Sampling from Hyper-Representations to Generate Neural Network Weights

Anonymous Author(s)
Affiliation
Address
email

Abstract

Learning representations of neural network weights given a model zoo is an emerging and challenging area with many potential applications from model inspection, to neural architecture search or knowledge distillation. Recently, an autoencoder trained on a model zoo was able to learn a *hyper-representation*, which captures intrinsic and extrinsic properties of the models in the zoo. In this work, we extend hyper-representations for generative use to sample new model weights. We propose layer-wise loss normalization which we demonstrate is key to generate high-performing models and several sampling methods based on the topology of hyper-representations. The models generated using our methods are diverse, performant and capable to outperform strong baselines as evaluated on several downstream tasks: initialization, ensemble sampling and transfer learning. Our results indicate the potential of knowledge aggregation from model zoos to new models via hyper-representations thereby paving the avenue for novel research directions.

1 Introduction

Over the last decade, countless neural network models have been trained and uploaded to different model hubs. Many factors such as random initialization and no global optimum ensure that the trained models are different from one another. What could we learn from such a population of neural network models? Since the parameter space of neural networks is complex and high-dimensional, representation learning from such populations (often referred to as model zoos) has become an emerging and challenging area.

Recent work along that direction has demonstrated the ability of such learned representations to capture intrinsic and extrinsic properties of the models in a zoo [36, 33, 25]. According to [33], NNs populate a low dimensional manifold, which can be learned with an autoencoder via self-supervised learning directly from the weights[1] without access to the original image data and labels. This so called *hyper-representation* has been demonstrated to be useful to predict several model properties such as accuracy, hyperparameters or architecture configurations.

However, [33] focused on discriminative downstream tasks by exploiting the encoder only. We take one step further and extend their work towards the generative downstream tasks by sampling model weights directly from the task-agnostic hyper-representation. To that end, we introduce a layer-wise normalization that improves the quality of decoded neural network weights significantly. Based on a careful analysis of the geometry, smoothness and robustness of this space, we also propose several sampling methods to generate weights in a single forward pass from the hyper-representation. We evaluate our approach on four image datasets and three generative downstream tasks of (i) model

[1] For conciseness, we refer to all weights and biases as weights.

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

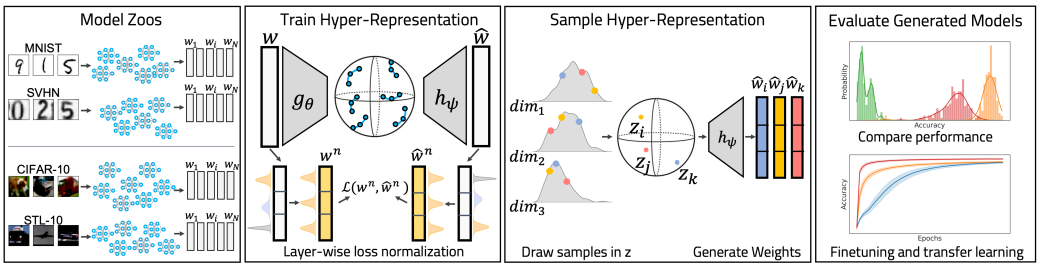

Figure 1: Outline of our approach: Model zoos are trained on image classification tasks. Hyper-representations are trained with self-supervised learning on the weights of the model zoos using layer-wise loss normalization in the reconstruction loss. We sample new embeddings in hyper-representation space and decode to weights. Generated models perform significantly better than random initialization or models sampled from baseline hyper-representations. They are diverse and can be combined to high-performant ensembles. Sampled models achieve high performance fine-tuned in-dataset and transfer learned on new datasets.

initialization, (ii) ensemble sampling, and (iii) transfer learning. Our results demonstrate its capability to out-perform previous hyper-representation learning and conventional baselines.

Previous work on generating model weights proposed (Graph) HyperNetworks [13, 39, 18], Bayesian HyperNetworks [7], HyperGANs [31] and HyperTransformers [40] for neural architecture search, model compression, ensembling, transfer- or meta-learning. These methods learn representations by using images and labels of the target domain. In contrast, our approach only uses model weights and does not need access to underlying data samples and labels – an emergent use case, e.g. of deep learning monitoring services or model hubs. In addition to the ability to generate novel and diverse model weights, compared to previous works our approach (a) can generate novel weights conditionally on model zoos from unseen tasks and (b) can be conditioned on the latent factors of the underlying hyper-representation. Notably, both (a) and (b) can be done without the need to retrain hyper-representations.

The results suggest our approach (Figure 1) to be a promising step towards a general purpose hyper-representation encapsulating knowledge of model zoos to advance different downstream tasks. The hyper-representations and code to reproduce our results will be made available upon publication.

2 Background: Training Hyper-Representations

We summarize the first stage of our method that corresponds to learning a hyper-representation of a population of neural networks, called a *model zoo* [33]. In [33] and this paper, a model zoo consists of models trained on the same task such as CIFAR-10 image classification [20]. Specifically, a hyper-representation is learned using an autoencoder $\hat{\mathbf{w}}_i = h(g(\mathbf{w}_i))$ on a zoo of M models $\{\mathbf{w}_i\}_1^M$, where $\mathbf{w}_i$ is the flattened vector of dimension N of all the weights of the i-th model. The encoder g compresses vector $\mathbf{w}_i$ to fixed-size hyper-representation $\mathbf{z}_i = g(\mathbf{w}_i)$ of lower dimension. The decoder h decompresses the hyper-representation to the reconstructed vector $\hat{\mathbf{w}}_i$. Both encoder and decoder are built on a self-attention blocks. The samples from model zoos are understood as sequences of convolutional or fully connected neurons. Each of the these is encoded as a token embedding and concatenated to a sequence. The sequence is passed through several layers of multi-head self-attention. Afterwards, a special compression token summarizing the entire sequence is linearly compressed to the bottleneck. The output is fed through a tanh-activation to achieve a bounded latent space $\mathbf{z}_i$ for the hyper-representation. The decoder is symmetric to the encoder, the embeddings are linearly decompressed from hyper-representations $\mathbf{z}_i$ and position encodings added.

Training is done in a multi-task fashion, minimizing the composite loss $\mathcal{L} = \beta\mathcal{L}_{MSE} + (1-\beta)\mathcal{L}_c$, where $\mathcal{L}_c$ is a contrastive loss and $\mathcal{L}_{MSE}$ is a weight reconstruction loss (see details in [33]). We can write the latter in a layer-wise way to facilitate our discussion in § 3.1:

$$\mathcal{L}_{MSE} = \frac{1}{MN} \sum\nolimits_{i=1}^{M} \sum\nolimits_{l=1}^{L} ||\hat{\mathbf{w}}_i^{(l)} - \mathbf{w}_i^{(l)}||_2^2, \tag{1}$$

where $\hat{\mathbf{w}}_i^{(l)}$, $\mathbf{w}_i^{(l)}$ are reconstructed and original weights for the l-th layer of the i-th model in the zoo. The contrastive loss $\mathcal{L}_c$ leverages two types of data augmentation at train time to impose structure on the latent space: permutation exploiting inherent symmetries of the weight space and random erasing.

3 Methods

In the following, we present (i) layer-wise loss normalization to ensure that decoded models are performant, and (ii) sampling methods to generate diverse populations of models.

3.1 Layer-Wise Loss Normalization

We observed that hyper-representations as proposed by [33] decode to dysfunctional models, with performance around random guessing. To alleviate that, we propose a novel layer-wise loss normalization (LWLN), which we motivate and detail in the following.

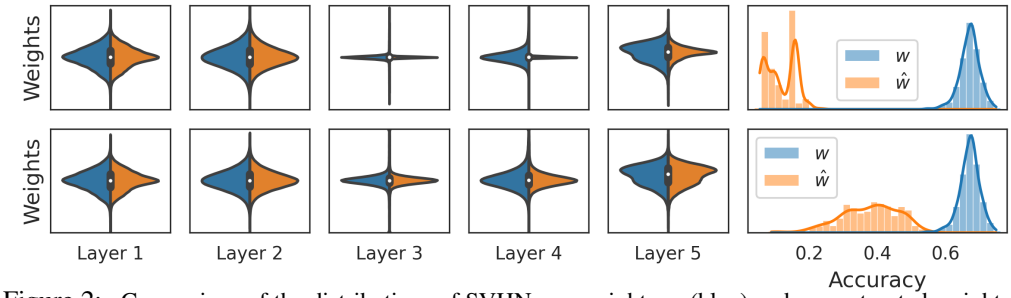

Figure 2: Comparison of the distributions of SVHN zoo weights $\mathbf{w}$ (blue) and reconstructed weights $\hat{\mathbf{w}}$ (orange) as well as their accuracy on the SVHN test set. **Top:** Baseline hyper-representation as proposed by [33], the weights of layers 3, 4 collapse to the mean. These layers form a weak link in reconstructed models. The accuracy of reconstructed models drops to random guessing. **Bottom:** Hyper-representation trained with layer-wise loss normalization (LWLN). The normalized distributions are balanced, all layers are evenly reconstructed, and the accuracy of reconstructed models is significantly improved.

Due to the MSE training loss in (1), the reconstruction error can generally be expected to be uniformly distributed over all weights and layers of the weight vector $\mathbf{w}$. However, the weight magnitudes of many of our zoos are unevenly distributed across different layers. In these zoos, the even distribution of reconstruction errors lead to undesired effects. Layers with broader distributions and large weights are reconstructed well, while layers with narrow distributions and small weights are disregarded. These layers can become a weak link in the reconstructed models, causing performance to drop significantly down to random guessing. The top row of Figure 2 shows an example of a baseline hyper-representation learned on the zoo of SVHN models [29]. Common initialization schemes [14, 10] produce distributions with different scaling factors per layer, so the issue is not an artifact of the zoos, but can exist in real world model populations. Similarly, recent work on generating models normalizes weights to boost performance [18]. In order to achieve equally accurate reconstruction across the layers, we introduce a layer-wise loss normalization (LWLN) with the mean μ_l and standard deviation σ_l of all weights in layer l estimated over the train split of the zoo:

$$\mathcal{L}_{\bar{MSE}} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{l=1}^{L} \left\| \frac{\hat{\mathbf{w}}_i^{(l)} - \mu_l}{\sigma_l} - \frac{\mathbf{w}_i^{(l)} - \mu_l}{\sigma_l} \right\|_2^2 = \frac{1}{MN} \sum_{i=1}^{M} \sum_{l=1}^{L} \frac{\|\hat{\mathbf{w}}_i^{(l)} - \mathbf{w}_i^{(l)}\|_2^2}{\sigma_l^2}. \tag{2}$$

3.2 Sampling from Hyper-Representations

We introduce methods to draw diverse and high-quality samples $\mathbf{z}^* \sim p(\mathbf{z})$ from the learned hyper-representation space to generate model weights $\mathbf{w}^* = h(\mathbf{z}^*)$. Such sampling is facilitated if there is knowledge on the topology of the space spanned by $\mathbf{z}$. One way to achieve that is to train a variational autoencoder (VAE) with a predefined prior [17] instead of the autoencoder of [33]. While training VAEs on common domains such as images has become well-understood and feasible, in our relatively novel weights domain, we found it problematic (see details in Appendix E). Other methods avoid a predefined prior of VAEs, either by analyzing the topology of the space learned by the autoencoder or fitting a separate density estimation model on top of the learned representation [24, 12]. These methods assume the representation space to have strong regularities. The hyper-representation space learned by the autoencoder of [33] is already regularized by dropout regularization applied to the encoder and decoder as in [9]. The contrastive loss component requiring similar models to be embedded close to each other may also improve the regularity of the representation space. Empirically,

we found our layer-wise loss normalization (LWLN) to further regularize the representation space by ensuring robustness and smoothness (see Figure 3 in § 4).

Given the smoothness and robustness of the learned hyper-representation space, we follow [24, 12, 9] in estimating the density and topology to draw samples from a regularized autoencoder. To that end, we introduce three strategies to sample from that space: $S_{\text{KDE}}, S_{\text{Neigh}}, S_{\text{GAN}}$. To model the density and topology in representation space, we use the embeddings of the train set as anchor samples $\{\mathbf{z}_i\}$. We observe that many anchor samples from $\{\mathbf{z}_i\}$ correspond to the models with relatively poor accuracy (Figure 2), so to improve the quality of sampled weights, we consider the variants of these methods using only those embeddings of training samples corresponding to the top 30% performing models. We denote these sampling methods as $S_{\text{KDE30}}, S_{\text{Neigh30}}, S_{\text{GAN30}}$ respectively. These methods can potentially decrease sample diversity, however, we found that the generated weights are still diverse enough (e.g. to construct high-performant ensembles, Figure 5). Finally, as baseline and sanity check we explore sampling uniformly in representation space S_U and sampling in low-probability regions S_C.

3.2.1 Uniform S_U

As a naive baseline, we draw samples uniformly in hyper-representation space (bounded by tanh, § 2) and denote it as S_U. This is naive, because we found that the embeddings $\mathbf{z}$ populate only sections of a shell of a high-dimensional sphere (see Figures 10 and 11 in Appendix D). So most of the uniform samples lie in the low-probability regions of the space and are not expected to be decoded to useful models.

3.2.2 Density estimation S_{KDE} and counterfactual sampling S_C

The dimensionality D of hyper-representations $\mathbf{z}$ in [33], as well as in our work, is relatively high due to the challenge of compressing weights $\mathbf{w}$. Fitting a probability density model to such a high-dimensional distribution is feasible by making a conditional independence assumption: $p(\mathbf{z}^{(j)}|\mathbf{z}^{(k)}, \mathbf{w}) = p(\mathbf{z}^{(j)}|\mathbf{w})$, where $\mathbf{z}^{(j)}$ is the j-th dimensionality of the embedding $\mathbf{z}$. To model the distribution of each j-th dimensionality, we choose kernel density estimation (KDE), as it is a powerful yet simple, non-parametric and deterministic method with a single hyperparameter. We fit a KDE to the M anchor samples $\{\mathbf{z}_i^{(j)}\}_{i=1}^{M}$ of each dimension j, and draw samples $z^{(j)}$ from that distribution: $z^{(j)} \sim p(\mathbf{z}^{(j)}) = \frac{1}{Mh}\sum_{i=1}^{M} K(\frac{\mathbf{z}^{(j)}-\mathbf{z}_i^{(j)}}{h})$, where $K(x) = (2\pi)^{-1/2}\exp(-\frac{x^2}{2})$ is the Gaussian kernel and h is a bandwidth hyperparameter. The samples of each dimension $z^{(j)}$ are concatenated to form samples $\mathbf{z}^* = [z^{(1)}, z^{(2)}, \cdots, z^{(D)}]$. This method is denoted as S_{KDE}.

As a sanity check, we invert the S_{KDE} method and explicitly draw samples from regions not populated by anchor samples, i.e. with low probability according to the KDE. This method, denoted as S_C, essentially samples counterfactual embeddings and similarly to S_U is expected to perform poorly.

3.2.3 Neighbor sampling S_{Neigh}

This method models the neighborhood topology to reduce the dimensionality and facilitate sampling. We approximate the neighborhood topology of the anchor samples $\mathbf{z}_i$ using an invertible function $k : \mathbb{R}^D \rightarrow \mathbb{R}^d$ with $\mathbf{n}_i = k(\mathbf{z}_i)$. We choose $d = 3$ as a sufficiently low dimensional space to sample from. Samples $\mathbf{n}^*$ are uniformly drawn from the box surrounding the d-dimensional reduction $\mathbf{n}^* \sim U(min(\mathbf{n}), max(\mathbf{n}))$ and mapped back to hyper-representations $\mathbf{z}^* = k^{-1}(\mathbf{n}^*)$. For k, we use a neighborhood-based dimensionality reduction method UMAP [26].

3.2.4 Latent space GAN S_{GAN}

A common choice for generative representation learning is generative adversarial networks (GANs) [11]. While training a GAN directly to generate weights is a promising yet challenging avenue for future research [31], we found the GAN framework to work reasonably well when trained on the hyper-representations. This idea follows [24, 12] that showed improved training stability and efficiency compared to training GANs on inputs directly. We train a generator $G : \mathbb{R}^d \rightarrow \mathbb{R}^D$ with $\mathbf{z}^* = G(\mathbf{n}^*)$ to generate samples in hyper-representation space from the Gaussian noise $\mathbf{n}^*$. We choose $d = 16$ as a compromise between size and capacity. See a detailed architecture of our GAN in Appendix E.

4 Experiments

4.1 Experimental Setup

We train and evaluate our approaches on four image classification datasets: MNIST [21], SVHN [29], CIFAR-10 [20] and STL-10 [4]. For each dataset, there is a model zoo that we use to train an autoencoder following [33].

Model zoos: In each image dataset, a zoo contains $M = 1000$ convolutional networks of the same architecture with three convolutional layers and two fully-connected layers. Varying only in the random seeds, all models of the zoo are trained for 50 epochs with the same hyperparameters following [33]. To integrate higher diversity in the zoo, initial weights are uniformly sampled from a wider range of values rather than using well-tuned initializations of [10, 14]. Each zoo is split in the train (70%), validation (15%) and test (15%) splits. To incorporate the learning dynamics, we train autoencoders on the models trained for 21-25 epochs following [33]. Here the models have already achieved high performance, but have not fully converged. The development in the remaining epochs of each model are treated as hold-out data to compare against. We use the MNIST and SVHN zoos from [33] and based on them create the CIFAR-10 and STL-10 zoos. Details on the zoos can be found in Appendix A.

Training details: We train separate hyper-representations on each of the model zoos. Images and labels are not used to train the hyper-representations (see § 2). Using the proposed sampling methods (§ 3.2), we generate new embeddings and decode them to weights. We evaluate sampled populations as initializations (epoch 0) and by fine-tuning for up to 25 epochs. We distinguish between in-dataset and transfer-learning. For in-dataset, the same image dataset is used for training and evaluating our hyper-representations and baselines. For transfer-learning, hyper-representations (and pre-trained models in baselines) are trained on a source dataset. All populations are evaluated and fine-tuned on a different target dataset. Full details on training, including infrastructure and compute is detailed in the Appendix B.

Baselines: As the first baseline, we consider the autoencoder of [33], which is same as ours but without the proposed layer-wise loss-normalization (LWLN, § 3.1). We combine this autoencoder with the S_{KDE30} sampling method and, hence, denote it as B_{KDE30}. We consider two other baselines based on training models with stochastic gradient descent (SGD): training from scratch on the target classification task B_T, and training on a source followed by fine-tuning on the target task B_F. The latter remains one of the strongest transfer learning baselines [3, 8, 19].

Reproducibility, reliability and comparability: We compare populations of at least 50 models to evaluate each method reliably. We report standard deviation in Tables 1-2 and statistical significance, effect size and 95% confidence interval in Appendix F. To ensure fairness and comparability, all methods share training hyperparameters. Fine-tuning uses the hyperparameters of the target domain.

4.2 Results

In the following, we first analyze the learned hyper-representations further justifying our sampling methods and assumptions made in § 3.2. We then confirm the effectiveness of our approach for model initialization without and with fine-tuning in the in-dataset and transfer learning settings.

4.2.1 Hyper-Representations are Robust and Smooth

We evaluate the robustness and smoothness of the hyper-representation space with two experiments. First, to evaluate robustness, we add different levels of noise to the embeddings of the test set to create $\tilde{\mathbf{z}}$, decode them to model weights $\tilde{\mathbf{w}}$ and compute models' accuracies on the classification task. We found that hyper-representations are robust to noise as large levels of relative noise >10% are required to affect performance (Figure 3, c). Second, to probe for smoothness, we linearly interpolate between the test set embeddings (i) along the trajectory of the same model at different epochs ($\mathbf{z}_{i,ep5}$ and $\mathbf{z}_{i,ep25}$) and (ii) between 250 random pairs of embeddings on the trajectories of different models ($\mathbf{z}_i$ and $\mathbf{z}_j$). We decode the interpolated embeddings and compute models' accuracies on the classification task. We found remarkably smooth development of accuracy along the interpolation in both schemes (Figure 3, d). The lack of fluctuations along and between trajectories support both local and global notions of smoothness in hyper-representation space.

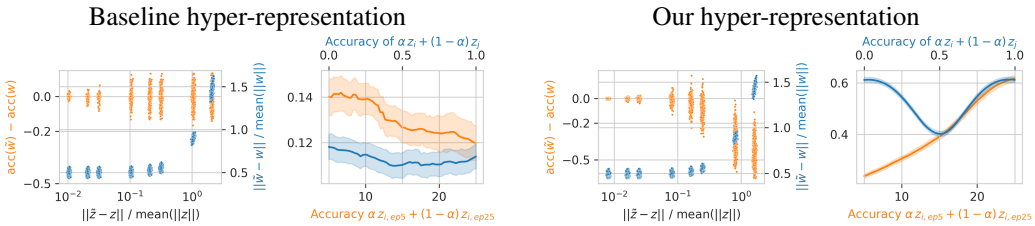

Figure 3: **(a,c):** Robustness of hyper-representations. Relatively large levels of relative noise >10% are necessary to affect the accuracy (orange) or reconstruction loss (blue). **(b,d):** Smoothness of hyper-representations. Orange: interpolations along model trajectories. Blue: interpolations between $\mathbf{z}$ of different models.

For the baseline autoencoder (without LWLN) decoded models all perform close to 10% accuracy, so these representations do not support similar notions of smoothness (Figure 3, b), while robustness can be misleading, since the accuracy even without adding noise is already low (Figure 3, a). Therefore, LWLN together with regularizations added to the autoencoder allow for learning robust and smooth hyper-representation. This property makes sampling from that representation more meaningful as we show next.

4.2.2 Sampling for In-dataset Initialization

Comparison between sampling methods: We evaluate the performance of different sampled populations (obtained with LWLN) *without fine-tuning* generated weights. On MNIST, all sampled models except those obtained using S_U and S_C perform better than random initialization (10% accuracy), but worse than models trained from scratch B_T for 50 epochs (Figure 4). Distribution-based samples (S_{KDE} and S_{GAN}) perform better than neighborhood based samples (S_{Neigh}). The populations based on the top 30% perform better than their 100% counterparts with S_{KDE30} as the strongest sampling method overall. This demonstrates that the learned hyper-representation and sampling methods are able to capture complex subtleties in weight space differentiating high and low performing models.

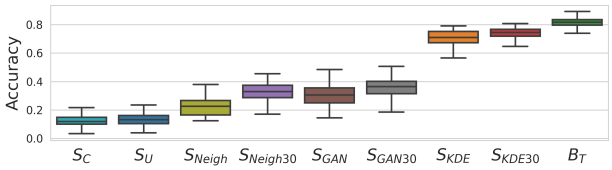

Figure 4: MNIST results of different populations as initialization.

Comparison to the baseline hyper-representations: We also compare S_{KDE30} that is based on our autoencoder with layer-wise loss normalization (LWLN) to the baseline autoencoder using the same sampling method (B_{KDE30}) without fine-tuning. On all datasets except for MNIST, S_{KDE30} considerably outperform B_{KDE30} with the latter performing just above 10% (random guessing), see Table 1 (rows with epoch 0). We attribute the success of LWLN to two main factors. First, LWLN prevents the collapse of reconstruction to the mean (compare Figure 2-top to Figure 2-bottom). Second, by fixing the weak links, the reconstructed models perform significantly better. We also evaluated input-to-output feature normalization, s.t. encoder and decoder operate on normalized weights, but empirically found it did not work as well as a normalization just for the loss. More details are provided in Appendix C.

Table 1: Mean and std accuracy (%) of sampled populations with LWLN (S_{KDE30}) and without (B_{KDE30}) compared to models trained from scratch B_T. Best results for each epoch and dataset are bolded.

Method	Ep.	MNIST	SVHN	CIFAR-10	STL-10
B_T	0		≈10% (random guessing)		
B_{KDE30}	0	63.2 | 7.2	10.1 | 3.2	15.5 | 3.4	12.7 | 3.4
S_{KDE30}	0	**68.6 | 6.7**	**51.5 | 5.9**	**26.9 | 4.9**	**19.7 | 2.1**
B_T	1	20.6 | 1.6	19.4 | 0.6	27.5 | 2.1	15.4 | 1.8
B_{KDE30}	1	83.2 | 1.2	67.4 | 2.0	39.7 | 0.6	**26.4 | 1.6**
S_{KDE30}	1	**83.7 | 1.3**	**69.9 | 1.6**	**44.0 | 0.5**	25.9 | 1.6
B_T	25	83.3 | 2.6	66.7 | 8.5	46.1 | 1.3	35.0 | 1.3
B_{KDE30}	25	**93.2 | 0.6**	**75.4 | 0.9**	48.1 | 0.6	**38.4 | 0.9**
S_{KDE30}	25	93.0 | 0.7	74.2 | 1.4	**48.6 | 0.5**	38.1 | 1.1
B_T	50	91.1 | 2.6	70.7 | 8.8	48.7 | 1.4	39.0 | 1.0

In-dataset fine-tuning: When fine-tuning, our S_{KDE30} and baseline B_{KDE30} appear to gradually converge to similar performance (Table 1). While unfortunate, this result aligns well with previous

findings that longer training and enough data make initialization less important [28, 15, 30]. We also compare S_{KDE30} and B_{KDE30} to training models from scratch (B_T). On all four datasets, both ours and the baseline hyper-representations outperform B_T when generated weights are fine-tuned for the same number of epochs as B_T. Notably, on MNIST and SVHN generated weights fine-tuned for 25 epochs are even better than B_T run for 50 epochs. Comparison to 50 epochs is more fair though, since the hyper-representations were trained on model weights trained for up to 25 epochs. These findings show that the models initialized with generated weights learn faster achieving better results in 25 epochs than B_T in 50 epochs.

Sampling ensembles: We found that a useful by-product of learning hyper-representations is the ability to generate high-performant ensembles at almost no extra computational cost, since both sampling and generation are computationally cheap. To demonstrate this effect, we compare ensembles formed using the baseline autoencoder (B_{KDE30}) and ours (S_{KDE30}) to the ensembles composed of models trained from scratch for 25 epochs (B_T). On SVHN, ensembles generated using the baseline B_{KDE30} stagnate below 20% (Figure 5). In contrast, ensembles generated using our S_{KDE30} gracefully improve with the ensemble size outperforming single B_T models and almost matching B_T ensembles on both datasets with enough models in the ensembles. Remarkably, on SVHN, the average test accuracy of generated ensembles of 15 models is 77.6%, which is considerably higher than 70.7% of models trained on SVHN for 50 epochs. We conclude, hyper-representations learned with LWLN generate models which are not only performant, but also diverse. Although generating ensembles requires learning hyper-representation and model zoo first, we assume that in future such a hyper-representation can be trained once and reused in unseen scenarios as we tentatively explore below (see results in Table 3 and the discussion therein).

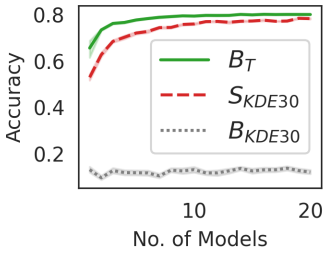

Figure 5: Generated ensembles evaluated on SVHN. Accuracies are averaged over 15 ensembles of randomly chosen models.

Do reconstructed models become similar to the original during fine-tuning? Sampled hyper-representations often learn faster and to a higher performance than the population of models they were trained on (Table 1). We therefore explore the question, if reconstructed models develop in weight space in the same direction as their original, or find a different solution. We found that the reconstructed models after one epoch of fine-tuning perform similarly well as their originals and slightly outperform from there on (Figure 6, left). At the same time, pairs of original and reconstructed models move further apart and become less aligned in weight space (Figure 6, right). It appears that reconstructed models perform better and explore different solutions in weight space to do so. This confirms the intuition, that hyper-representations impress useful structure on decoded weights. A pass through encoder and decoder thus results not just in a noisy reconstruction of the original sample. Instead, it maps to a different region on the loss surface, which leads to faster learning and better solutions. Considering the ensembling results above (Figure 5), hyper-representations do not collapse on a single good solution, but decode to diverse, useful weights.

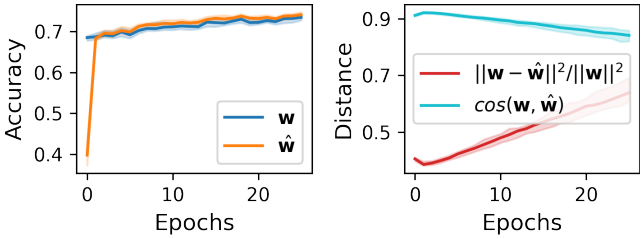

Figure 6: Original ($\mathbf{w}$) and reconstructed ($\hat{\mathbf{w}}$) SVHN models over 25 in-dataset fine-tuning epochs.

4.2.3 Sampling for Cross-dataset Initialization

Setup: We investigate the effectiveness of our method in a transfer-learning setup across image datasets. In particular, we report transfer learning results from SVHN to MNIST and from STL-10 to CIFAR-10 as two representative scenarios. Results on the other pairs of datasets can be found in Appendix F. In these experiments, pre-trained models B_F and the hyper-representation model are trained on a source domain. Subsequently, the pre-trained models B_F and the samples S_{KDE}, S_{Neigh} and S_{GAN} are fine-tuned on the target domain. The baseline approach (B_T) is based on training models from scratch on the target domain.

Table 2: Transfer-learning results (mean and standard deviation of the accuracy in %). Note that for STL-10 to CIFAR-10 the performance of all methods saturate quickly due to the limited capacity of models in the zoo making further improvements challenging as we discuss in § 4.3.

Method	SVHN to MNIST			STL-10 to CIFAR-10		
	Ep. 0	**Ep. 1**	**Ep. 50**	**Ep. 0**	**Ep. 1**	**Ep. 50**
B_T	10.0 | 0.6	20.6 | 1.6	91.1 | 1.0	10.1 | 1.3	27.5 | 2.1	48.7 | 1.4
B_F	**33.4 | 5.4**	84.4 | 7.4	95.0 | 0.8	**15.3 | 2.3**	29.4 | 1.9	**49.2 | 0.7**
S_{KDE30}	31.8 | 5.6	**86.9 | 1.4**	**95.5 | 0.4**	14.5 | 1.9	**29.6 | 2.0**	48.8 | 0.9
S_{Neigh30}	10.7 | 2.7	79.2 | 3.3	**95.5 | 0.7**	10.1 | 2.1	29.2 | 1.9	48.9 | 0.7
S_{GAN30}	10.4 | 2.4	75.0 | 6.3	94.9 | 0.7	10.2 | 2.5	28.6 | 1.8	48.8 | 0.8

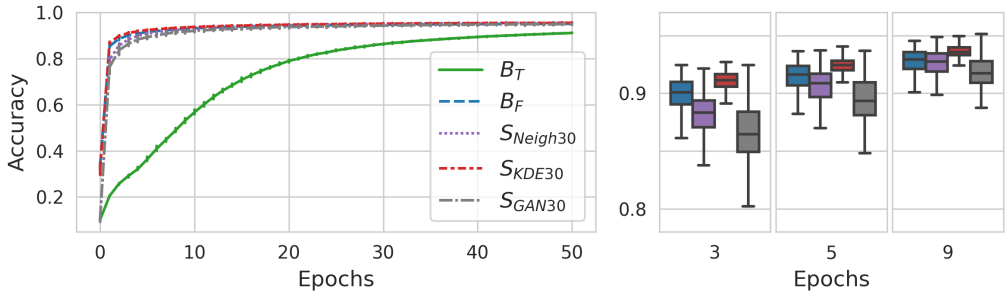

Figure 7: SVHN to MNIST transfer learning experiment: accuracy over epochs. Our sampling methods outperform the baselines after the first epoch. **Left:** epochs from 0 to 50. **Right**: epochs from 3 to 9, where B_T is significantly lower than 80% and thus is not visible.

Results: When transfer learning is performed from SVHN to MNIST, the sampled populations on average learn faster and achieve significantly higher performance than the B_T baseline and generally compares favorably to B_F (Figure 7, Table 2). In the STL-10 to CIFAR-10 experiment, all populations appear to saturate with only small differences in their performances (Table 2). Different sampling methods perform differently at the beginning versus the end of transfer learning. Generally, S_{KDE30} performs better in the first epochs, while all methods perform comparably at the end of transfer-learning. These discrepancies underline the difficulty of developing a single strong sampling method, which is an interesting area of future research. We further found that all datasets are useful sources for all targets (see Appendix F). Interestingly and other than in related work [27], even transfer from the simpler to harder datasets (e.g., MNIST to SVHN) improves performance.This might be explained by the ability of hyper-representations to capture a generic inductive prior useful across different domains, which we further investigate next.

Conditioning on unseen zoos: We explore if the hyper-representation trained on the models of one zoo (e.g. MNIST) can reconstruct the weights of another unseen zoo (e.g. SVHN). This can be useful to enable generation of weights for novel tasks without the need to retrain a hyper-representation. This is analogous to instance-conditioned GANs that recently were able to generate images from unseen domains without retraining GANs [2]. Our results in Table 3 show that while the performance on the unseen zoos is reduced, it is still well above random guessing (10%), especially when multiple model weights are sampled and ensembled. This is notable, as the hyper-representations were trained on single-dataset zoos.

Table 3: Accuracies (%) of models generated conditioned on the models of unseen zoos.

Training zoo	Conditioning (unseen)	Mean / max (bolded) accuracy One model	Ensemble
MNIST	SVHN	12.7 / **19.8**	13.4 / **18.7**
SVHN	MNIST	16.2 / **26.0**	22.1 / **29.8**
CIFAR-10	STL-10	18.0 / **24.4**	23.8 / **26.7**
STL-10	CIFAR-10	16.3 / **21.2**	20.0 / **23.0**

4.3 Limitations of Zoos with Small Models

To thoroughly investigate different methods and make experiments feasible, we chose to use the model zoos of the same scale as in [33]. While on MNIST and SVHN, the architectures of such model zoos allowed us to achieve a reasonably high performance, on CIFAR-10 and STL-10, the performance of all populations is limited by the low capacity of the models architecture. The models saturate at around 50% and 40% accuracy, respectively. The sampled populations reach the saturation point and fluctuate, but cannot outperform the baselines, see Appendix F. We hypothesize that due to the high remaining loss, the weight updates are correspondingly large without converging or improving performance. This may cause the weights to contain relatively little signal and high noise. Indeed, learning hyper-representations on the CIFAR-10 and STL-10 zoos was difficult and never reached similar reconstruction performance as in the MNIST and SVHN zoos. That in turn additionally limits the performance of the sampled populations. Future work will therefore focus on architectures with higher capacity.

5 Related Work

HyperNetworks: Representation learning on neural networks is typically based on HyperNetworks that learn low-dimensional structure of model weights to generate weights in a deterministic fashion [13, 1, 18, 39]. HyperNetworks have also been extended to meta-learning by conditioning weight generation on data [40, 32]. Closely related to our work, HyperGANs [31] can sample model weights by combining the hypernetworks and the GAN framework. Similarly, [7] allow for sampling model weights by conditioning the hypernetwork on a noise vector. However, training hypernetwork-based methods require input data (e.g. images) to feed to the neural networks. In practice, there may already be large collections of trained models, while their training data may not always be accessible. Learning representations of model weights without data, called hyper-representations, has been recently introduced in [33]. Our methods build on that work to allow for better reconstruction and sampling. [6] showed that network weights of a single model can be accurately reconstructed, however in our work we leverage the autoencoder to train a representation of the entire model zoo.

Transfer Learning: Transfer learning via fine-tuning aims at re-using models and their learned knowledge from a source to a target task [38, 3, 8, 27, 19]. Transfer learning models makes training less expensive, boosts performance, or allows to train on datasets with very few samples and has been applied on a wide range of domains [42]. The common transfer learning methods however only consider transferring from a single model, and so disregard the large variety of pre-trained models and potential benefit of combining them.

Knowledge distillation: Our work is related to [37, 22, 34] that allow to distill knowledge from a model zoo into a single network. Knowledge distillation overcomes the inherent limitation of transfer learning by transferring the knowledge from many large teacher models to a relatively small student model [23, 35]. Knowledge distillation however requires the source models at training as in [23] and at inference as in[35] thus increasing memory cost. Further, the learned knowledge cannot be shared between different target models. **Learnable initialization** [5, 41] provide methods to improve initialization by leveraging the meta-learning and gradient-flow ideas. In contrast to knowledge distillation and learnable initialization, we train a hyper-representation of a model zoo in a latent space, which is a more general and powerful approach that can enable sampling an ensemble, property estimation, improved initialization and implicit knowledge distillation across datasets.

6 Conclusion

In this paper, we propose a new method to sample from hyper-representations to generate neural network weights in one forward pass. We extend the training objective of hyper-representations by a novel layer-wise loss normalization which is key to the capability of generating functional models. Our method allows us to generate diverse populations of model weights, which show high performance as ensembles. We evaluate sampled models both in-dataset as well as in transfer learning and find them capable to outperform both models trained from scratch, as well as pre-trained and fine-tuned models. Populations of sampled models generally learn faster and achieve statistically significantly higher performance. This demonstrates that such hyper-representation can be used as a generative model for neural network weights and therefore might serve as a building block for transfer learning from different domains, meta learning or continual learning.

References

[1] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. *Advances in neural information processing systems*, 29, 2016. 9

[2] Arantxa Casanova, Marlène Careil, Jakob Verbeek, Michal Drozdzal, and Adriana Romero Soriano. Instance-conditioned gan. *Advances in Neural Information Processing Systems*, 34, 2021. 8

[3] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019. 5, 9

[4] Adam Coates, Honglak Lee, and Andrew Y Ng. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *Proceedings of the 14th International Con- Ference on Artificial Intelligence and Statistics (AISTATS)*, page 9, 2011. 5

[5] Yann N Dauphin and Samuel Schoenholz. MetaInit: Initializing learning by learning to initialize. In *Neural Information Processing Systems*, page 13, 2019. 9

[6] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *Advances in neural information processing systems*, 26, 2013. 9

[7] Lior Deutsch. Generating Neural Networks with Neural Networks. *arXiv:1801.01952 [cs, stat]*, April 2018. 2, 9

[8] Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. *arXiv preprint arXiv:1909.02729*, 2019. 5, 9

[9] Partha Ghosh, Mehdi S. M. Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From Variational to Deterministic Autoencoders. *arXiv:1903.12436 [cs, stat]*, May 2020. 3, 4

[10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. page 8, 2010. 3, 5

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. page 9. 4

[12] Yong Guo, Qi Chen, Jian Chen, Qingyao Wu, Qinfeng Shi, and Mingkui Tan. Auto-embedding generative adversarial networks for high resolution image synthesis. *IEEE Transactions on Multimedia*, 21(11):2726–2737, 2019. 3, 4

[13] David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. *arXiv:1609.09106 [cs]*, December 2016. 2, 9

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, February 2015. 3, 5

[15] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019. 7

[16] Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding Dimensional Collapse in Contrastive Self-supervised Learning. In *International Conference on Learning Representations*, September 2021. 18

[17] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014. 3

[18] Boris Knyazev, Michal Drozdzal, Graham W. Taylor, and Adriana Romero-Soriano. Parameter Prediction for Unseen Deep Architectures. *arXiv:2110.13100 [cs, stat]*, October 2021. 2, 3, 9

[19] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pages 491–507. Springer, 2020. 5, 9

[20] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. page 60. 2, 5

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 1558-2256. doi: 10.1109/5.726791. 5

[22] Iou-Jen Liu, Jian Peng, and Alexander G Schwing. Knowledge flow: Improve upon your teachers. *arXiv preprint arXiv:1904.05878*, 2019. 9

[23] Iou-Jen Liu, Jian Peng, and Alexander G. Schwing. Knowledge Flow: Improve Upon Your Teachers. *arXiv:1904.05878 [cs, stat]*, April 2019. 9

[24] Jinlin Liu, Yuan Yao, and Jianqiang Ren. An acceleration framework for high resolution image synthesis. *arXiv preprint arXiv:1909.03611*, 2019. 3, 4

[25] Charles H Martin, Tongsu Serena Peng, and Michael W Mahoney. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*, 12(1):1–13, 2021. 1

[26] Leland McInnes, John Healy, and Nathaniel Saul. UMAP: Uniform Manifold Approximation and Projection. 2018. 4

[27] Thomas Mensink, Jasper Uijlings, Alina Kuznetsova, Michael Gygli, and Vittorio Ferrari. Factors of Influence for Transfer Learning across Diverse Appearance Domains and Task Types. *arXiv:2103.13318 [cs]*, November 2021. 8, 9

[28] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015. 7

[29] 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*, page 9, 2011. 3, 5

[30] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. *Advances in neural information processing systems*, 28, 2015. 7

[31] Neale Ratzlaff and Li Fuxin. HyperGAN: A Generative Model for Diverse, Performant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5361–5369. PMLR, May 2019. 2, 4, 9

[32] James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E Turner. Fast and flexible multi-task classification using conditional neural adaptive processes. *Advances in Neural Information Processing Systems*, 32, 2019. 9

[33] Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-Supervised Representation Learning on Neural Network Weights for Model Characteristic Prediction. In *NeurIPS)*, volume 35, page 13, 2021. 1, 2, 3, 4, 5, 9

[34] Yang Shu, Zhi Kou, Zhangjie Cao, Jianmin Wang, and Mingsheng Long. Zoo-tuning: Adaptive transfer from a zoo of models. In *International Conference on Machine Learning*, pages 9626–9637. PMLR, 2021. 9

[35] Yang Shu, Zhi Kou, Zhangjie Cao, Jianmin Wang, and Mingsheng Long. Zoo-Tuning: Adaptive Transfer from a Zoo of Models. In *ICML*, page 12, 2021. 9

[36] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting Neural Network Accuracy from Weights. *arXiv:2002.11448 [cs, stat]*, February 2020. 1

[37] Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, and Richard Zemel. Adversarial distillation of bayesian neural network posteriors. In *International conference on machine learning*, pages 5190–5199. PMLR, 2018. 9

[38] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *arXiv:1411.1792 [cs]*, November 2014. 9

[39] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph HyperNetworks for Neural Architecture Search. *arXiv:1810.05749 [cs, stat]*, December 2020. 2, 9

[40] Andrey Zhmoginov, Mark Sandler, and Max Vladymyrov. HyperTransformer: Model Generation for Supervised and Semi-Supervised Few-Shot Learning. *arXiv:2201.04182 [cs]*, January 2022. 2, 9

[41] Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W Ronny Huang, and Tom Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *Advances in Neural Information Processing Systems*, 34, 2021. 9

[42] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning. *arXiv:1911.02685 [cs, stat]*, June 2020. 9

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 4.3 and 6
   (c) Did you discuss any potential negative societal impacts of your work? [No] The method we propose is a general machine learning method. It comes with all the risks involved, but in our opinion does not pose any particular risk.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code, hyper-representations of selected model zoos are included in the supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All the details are specified in the Appendix A and B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report means and std for all experiments in the main paper and provide additional details about median, CI95 on the median, p-value and effect size in the Appendix F
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix B
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See Section A
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We have trained new model zoos and will make them publicly available via an URL
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Our experiments use publicly available image datasets, which we cite. The model zoos and code for hyper-representation learning was published on github and zenodo by the authors of the original paper under Creative Commons Attribution 4.0.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Our experiments rely on common computer vision datasets and do not contain personally identifiable information nor offensive content.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]