the margin distribution and predict the generalization gap. (**?**) proposed persistent homology by using a connectivity patterns in the NN activation, and compute topological summaries.

While previously mentioned related work studied measures defined on the activations for insights about the NN characteristics, the methods applied on populations of NN weights have not received much attention for the same purpose. (**?**) relate the empirical spectral density of the weight matrices to accuracy, indicating that the weight space alone contains relevant information about the model. (**?**) evalutaed a classifier for hyper-parameter prediction directly from the weights. In contrast, we learn a general purpose neural representations in self-supervised fashion. (**?**) proposed layer-wise statistics derived from the NN models to predict the test accuracy of NN model. In (**?**), subspaces of the weights space of populations of NNs are investigated for model uniqueness and epoch order.

In the proposed work, we model associations between the different weights in NN using an attention-based module. This helps us to learn representations that compactly extract the relevant and meaningful information while considering the correlations between the weights in an NN.

6 Conclusions

In this work, we present a novel approach to learn neural representations from the weights of neural networks. To that end, we proposed novel augmentations, self-supervised learning losses and adapted multi-head attention-based architectures with suitable weight encoding for this domain. Further, we introduced several new model zoos and investigated their properties. We showed that not only learned neural networks but also their neural representations contain the latent footprint of their training data. We demonstrated high performance on downstream tasks, exceeding existing methods in hyper-parameters, test accuracy, and generalization gap prediction and showing the potential in an out-of-distribution setting.

Acknowledgements Leading up to this paper, there were many discussions with colleagues, which we would like to acknowledge. We are particularly grateful to Marco Schreyer, Xavier Giró-i-Nieto, Pol Caselles Rico and Diyar Taskiran.

Funding Disclosure This work is supported by the University of St.Gallen Basic Research Fund.

References

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv:2104.13478 [cs, stat]*, May 2021. URL `http://arxiv.org/abs/2104.13478`. arXiv: 2104.13478.

Ting Chen and Lala Li. Intriguing Properties of Contrastive Losses. *arXiv:2011.02803 [cs, stat]*, November 2020. URL `http://arxiv.org/abs/2011.02803`. arXiv: 2011.02803.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *arXiv:2002.05709 [cs, stat]*, June 2020a. URL http://arxiv.org/abs/2002.05709. arXiv: 2002.05709.

Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning. *CVPR*, page 9, 2021.

Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved Baselines with Momentum Contrastive Learning. *arXiv:2003.04297 [cs]*, March 2020b. URL http://arxiv.org/abs/2003.04297. arXiv: 2003.04297.

Ciprian A. Corneanu, Sergio Escalera, and Aleix M. Martinez. Computing the Testing Error Without a Testing Set. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2674–2682, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. doi: 10.1109/CVPR42600.2020.00275. URL https://ieeexplore.ieee.org/document/9156398/.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp Minima Can Generalize For Deep Nets. *arXiv:1703.04933 [cs]*, May 2017. URL http://arxiv.org/abs/1703.04933. arXiv: 1703.04933.

A. Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, M. Dehghani, Matthias Minderer, Georg Heigold, S. Gelly, Jakob Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.

Gabriel Eilertsen, Daniel Jönsson, Timo Ropinski, Jonas Unger, and Anders Ynnerman. Classifying the classifier: dissecting the weight space of neural networks. *arXiv:2002.05688 [cs]*, February 2020. URL http://arxiv.org/abs/2002.05688. arXiv: 2002.05688.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. Qualitatively characterizing neural network optimization problems, 2015.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised Learning. *arXiv:2006.07733 [cs, stat]*, September 2020. URL http://arxiv.org/abs/2006.07733. arXiv: 2006.07733.

Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture, 2018.

Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General Perception with Iterative Attention. *arXiv:2103.03206 [cs, eess]*, March 2021. URL http://arxiv.org/abs/2103.03206. arXiv: 2103.03206.

Yuting Jia, Haiwen Wang, Shuo Shao, Huan Long, Yunsong Zhou, and Xinbing Wang. On Geometric Structure of Activation Spaces in Neural Networks. *arXiv:1904.01399 [cs, stat]*, April 2019. URL http://arxiv.org/abs/1904.01399. arXiv: 1904.01399.

Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the Generalization Gap in Deep Networks with Margin Distributions. *arXiv:1810.00113 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1810.00113. arXiv: 1810.00113.

Jeremiah Johnson. Subspace Match Probably Does Not Accurately Assess the Similarity of Learned Representations. *arXiv:1901.00884 [cs, math, stat]*, January 2019. URL http://arxiv.org/abs/1901.00884. arXiv: 1901.00884.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and Understanding Recurrent Networks. *arXiv:1506.02078 [cs]*, June 2015. URL http://arxiv.org/abs/1506.02078. arXiv: 1506.02078.

M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014. URL http://arxiv.org/abs/1312.6114. arXiv: 1312.6114.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of Neural Network Representations Revisited. *arXiv:1905.00414 [cs, q-bio, stat]*, May 2019. URL http://arxiv.org/abs/1905.00414. arXiv: 1905.00414.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

Aarre Laakso and Garrison Cottrell. Content and cluster analysis: Assessing representational similarity in neural systems. *Philosophical Psychology*, 13(1):47–76, March 2000. ISSN 0951-5089, 1465-394X. doi: 10.1080/09515080050002726. URL http://www.tandfonline.com/doi/abs/10.1080/09515080050002726.

Yann LeCun and Corinna Cortes. The MNIST database of handwritten digits. URL http://yann.lecun.com/exdb/mnist/.

Yann LeCun and Ishan Misra. Self-supervised learning: The dark matter of intelligence, April 2021. URL https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent Learning: Do different neural networks learn the same representations? *arXiv:1511.07543 [cs]*, November 2015. URL http://arxiv.org/abs/1511.07543. arXiv: 1511.07543.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

Charles H. Martin and Michael W. Mahoney. Traditional and Heavy-Tailed Self Regularization in Neural Network Models. *arXiv:1901.08276 [cs, stat]*, January 2019. URL http://arxiv.org/abs/1901.08276. arXiv: 1901.08276.

Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv:1802.03426 [cs, stat]*, December 2018. URL http://arxiv.org/abs/1802.03426. arXiv: 1802.03426.

Ishan Misra and Laurens van der Maaten. Self-Supervised Learning of Pretext-Invariant Representations. *arXiv:1912.01991 [cs]*, December 2019. URL http://arxiv.org/abs/1912.01991. arXiv: 1912.01991.

Ari S. Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. *arXiv:1806.05759 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1806.05759. arXiv: 1806.05759.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. *arXiv:1706.05806 [cs, stat]*, June 2017. URL http://arxiv.org/abs/1706.05806. arXiv: 1706.05806.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. Publisher: Nature Publishing Group.

Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron Courville, and Philip Bachman. Data-Efficient Reinforcement Learning with Self-Predictive Representations. *ICLR*, May 2021. URL http://arxiv.org/abs/2007.05929. arXiv: 2007.05929.

Konstantin Schürholt and Damian Borth. An investigation of the weight space to monitor the training progress of neural networks, 2021. URL https://arxiv.org/abs/2006.10424.

Connor Shorten and Taghi M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60, December 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0197-0. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0.

Andrey N. Tikhonov and Vasiliy Y. Arsenin. *Solutions of ill-posed problems*. V. H. Winston & Sons, 1977. Translated from the Russian, Preface by translation editor Fritz John, Scripta Series in Mathematics.

Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting Neural Network Accuracy from Weights. *arXiv:2002.11448 [cs, stat]*, February 2020. URL http://arxiv.org/abs/2002.11448. arXiv: 2002.11448.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

Liwei Wang, Lunjia Hu, Jiayuan Gu, Yue Wu, Zhiqiang Hu, Kun He, and John Hopcroft. Towards Understanding Learning Representations: To What Extent Do Different Neural Networks Learn the Same Representation. *arXiv:1810.11750 [cs, stat]*, October 2018. URL http://arxiv.org/abs/1810.11750. arXiv: 1810.11750.

Sewall Wright. Correlation and causation. *J. agric. Res.*, 20:557–580, 1921.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

Scott Yak, Javier Gonzalvo, and Hanna Mazzawi. Towards Task and Architecture-Independent Generalization Gap Predictors. *arXiv:1906.01550 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1906.01550. arXiv: 1906.01550.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding Neural Networks Through Deep Visualization. *arXiv:1506.06579 [cs]*, June 2015. URL http://arxiv.org/abs/1506.06579. arXiv: 1506.06579.

Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, volume 8689, pages 818–833. Springer International Publishing, Cham, 2014. ISBN 978-3-319-10589-5 978-3-319-10590-1. doi: 10.1007/978-3-319-10590-1_53. URL http://link.springer.com/10.1007/978-3-319-10590-1_53.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Appendix A. Permutation Augmentation

In this appendix section, we give the full derivation about the permutation equivalence in the proposed permutation augmentation (Section 2 in the paper). In the following appendix subsections, we show the equivalence in the *forward* and *backward* pass through the neural network with original learnable parameters and the permutated neural network.

A.1 Neural Networks and Back-propagation

Consider a common, fully-connected feed-forward neural network (FFN). It maps inputs $\mathbf{x} \in \mathbb{R}^{N_0}$ to outputs $\mathbf{y} \in \mathbb{R}^{N_L}$. For a FFN with L layers, the forward pass reads as

$$\begin{aligned} \mathbf{a}^0 &= \mathbf{x}, \\ \mathbf{n}^l &= \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l, \qquad l \in \{1, \cdots, L\}, \\ \mathbf{a}^l &= \sigma(\mathbf{n}^l), \qquad l \in \{1, \cdots, L\}. \end{aligned} \tag{4}$$

Here, $\mathbf{W}^l \in \mathbb{R}^{N_l \times N_{l-1}}$ is the weight matrix of layer l, $\mathbf{b}^l$ the corresponding bias vector. Where N_l denotes the dimension of the layer l. The activation function is denoted by σ, it processes the layer's weighted sum $\mathbf{n}^l$ to the layer's output $\mathbf{a}^l$.

Training of neural networks is defined as an optimization against a objective function on a given dataset, *i.e.* their weights and biases are chosen to minimize a cost function, usually called *loss*, denoted by $\mathcal{L}$. The training is commonly done using a gradient based rule. Therefore, the update relies on the gradient of $\mathcal{L}$ with respect to weight $\mathbf{W}^l$ and the bias $\mathbf{b}^l$, that is it relies on $\nabla_{\mathbf{W}}\mathcal{L}$ and $\nabla_{\mathbf{b}}\mathcal{L}$, respectively. Back-propagation facilitates the computation of these gradients, and makes use of the chain rule to back-propagate the prediction error through the network (**?**). We express the error vector at layer l as

$$\delta^l = \nabla_{\mathbf{n}^l}\mathcal{L}, \tag{5}$$

and further use it to express the gradients as

$$\begin{aligned} \nabla_{\mathbf{W}^l}\mathcal{L} &= \delta^l (\mathbf{a}^{l-1})^{\mathrm{T}}, \\ \nabla_{\mathbf{b}^l}\mathcal{L} &= \delta^l. \end{aligned} \tag{6}$$

The output layer's error is simply given by

$$\delta^L = \nabla_{\mathbf{a}^L}\mathcal{L} \odot \sigma'(\mathbf{n}^L), \tag{7}$$

where $\odot$ denotes the Hadamard element-wise product and σ' is the activation's derivative with respect to its argument. Subsequent earlier layer's error are computed with

$$\delta^l = (\mathbf{W}^{l+1})^{\mathrm{T}} \delta^{l+1} \odot \sigma'(\mathbf{n}^l), \qquad l \in \{1, \cdots, L-1\}. \tag{8}$$

A usual parameter update takes on the form

$$(\mathbf{W}^l)_{\text{new}} = \mathbf{W}^l - \beta \nabla_{\mathbf{W}^l}\mathcal{L}, \tag{9}$$

where β is a positive learning rate.

A.2 Proof: Permutation Equivalence

In the following appendix subsection, we show the permutation equivalence for feed-forward and convolutional layers.

Permutation Equivalence for Feed-forward Layers Consider the permutation matrix $\mathbf{P}^l \in \mathbb{N}^{N_i \times N_l}$, such that $(\mathbf{P}^l)^{\mathrm{T}}\mathbf{P}^l = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. We can write the weighted sum for layer l as

$$\begin{aligned} \mathbf{n}^{l+1} &= \mathbf{W}^{l+1}\, \mathbf{a}^l + \mathbf{b}^{l+1} \\ &= \mathbf{W}^{l+1}\, \sigma(\mathbf{n}^l) + \mathbf{b}^{l+1} \\ &= \mathbf{W}^{l+1}\, \sigma(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l) + \mathbf{b}^{l+1}. \end{aligned} \tag{10}$$

As $\mathbf{P}^l$ is a permutation matrix and since we use the element-wise nonlinearity $\sigma(.)$, it holds that

$$\mathbf{P}^l\sigma(\mathbf{n}^l) = \sigma(\mathbf{P}^l\mathbf{n}^l), \tag{11}$$

which implies that we can write

$$\begin{aligned}
\mathbf{n}^{l+1} &= \mathbf{W}^{l+1}\,\mathbf{I}\,\sigma(\mathbf{W}^l\,\mathbf{a}^{l-1} + \mathbf{b}^l) + \mathbf{b}^{l+1} \\
&= \mathbf{W}^{l+1}\,(\mathbf{P}^l)^{\mathrm{T}}\,\mathbf{P}^l\,\sigma(\mathbf{W}^l\,\mathbf{a}^{l-1} + \mathbf{b}^l) + \mathbf{b}^{l+1} \\
&= \mathbf{W}^{l+1}\,(\mathbf{P}^l)^{\mathrm{T}}\,\sigma(\mathbf{P}^l\,\mathbf{W}^l\,\mathbf{a}^{l-1} + \mathbf{P}^l\,\mathbf{b}^l) + \mathbf{b}^{l+1} \\
&= \hat{\mathbf{W}}^{l+1}\,\sigma(\hat{\mathbf{W}}^l\,\mathbf{a}^{l-1} + \hat{\mathbf{b}}^l) + \mathbf{b}^{l+1},
\end{aligned} \tag{12}$$

where $\hat{\mathbf{W}}^{l+1} = \mathbf{W}^{l+1}(\mathbf{P}^l)^{\mathrm{T}}$, $\hat{\mathbf{W}}^l = \mathbf{P}^l\mathbf{W}^l$ and $\hat{\mathbf{b}}^l = \mathbf{P}^l\mathbf{b}^l$ are the permuted weight matrices and bias vector.

Note that rows of weight matrix and bias vector of layer l are exchanged together with columns of the weight matrix of layer $l+1$. In turn, $\forall l \in \{1, \cdots, L-1\}$, (**??**) holds true. At any layer l, there exist N_l different permutation matrices $\mathbf{P}^l$. Therefore, in total there are $\prod_{l=1}^{L-1} N_l!$ equivalent networks.

Additionally, we can write

$$\begin{aligned}
(\mathbf{P}^l\mathbf{W}^l)_{\text{new}} =& \mathbf{P}^l\mathbf{W}^l - \alpha\mathbf{P}^l\nabla_{\mathbf{W}^l}\mathcal{L} \\
=& \mathbf{P}^l\mathbf{W}^l - \alpha\mathbf{P}^l\delta^l(\mathbf{a}^{l-1})^{\mathrm{T}} \\
=& \mathbf{P}^l\mathbf{W}^l - \alpha\mathbf{P}^l\left[(\mathbf{W}^{l+1})^{\mathrm{T}}\delta^{l+1} \odot \sigma'(\mathbf{n}^l)\right]\,(\mathbf{a}^{l-1})^{\mathrm{T}} \\
=& \mathbf{P}^l\mathbf{W}^l - \alpha\left[(\mathbf{W}^{l+1}\mathbf{P}^{\mathrm{T}})^{\mathrm{T}}\delta^{l+1} \odot \sigma'(\mathbf{P}^l\mathbf{n}^l)\right]\,(\mathbf{a}^{l-1})^{\mathrm{T}} \\
=& \mathbf{P}^l\mathbf{W}^l - \alpha[(\mathbf{W}^{l+1}(\mathbf{P}^l)^{\mathrm{T}})^{\mathrm{T}}\delta^{l+1} \odot \sigma'(\mathbf{P}^l\mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{P}^l\mathbf{b}^l)](\mathbf{a}^{l-1})^{\mathrm{T}}.
\end{aligned} \tag{13}$$

If we apply a permutation $\mathbf{P}^l$ to our update rule (equation **??**) at any layer except the last, then using the above, we can express the gradient based update as

$$(\hat{\mathbf{W}}^l)_{\text{new}} = \hat{\mathbf{W}}^l - \alpha\left[(\hat{\mathbf{W}}^{l+1})^{\mathrm{T}}\delta^{l+1} \odot \sigma'(\hat{\mathbf{W}}^l\mathbf{a}^{l-1} + \hat{\mathbf{b}}^l)\right]\,(\mathbf{a}^{l-1})^{\mathrm{T}}\square \tag{14}$$

The above implies that the permutations not only preserve the structural flow of information in the forward pass, but also preserve the structural flow of information during the update with the backward pass. That is we preserve the structural flow of information about the gradients with respect to the parameters during the backward pass trough the feed-forward layers.

Permutation Equivalence for Convolutional Layers We can easily extend the permutation equivalence in the feed-forward layers to convolution layers. Consider the 2D-convolution with input channel $\mathbf{x}$ and O output channels $\mathbf{a}_{s_1} = [\begin{smallmatrix}\mathbf{a}_1\\ \vdots \\ \mathbf{a}_O\end{smallmatrix}]$. We express a single convolution operation for the input channel $\mathbf{x}$ with a convolutional kernel $\mathbf{K}_o, o \in \{1, ..., O\}$ as

$$\mathbf{a}_o = \mathbf{b}_o + \mathbf{K}_o \star \mathbf{x}, o \in \{1, \cdots, O\}, \tag{15}$$

where $\star$ denotes the discrete convolution operation.

Note that in contrast to the hole set of permutation matrices $\mathbf{P}^l$ (which where introduced earlier) now we consider only a subset that affects the order of the input channels (if we have multiple) and the order of the concatenation of the output channels.

We now show that changing the order of input channels does not affect the output if the order of kernels is changed accordingly.

The proof is similar with the permutation equivalence for feed-forward layer. The difference here is that we take into account only the change in the order of channels and kernels. In order to prove permutation equivalence here it suffices to show that we can represent the convolution of multiple input channels by multiple convolution kernels in an alternative form, that is as matrix vector operation.

To do so we fist show that we can express the convolution of one input channel with one kernel to its equivalent matrix vector product form. Formally, we have that

$$\mathbf{a}_o = \mathbf{b}_o + \mathbf{K}_o \star \mathbf{x} = \quad \mathbf{b}_o + \mathbf{R}_o\mathbf{x}, \tag{16}$$

where $\mathbf{R}_o$ is the convolution matrix. We build the matrix $\mathbf{R}_o$ in this alternative form (**??**) for the convolution operation from the convolutional kernel $\mathbf{K}_o$. $\mathbf{R}_o$ has a special structure (if we have 1D convolution then it is known as a circulant convolution matrix), while the input channel $\mathbf{x}$ remains the same. The number of columns in $\mathbf{R}_o$ equals the dimension of the input channel, while the number of rows in $\mathbf{R}_o$ equals the dimension of the output channel. In each row of $\mathbf{R}_o$, we store the elements of the convolution kernel. That is we sparsely distribute the kernel elements such that the multiplication of one row $\mathbf{R}_{o,j}$ of $\mathbf{R}_o$ with the input channel $\mathbf{x}$ results in the convolution output $a_{o,j}$ for the corresponding position j at the output channel $\mathbf{a}_o$.

The convolution of one input channels by multiple kernels can be expressed as a matrix vector operation. In that case, the matrix in the equivalent form for the convolution with multiple kernels over one input represents a block concatenated matrix, where each of the block matrices has the previously described special structure, *i.e.*,

$$\mathbf{a}_{s_1} = \begin{bmatrix} \mathbf{a}_1 \\ . \\ \mathbf{a}_O \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ . \\ \mathbf{b}_O \end{bmatrix} + \begin{bmatrix} \mathbf{R}_1 \\ . \\ \mathbf{R}_O \end{bmatrix} \mathbf{x} = \mathbf{b}_f + \mathbf{R}_f \mathbf{x}, \tag{17}$$

where $\mathbf{b}_f = \begin{bmatrix} \mathbf{b}_1 \\ . \\ \mathbf{b}_O \end{bmatrix}$ and $\mathbf{R}_f = \begin{bmatrix} \mathbf{R}_1 \\ . \\ \mathbf{R}_O \end{bmatrix}$.

In the same way the convolution of multiple input channels $\mathbf{x}_1, ..., \mathbf{x}_S$ by multiple kernels $\mathbf{K}_1, ..., \mathbf{K}_O$ can be expressed as a matrix vector operation. In that case, the matrix in the equivalent form for the convolution with multiple kernels over multiple inputs represents a block diagonal matrix, where each of the blocks in the block diagonal matrix has the previously described special structure, *i.e.*,

$$\begin{bmatrix} \mathbf{a}_{s_1} \\ . \\ \mathbf{a}_{s_s} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_f \\ . \\ \mathbf{b}_f \end{bmatrix} + \begin{bmatrix} \mathbf{R}_f & \mathbf{0} & ... & \mathbf{0} \\ . & . & . & . \\ \mathbf{0} & ... & \mathbf{0} & \mathbf{R}_f \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ . \\ \mathbf{x}_S \end{bmatrix} = \begin{bmatrix} \mathbf{b}_f \\ . \\ \mathbf{b}_f \end{bmatrix} + \mathbf{R} \begin{bmatrix} \mathbf{x}_1 \\ . \\ \mathbf{x}_S \end{bmatrix}, \tag{18}$$

where $\mathbf{R} = \begin{bmatrix} \mathbf{R}_f & \mathbf{0} & ... & \mathbf{0} \\ . & . & . & . \\ \mathbf{0} & ... & \mathbf{0} & \mathbf{R}_f \end{bmatrix}$, which we can also express as

$$\mathbf{a} = \mathbf{b} + \mathbf{R} \begin{bmatrix} \mathbf{x}_1 \\ . \\ \mathbf{x}_S \end{bmatrix}, \tag{19}$$

where $\mathbf{a} = \begin{bmatrix} \mathbf{a}_{s_1} \\ . \\ \mathbf{a}_{s_s} \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} \mathbf{b}_f \\ . \\ \mathbf{b}_f \end{bmatrix}$.

Note that the above equation has equivalent form with equation (**??**), therefore, the previous proof is valid for the update with respect to hole matrix $\mathbf{R}$. However, $\mathbf{R}$ has a special structure, therefore for the update of of each element in $\mathbf{R}$, we can use the chain rule, which results in

$$\frac{\partial f(\mathbf{R})}{\partial R_{ij}} = \sum_k \sum_l \frac{\partial f(\mathbf{R})}{\partial R_{kl}} \frac{\partial R_{kl}}{\partial R_{ij}} = Tr \left[\left[\frac{\partial f(\mathbf{R})}{\partial \mathbf{R}} \right]^T \frac{\partial \mathbf{R}}{\partial R_{ij}} \right]. \tag{20}$$

Replacing $f()$ by $\mathcal{L}$ in the above and using the update rule equation (**??**) gives us the update equation for R_{ij}. Using similar argumentation and derivation that leads to equation (**??**) concludes the proof for permutation equivalence for a convolutional layer □

Empirical Evaluation We empirically confirm the permutation equivalence (see Figure **??**). We begin by comparing accuracies of permuted models (Figure **??** left). To that end, we randomly initialize model A and train it for 10 epochs. We pick one random permutation, and permute all epochs of model A. For the permuted version Ap we compute the test accuracy for all epochs. The test accuracy of model A and Ap lie on top of each other, so the permutation equivalence holds for the forward pass. To test the backwards pass, we create model B as a copy of Ap at initialization, and train for 10 epochs. Again, train and test epochs of A and B lie on top of each other, which indicates that the equivalence empirically holds for the backwards pass, too.

To track how models develop in weight space, we compute the mutual ℓ_2-distances between the vectorized weights (Figure **??** right). The distance between A and Ap, as well as between A and B is high and identical. Therefore, model A is far away from models Ap and B. Further, the distance

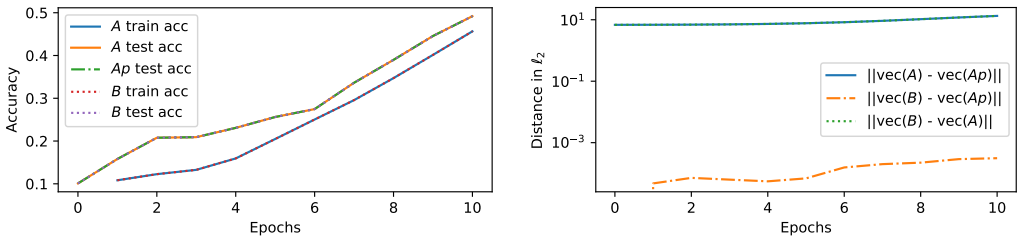


Figure 5: Empirical Evaluation of Permutation Equivalence. **Left:** Accuracies of orignal model A, permuted model Ap and model B trained from Ap's initialization. All models are indistinguishable in their accuracies. **Right:** pairwise distances of vectorized weights over epochs of models A, Ap and B. The distance between A and Ap as well as A and B is equally large and does not change much over the epochs. The distance between Ap and B, which start from the same point in weight space, is small and remains small over the epochs. **both figures:** permuted versions of models are indistinguishable in their mapping, but far apart in weight space.

between Ap and B is small, confirming the backwards pass equivalence. We attribute the small difference to numerical errors.

Further Weight Space Symmetries It is important to note that besides the symmetry used above, other symmetries exist in the model weight space, which change the representation of a NN model, but not it's mapping, *e.g.*, scaling of subsequent layers with piece-wise linear activation functions (**?**). While some of these symmetries may be used as augmentation, these particular mappings only create equivalent networks in the forward pass, but different gradients and updates in the backward pass when back propagating. Therefore, we did not consider them in our work.

Appendix B. Downstream Tasks Additional Details

In this appendix section, we provide additional details about the downstream tasks which we use to evaluate the utility of the neural representations obtained by our self-supervised learning approach.

B.1 Downstream Tasks Problem Formulation

We use linear probing as a proxy to evaluate the utility of the learned neural representations, similar to **?**.

We denote the training and testing neural representations as $\mathbf{Z}_{train}$ and $\mathbf{Z}_{test}$. We assume that training $\mathbf{t}_{train}$ and testing $\mathbf{t}_{test}$ target vectors are given. We compute the closed form solution $\hat{\mathbf{r}}$ to the regression problem

$$(Q1) : \hat{\mathbf{r}} = \arg\min_{\mathbf{r}} \|\mathbf{Z}_{train}\mathbf{r} - \mathbf{t}_{train}\|_2^2,$$

and evaluate the utility of $\mathbf{Z}_{test}$ by measuring the R^2 score **?** as discrepancy between the predicted $\mathbf{Z}_{test}\hat{\mathbf{r}}$ and the true test targets $\mathbf{t}_{test}$.

Note that by using different targets $\mathbf{t}_{train}$ in $(Q1)$, we can estimate different $\mathbf{r}$ coefficients. This enables us to evaluate on different downstream tasks, including, accuracy prediction (Acc), epoch prediction (Eph) as proxy to model versioning, F-Score **?** prediction (F_c), learning rate (LR), ℓ_2-regularization (ℓ_2-reg), dropout (Drop) and training data fraction (TF). For these target values, we solve $(Q1)$, but for categorical hyper-parameters prediction, like the activation function (Act), optimizer (Opt), initialization method (Init), we train a linear perceptron by minimizing a cross entropy loss **?**. Here, instead of R^2 score, we measure the prediction accuracy.

B.2 Downstream Tasks Targets

In this appendix subsection, we give the details about how we build the target vectors in the respective problem formulations for all of the downstream tasks.

Accuracy Prediction (Acc). In the accuracy prediction problem, we assume that for each trained NN model on a particular data set, we have its accuracy. Regarding the task of accuracy prediction, the value $a_{train,i}$ for the training NN models represents the training target value $t_{train,i} = a_{train,i}$, while the value $a_{test,j}$ for the testing NN models represents the true testing target value $t_{test,j} = a_{test,j}$.

Generalization Gap Prediction (GGap). In the generalization gap prediction problem, we assume that for each trained NN model on a particular data set, we have its train and test accuracy. The generalization gap represents the target value $g_i = a_{train,i} - a_{test,i}$ for the training NN models, while $g_j = a_{train,j} - a_{test,j}$ is the true target $t_{test,j} = g_j$ for the testing NN models.

Epoch Prediction (Eph). We consider the simplest setup as a proxy to model versioning, where we try to distinguish between NN weights and biases recorded at different epoch numbers during the training of the NNs in the model zoo. To that end, we assume that we construct the model zoo such that during the training of a NN model, we record its different evolving versions, *i.e.*, the zoo includes versions of one NN model at different epoch numbers e_i. Similarly to the previous task, our targets for the task of epoch prediction are the actual epoch numbers, $t_{train,i} = e_{train,i}$ and $t_{test,j} = e_{test,j}$, respectively.

F Score Prediction (F_c). To identify more fine-grained model properties, we therefore consider the class-wise F score. We define the F score prediction task similarly as in the previous downstream task. We assume that for each NN model in the training and testing subset of the model zoo, we have computed F score **?** for the corresponding class with label c that we denote as $\mathrm{F}_{train,c,i}$ and $\mathrm{F}_{test,c,j}$, respectively. Then we use $\mathrm{F}_{train,c,i}$ and $\mathrm{F}_{test,c,i}$ as a target value $t_{train,c,i} = \mathrm{F}_{train,c,i}$ in the regression problem and set $t_{test,c,j} = \mathrm{F}_{test,c,j}$ during the test evaluation.

Hyper-parameters Prediction. We define the hyper-parameter prediction task identically as the previous downstream tasks. Where for continuous hyper-parameters, like learning rate (LR), ℓ_2-regularization (ℓ_2-reg), dropout (Drop), nonlinear thresholding function (TF), we solve the linear regression problem $(Q1)$. Similarly to the previous task, our targets for the task of hyper-parameters prediction are the actual hyper-parameters values.

In particular, for learning rate $t_{train,i} = learning\,rate$, for ℓ_2-regularization $t_{train,i} = \ell_2\text{-}regularization\,type$, for dropout (Drop) $t_{train,i} = dropout\,value$ and for nonlinear thresholding function (TF) $t_{train,i} = nonlinear\,thresholding\,function$. In a similar fashion, we also define the test targets $\mathbf{t}_{test}$.

For categorical hyper-parameters, like activation function (Act), optimizer (Opt), initialization method (Init), instead of regression loss $(Q1)$, we train a linear perception by minimizing a cross entropy loss **?**. Here, also we also define the targets as detailed above. The only difference here is that the targets here have discrete categorical values.

OUR ZOOS	DATA	NN TYPE	NO. PARAM.	VARYING PROP.	NO. EPH	NO. NNS
TETRIS-SEED	TETRIS	MLP	100	SEED (1-1000)	75	1000*75
TETRIS-HYP	TETRIS	MLP	100	SEED (1-100), ACT, INIT, LR	75	2900*75
MNIST-SEED	MNIST	CNN	2464	SEED (1-1000)	25	1000*25
FASHION-SEED	F-MNIST	CNN	2464	SEED (1-1000)	25	1000*25
MNIST-HYP-1-FIX-SEED	MNIST	CNN	2464	FIXED SEED, ACT, INT, LR	25	$\sim$ 1152*25
MNIST-HYP-1-RAND-SEED	MNIST	CNN	2464	RANDOM SEED, ACT, INT, LR	25	$\sim$ 1152*25
MNIST-HYP-5-FIX-SEED	MNIST	CNN	2464	5 FIXED SEEDS, ACT, INT, LR	25	$\sim$ 1280*25
MNIST-HYP-5-RAND-SEED	MNIST	CNN	2464	5 RANDOM SEEDS, ACT, INT, LR	25	$\sim$ 1280*25

EXISTING ZOOS	DATA	NN TYPE	NO. PARAM.	VARYING PROP.	NO. EPH	NO. NNS
MNIST-HYP	MNIST	CNN	4970	ACT, INIT, OPT, LR, ℓ_2-REG, DROP, TF	9	$\sim$ 30000*9
FASHION-HYP	F-MNIST	CNN	4970	ACT, INIT, OPT, LR, ℓ_2-REG, DROP, TF	9	$\sim$ 30000*9
CIFAR10-HYP	CIFAR10	CNN	4970	ACT, INIT, OPT, LR, ℓ_2-REG, DROP, TF	9	$\sim$ 30000*9
SVHN-HYP	SVHN	CNN	4970	ACT, INIT, OPT, LR, ℓ_2-REG, DROP, TF	9	$\sim$ 30000*9

	1 INIT	M INIT	NO DATA LEAKAGE	DENSE CHECK POINTS
?	$\surd$	$\surd$	$\times$	$\times$
?	$\surd$	$\times$	$\surd$	$\times$
PROPOSED ZOOS	$\surd$	$\surd$	$\surd$	$\surd$

Table 8: Overview of the characteristics for the model zoos proposed and used (existing) in this work.

Appendix Appendix C. Model Zoos Details

In Table **??**, we give an overview of the characteristics for the used model zoos in this paper. This includes

- The data sets used for zoo creation.
- The type of the NN models in the zoo.
- Number of learnable parameters for each of the NNs.
- Used number of model versions that are taken at the corresponding epochs during training.
- Total number of NN models contained in the zoo.

In Table **??** we also compare the existing and the introduced model zoos in prior and this work in terms of properties like initialization, data leakage and presence of dense model versions obtained by recording the NN model during training evolution.

In Table **??** we provide the architecture configurations and exact modes of variation of our model zoos.

OUR ZOOS	INIT	SEED	OPT	ACT	LR	DROP	ℓ_2-REG
TETRIS-SEED	UNIFORM	1-1000	ADAM	TANH	3E-5	0.0	0.0
TETRIS-HYP	UNIFORM, NORMAL, KAIMING-NO, KAIMING-UN, XAVIER-NO, XAVIER-UN,	1-100	ADAM	TANH, RELU	1E-3, 1E-4, 1E-5	0.0	0.0
MNIST-SEED	UNIFORM	1-1000	ADAM	TANH	3E-4	0.0	0.0
MNIST-HYP-1-FIX-SEED	UNIFORM, NORMAL, KAIMING-UN, KAIMING-NO	42	ADAM, SGD	TANH, RELU, SIGMOID, GELU	3E-3, 1E-3, 3E-4, 1E-4	0.0, 0.3, 0.5	0, 1E-3, 1E-1
MNIST-HYP-1-RAND-SEED	UNIFORM, NORMAL, KAIMING-UN, KAIMING-NO	$1 \in [1e0, 1e6]$	ADAM, SGD	TANH, RELU, SIGMOID, GELU	3E-3, 1E-3, 3E-4, 1E-4	0.0, 0.3, 0.5	0, 1E-3, 1E-1
MNIST-HYP-5-FIX-SEED	UNIFORM, NORMAL, KAIMING-UN, KAIMING-NO	1,2,3,4,5	ADAM, SGD	TANH, RELU, SIGMOID, GELU	1E-3, 1E-4	0.0, 0.5	1E-3, 1E-1
MNIST-HYP-5-RAND-SEED	UNIFORM, NORMAL, KAIMING-UN, KAIMING-NO	$5 \in [1e0, 1e6]$	ADAM, SGD	TANH, RELU, SIGMOID, GELU	1E-3, 1E-4	0.0, 0.5	1E-3, 1E-1
FASHION-SEED	UNIFORM	1-1000	ADAM	TANH	3E-4	0.0	0.0

Table 9: Architecture configurations and modes of variation of our model zoos.

Figure 6: Visualization of samples representing the four basic shapes in our Tetris data set.

C.1 Zoos Generation Using Tetris Data

As a toy example, we first create a 4x4 grey-scaled image data set that we call *Tetris* by using four tetris shapes. In Figure **??** we illustrate the basic shapes of the tetris data set. We introduce two zoos, which we call `TETRIS-SEED` and `TETRIS-HYP`, which we group under *small*. Both zoos contain FFN with two layers. In particular, the FFN has input dimension of 16 a latent dimension of 5 and output dimension of 4. In total the FFN has $16 \times 5 + 5 \times 4 = 100$ learnable parameters (see Table **??**). We give an illustration of the used FFN architecture in Figure **??**.

In the `TETRIS-SEED` zoo, we fix all hyper-parameters and vary only the seed to cover a broad range of the weight space. The `TETRIS-SEED` zoo contains 1000 models that are trained for 75 epochs. In total, this zoo contains $1000 \times 75 = 75000$ trained NN weights and biases.

To enrich the diversity of the models, the `TETRIS-HYP` zoo contains FFNs, which vary in activation function [`tanh`, `relu`], the initialization method [`uniform`, `normal`, `kaiming normal`, `kaiming uniform`, `xavier normal`, `xavier uniform`] and the learning rate [$1e$-3, $1e$-4, $1e$-5]. In addition,

each combination is trained with 100 different seeds. Out of the 3600 models in total, we have successfully trained 2900 for 75 epochs - the remainders crashed and are disregarded. So in total, this zoo contains $2900 \times 75 = 217500$ trained NN weights and biases.

C.2 Zoos Generation Using MNIST Data

Similarly to `TETRIS-SEED`, we further create medium sized zoos of CNN models. In total the CNN has 2464 learnable parameters, distributed over 3 convolutional and 2 fully connected layers. The full architecture is detailed in Table **??**. We give an illustration of the used CNN architecture in Figure **??**. Using the MNIST data set, we created five zoos with approximately the same number of CNN models.

In the `MNIST-SEED` zoo we vary only the random seed (1-1000), while using only one fixed hyper-parameter configuration. In particular,

In `MNIST-HYP-1-FIX-SEED` we vary the hyper-parameters. We use only one fixed seed for all the hyper-parameter configurations (similarly to (**?**)). The `MNIST-HYP-1-RAND-SEED` model zoo contains CNN models, where per each model we draw and use 1 random seeds and different hyper-parameter configuration.

We generate `MNIST-HYP-5-FIX-SEED` insuring that for each hyper-parameter configurations we add 5 models that share 5 fixed seed. We build `MNIST-HYP-5-RAND-SEED` such that for each hyper-parameter configurations we add 5 models that have different random seeds.

We grouped these model zoos as *medium*. In total, each of these zoos approximately $1000 \times 25 = 25000$ trained NN weights and biases.

In Figure **??** we provide a visualization for different properties of the `MNIST-SEED`, `MNIST-HYP-1-FIX-SEED`, `MNIST-HYP-1-RANDOM-SEED`, `MNIST-HYP-5-FIX-SEED` and `MNIST-HYP-5-RANDOM-SEED` zoos. The visualization supports the empirical findings from the paper, that zoos which vary in seed only appear to contain a strong correlation between the mean of the weights and the accuracy. In contrast, the same correlation is considerably lower if the hyper-parameters are varied. Further, we also observe clusters of models with shared initialization method and activation function for zoos with fixed seeds. Random seeds seem to disperse these clusters to some degree. This additionally confirms our hypothesis about the importance of the generating factors for the zoos. We find that zoos containing hyper-parameters variation and multiple (random) seeds, have a rich set of properties, avoid 'shortcuts' between the weights (or their statistics) and properties, and therefore benefits neural representation learning.

In Figure **??**, we show additional UMAP reductions of `MNIST-HYP`, which confirm our previous findings. Similarly to the UMAP for the `MNIST-HYP-1-FIX-SEED` zoo, the UMAP for the `MNIST-HYP` has distinctive and recognizable initialization points. The categorical hyper-parameters are visually separable in weight space. As we can see in the same figure, it seems that the UMAP for the `MNIST-HYP` zoo contains very few paths along which the evolution during learning of all the models can be tracked in weight space, facilitating both epoch and accuracy prediction.

C.3 Zoo Generation Using F-MNIST Data

We used the F-MNIST data set. As for the previous zoos for the MNIST data set, we have created one zoos with exactly the same number of CNN models as in `MNIST-SEED`. In this zoo that we call `FASHION-SEED`, we vary only the random seed (1-1000), while using only one fixed hyper-parameter configuration.

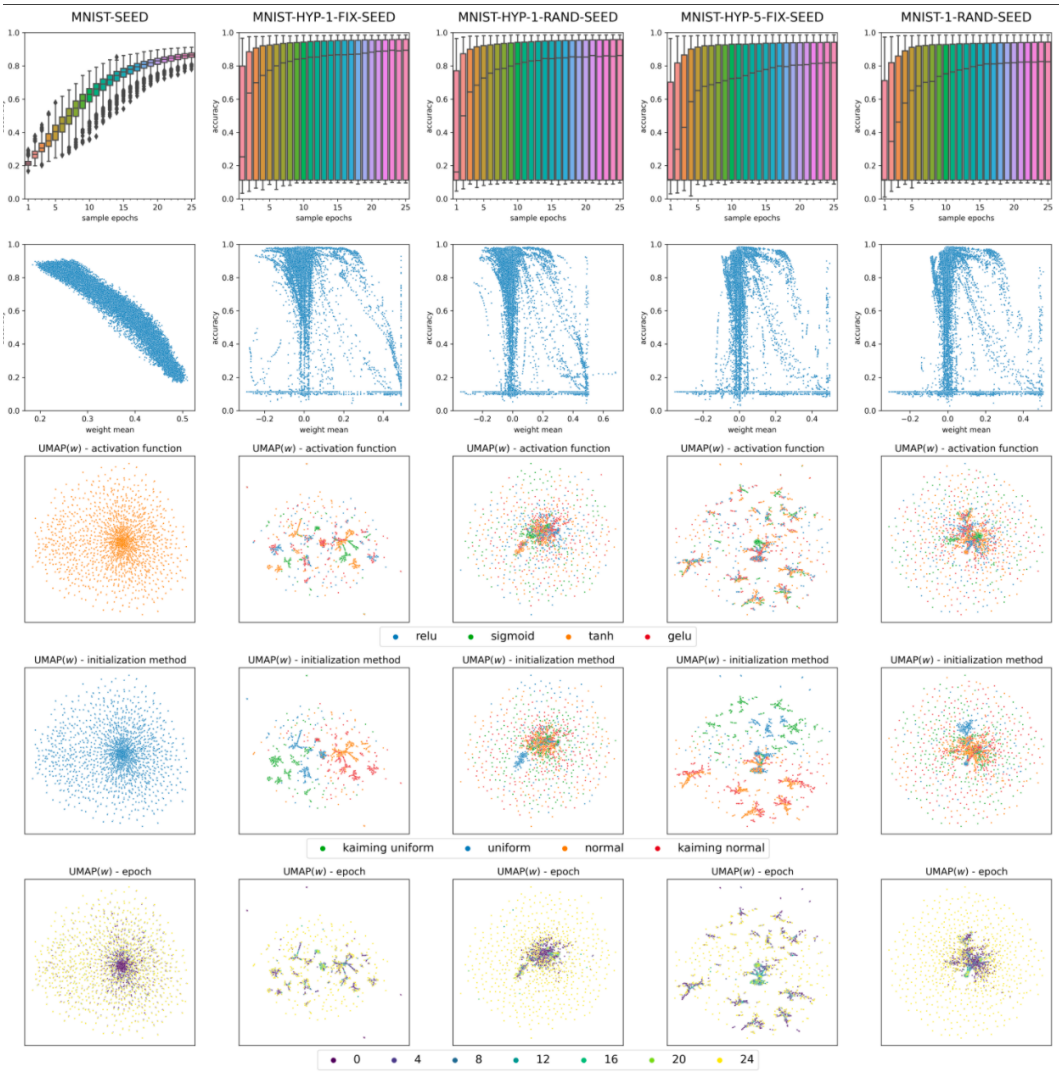


Figure 7: Visualization on the properties for the `MNIST-SEED`, `MNIST-HYP-1-FIX-SEED`, `MNIST-HYP-1-RANDOM-SEED`, `MNIST-HYP-5-FIX-SEED` and `MNIST-HYP-5-RANDOM-SEED` zoos. **Row One**. Boxplot of NNs accuracy over the epoch ids. **Row Two**. NNs accuracy plotted over the mean of the NNs weights of each sample. `MNIST-SEED` shows homogeneous development and a strong correlation between weight mean and accuracy, while varying the hyperparameters yields heterogeneous development without that correlation. **Rows Three to Five**. UMAP reductions of the weight space coloured by activation function, initialization method and sample epoch. Zoos with fixed seeds contain visible clusters of NNs that share same initialization method or activation function. Zoos with varying hyperparameters and random seeds do not contain such clear clusters.

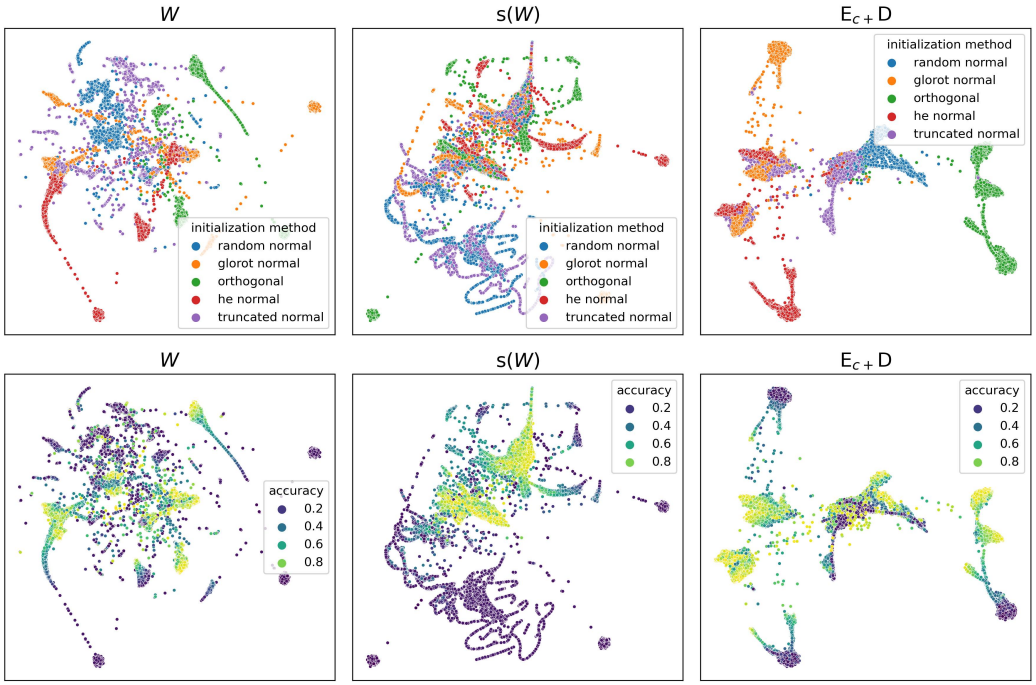


Figure 8: UMAP dimensionality reduction of the weight space (left), weight statistics (middle) and learned neural representations (right) for the MNIST-HYP zoo **?**. The initialization methods for the trained NN weights are already visually separable to a high degree in weight space, which carries over to the learned embedding space, while the statistics introduce a mix between the initialization methods. For accuracy, in seems that the statistics filter out and contain more relevant information than the weight space. Learned embeddings appears to cluster the models according to their initialization methods and within the clusters help to preserve high accuracy.

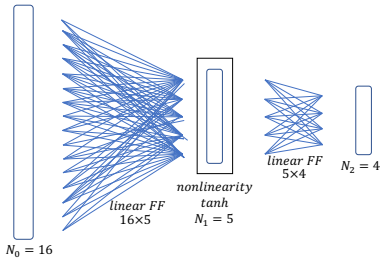


Figure 9: A diagram for the feed-forward architecture of the NNs in the `TETRIS-SEED` and `TETRIS-HYP` zoos.

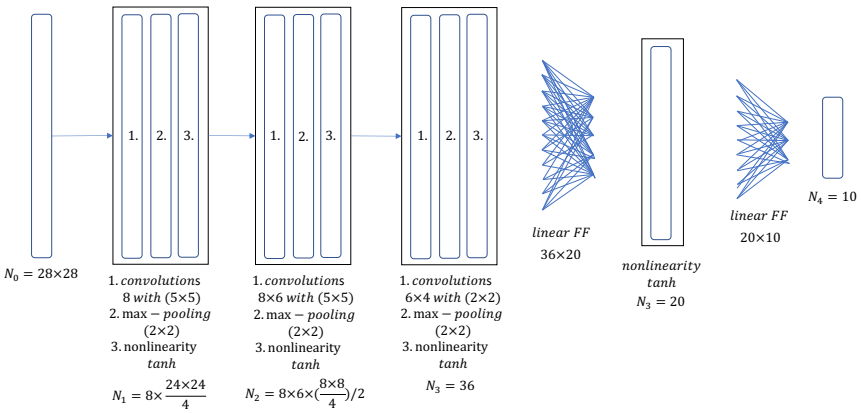


Figure 10: A diagram for the CNN architecture of the NNs in the `MNIST` zoos.

	TYPE	DETAILS	PARAMS
1.1	LINEAR	CH-IN=16, CH-OUT=5	80
1.2	NONLIN.	TANH	
2.1	LINEAR	CH-IN=5, CH-OUT=4	20

Table 10: FFN Architecture Details. CH-IN describes the number of input channels, CH-OUT the number of output channels.

	TYPE	DETAILS	PARAMS
1.1	CONV	CH-IN=1, CH-OUT=8, KS=5	208
1.2	MAXPOOL	KS= 2	
1.3	NONLIN.	TANH	
2.1	CONV	CH-IN=8, CH-OUT=6, KS=5	1206
2.2	MAXPOOL	KS= 2	
2.3	NONLIN.	TANH	
3.1	CONV	CH-IN=6, CH-OUT=4, KS=2	100
3.2	MAXPOOL	KS= 2	
3.3	NONLIN.	TANH	
4	FLATTEN		
5.1	LINEAR	CH-IN=36, CH-OUT=20	740
5.2	NONLIN.	TANH	
6.1	LINEAR	CH-IN=20, CH-OUT=10	200

Table 11: CNN Architecture Details. CH-IN describes the number of input channels, CH-OUT the number of output channels. KS denotes the kernel size, kernels are always square.

Appendix D. Additional Results

In this appendix section, we provide additional results about the impact of the compression ratio $c = N/L$.

D.1 Impact of the Compression Ratio N/L

In this subsection, we first explain the experiment setup and then comment on the results about the impact of the compression ratio on the performance for downstream tasks.

Experiment Setup. To see the impact of the compression ratio $c = N/L$ on the performance over the downstream tasks, we use our neural representation learning approach under different types of architectures, including E_c, ED and $\mathrm{E}_c\mathrm{D}$ (see Section 3 in the paper). As encoders E and decoders D, we used the attention-base modules introduced in Section 3 in the paper. The attention-based encoder and decoder, on the `TETRIS-SEED` and `TETRIS-HYP` zoos, we used 2 attention blocks with 1 attention head each, token dimensions of 128 and FC layers in the attention module of dimension 512.

We use our weight augmentation methods for representation learning (please see Section 3.1 in the paper). We run our representation learning algorithm for up to 2500 epochs, using the adam optimizer **?**, a learning rate of 1e-4, weight decay of 1e-9, dropout of 0.1 percent and batch-sizes of 500. In all of our experiments, we use 70% of the model zoos for training and 15% for validation and testing each. We use checkpoints of all epochs, but ensure that samples from the same models are either in the train or in the test split of the zoo. As quality metric for the self-supervised learning, we track the reconstruction R^2 on the test split of the zoo.

Results. As Table **??** shows, all NN architectures decrease in performance, as the compression ratio increases. The purely contrastive setup E_c generally learns embeddings which are useful for the downstream tasks, which are very stable under compression. These results strongly depend on a projection head with enough complexity. The closer the contrastive loss comes to the bottleneck of the encoder, the stronger the downstream tasks suffer under compression. Notably, the reconstruction of ED is very stable, even under high compression ratios. However, higher compression ratios appear to negatively impact the neural representations for the downstream tasks we consider here. The combination of reconstruction and contrastive loss shows the best performance for $c = 2$, but suffers under compression. Higher compression ratios perform comparably on the downstream tasks, but don't manage high reconstruction R^2. We interpret this as sign that the combination of losses requires high capacity bottlenecks. If the capacity is insufficient, the two objectives can't be both satisfied.

D.2 NN Model Characteristics Prediction on FASHION-SEED

Due to space limitations, here in Figure **??**, we present the results on the `FASHION-SEED` together with the results on `MNIST-SEED`. The experimental setup is same as for the `MNIST-SEED` zoo, which is explained in the paper. Here, we add a complementary result to our ablation study about the seed variation, that we presented in section 4.3 in the paper. Similarly to the discussion in the paper, random seeds variation in the `FASHION-SEED` again appears to make the prediction more challenging. The results show that the proposed approach is on-par with the comparing $s(W)$ for this type of model zoos.

D.3 In-distribution and Out-of-distribution Prediction

In Figures **??**, **??** and **??** we show in-distribution and out-of-distribution comparative results for test accuracy, epoch id and generalization gap prediction using the `MNIST-HYP` zoo.

In the majority of the results for accuracy and generalization gap prediction, our learned representations have higher R^2 and Kendall's τ score. Also, the baseline methods the distribution of predicted target values is more dispersed compared to the true target values. On the epoch id prediction we have comparable results but with lower score, we attribute this to the fact that the zoos contain sparse check points and we suspect that there are not enough so that our learning model could capture the present variability. Overall in the in-distribution and out-of-distribution results for test accuracy, epoch id and generalization gap prediction, the proposed approach has a slight advantage.

Encoder with contrastive loss E_c

c	REC	EPH	ACC	GGAP	F_{C0}	F_{C1}	F_{C2}	F_{C3}
2	–	96.7	90.8	82.5	67.7	72.0	74.4	85.8
3	–	96.6	89.4	81.5	68.4	69.4	71.1	85.1
5	–	96.4	89.5	81.8	67.1	68.7	69.7	84.0

Encoder and decoder with reconstruction loss ED

c	REC	EPH	ACC	GGAP	F_{C0}	F_{C1}	F_{C2}	F_{C3}
2	96.1	88.3	68.9	69.9	47.8	57.2	33.0	58.1
3	93.0	74.6	69.4	66.9	53.5	46.5	38.9	48.3
5	87.7	80.5	60.0	63.3	37.9	48.8	24.4	52.6

Encoder and decoder with reconstruction and contrastive loss E_cD

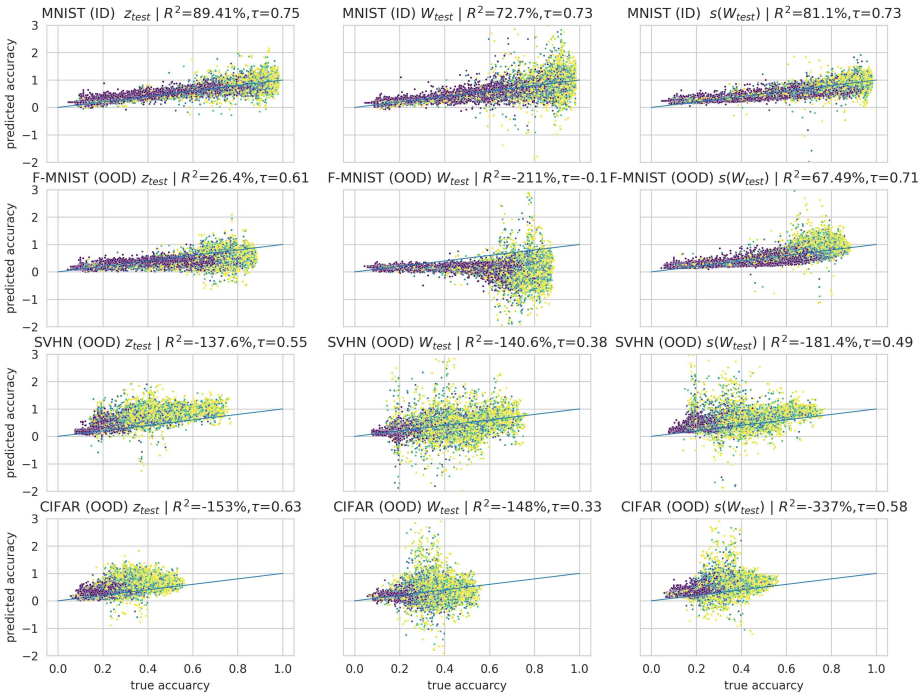
c	REC	EPH	ACC	GGAP	F_{C0}	F_{C1}	F_{C2}	F_{C3}
2	84.1	97.0	90.2	81.9	70.7	75.9	69.4	86.6
3	75.6	96.3	88.3	80.7	66.9	70.8	66.1	83.2
5	64.5	96.3	85.2	80.0	63.5	68.0	61.3	73.6

Table 12: The impact of the compression ratio $c = N/L$ in the different NN architectures of our approach for learning neural representations over the `Tetris-Seed` Model Zoo. All values are R^2 scores and given in %.

	MNIST-SEED			FASHION-SEED		
	W	S(W)	$E_{c+}D$	W	S(W)	$E_{c+}D$
EPH	84.5	**97.7**	97.3	87.5	**97.0**	95.8
ACC	91.3	98.7	**98.9**	88.5	97.9	**98.0**
GGAP	56.9	66.2	**66.7**	70.4	81.4	**83.2**

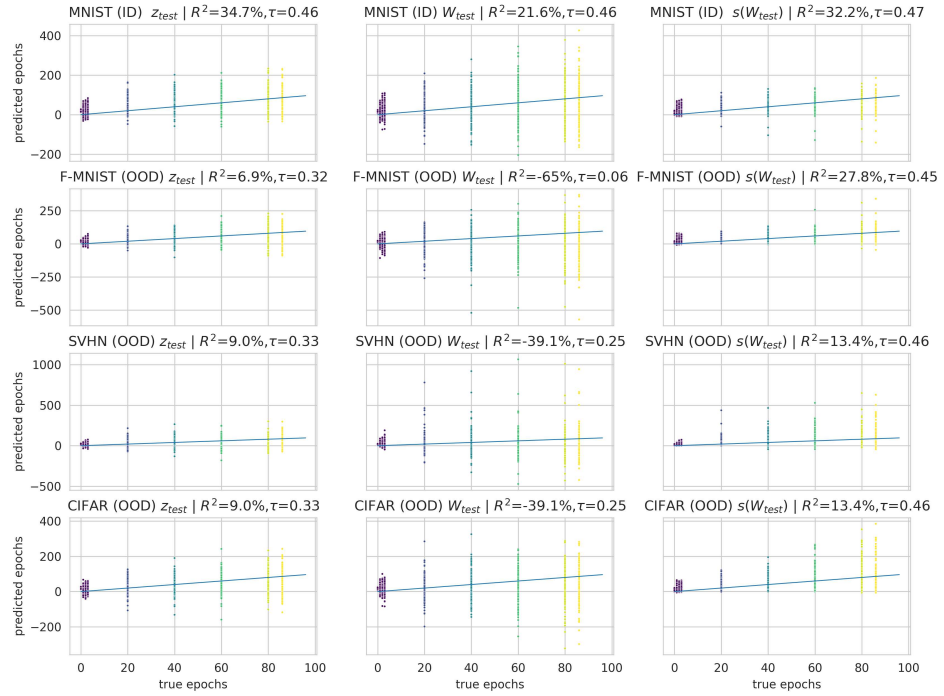
Table 13: R^2 score in % for epoch, accuracy and generalization gap.

Due to space limitations, for the `MNIST-SEED`, `FASHION-SEED` zoos and an additional `SVHN-SEED` zoo we only include out-of-distribution results for accuracy prediction in Figure **??**. Here, too, our learned representations have higher scores in both Kendall's τ as well as R^2. Further, the accuracy prediction for `SVHN-SEED` clearly preserves the order, but has a noticable bias. We attribute that effect to the different accuracy distributions of `MNIST-SEED` (ID, accuracy: [0.2,0.95]) and `SVHN-SEED` (OOD, accuracy: [0.2,0.75]). Due to the higher accuracy in `MNIST-SEED`, we suspect that the accuracy in `SVHN-SEED` is overestimated.



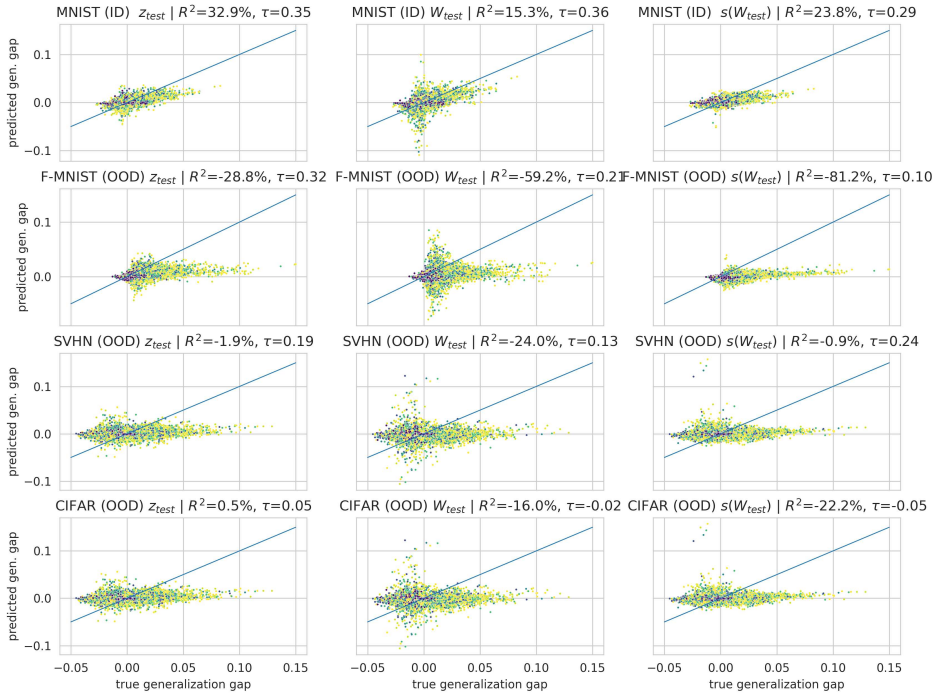
	MNIST-HYP			FASHION-HYP			SVHN-HYP			CIFAR10-HYP		
	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$
MNIST-HYP (τ)	.73	.73	**.75**	-.08	**.71**	.61	.38	.49	**.55**	.33	.58	**.63**
MNIST-HYP (R^2)	72.7	81.1	**89.4**	-211	**67**	26	-140	-180	**-137**	**-148**	-337	-153

Figure 11: In-distribution and out-of-distribution results for test accuracy prediction. Representation learning model and linear probes are trained on MNIST-HYP, and evaluated on MNIST-HYP, FASHION-HYP, SVHN-HYP and CIFAR-HYP.



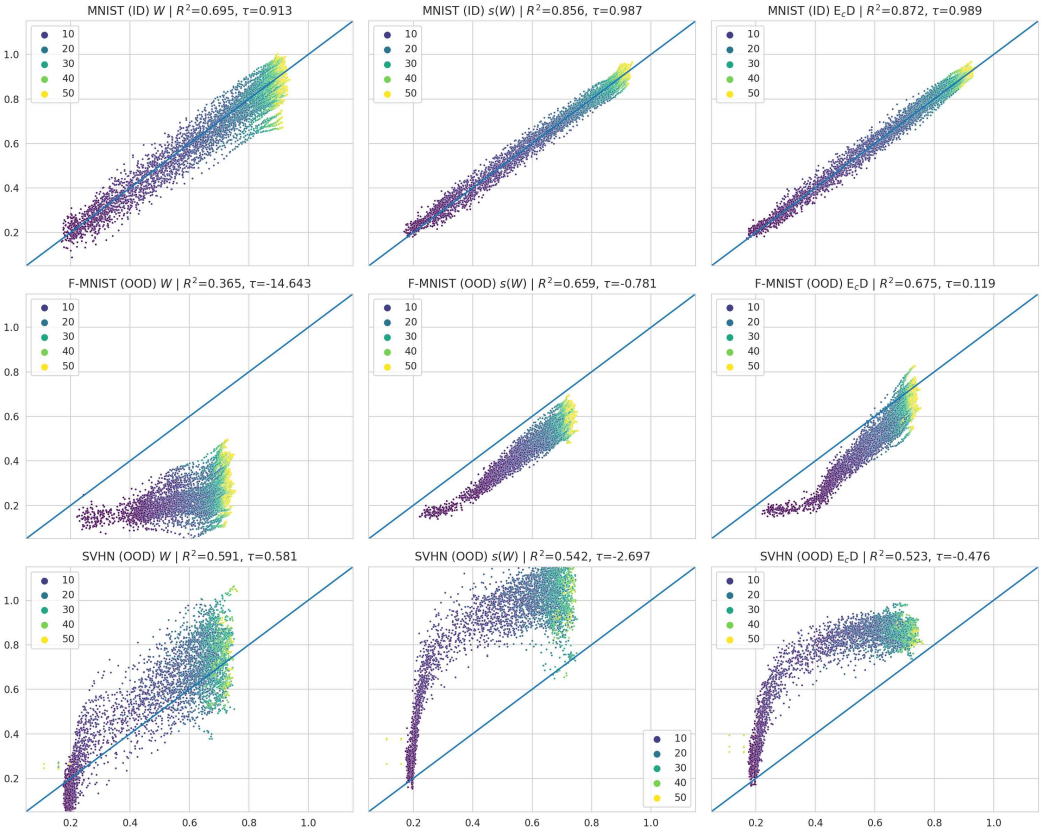
	MNIST-HYP			FASHION-HYP			SVHN-HYP			CIFAR10-HYP		
	W	S(W)	$E_{c+}D$	W	S(W)	$E_{c+}D$	W	S(W)	$E_{c+}D$	W	S(W)	$E_{c+}D$
MNIST-HYP (τ)	.46	**.47**	.46	.06	**.45**	.32	.25	**.46**	.33	.18	**.41**	.16
MNIST-HYP (R^2)	21.6	32.2	**34.7**	-64.9	**27.8**	6.9	-39.1	**13.4**	9.	-21.9	**19.2**	-13.

Figure 12: In-distribution and out-of-distribution results for the epoch id predictions. Representation learning model and linear probes are trained on MNIST-HYP, and evaluated on MNIST-HYP, FASHION-HYP, SVHN-HYP and CIFAR-HYP.



	MNIST-HYP			FASHION-HYP			SVHN-HYP			CIFAR10-HYP		
	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$
MNIST-HYP (τ)	**.36**	.29	.35	.20	.10	**.32**	.13	**.24**	.19	-.05	-.02	**.05**
MNIST-HYP (R^2)	15.3	24.8	**32.9**	-56.2	-81.8	**-27.8**	-24.	**-.9**	-1.9	-16.	-22.2	**.5**

Figure 13: In distribution and out-of-distribution results for the generalization gap predictions. Representation learning model and linear probes are trained on MNIST-HYP, and evaluated on MNIST-HYP, FASHION-HYP, SVHN-HYP and CIFAR-HYP.



	MNIST-SEED			FASHION-SEED			SVHN-SEED		
	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$
MNIST-SEED (τ)	.913	.987	**.989**	-14	-.781	**.12**	**.581**	-2.7	-.48
MNIST-SEED (R^2)	69.5	85.6	**87.2**	36.5	65.9	**67.5**	**.591**	.542	.523

Figure 14: In-distribution and out-of-distribution results for test accuracy prediction. Representation learning model and linear probes are trained on MNIST-SEED, and evaluated on MNIST-SEED, FASHION-SEED and SVHN-SEED.