HYPER-REPRESENTATIONS: LEARNING FROM POPULATIONS OF NEURAL NETWORKS

DISSERTATION

of the University of St.Gallen,
School of Management,
Economics, Law, Social Sciences,
International Affairs and Computer Science,

to obtain the title of
Doctor of Philosophy in Computer Science

submitted by

Konstantin Schürholt

from

Germany

Approved on the application of

Prof. Dr. Damian Borth

and

Prof. Michael Mahoney, Ph.D.

and

Prof. Xavier Giró-i-Nieto, Ph.D.

Table of Contents

List of Figures

List of Tables

Abstract

Neural Networks (NNs) have undergone a remarkable evolution, transitioning from academic labs to key technologies across various domains. This proliferation underscores their capability and versatility. As these models become integral to critical decision-making processes, the demand for methods to understand their inner workings likewise becomes more pronounced. This thesis addresses the challenge of understanding NNs through the lens of their most fundamental component: the weights, which encapsulate the learned information and determine the model behavior.

The NN weight space contains complex local and global structures which makes it a challenging domain. Addressing these challenges, this thesis develops innovative representation learning methods for the domain of weight spaces. The proposed methods embed and disentangle model weights in a representation space. The representation space allows not only to analyze existing models but also to generate new models with specified characteristics. Such an analysis builds on populations of models, to develop a nuanced understanding of the structure of NN weights.

At the core of this thesis is a fundamental question: Can we learn general, task-agnostic representations from populations of Neural Network models? The key contribution of this thesis to answer that question are *hyper-representations*, a self-supervised method to learn representations of NN weights. Work in this thesis finds that trained NN models indeed occupy meaningful structures in the weight space, that can be learned and used. Through extensive experiments, this thesis demonstrates that *hyper-representations* uncover model properties, such as their performance, state of training, or hyperparameters.

Moreover, the identification of regions with specific properties in *hyper-representation* space allows to sample and generate model weights with targeted properties. This thesis demonstrates applications for fine-tuning, and transfer learning to great success. Lastly, it presents methods that allow *hyper-representations* to generalize beyond model sizes, architectures, and tasks. The practical implications of that are profound, as it opens the door to foundation models of Neural Networks, which aggregate and instantiate their knowledge across models and architectures.

Ultimately, this thesis contributes to the deeper understanding of Neural Networks by investigating structures in their weights which leads to more interpretable, efficient, and adaptable models. By laying the groundwork for representation learning of NN weights, this research demonstrates the potential to change the way Neural Networks are developed, analyzed, and used.

Zusammenfassung

Neuronale Netze (NNs) haben eine bemerkenswerte Evolution durchlaufen, von akademischen Laboren zu Schlüsseltechnologien in verschiedenen Bereichen. Diese Ausbreitung unterstreicht ihre Vielseitigkeit und Fähigkeiten. Da diese Modelle integraler Bestandteil kritischer Prozesse werden, wird auch die Nachfrage nach Methoden, ihre inneren Abläufe zu verstehen, immer ausgeprägter. Diese Dissertation adressiert die Herausforderung, NNs durch die Linse ihrer grundlegendsten Komponente zu verstehen: die Gewichte, welche die gelernte Information beinhalten und das Modellverhalten bestimmen.

Angesichts der hohen Informationsmenge und der herausfordernden Struktur der NN-Gewichte entwickelt diese Arbeit innovative Methoden des Repräsentationslernens für diesen Domäne. Diese Methoden betten Modellgewichte in einen aussagekräftigen Repräsentationsraum ein. Solchhe Repräsentationen ermöglichen nicht nur die Analyse bestehender Modelle, sondern auch die Generierung neuer Modelle mit spezifizierten Eigenschaften. Eine solche Analyse baut auf Populationen von Modellen auf, um ein nuanciertes Verständnis der Gewichtsstruktur zu entwickeln.

Im Kern dieser Dissertation steht eine grundlegende Frage: Können allgemeine, aufgabenagnostische Repräsentationen aus Populationen von neuronalen Netzwerkmodellen gelernt werden? Der Schlüsselbeitrag dieser Dissertation zur Beantwortung dieser Frage sind *hyper-representations*, eine selbstüberwachte Methode, um Strukturen innerhalb der NN-Gewichte zu lernen. Beiträge in dieser Dissertation finden heraus, dass trainierte NN-Modelle tatsächlich bedeutungsvolle Strukturen im Gewichtsraum besetzen, die gelernt und genutzt werden können. Durch umfangreiche Experimente demonstriert diese Dissertation, dass *hyper-representations* Modellcharakteristiken, wie ihre Leistung, ihren Lernfortschritt oder Hyperparameter, aufdecken.

Darüber hinaus ermöglicht die Identifikation von Regionen mit spezifischen Eigenschaften im *hyper-representations* Raum das Generieren von Modellgewichten mit gezielten Eigenschaften. Diese Dissertation zeigt erfolgreiche Anwendungen für Feinabstimmung und Transferlernen auf. Zuletzt präsentiert sie Methoden, die es *hyper-representations* ermöglichen, über Modellgrößen, Architekturen und Aufgaben hinaus zu generalisieren. Dadurch sind erstmals Grundlagenmodellen von NNs möglich, die Wissen über Modelle und Architekturen aggregieren und instanziieren können.

Letztendlich trägt diese Dissertation zum tieferen Verständnis Neuronaler Netze bei, indem sie Strukturen in ihren Gewichten untersucht, was zu interpretierbareren, effizienteren und anpassungsfähigeren Modellen führt. Indem sie die Grundlage für das Repräsentationslernen von NN-Gewichten legt, zeigt diese Forschung das Potenzial, die Art zu verändern, wie Neuronale Netze entwickelt, analysiert und genutzt werden.

Chapter 1

Introduction

Over the past years, Neural Networks (NNs) have transitioned from experimental tools in laboratory environments to cornerstone technologies in production systems across the globe. Their applications range from enabling autonomous vehicles to detect and navigate through complex environments to curating personalized news feeds in traditional and social media platforms. NNs can digest and generate content across various mediums—text, audio, images, and video—and even determine insurance policies. As NNs become increasingly integrated into applications with significant societal impact, the demand for their trustworthiness and safety becomes paramount.

Trustworthiness in NNs is multifaceted. It is not solely about achieving high performance and making accurate decisions in challenging conditions. Trustworthiness also hinges on transparency and explainability. A core part of this is understanding how and why decisions are made, especially in high-stakes applications. Furthermore, accountability is integral to trust. This requires implementing robust mechanisms for identity and version control of models, ensuring that modifications and deployments are traceable and justifiable. Fundamentally, achieving this level of trust, transparency, and accountability necessitates a profound understanding of NN models themselves.

Consequently, there is a need for technical solutions to understand the inner working of NN models. An improved understanding also helps guide NN training and thus improve the performance of models. On a high level, improved NN model understanding can serve two purposes: **(i)** model analysis and **(ii)** model generation. These two purposes make up one axis on the landscape considered for this thesis, see Figure 1.1. For both, different perspectives can be considered, which make up the other axis of the landscape. Along the machine learning (ML) pipeline, three elements can be used to operate on models: **(a)** their behavior when confronted with data; **(b)** their generating factors, i.e., their hyperparameters, as proxies for model behavior; and **(c)** the trainable weights of NN models.

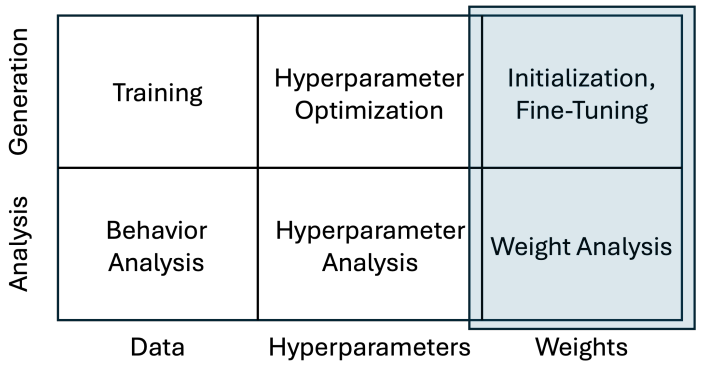


Figure 1.1: Overview of the landscape spanned by applications for and perspectives on Neural Network models. Due to the rich information and application potential, the focus of this thesis lies on NN weights for model analysis and generation.

Data is used for model generation in regular gradient-based training to minimize the loss. More complex hyper-networks generate weights but likewise use gradient signals from data [61]. Data-based analysis is realized by evaluating model behavior on holdout and test data. The advantage of using data is that it is close to real-world usage. On the other hand, it can be limited by the availability and expressiveness of data as well as suitable evaluation metrics. Hyperparameter analysis predicts model performance based on hyperparameters. For model generation, hyperparameters are used as inputs in hyperparameter optimization[75]. These techniques implicitly connect hyperparameters with model behavior for both analysis and generation. While widely used, the connection between hyperparameters and behavior is indirect and nonlinear, which makes modeling and using it a challenging task. Lastly, weight initialization, fine-tuning, or transfer learning from pre-trained models are an inherent part of NN training. The weights can also be used directly as inputs for model analysis [43, 119, 159] or model generation [1, 168]. NN model weights are the immediate outcome of NN training, and as such determine model behavior. However, they also encode training information like model accuracy, epoch, and hyperparameters. Therefore, this thesis focuses on using weights of trained NNs for both model analysis and model generation.

The relation between behavior and weights has complex local and global structures. To make statements beyond one model and improve understanding of NN models more broadly therefore requires sufficient coverage of these structures. Hence, research in this domain requires analyzing not just single models, but a broad spectrum of trained NNs. By varying the generating factors such as hyperparameters, datasets, and architectures and training multiple models, diverse populations of trained NN models can be generated. Studying diverse populations of models allows the findings to generalize to real-world models. Therefore, the scope of this thesis is on model populations.

Prior to the work on this thesis, a unified, task-agnostic representation learning method on populations of NN weights that is suitable for both model analysis and model generation did not exist.

The goal of this thesis is therefore to develop suitable representation learning methods for Neural Network weight spaces, which can be used to analyze models in a model representation space, as well as generate new weights with targeted properties.

The following paragraphs discuss the complexities in NN training, structures, and challenges in NN weight spaces to identify the research gap and research questions. Subsequently, the contribution of this thesis to address these questions is summarized.

Neural Network Weight Spaces

The Unreasonable Success of Neural Networks Over the past decade, Neural Networks (NNs) have been improved tremendously, and are the state of the art across many domains, such as computer vision [40, 65], natural language processing [15, 37, 158], text-to-speech systems [138], or even video generation [13]. The success of NNs is impressive, considering that the training of NNs is a hard optimization problem. NN training is NP-complete [11]. Further, the loss surface and optimization problem are highly non-convex [32, 57, 101]. This makes navigating the loss surface to a global minimum a more challenging task. Due to the non-convexity, NN models with different random initialization or hyperparameters may end up in different local minima on the loss surface and therefore also have different model weights. With recent growing model sizes, NN training is also increasingly high dimensional, which requires the tuning of ever more optimization parameters. These properties of the optimization problem not only make training difficult. The trained model as the outcome of the optimization is also sensitive to hyperparameter choices [62, 103, 174]. Recent work investigates the mode connectivity of different training outcomes [1, 6, 41, 50, 51, 131]. Nonetheless, it remains an open question, whether trained models with different model weights learn qualitatively different representations.

The gaps in understanding the relation between the trained NN weights and the model behavior create the two major challenges for NNs identified earlier: **(i) predicting model behavior** to diagnose trained models [26, 117, 174], for hyperparameter optimization [8, 22], or neural architecture search[44]; and **(ii) generating weights** with desirable properties as initializations [30, 54, 64, 187], for fine-tuning or transfer-learning [122, 176].

Structures in Neural Network Weights This thesis attempts to address both challenges by using the weights only. This assumes sufficient structure in the weights of populations of NN models, and that such structure encodes latent factors of the models. From an information theory standpoint, NNs learn structured information in the data, during which the weights also become structured. Indeed, weight matrix entropy as a proxy for disorder is reduced during training [117].

Formally, NN training is a combination of a dataset $\mathcal{D}$, architecture $\mathcal{A}$, task $\mathcal{T}$, loss $\mathcal{L}$ and training hyperparameters λ, all of which are structured. Concretely, the dataset $\mathcal{D}$ contains samples x which are structured, i.e. images. Datasets for supervised learning tasks further contain labels y for each sample. The training task $\mathcal{T}$ and corresponding loss $\mathcal{L}$ determine what signal from the data is learned. The NN architecture $\mathcal{A}$ with training hyperparameters λ imposes structure in how data and weights are processed. NN training finds optimal weights $\mathbf{W}^*$ by minimizing the training loss as $\mathbf{W}^* = \arg\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}, \mathcal{D}, \mathcal{A}, \lambda)$. Through this optimization, the information from the data is encoded in the model weights $\mathbf{W}$, imposing structure on $\mathbf{W}$. Consequently, the structure in the weights $\mathbf{W}^*$ reflects their latent generating factors $\mathcal{D}$, $\mathcal{L}$, λ and $\mathcal{A}$. Since model behavior and performance are also a consequence of $\{\mathcal{D}, \mathcal{L}, \lambda, \mathcal{A}\}$, they, too, are reflected in the NN weights. The perspective on structure in a single model can be extended to structure in the *weight space*, the space spanned by the individual weight dimensions, in which a single model is one point. Extending the earlier thoughts, by induction, the notion of structure in weights of individual models also implies structure in populations of trained models.

This leads to the main hypothesis for this thesis:

(i) Neural Network models populate a structure in weight space;

(ii) These structures encode properties and generating factors of models.

(iii) Such structures can be exploited for discriminative and generative tasks.

Challenges, Thesis Objectives, and Contributions

Challenges of Neural Network Weight Spaces Operating in weight space to identify such structure poses several challenges. First, with growing models, the size of the parameter space also grows. The larger space requires more samples to have sufficient coverage. The correspondingly increasing cost of training those models exacerbates the curse of dimensionality of weight spaces. Secondly, changes to the architecture affect the parameter count and thus the dimensionality of the space, complicating the comparison of models. In addition, NN weights contain invariances and equivariances that translate to symmetries in weight space. For example, changing the order of two neurons in a layer with all ingoing and outgoing connections does not change the underlying function of the model, but it does change the position in weight space [67]. Similarly, piece-wise linear activation functions allow for linear up and down scaling in subsequent layers which results in unchanged model behavior[39]. Further symmetries can arise if a model has excess capacity [58]. Due to these symmetries, there is a large finite number of equivalent versions of every model in weight space. The number of equivalent versions grows with the factorial of layer width, and can even be infinite for continuous equivalence classes. This property of the NN weight space forms complex local and global structures which complicate working with weights to identify structure and renders notions of neighborhood or distance murky.

Learning Representations of Neural Network Weights To address these challenges, previous work extracts robust features [43, 119, 159], aligns models in weight space [1], uses permutation invariant or equivariant architectures [2, 128, 184, 185], or prioritizes single modes in weight generation [61, 88, 89, 168, 169, 179, 182]. These individual approaches are designed to either a) extract features for model analysis, or to b) generate weights. However, there may be synergies between the two. Generating weights may profit from an understanding of beneficial structures. Likewise, generative capabilities may provide more generalizing features to use for analysis. Similar synergies have been demonstrated on other domains [18, 137, 181] Prior to the work in this thesis, no general representations of NN models existed that are suitable for both model analysis and model generation downstream tasks.

Therefore, work in this thesis fundamentally is concerned with the following question.

Can general, task-agnostic representations be learned from populations of Neural Network models?

This overarching question can be broken down into smaller parts:

(i) Do trained NN models populate a structure in weight space?

(ii) How can diverse populations of neural networks be created?

(iii) What are suitable methods to learn model structures in weight space?

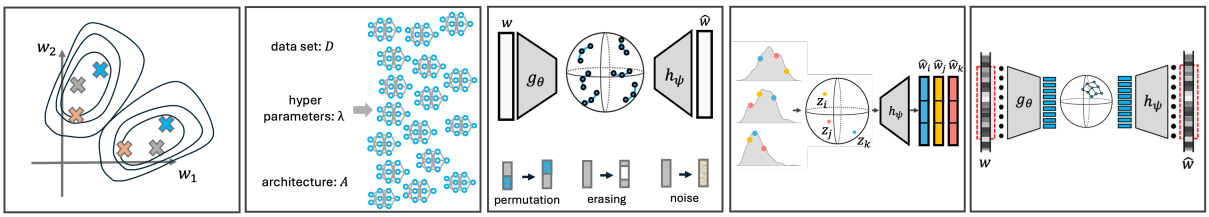
(iv) Are model structures in weight space predictive of model properties, such as model performance or latent generating factors?

(v) Can models be generated by sampling from structures in weight space?

(vi) What are suitable specific downstream tasks to test discriminative (iv) and generative (v) applications?

Contributions of this Thesis

The work collected in this thesis addresses these questions directly as outlined in Figure 1.2. Fundamentally, it establishes that trained NN models populate meaningful structures in weight space. It proposes *hyper-representations* as a self-supervised method to learn these structures from NN weights. Experiments demonstrate that such representations of structure in weight space encode information on latent model properties, such as performance, or hyper-parameters. Further, sampling *hyper-representations* generates model weights that are competitive in fine-tuning and transfer-learning, and generalize to new architectures and tasks. The individual contributions are organized as follows.



(a) Weight spaces (b) Model Zoos (c) Hyper-Representations (d) Model Generation (e) Scaling Hyper-Reps.

Figure 1.2: Overview of the contribution of the thesis. **(a)**: Chapter 2 investigates local and global structure in weight spaces as well the potential and challenges of operations on NN weights. **(b)**: Chapter 3 proposes a blueprint for diverse populations of NNs as a dataset for the work in this thesis. **(c)**: Chapter 4 introduces *hyper-representations* as a self-supervised representation learning method on NN weights, as well as NN weight augmentation methods. **(d)**: Chapter 5 extends *hyper-representations* for model generation. **(e)**: Chapter 6 proposes methods to scale *hyper-representations* to large models and diverse architectures.

Chapter 2 Challenges in Neural Network Weight Spaces

The contributions of **Chapter 2** are centered around establishing properties of the domain of NN weight spaces. It builds on existing work, it describes the complex global and local structure in NN weight spaces. For this work, the primary task is to evaluate the impact of these global and local structures on operations in weight space.

Proposed Methods: The research presented in Chapter 2 is based on the proposition, implementation, and evaluation of the following methods:

1. *Augmentations* of NN models to explore local and global structure in weight spaces.
2. *Similarity of Behavior* to assess similarity in weight space.
3. *Analysis and Generation* to evaluate the robustness of weight operations.

The empirical evaluation of the NN weight spaces validates the notion of structure through training. It demonstrates the existence of local and global structures. Further, it is shown that the operations on weights can become unstable when these complex relations are not considered.

Conclusion: This chapter establishes fundamental properties of the NN weight space domain for this thesis. It shows the potential for analysis and generation of NN weights, but also the need for feature extractors that consider the complex local and global structure of NN weights.

The work in this chapter has originally appeared as Konstantin Schürholt; *Challenges in Neural Network Weight Spaces*; 2024.

Chapter 3: Model Zoos

The contributions of **Chapter 3** are centered around the generation of diverse datasets of trained NN models. In the absence of suitable model populations, this chapter proposes a blueprint for model zoo creation, as well as different diversity metrics. For this work, the primary task is to identify generating factors of model zoos that result in sufficient diverse model populations.

Proposed Methods: The research presented in Chapter 3 is based on the proposition, implementation, and evaluation of the following methods:

1. *Supervised Training* for populations of NN models.
2. *Diversity Metrics* to assess the spread of populations in different spaces.
3. *Application Examples and Baselines* for model zoos.

The empirical evaluation of the model zoo approach demonstrates the ability to control different aspects of diversity in model populations. It is shown that the diversity properties of model zoos generalize to different tasks and architectures. The established baselines in model analysis indicate nontrivial structures in the model zoos.

Conclusion: This chapter contributes the model zoo datasets for all other work in this thesis. It also outlines the potential for applications of model zoos beyond *hyper-representations* and provides a starting point for future research in this domain.

The work in this chapter has originally appeared as Konstantin Schürholt, Diyar Taskiran, Boris Knyazev, Xavier Giró-i-Nieto, Damian Borth; *Model Zoo: A Dataset of Diverse Populations of Neural Network Models*; Conference on Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track, 2022.

Chapter 4: Hyper-Representations

The contributions of **Chapter 4** are centered around method development and application of self-supervised representation learning on weights of trained NNs. Going beyond previous supervised work with hand-crafted feature extractors, this chapter proposes *hyper-representations* with corresponding self-supervised learning task, architecture, and augmentations for NN weights. For this work, the primary task is to establish structure in NN weights and uncover latent generating factors of the NN models.

Proposed Methods: The research presented in Chapter 4 is based on the proposition, implementation, and evaluation of the following methods:

1. *Self-Supervised Learning Task* for representation learning on NN weights.
2. *Data Augmentation* to increase sample efficiency and include inductive biases.
3. *Transformer Architecture* with tokenization scheme for NN Weights.

The empirical evaluation of the implemented hyper-representation approach demonstrates the ability to learn task-agnostic, lower-dimensional representations of NN weights. It is shown that these representations are highly predictive of model properties such as accuracy or hyperparameters, and generalize to new tasks and models. Furthermore, the proposed augmentations for NN weights improve generalization.

Conclusion: This chapter contributes to the core of this thesis by demonstrating the feasibility, utility, and effectiveness of *Representation Learning* on NN weights. It also outlines the potential for applications of hyper-representations and lays the groundwork for future research in this domain.

The work in this chapter has originally appeared as Konstantin Schürholt, Dimche Kostadinov, Damian Borth; *Self-Supervised Representation Learning on Neural Network Weights for Model Characteristic Prediction*; Conference on Neural Information Processing Systems (NeurIPS), 2021.

Chapter 5: Generative Hyper-Representations

The contributions of **Chapter 5** extends *hyper-representations* for generative tasks to allow for NN weight generation. Compared to previous *hyper-representations*, this chapter adjusts the self-supervised learning task to improve reconstruction quality, robustness, and smoothness. Further, it proposes sampling methods to target specific model properties. For this work, the primary task is to identify the distribution of targeted models in latent space.

Proposed Methods: The research presented in Chapter 5 is based on the proposition, implementation, and evaluation of the following methods:

1. *Layer-Wise Normalization* of NN weights to improve reconstruction.
2. *Sampling Schemes* to identify distributions of targeted properties.
3. *Fine-tuning and Transfer Learning* of sampled models.

The empirical evaluation of the generative *hyper-representation* approach demonstrates the ability to improve reconstruction quality. It is shown that sampling methods can target the distributions of specific properties in latent space. Furthermore, the sampled models match or outperform baselines in fine-tuning and transfer learning.

Conclusion: This chapter builds on previous work by demonstrating the feasibility, utility, and effectiveness of generative models on NN weights. It shows that a task-agnostic representation learning method is suitable for analysis and generation tasks.

The work in this chapter has originally appeared as Konstantin Schürholt, Boris Knyazev, Xavier Giró-i-Nieto, Damian Borth; *Hyper-Representations as Generative Models: Sampling Unseen Neural Network Weights*; Conference on Neural Information Processing Systems (NeurIPS), 2022.

Chapter 6: Scalable Hyper-Representations

The contributions of **Chapter 6** proposes methods for scaling *hyper-representations* to much larger models of varying architectures. This work extends previous *hyper-representations* in data pre-processing and representation, representation learning architecture, and sampling methods. For this work, the primary task is to decouple the representation learning model from the architecture and size of the model population by consistent tokenization of models.

Proposed Methods: The research presented in Chapter 6 is based on the proposition, implementation, and evaluation of the following methods:

1. *Representation Learning* on sequences to scale to large and varying architectures.
2. *Model Pre-pocessing* by aligning, standardization and sequentialization.
3. *Unified Models* for discriminative and generative tasks on large models.
4. *Novel Sampling Schemes* to target new architectures and tasks.

The empirical evaluation of the scalable *hyper-representation* approach demonstrates the ability to extend representation learning on NN weights to large models and new architectures. It is shown that significant trends in models are preserved in embedding space. Furthermore, the predictive power as well as model sampling generalizes to large models and unseen architectures.

Conclusion: This chapter builds on previous work by demonstrating the feasibility, utility, and effectiveness of *hyper-representations* on ResNet-18 models and beyond. It shows that a task-agnostic representation learning method is suitable across model sizes for both analysis and generation tasks.

The work in this chapter has originally appeared as Konstantin Schürholt, Michael Mahoney, Damian Borth; *Towards Scalable and Versatile Hyper-Representation Learning*; 2024.

Relevance and Potential Future Impact

Over the past years, *weight space learning* has become a dynamic research topic and has been recognized by the Machine Learning community. The project that this thesis is part of has received a Google Research Scholar Award and a HSG Impact Award. In the context of this thesis, work on populations of models has continued, extending populations towards sparsified models [71] and models on remote sensing data [72]. Recently, work done in another lab extends *hyper-representations* to detect backdoors [95]. In parallel, other groups, too, have begun to work on *weight space learning* similar to the work presented in this thesis. Berardi et al. [7] train auto-encoders on weights of CNN models. Peebles et al. [136] propose a conditional diffusion approach to generate weights. Several approaches have been proposed to encode implicit neural representations or neural radiance field models [2, 3, 33, 128, 180, 184] or on RNNs [68].

Work on *weight space learning* contributes to a deeper understanding of Neural Networks. Understanding and identifying structure in NN weight spaces can help analyze models to select ideal candidates and reveal weaknesses or backdoors in models. It may lead to meaningful notions of provenance for model versioning and intellectual property protection, and a meaningful similarity metric for model governance and certification. The general trend of large foundation models for individual domains can arguably be extended to the domain of NN weights. With the work in Chapter 6, *Hyper-representations* trained on the corpus of publicly available models can become *foundation models of Neural Networks* that incorporate their collective knowledge. Such models could be used to generate initializations for new datasets or tasks, meta-learn in latent space, or manipulate model properties like robustness or sparsity. The future holds the promise of advancements in this area which may change the way NNs are trained and used. I hope that the research provided in this thesis can spark new and exciting work in this direction and provide a foundation in the promising area of *weight space learning*.

Chapter 2

Challenges in Neural Network Weight Spaces

Abstract

The success of Neural Networks (NNs) raises the demand for robustness and analysis of models. Among the available perspectives on trained models, their weights are especially interesting as they contain the information the models have learned. We argue, summarizing previous work, that the weights become structured during training, and that this structure encodes latent information of the models. However, we also identify challenges of operating in NN weight spaces. Their high dimensionality, lack of interoperability, and symmetries form complex local and global relations between weights and model behavior. We perform experiments to evaluate the effect of these challenges and demonstrate that they affect model analysis and generation in weight space so much that it can render results random. These results indicate that operating directly in weight space is inadvisable. Instead, we call for robust methods that consider the properties of NN weight spaces.

2.1 Introduction

In recent years, Neural Networks have become state of the art for complex challenges across a multitude of fields, demonstrating remarkable success in areas such as natural language processing with advancements like GPT-3 [15], computer vision through breakthroughs in image recognition [40, 65] and generative models [28], and reinforcement learning [42, 83, 154].

As the deployment of Neural Networks increases, the need to understand their inner workings intensifies. Achieving a better understanding of these models is necessary for ensuring their reliability, improving their interpretability, and thus enabling trust in their

applications. To this end, there are three principal avenues through which we can gain insights into Neural Network models: (i) examining their behavior, e.g, by computing the prediction error on test data; (ii) understanding their generating factors, e.g., by optimizing the training hyperparameters; and (iii) analyzing the weights as the outcome of their training, e.g., evaluating the distribution of weights to determine overfitting. These perspectives offer a holistic framework for navigating the intricacies of Neural Networks, providing a structured approach to dissecting and comprehending their functionality.

Model Behavior: Exploring model behavior offers direct insights into how a Neural Network interacts with data, revealing its strengths and predictive capabilities across diverse scenarios. Formally, model behavior describes the output $y = f(x)$ of a model f confronted with data x, but can extend to intermediate internal representations of the data within the model. Using model behavior is grounded in empirical evidence, as it assesses the model's outputs against real-world or synthetic data. However, this perspective is inherently limited by the quality and diversity of the data used for evaluation. If the datasets do not fully represent the complexity of real-world applications or omit critical edge cases, the analysis cannot uncover significant model limitations.

Training Hyperparameters: There is abundant work trying to connect the generating factors of models, i.e. their hyperparameters, to model behavior [23, 75]. However, the relationship between hyperparameters, architectural choices, and model performance is complex. The search space is vast and often requires substantial computational resources to navigate effectively. Moreover, the relation between generating factors and the performance of models is highly non-linear, indirect, and incomplete, which makes modeling the relation a challenging task.

Neural Network Weights: The weights of a Neural Network are the outcome of the learning process. As such, they encode information on the generating factors, learned features, and training progress [43, 119, 147, 159]. Despite their informative potential, the weight space of Neural Networks poses significant challenges. Navigating the weight space of Neural Networks presents a difficult challenge due to its inherent high dimensionality, architectural incompatibilities, as well as complex local and global structures within.

In this work, we explore the potential and pitfalls of working directly with weights. We begin by establishing that Neural Network weight spaces become structured during training via the entropy of weight matrices in Section 2.3. Given that weights are structured and contain latent information on models, we then identify specific challenges of weight spaces in Section 2.4. Subsequently, we investigate the impact of these challenges on the relation between similarity in weights and behavior, the usefulness of weights as inputs to predict model properties, and on aggregation of weights in Sections 2.5 and 2.6. Our experiments demonstrate that there are complex local and

global relations between weights and model behavior, which can lead to unintended behavior in downstream tasks. Consequently, we argue against direct operations in the raw weight space. Our results highlight the need for robust methods that consider the global and local structure of weight spaces.

2.2 Related Work

Structure in Neural Network Weights. Different strands of work explicitly or implicitly identify structure in the weights of trained Neural Networks. Fort and Jastrzebski [49] identify wedge-shaped substructures, while Benton et al. [6], Draxler et al. [41], Garipov et al. [51] identify simplexes of connected low-loss regions. Along similar lines, other work identifies local or global connected substructures in weight space with homogeneous properties [167, 174]. Martin and Mahoney [117] investigate the eigenvalue spectrum of weight matrices from a random matrix theory perspective. They describe the evolution from random to heavy-tailed spectra and apply matrix entropy to describe increasing order in weights. Different work generates weights for Neural Network from some latent factors, which implies structure in those weights [61, 88, 89, 136, 148, 163, 182].

Symmetries in Neural Network Weights. Permutation and sign symmetries in Neural Networks have been known for a long time [10, 63]. Later work added more continuous equivariances caused by piece-wise linear activation functions [39, 58] Methods have been proposed to align models in weight space and map Neural Network to a pseudo-canonical form [1]. Some of these symmetries have also been used as data augmentation and as inductive bias [136, 147].

Feature extractors. Several approaches have been proposed to extract features from trained weights. Corneanu et al. [26], Eilertsen et al. [43], Unterthiner et al. [159] use weights, weight-statistics, or derived features to predict properties, the last of which are invariant to permutation symmetries. Martin et al. [119] extract norm-based and eigenvalue-based features from the weight matrices, which are likewise invariant to weight permutations. Another line of work learns representations of weights, which are either equivariant or invariant by architecture [128, 180, 185] or approximately invariant via contrastive learning [147].

2.3 Structure in Neural Network Weights

Using the weights of trained Neural Networks as inputs for downstream tasks implies that there is information in the weights of converged models. In this context, a necessary condition for information is some degree of order in these weights. In this section, we argue that during training, models become structured. Subsequently, we empirically evaluate the order in weights during training using matrix entropy.

Training a Neural Network involves iteratively adjusting its weights, θ, to minimize a loss function $L(y, f_\theta(x))$, where $f_\theta(x)$ denotes the network's output for input x, and y is the true output. This process inherently imposes a structure on the weights θ, reflecting the patterns in the training data.

One way of thinking about the structuring of weights during training is the evolution of weight entropy. Entropy, represented as $H(\Theta)$, quantifies the level of disorder within the weight distribution of a network, where Θ is the distribution of weights. Entropy is defined as $H(\Theta) = -\sum_i P(\theta_i) \log P(\theta_i)$, with $P(\theta_i)$ being the probability of a specific weight configuration θ_i. However, in the context of Neural Networks, directly computing $H(\Theta)$ is impractical since modeling $P(\theta_i)$ is challenging.

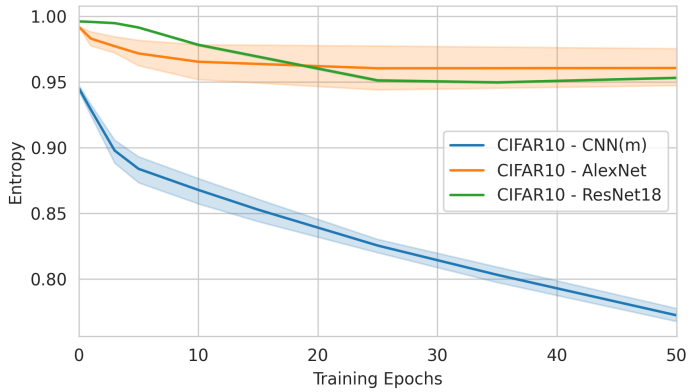


Figure 2.1: Weight entropy over training epochs for populations of CNNs (with $\sim 12k$ parameters), AlexNet and ResNet-18 models trained on CIFAR10. Weight entropy is approximated via the empirical spectral density of the weight matrices. During training, the entropy of weight matrices decreases as order is induced in the weights.

However, the evolution of the eigenvalue spectrum of weight matrices offers another perspective on order in weights. Specifically, analyzing the eigenvalue distribution $\rho(\lambda)$ of a weight matrix offers insights into the network's internal structure. A structured weight matrix will have a characteristic spectral density that differs significantly from that of a random matrix. For instance, a more sharply peaked $\rho(\lambda)$ suggests a higher degree of organization within the weights, acting as a proxy for lower entropy. This

spectral analysis serves as an indirect measure of the weight space's organization, reflecting how training imprints structure on the network's parameters. We follow Martin and Mahoney [117] in computing the matrix entropy $\mathcal{S}$ which we use as a proxy for the model's entropy. To get a strong signal, we considered the largest fully connected layer, which is usually the (pen-) ultimate layer. The empirical evaluation of that approach indeed shows decreasing entropy during training, see Figure 2.1. The effect varies between model sizes. Models with higher capacity relative to the task appear to go through more entropy reduction than smaller models.

The experiments show that indeed the weights of models become more structured during training. By induction, the weights of populations of models are also structured, and shaped by the individual generating factors. That structure can be used to analyze models or generate new weights. However, the weight space also poses challenges, to which we turn in the next section.

2.4 Challenges in Neural Network Weight Spaces

In the previous section, we argue that training imposes structure on Neural Network weights, which contains information on latent generating factors of the models and encodes their properties. However, using that structure by analyzing and manipulating Neural Network weights presents significant challenges. The following paragraphs discuss some of these challenges, the impact of which we experimentally evaluate in the following section.

Weight Space Dimension. The dimensionality and size of Neural Network weight spaces grow with network complexity. To reasonably identify structures in that space, more and more samples are required - a "curse of dimensionality" situation. Beyond the sheer vastness of this space, larger models incur higher computational costs for training and evaluation, exacerbating the challenge. This dual issue of increased dimensionality and computational expense makes optimizing and even comprehensively understanding the weight space increasingly difficult as models scale. Higher dimensional spaces also defeat the intuition of lower dimensional spaces [32]. By the law of large numbers, samples from Gaussian distributions focus almost all of their probability mass on the shell of a high-dimensional sphere. Further, the distance between points drawn from Gaussian distributions is almost all equally large [12]. Since Neural Networks are commonly initialized with Gaussian distributions, these findings carry over to high dimensional weight spaces and challenge the concept of distance between weights.

Architectural Incompatibility. The specificity of weights to their network architectures complicates the direct transfer of learned weights between models. For two architectures, A and B, with their respective weight spaces Θ_A and Θ_B, and a weight vector $\theta_A \in \Theta_A$, attempting to apply θ_A directly to B without addressing architectural disparities usually fails. Even if that is not the case, the mismatch affects the interpretation. Confronted with the same input x, the output $f(x)$ generally varies between parameters θ_A and θ_B: $f_{\theta_A}(x) \neq f_{\theta_B}(x)$. This limitation highlights the challenge of leveraging knowledge across different models, necessitating architecture-invariant strategies for weight analysis and transfer.

Weight Space Symmetries. In addition to their scale, weight spaces exhibit multiple forms of symmetries and invariances, which complicate their interpretation. First, most layers contain permutation invariances: For any layer l, swapping the order of neurons (and their corresponding weights) does not change the network's output [63]. If $\mathbf{P}$ is a permutation matrix, then for weight matrix $\mathbf{W}$, $\mathbf{W}' = \mathbf{PW}$ (or $\mathbf{W}' = \mathbf{WP}^T$) maintains the same function: $f_{\mathbf{W}}(x) = f_{\mathbf{W}'}(x)$. If P is of shape $d_p \times d_p$, there are $d_p!$ unique permutation matrices. That is, the number of distinct symmetric versions of the same function grows with the factorial of a model's width. These multitudes of replications of the loss landscape over weight space make identifying any structure a very hard problem, as they introduce a complex global relation between distance and behavior. Point-symmetric activations like the Sigmoid function introduce additional sign-change symmetries. In networks with piece-wise linear or point-symmetric activations (e.g., ReLU), there are further, continuous invariances. Scaling the weights and biases in one layer can be compensated by inverse scaling in the subsequent layer, leaving the overall network function unchanged [39]. Lastly, in highly complex networks, certain weights or combinations thereof may not significantly contribute to the network's function, leading to redundancy. This excess capacity means that multiple, significantly different weight configurations can produce the same output, obscuring the direct relationship between specific weights and network behavior [58]. These symmetries in weight space create a global structure and cause highly nontrivial relations between distance and behavior.

Sensitivity to Perturbations. Training of Neural Network is an inherently noisy process, where noise can be contributed from the data, the parameter updates, or explicitly as regularization [115, 174]. From a weight space perspective, adding perturbations to weights explores the local relation between weight and behavior. The effect of perturbations on the weight has been studied as the shape of the Neural Network loss landscape [103]. It is characterized by the function $L(\theta)$ and illustrates the model's sensitivity

to weight perturbations. This landscape features regions with sharp minima, where small deviations in weights can lead to substantial increases in loss, indicating high sensitivity to noise. Conversely, flat minima represent regions where the model exhibits greater robustness to changes in weights [39]. The presence of these diverse topological features in the loss landscape underlines the variable impact of noise on model performance and stability, making the identification of structure and relating it to properties a difficult task.

2.5 Experiments

In the previous sections, we argued that the weight space contains latent information on the models, but also complex local and global structure that poses challenges for operations in weight space. Here, we test how these challenges affect applications in weight space in a set of experiments. With these experiments, we test the underlying properties of weight space and evaluate how suitable operations in a local or global scope are. Further, we perform experiments as proxies for two general types of applications in weight space introduced in the introduction: (i) predicting model properties from weights as one way of model analysis, and (ii) aggregating the weights of several pretrained models into one to evaluate generative applications of weights.

Correlating Weight Similarity and Behavioral Similarity. To perform operations in weight space, there is often an implicit or explicit assumption on the relation between changes in weight space and changes in behavior of models, e.g. [76]. In task arithmetic and model souping, changes in weights are expected to translate to proportional changes in behavior [169]. We test this assumption by examining how similarities in model behavior correlate with their similarities in weight space. We measure behavioral similarity using Centered Kernel Alignment (CKA) [91] and weight space similarity through cosine or l_2 similarity metrics. CKA correlates the activations of models at intermediate layers and is permutation invariant, which makes it ideal to compare the behavior of Neural Networks processing the same data. The cosine similarity is computed on the vectorized weights θ_A and $\theta_\mathbf{B}$ as $sim_{cos} = \frac{\theta_A \theta_\mathbf{B}}{\|\theta_A\| \|\theta_\mathbf{B}\|}$. We compute l_2 similarity as $sim_{l_2} = \exp(-\|\theta_A - \theta_B\|_2^2)$.

Predicting Neural Network Accuracy from Weights. As a second experiment, we evaluate the suitability of raw weights as input to infer model properties. Previous work has demonstrated that such weights can be used to predict model properties like test accuracy, generalization gap, or hyperparameters [43, 119, 147, 159]. To evaluate the usefulness and sensitivity of weights for model analysis, we linear probe from weights

for model test-accuracy following previous work [43, 147, 159]. In these experiments, we fit the linear probe to a train set of aligned models and evaluate how performance varies under changes to the test set.

Merging Weights of Pretrained Models. In the last set of experiments, we evaluate the sensitivity of methods that merge weights of models. Re-using weights of pretrained models for continued training is a common strategy. In conventional fine-tuning or transfer learning, the weights of one model are re-used directly [176]. Various factors affect the success of transfer learning, such as the domain overlap, dataset and model size, and complexity [122]. Recently, several methods have attempted to break up the 1-to-1 match between models and combine the weights of several source models into one target model. Among those, there are learned re-combinations of weights such as zoo-tuning [153] or knowledge flow [108], but also much simpler interpolating or averaging of weights [167, 168, 169]. Averaging model weights, often called *model soup*, has been successfully applied to improve model performance, robustness or combine task knowledge [1, 24, 76]. With similar goals in mind, other work learns latent representations of Neural Network weights [148]. Experimental evaluation shows sampling from latent distributions of models generates weights with high zero-shot performance. That raises the question if learning latent representations is necessary, or if similar methods cannot be employed in weight space directly. We therefore experiment with weight averaging (model soup) and weight sampling similar to [148] and evaluate the sensitivity of both methods to variations in weights. Following Schürholt et al. [148], weight sampling models the weight distribution per weight dimension via the Kernel Density Estimation of base models, and then draws samples from the estimated weight distribution.

Base Models and Weight Variations. In all three experiments, we evaluate sets of trained models as a proxy for real-world models. These models are taken from the modelzoo repository [150]. We use models of small and medium CNNs as well as ResNet18 trained on common computer vision datasets. Within architecture and task, the models are varied only in seed, which causes them to have similar performance but different weights. We evaluate how the results of the experiments change under specific changes in the weights of the models. Specifically, we i) change the number of base models, which extends the global coverage of the weight space; ii) align or permute models as proposed by Ainsworth et al. [1] following the methodology of [147], which explores local or global structure of the weight space; iii) add noise to the weights $\tilde{\mathbf{W}} = \mathbf{W} + r * \mathcal{N}(0, 1)$ where r denotes relative noise ratio; adding noise explores the local structure around models; and iv) change in model and task complexity to the impact of weight space dimensionality and its relation to the shape of the loss surface.

2.6 Results

In this section, the results of the three sets of experiments are presented and discussed.

2.6.1 Relation Between Weight Distance and Behavior

In the first set of experiments, we compute pairwise similarities in weights and behavior for sets of trained models. We compute the correlation between weight similarity and behavioral similarity to evaluate how much changes in weights correspond to changes in behavior. We vary the number of permutations, the number of models, and the amount of noise added. Due to the nonlinear interaction of weights and behavior, an increasing number of pairs that mix local and global relationships through more models, permutations, or noise, the correlation between similarities should decrease. Vice-versa, aligning models should simplify global relations and thus increase the correlation between weight and behavior similarity.

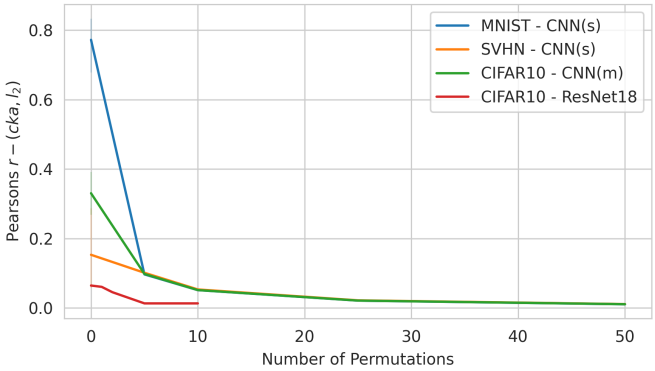


Figure 2.2: Absolute correlation coefficient between pairwise CKA and l_2 similarity scores over number of permutations of 15 models. Increasing the number of permutations increases the number of pairs and the global coverage of the weight space.

Empirical evaluations largely support these expectations. In populations of fully aligned models, we observe absolute correlation coefficients in the range of 0.1 and 0.78 between behavioral similarity and l_2 similarity, see Figure 2.2. This correlation strength diminishes as model and task complexity increase. Introducing more permutations significantly lowers the absolute correlation between behavioral and Euclidean similarities to near zero. These results demonstrate the complex global structures in unaligned weight space.

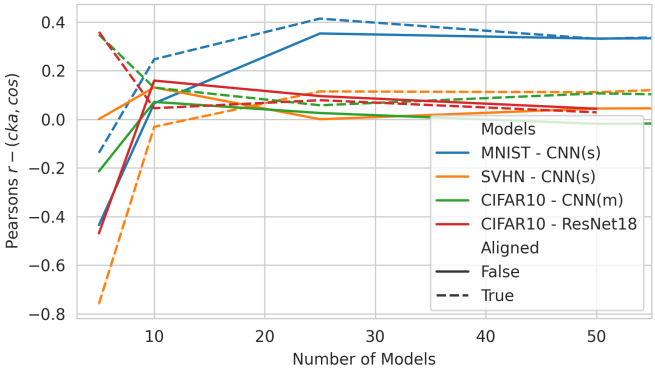


Figure 2.3: Correlation between pairwise CKA and *cos* similarity over the number of models. Increasing the number of models increases the global coverage of the weight space.

Similar patterns are observed with an increasing number of models (Figure 2.3), which likewise affects the global relation between models. Aligning models generally leads to stronger correlations. Local changes by adding noise to the weights have a similar decreasing effect on correlation, but it is less abrupt than increasing permutations (Figure 2.4). These findings highlight the complex global and local relationship between models in weight space and their behavior, emphasizing the need for careful consideration of underlying weight structures and changes. Specifically, proportional relations between weight changes and behavior changes only seem to be supported in local neighborhoods around models, while global relations are far more complex.

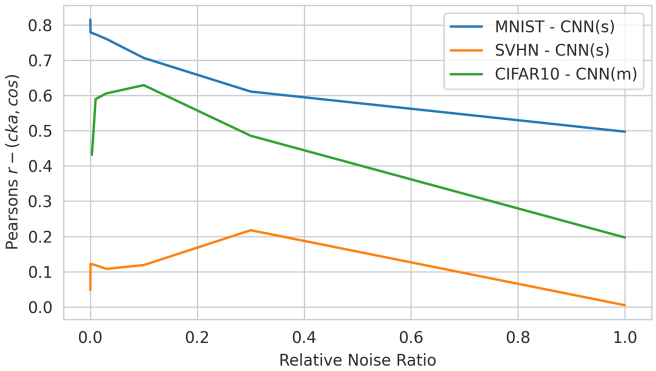


Figure 2.4: Correlation between pairwise CKA and *cos* similarity over relative noise added to model weights. Increasing the noise increases the local coverage of the weight space.

2.6.2 Weights for Model Anlaysis

In the second set of experiments, we predict model accuracy using linear probes from the weights. We fit linear probes to aligned train sets and evaluate on different test sets to test the effects of variations. As a baseline, the linear probe is evaluated on aligned test sets, where linear probes achieve regression-R^2 of above 90%, see Figure 2.5. Harder tasks and larger models reduce the performance. However, if the test sets are not aligned or if linear probes are evaluated on permuted versions of the train set, both of which increase the global coverage of the weight space, the performance is significantly reduced, in many cases to R^2 far below zero. These results again indicate the complex global structure in weight space.

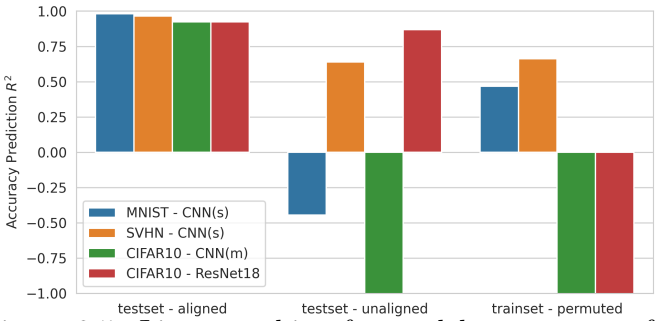


Figure 2.5: Linear probing for model accuracy performance in R^2 for different model and task complexities. The linear probes are fitted to aligned train sets and evaluated on different test sets. Deviations to aligned test sets increase global coverage of the weight space.

Adding noise to models explores the local neighborhood in weight space. Experimental evaluations show that predicting accuracy for such models collapses at a certain threshold for r, see Figure 2.6. While the collapse happens at relatively large levels of relative noise, it decreases with model and task complexity.

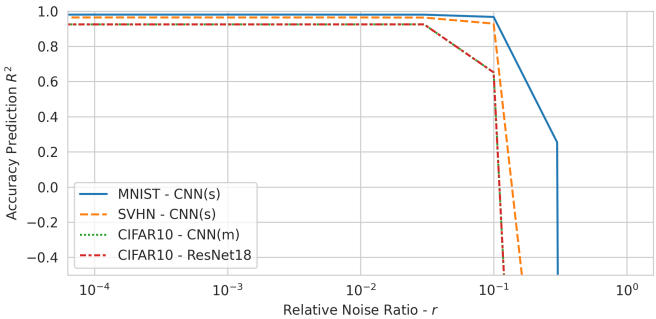


Figure 2.6: Linear probing for model accuracy performance in R^2 for different model and task complexities over relative noise added to model weights. The linear probes are fitted to aligned train sets and evaluated on aligned test sets. Noise increases the local coverage of the weight space.

Fitting predictors on relatively homogeneous, fully aligned populations is not implausible in practice. Such settings can occur if random seeds are shared, or models are fine-tuned from only a few pre-trained models. The experiments show that making predictions on weights based on such populations breaks very quickly for models that are not aligned or explore local regions, which can not always be ruled out. On the other hand, methods that are more robust to changes, such as weight statistics [43, 159], the eigenvalue spectrum [119] or learned representations [147] are also more reliable for downstream predictions of model properties, see Table 2.1.

Table 2.1: R^2 of linear probing for test accuracy using layer-wise weight statistics. Using aligned trainsets and variations of the testset. Weight statistics are invariant to permutations and therefore robust to lack of alignment or random permutations, but are sensitive to noise.

	Testset				
	aligend	not aligned	permuted	noise $r = 0.1$	noise $r = 0.3$
MNIST - CNN(s)	0.989	0.989	0.988	0.972	-0.119
SVHN - CNN(s)	0.988	0.988	0.988	0.955	-2.302
CIFAR10 - CNN(m)	0.970	0.970	0.970	0.546	-9.811
CIFAR10 - ResNet18	0.970	0.970	0.963	0.546	-9.811

2.6.3 Combining Model Weights

In this last set of experiments, we evaluate the sensitivity of weight averaging and weight sampling to variations in weights. Specifically, we evaluate the effects of alignment, permutations, and the number of base models. Since both methods operate in raw weight space to achieve a specific behavior, we expect effects similar to those of the previous experiments. That is, combining models with a clearer signal improves performance; with more models and permutations the signal in weights becomes less clear and performance decreases.

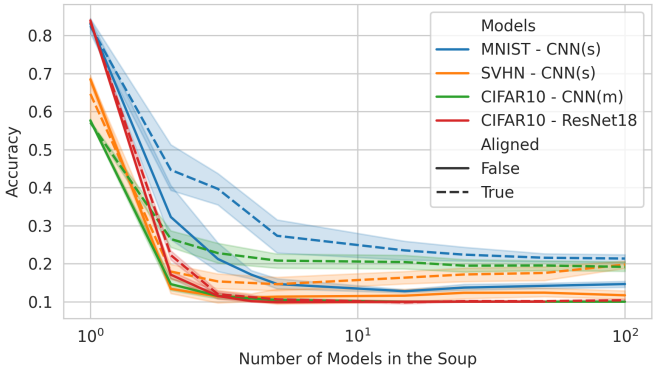


Figure 2.7: Model soup test accuracy over number of averaged models. Increasing the number of models increases the global coverage of the weight space.

Experimental evaluations on model soups with averaged weights largely support those expectations, see Figure 2.7 and 2.10. The performance of individual models is demonstrated as the performance of soups with 1 model. Combining more than one model decreases the performance in all our experiments. Aligning models improves performance over non-aligned source models, adding permutations decreases performance. Similarly, using more source models to combine into a single target model generally hurts performance, more base models seem to make the target weights noisier. Further, performance decreases with task and model complexity. Notably, even averaging aligned models decreases performance over the base population. This indicates that averaging weights of models that are not very close to each other does not generally improve performance.

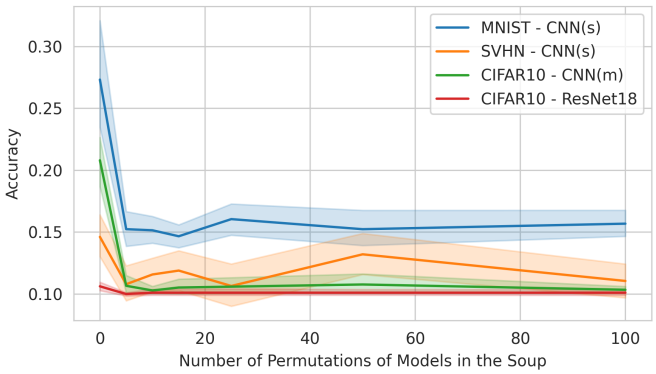


Figure 2.8: Test accuracy of soup of 5 models over number of permutations. Increasing the number of permutations increases the global coverage of the weight space.

Interestingly, experiments sampling in weight space directly shows quite different behavior, see Figures 2.9 and 2.10. On the simpler MNIST and SVHN datasets, performance is higher than that of comparable model soups. On the other hand, with increasing task and model complexity (CIFAR10 CNN or ResNet-18), sampled models default to random guessing. Further, there does not appear to be any significant impact of aligning, permutations, or number of base models. More work is necessary to gain a better understanding of the mechanics, as the sampling method and hyper-parameters may overshadow the source model impact. Also, the overall performance of models sampled in weight space is significantly lower than models sampled in a learned representation space [148], which indicates again that the weight space may not be suitable for such operations, and they instead may benefit from an abstract (learned) feature space.

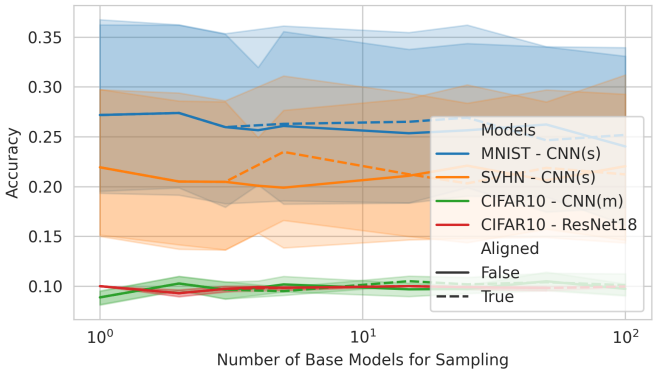


Figure 2.9: Test accuracy for sampled models over number of base models. Increasing the number of base models increases the global coverage of the weight space.

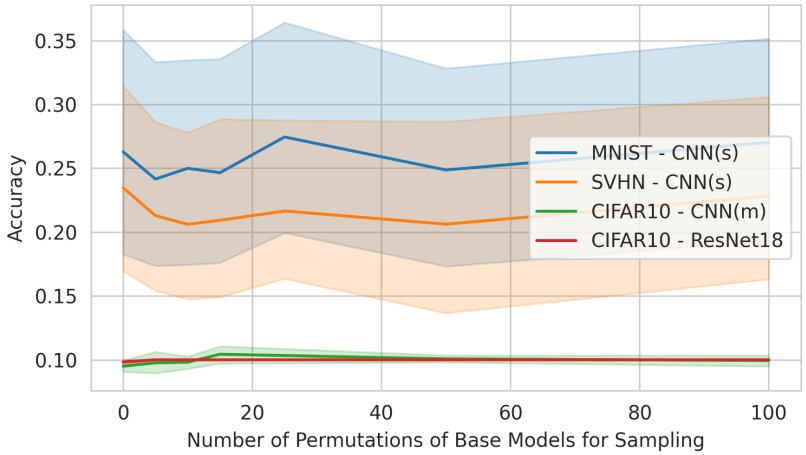


Figure 2.10: Test accuracy for sampled models over number permutations of the base models used to model the weight distribution. Increasing the number of permutations increases the global coverage of the weight space.

2.7 Discussion

The weights of Neural Networks are not only the byproduct of training but can be used for model analysis, control training, or the generation of new models. The training process structures the weights and embeds information of the model, which makes them suitable for model evaluation and the exploration of novel training strategies.

This work confirms that under ideal circumstances, weights are a good input for model analysis and model generation. Yet, we also identify challenges in dealing with Neural Network weights rooted in local and global relations between weights and behavior of models. We observe that even minor deviations from optimal conditions can introduce instability. Effects are noticeable for lack of model alignment, varying model size, and local variations similar to training noise. In real-world scenarios, such deviations cannot be ruled out and can make operations in weight space hard to control.

Our results highlight the need for methods that focus on extracting robust features from weights like weight statistics or learned abstract representations. These features should be resilient to the inherent local and global structure within weight spaces, ensuring stable and reliable outcomes.

Chapter 3

Model Zoos: A Dataset of Diverse Populations of Neural Network Models

Abstract

In the last years, neural networks (NN) have evolved from laboratory environments to the state-of-the-art for many real-world problems. It was shown that NN models (i.e., their weights and biases) evolve on unique trajectories in weight space during training. Following, a population of such neural network models (referred to as *model zoo*) would form structures in weight space. We think that the geometry, curvature and smoothness of these structures contain information about the state of training and can reveal latent properties of individual models. With such *model zoos*, one could investigate novel approaches for (i) model analysis, (ii) discover unknown learning dynamics, (iii) learn rich representations of such populations, or (iv) exploit the *model zoos* for generative modelling of NN weights and biases. Unfortunately, the lack of standardized *model zoos* and available benchmarks significantly increases the friction for further research about populations of NNs. With this work, we publish a novel dataset of *model zoos* containing systematically generated and diverse populations of NN models for further research. In total the proposed model zoo dataset is based on eight image datasets, consists of 27 *model zoos* trained with varying hyperparameter combinations and includes 50'360 unique NN models as well as their sparsified twins, resulting in over 3'844'360 collected model states. Additionally, to the model zoo data we provide an in-depth analysis of the zoos and provide benchmarks for multiple downstream tasks. The dataset can be found at www.modelzoos.cc.

This work originally was accepted for publication at NeurIPS 2022 [150]

3.1 Introduction

The success of Neural Networks (NN) is surprising, considering the hard optimization problem to be solved during training of NNs. Specifically, NN training is NP-complete [11], the loss surface and optimization problem are non-convex [32, 57, 101] and the parameter space to fit during training is high dimensional [15]. Additionally, NN training is sensitive to random initialization and hyperparameter selection [62, 103]. Together, this leads to an interesting characteristic of NN training: given a dataset and an architecture, different random initializations or hyperparameters lead to different minima on the loss surface and therefore result in different model parameters (i.e., weights and biases). Consequently, multiple training results in different NN models. The resulting population of NN (referred to as *model zoo*) is an interesting object to study: Do individual models of the model zoo have something in common? Do they form structures in weight space? What can we infer from such structures? Can we learn representations of them? Lastly, can such structures be exploited to generate new models with controllable properties?

These questions have been partially answered in prior work. Theoretical and empirical work demonstrates increasingly well-behaved loss surfaces for growing number of parameters [31, 57, 103]. The shape of the loss surface and the starting point is determined by hyperparameters and the initialization, respectively [103]. NN training navigates the loss surface with iterative, gradient-based update schemes smoothed by momentum. The step length along a trajectory as well as the curvature are determined by the change of the loss as well as how aligned the subsequent updates are [17, 146]. Together, these findings suggest that populations of NN models evolve on unique and smooth trajectories in weight space. Related work has empirically confirmed the existence of such structures in NNs [35], demonstrated the feasibility to learn representations of them, showed that they encode information on model properties [43, 147, 159] and can be used to generate unseen models with desirable properties [88, 148, 149, 182] To thoroughly answer the questions above, a large and systematically created dataset of model weights is necessary.

Unfortunately, so far only few model zoos with specific properties have been published [43, 147, 156, 159]. While many machine learning domains have standardized datasets, there is no model zoo nor a benchmark to evaluate and compare against. The lack of a standardized model zoos has three significant disadvantages: (i), existing model zoos are usually designed for a specific purpose and of limited general utility. Their design space is rather sparse, covering only small portions of all available hyperparameter combinations. Moreover, some existing zoos are generated on synthetic tasks

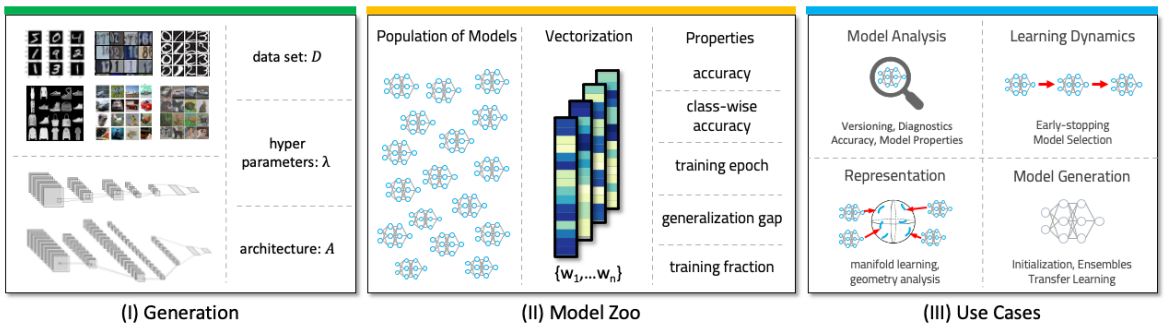


Figure 3.1: The proposed dataset of model zoos is trained on several image datasets with two CNN architectures and multiple configurations of hyperparameters. The resulting population of neural network models is vectorized and made available with all meta-data such as the generating factors of the model zoo as well as the model properties such as accuracy, generalization gap, and others. Potential use cases are (a) model property prediction, (b) inference of learning dynamics, (c) representation learning, or (d) model generation.

and are small, containing only a small population of models; (ii), researchers have to choose between using an existing zoo or generating a new one for each new experiment, weighing the disadvantages of existing zoos against the effort and computational resources required to generate a new zoo; (iii), a new model zoo causes subsequent work to lose comparability to existing research. Therefore, the lack of a benchmark model zoo significantly increases the friction for new research.

Our contributions: To study the behaviour of populations of NNs, we publish a large-scale model zoo of diverse populations of neural network models with controlled generating factors of model training. Special care has been taken in their design and the used protocols for training. To do so, we have defined and restricted the generating factors of model zoo training to achieve desired zoo characteristics.

The zoos are trained on eight standard image classification datasets, with a broad range of hyperparameters and contain thousands of configurations. Further, we add sparsified *model zoo twins* to each of these zoos. Alltogether, the zoos include a total of 50'360 unique image classification NNs, resulting in over 3'844'360 collected model states.

Potential use cases for the model zoo include (a) model analysis for reliability, bias, fairness, or adversarial vulnerability, (b) inference of learning dynamics for efficiency gain, model selection or early stopping, (c) representation learning of such populations, or (d) model generation. Additionally, we present an analysis of the model zoos and a set of experimental setups for benchmarks on these use cases and initial results as foundation for evaluation and comparison.

With this work we provide a standardized dataset of diverse model zoos connected to popular image datasets, its corresponding meta-data and performance evaluations to the machine learning research community. All data is made publicly available to foster community building around the topic and to provide a ground for use beyond the defined benchmark tasks. An overview of the proposed dataset and benchmark as well as potential use cases can be found in Fig. 3.1

3.2 Existing Populations of Neural Networks Models

With the increase in usage of neural networks, requirements for evaluation, testing and certification have grown. Methods to analyze NN models may attempt to visualize salient features for a given class [84, 177, 178], investigate the robustness of models to specific types of noise [27, 189], predict model properties from model features [26, 80, 172] or compare models based on their activations [126, 132, 139] However, while most of these methods rely on common (image) datasets to train and evaluate their models, there is no common dataset of neural network models to compare the evaluation methods on. Model zoos as common evaluation datasets can be a step up to evaluate the evaluation methods.

There are only few publications who use model zoos. In [108], zoos of pre-trained models are used as teacher models to train a target model. Similarly, [153] propose a method to learn a combination of the weights of models from a zoo for a new task. [186] uses a zoo of GAN models trained with different methods to accelerate GAN training. To facilitate continual learning, [140] propose to generate zoos of models trained on different tasks or experiences, and to ensemble them for future tasks.

Larger model zoos containing a few thousand models are used in [159] to predict the accuracy of the models from their weights. Similarly, [43] use zoos of larger models to predict hyperparameters from the weights. In [52], a large collection of 3x3 convolutional filters trained on different datasets is presented and analysed. Other work identifies structures in the form of subspaces with beneficial properties [6, 111, 167]. [147] use zoos to learn self-supervised representations on the weights of the models in the zoo. The authors demonstrate that the learned representations have high predictive capabilities for model properties such as accuracy, generalization gap, epoch and various hyperparameters. Further, they investigate the impact of the generating factors of model zoos on their properties. [148, 149] demonstrate that learned representations can be instantiated in new models, as initialization for fine-tuning or transfer learning. This work systematically extends their zoos to more datasets and architectures.

3.3 Model Zoo Generation

The proposed model zoo datasets contain systematically generated and diverse populations of neural networks. Since the applicability of the model zoos for downstream tasks largely depends on the composition and properties of the zoos, special care has to be taken in their design and the used protocol for training. The entire procedure can be considered as defining and restricting the generating factors of model zoo training with respect to their latent relation of desired zoo characteristics. The described procedure and protocol could be also used as a general blueprint for the generation of model zoos.

In our paper, the term architecture means the structure of an NN, i.e., a set of operations and their connectivity. We use 'model' to denote an instantiating of an architecture with weights over all stages of training, 'model state' to denote the model with the specific state of weights at a specific training epoch, and the weights $\mathbf{w}$ to denote all trainable parameters (weights and biases).

3.3.1 Model Zoo Design

Generating Factors Following [159], we define the tuple $\{\mathcal{D}, \lambda, \mathcal{A}\}$ as a configuration of a model zoo's generating factors. We denote the dataset of image samples with their corresponding labels as $\mathcal{D}$. The NN architecture is denoted by $\mathcal{A}$. We denote the set of hyperparameters used for training, (*e.g.*, loss function, optimizer, learning rate, weight initialization, seed, batch-size, epochs) as λ. While dataset $\mathcal{D}$ and architecture $\mathcal{A}$ are fixed for a model zoo, λ provides not only the set of hyperparameters but also configures the ranges for individual hyperparameter such as learning rate for model zoo generation. Training with such differing configurations $\{\mathcal{D}, \lambda, \mathcal{A}\}$ results in a population of NN models i.e., the model zoo. We convert the weights and biases of each model to a vectorized form. In the resulting model zoo $\mathcal{W} = \{\mathbf{w}_1, ..., \mathbf{w}_M\}$, $\mathbf{w}_i$ denotes the flattened vector of the weights and biases of one trained NN model from the set of M models of the zoo.

Configurations & Diversity The model zoos have to be representative of real-world models, but also diverse and span an interesting range of properties. The definition of the diversity of model zoos, as well as the choice of how much diversity to include, is as difficult as in image datasets, e.g. [34, 45]. Model zoos can be diverse in their properties (i.e., performance) as well as in their generating factors λ, or in their weights $\mathbf{w}$. We aim to generate model zoos with a rich set of models and diversity in these aspects. As these zoo properties are effects of the generating factors, we tune the diversity of the generating factors and evaluate the diversity in Section 3.4.

Table 3.1: Generating factors of the model zoos. Several values for each parameter define the grid. `Arch` denotes the architecture: CNN (s) - small CNN architecture, CNN (m) - medium CNN architecture, RN-18 - ResNet-18. `Init` denotes the initialization methods: U - uniform, N - normal, KU - Kaiming Uniform, KN - Kaiming Normal. `Activation` denotes the activation function: T - Tanh, S - Sigmoid, R - ReLU, G - GeLU. `Optim` denotes the optimizer: AD - Adam, SGD - Stochastic Gradient Descent. Models with learning rates denoted with * have been trained with a one-cycle LR scheduler, the listed LR is the maximum value.

Dataset	Arch	Config	`Init`	`Activation`	`Otpim`	`LR`	`WD`	`Dropout`	`Seed`
	CNN (s)	Seed	U	T	AD	3e-4	0	0	1-1000
MNIST	CNN (s)	Hyp-10-r	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-3, 1e-4	0, 0.5	~ 10
	CNN (s)	Hyp-10-f	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-3, 1e-4	0, 0.5	1-10
	CNN (s)	Seed	U	T	AD	3e-4	0	0	1-1000
F-MNIST	CNN (s)	Hyp-10-r	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-3, 1e-4	0, 0.5	~ 10
	CNN (s)	Hyp-10-f	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-3, 1e-4	0, 0.5	1-10
	CNN (s)	Seed	U	T	AD	3e-3	0	0	1-1000
SVHN	CNN (s)	Hyp-10-r	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-3, 1e-4, 0	0, 0.3, 0.5	~ 10
	CNN (s)	Hyp-10-f	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-3, 1e-4, 0	0, 0.3, 0.5	1-10
	CNN (s)	Seed	U	T	AD	3e-4	1e-3	0	1-1000
USPS	CNN (s)	Hyp-10-r	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-2, 1e-3	0, 0.5	~ 10
	CNN (s)	Hyp-10-f	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-2, 1e-3	0, 0.5	1-10
	CNN (s)	Seed	KU	G	AD	1e-4	1e-2	0	1-1000
CIFAR10	CNN (s)	Hyp-10-r	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3	1e-2, 1e-3	0, 0.5	~ 10
	CNN (s)	Hyp-10-f	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3	1e-2, 1e-3	0, 0.5	1-10
	CNN (m)	Seed	KU	G	AD	1e-4	1e-2	0	1-1000
CIFAR10	CNN (m)	Hyp-10-r	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3	1e-2, 1e-3	0, 0.5	~ 10
	CNN (m)	Hyp-10-f	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3	1e-2, 1e-3	0, 0.5	1-10
	CNN (s)	Seed	KU	T	AD	1e-4	1e-3	0	1-1000
STL (s)	CNN (s)	Hyp-10-r	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-2, 1e-3	0, 0.5	~ 10
	CNN (s)	Hyp-10-f	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-2, 1e-3	0, 0.5	1-10
	CNN (m)	Seed	KU	T	AD	1e-4	1e-3	0	1-1000
STL	CNN (m)	Hyp-10-r	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-2, 1e-3	0, 0.5	~ 10
	CNN (m)	Hyp-10-f	U, N, KU, KN	T, S, R, G	AD, SGD	1e-3,1e-4	1e-2, 1e-3	0, 0.5	1-10
CIFAR10	RN-18	Seed	KU	R	SGD	1e-4*	5e-4	0	1-1000
CIFAR100	RN-18	Seed	KU	R	SGD	1e-4*	5e-4	0	1-1000
t-Imagenet	RN-18	Seed	KU	R	SGD	1e-4*	5e-4	0	1-1000

Prior work discusses the impact of random seeds on the properties of model zoos. While [172] use multiple random seeds for the same hyperparameter configuration, [159] explicitly argues against that to prevent information leakage between models from train to test set. To achieve diverse model zoos and disentangle the generating factors (seeds and hyperparameters), we train model zoos in three different configurations, some with random seeds, others with fixed seeds.

`Random Seeds:` The first configuration, denoted as `Hyp-10-rand`, varies a broad range of hyperparameters to define a grid of hyperparameters. To include the effect of different random initializations, each of the hyperparameter nodes in the grid is repeated with ten randomly drawn seeds. One model is configured with the combination

of hyperparameters and seed, with a total of ten models per hyperparameter node. It is very unlikely for two models in the zoo share the same random seed. With this, we achieve the highest amount of diversity in properties, generating factors and weights.

`Fixed Seeds:` The second configuration, denoted as `Hyp-10-fix`, uses the same hyperparameter grid as but repeats each node with ten fixed seeds $[1, 2, ..., 10]$. Fixing the seeds allows evaluation methods to control for the seed, isolate the influence of hyperparameter choices, and still get robust results over 10 repetitions. A side effect of the (desired) isolation of factors of influence is, that fixing the seeds leads to repetitions of the starting point in weight space for models with the same seed and initialization methods. At the beginning of the training, these models may have similar trajectories.

`Fixed Hyperparameters:` For the third configuration, denoted as `Seed`, we fix one set of hyperparameters and repeat that with 1000 different seeds. With that, we achieve zoos that are very diverse in weights and covers a broad range in weight space. These zoos can be used to evaluate the impact of weights and their starting point on model performance. The hyperparameters for the `Seed` zoos are chosen such that there is still a level of diversity in model performance.

3.3.2 Specification of Generating Factors for Model Zoos

This section describes the systematic specification of the trained model zoos. Multiple generating factors define a configuration $\{\mathcal{D}, \lambda, \mathcal{A}\}$ for the model zoo generation, detailed in Table 3.1.

Datasets $\mathcal{D}$: We generate model zoos for the following image classification datasets: `MNIST` [98], `Fashion-MNIST` [171], `SVHN` [129], `CIFAR-10` [92], `STL-10` [25], `USPS` [74], `CIFAR-100` [92] and `Tiny Imagenet` [96].

Hyperparameter λ: varied hyperparameters to train models in zoos are: (1) `seed`, (2) `initialization method`, (3) `activation function`, (4) `dropout`, (4) `optimization algorithm`, (5) `learning rate`, and (6) `weight decay`. The batch size and number of training epochs are kept constant within zoos.

Architecture $\mathcal{A}$: To preserve the comparability within a model zoo, each zoo is generated using a single neural network architecture. One of three standard architectures is used to generate each zoo. Our intention with this dataset is similar to research

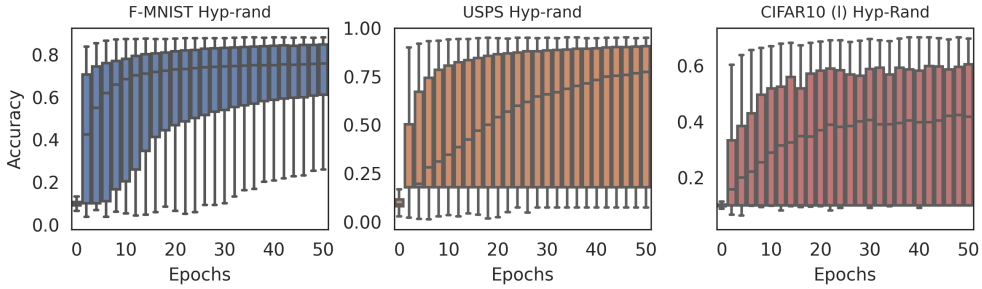


Figure 3.2: Accuracy distribution over epochs for the `F-MNIST Hyp-rand`, `USPS Hyp-rand`, and `CIFAR Hyp-rand` zoos. All zoos show training progress and considerable performance diversity.

communities such as Neural Architecture Search (NAS), Meta-Learning or Continual Learning (CL), where initial work started small scale [140, 182]. Hence, the first two architectures are a small and a slightly larger Convolutional Neural Network (CNN), both have three convolutional and two fully-connected layers, but different numbers of channels (details in Appendix 3.A). The third architecture is a standard ResNet-18 [65]. The (1) `small CNN` has a total of 2′464-2′864 parameters, the (2) `medium CNN` has 10′853 parameters, the (3) ResNet-18 has 11.2M-11.3M parameters.

Compared to (1), the medium architecture (2) provides additional diversity to the collection of model zoos and performs significantly better on more complex datasets `CIFAR-10` and `STL-10`. These architectures are similar to the one used in [148]. The ResNet-18 architecture is included to apply the model zoo blueprint to models of the widely used ResNet family and so facilitate research on populations of real-world-sized models.

3.3.3 Training of Model Zoos

Neural network models are trained from the previously defined three configurations $\{\mathcal{D}, \lambda, \mathcal{A}\}$ (Seed, Hyp-10-rand, Hyp-10-fix, see Sec 2.1). With the *8 image datasets* and the three configurations, this results in *27 model zoos*. The zoos include a total of around *50'360* unique neural network models.

Training Protocol: Every model in the collection of zoos is trained according to the same protocol. We keep the same train, validation, and test splits for each zoo, and train each model for 50 epochs with gradient descent methods (SGD+momentum or ADAM). At every epoch, the model checkpoint as well as accuracy and loss of all splits are recorded. Validation and test performance are also recorded before the first training epoch. This makes 51 checkpoints per model training trajectory including the starting checkpoint representing the model initialization before training starts. The ResNet-18

zoos on CIFAR100 and Tiny Imagenet require more updates and are trained for 60 epochs. In total, this results in a set of *2'585'360* collected model states.

Splits: To enable comparability, this set of models is split into `training` (70%), `validation` (15%), and `test` (15%) subsets. This split is done such that all individual checkpoints of one model training (i.e., the 51 checkpoints per training) is entirely in either `training`, `validation`, or `test` and therefore no information is leaked between these subsets.

Sparsified Model Zoo Twins: Model sparsification is an effective method to reduce the computational cost of models. However, methods to sparsify models to a high degree while preserving the performance are still actively researched [70]. In order to allow systematic studies of sparsification, we are extending the model zoos with sparsified *model zoo twins* serving as counterparts to existing zoos in the dataset. Using Variational Dropout (VD) [125], we sparsify the trained models from existing model zoos. VD generates a sparsification trajectory for each model, along which we track the performance, degree of sparsity, and the sparsified checkpoint. With 25 sparsification epochs, this yields 1'259'000 sparsification model states.

3.3.4 Data Management and Accessibility of Model Zoos

The model zoos are made publicly available in an accessible, standardized, and well-documented way to the research community under the Creative Commons Attribution 4.0 license (CC-BY 4.0). We ensure the technical accessibility of the data by hosting it on Zenodo, where the data will be hosted for at least 20 years. Further, we take steps to reduce access barriers by providing code for data loading and preprocessing, to reduce the friction associated with analyzing the raw zoo files. All code can be found on the model zoo website www.modelzoos.cc. To ensure conceptional accessibility, we include detailed insights, visualizations, and the analysis of the model zoo (Sec. 3.4) with each zoo. Further details can be found in Appendix 3.B.

3.4 Model Zoo Analysis

The model zoos have been created aiming at diversity in generating factors, weights, and performance. In this section, we analyze the zoos and their properties. Zoo cards with key values and visualizations are provided along with the zoos online. We consider models at their last epoch for the analysis. For all later analyses, non-viable checkpoints

Table 3.2: Analysis of the diversity of our 27 model zoos (one row per zoo). Mean (std) values in % per zoo, computed on the last epoch. Agreement is computed using samples from the test split of the image dataset pairwise over the entire zoo. Higher agreement values indicate more uniform behavior and less behavioral diversity. Distance in weight space are computed pairwise over the entire zoo. Higher distance values indicate larger diversity in weight space.

			Performance	Agreement		Weights		
Dataset	Architecture	Config	Accuracy	κ_{aggr}	κ_{cka}	$\mathbf{w}$	l2-dist	cos dist
MNIST	CNN (s)	Seed	91.1 (0.9)	88.5 (1.3)	77.2 (5.2)	18.9 (58.4)	124.1 (4.9)	77.1 (4.1)
	CNN (s)	Hyp-10-r	79.9 (30.7)	67.7 (35.5)	58.6 (25.9)	0.4 (46.5)	150.6 (66.5)	98.8 (7.2)
	CNN (s)	Hyp-10-f	80.3 (30.3)	68.3 (35.3)	58.8 (25.7)	0.3 (46.7)	149.7 (66.8)	97.7 (10.0)
F-MNIST	CNN (s)	Seed	72.7 (1.0)	79.8 (2.6)	82.3 (12.6)	22.6 (55.6)	122.0 (4.9)	74.5 (4.4)
	CNN (s)	Hyp-10-r	68.4 (23.7)	59.9 (29.1)	64.6 (23.5)	1.0 (46.0)	149.6 (62.2)	99.2 (6.8)
	CNN (s)	Hyp-10-f	68.7 (23.4)	60.4 (28.7)	64.6 (22.7)	0.9 (46.3)	148.5 (61.9)	97.9 (9.9)
SVHN	CNN (s)	Seed	71.1 (8.0)	67.2 (10.3)	67.7 (15.7)	7.1 (113.7)	137.6 (8.3)	94.5 (5.1)
	CNN (s)	Hyp-10-r	35.9 (24.3)	61.6 (35.9)	17.8 (28.0)	1.4 (42.2)	170.5 (149.4)	83.6 (30.4)
	CNN (s)	Hyp-10-f	36.0 (24.4)	61.4 (36.0)	18.1 (27.9)	1.3 (42.2)	170.0 (149.0)	83.2 (30.7)
USPS	CNN (s)	Seed	87.0 (1.7)	87.3 (2.2)	86.7 (6.3)	8.2 (26.9)	123.1 (5.2)	75.9 (5.0)
	CNN (s)	Hyp-10-r	64.7 (30.8)	55.3 (31.4)	50.9 (30.5)	2.1 (39.6)	155.5 (92.6)	99.1 (8.9)
	CNN (s)	Hyp-10-f	65.0 (30.7)	55.4 (31.3)	50.4 (30.4)	1.9 (40.1)	154.2 (93.1)	97.3 (13.7)
CIFAR10	CNN (s)	Seed	48.7 (1.4)	65.7 (3.1)	72.9 (11.3)	1.1 (11.0)	138.7 (5.6)	96.3 (5.1)
	CNN (s)	Hyp-10-r	35.1 (16.3)	33.3 (22.9)	47.5 (34.0)	-0.2 (17.0)	155.6 (71.0)	97.5 (10.8)
	CNN (s)	Hyp-10-f	35.1 (16.2)	33.3 (22.8)	47.3 (34.2)	-0.2 (16.9)	155.3 (70.0)	97.2 (11.1)
CIFAR10	CNN (m)	Seed	61.5 (0.7)	76.0 (1.6)	92.4 (1.7)	0.1 (18.2)	137.0 (7.9)	94.1 (9.2)
	CNN (m)	Hyp-10-r	39.6 (21.8)	34.5 (27.1)	43.2 (36.5)	-0.4 (23.0)	158.9 (79.9)	98.6 (12.2)
	CNN (m)	Hyp-10-f	39.6 (21.7)	34.4 (26.7)	42.8 (37.8)	-0.4 (22.9)	158.1 (77.2)	98.0 (13.1)
STL	CNN (s)	Seed	39.0 (1.0)	48.4 (3.0)	81.5 (3.9)	-0.1 (19.1)	141.2 (5.0)	99.8 (4.2)
	CNN (s)	Hyp-10-r	23.1 (12.3)	23.4 (20.9)	39.0 (30.7)	3.0 (40.0)	158.7 (107.3)	98.7 (10.9)
	CNN (s)	Hyp-10-f	23.0 (12.2)	23.3 (21.1)	38.1 (30.0)	3.0 (39.8)	157.1 (107.2)	96.8 (16.3)
STL	CNN (m)	Seed	47.4 (0.9)	53.9 (2.2)	83.3 (2.3)	0.1 (26.6)	141.3 (6.0)	99.9 (5.8)
	CNN (m)	Hyp-10-r	24.3 (14.7)	23.2 (24.2)	34.1 (30.0)	2.3 (45.7)	159.3 (103.0)	99.1 (12.5)
	CNN (m)	Hyp-10-f	24.4 (14.7)	23.7 (24.5)	34.6 (30.3)	2.3 (46.5)	157.4 (104.1)	97.6 (20.1)
CIFAR10	ResNet-18	Seed	92.1 (0.2)	93.4 (0.7)	–.- (-.-)	-0.01 (1.7)	122.1 (3.9)	72.2 (2.3)
CIFAR100	ResNet-18	Seed	74.2 (0.3)	77.6 (1.2)	–.- (-.-)	-0.1 (1.6)	130.8 (4.1)	83.1 (2.6)
Tiny ImageNet	ResNet-18	Seed	63.9 (0.7)	66.1 (1.9)	–.- (-.-)	-0.1 (1.9)	125.4 (4.9)	77.1 (3.0)

are excluded from each zoo. This includes the removal of every checkpoint with NaN values or values beyond a threshold. The threshold value is set for each zoo, such that it only excludes diverging models.

Performance To investigate the performance diversity, we consider the accuracy of the models in the zoo, see Table 3.2 and Figure 3.2. As expected, the zoos with variation only in the seed show the smallest variation in performance. Changing the hyperparameters induces a broader range of variation. Changing (Hyper-10-rand) or fixing (Hyper-10-fix) the seeds does not affect the accuracy distribution.

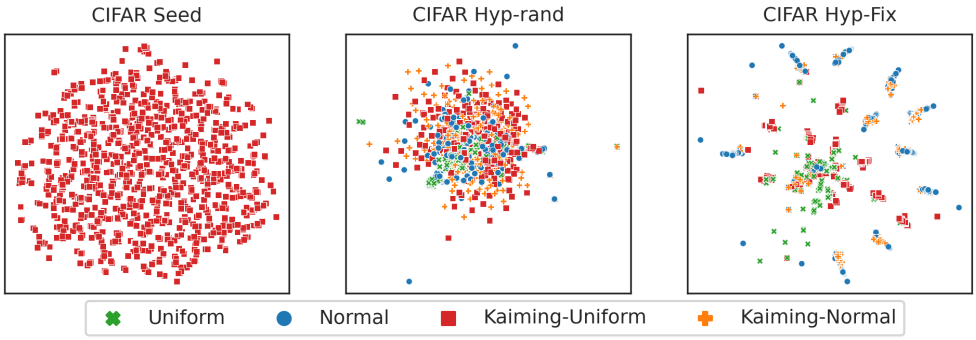


Figure 3.3: Visualization of the weights of the large CIFAR model zoos in different configurations. The weights are reduced to 2d using UMAP, preserving both local and global structure. In the `Seed` configuration, the UMAP reduction contains little structure. The `Hyp-rand` is equally little structured. In contrast, `Hyp-fix` contains visible clusters of initialization methods.

Model Agreement To get more in-depth insights into the diversity of model behavior, we investigate their pairwise agreement, see Table 3.2. To that end, we compute the rate of agreement of class prediction between two models as $\kappa_{aggr} = \frac{1}{N}\sum_{1=1}^{N}\delta_{y_i}$. Here y_i^k, y_i^l are the predictions of models k, l for sample i of N samples. Further, $\delta_{y_i} = 1$ if $y_i^k = y_i^l$ and otherwise $\delta_{y_i} = 0$. Further, we compute the pairwise centered kernel alignment (cka) score between intermediate and last layer outputs and denote it as κ_{cka}. The cka score evaluates the correlation of activations, compensating for equivariances typical for neural networks [132]. In empirical evaluations, we found the cka score robust for a relatively small number of image samples, and compute the score using 50 images to reduce the computational load. Both agreement metrics confirm the expectation and performance results. Zoos with higher overall performance naturally have a higher agreement on average, as there are fewer samples on which to disagree. Zoos with varying hyperparameters(Hyp-10-rand and Hyp-10-fix) agree less on average than zoos with changes in seed only (Seed). What is more, the distribution of κ_{aggr} and κ_{cka} in the `Seed` zoos is unimodal and approximately Gaussian. In the `Hyp-10` zoos, the distributions are bi-modal, with one mode around 0.1 (0.0) and the other around 0.9 (0.75) in hard agreement (cka score). In these zoos, models agree to a rather high degree with some models and disagree with others.

Weights Lastly, we investigate the diversity of the model zoos in weight space, see again Table 3.2. By design, the mean weight value of the zoos varying only in the seed is larger than in the other zoos, while the standard deviation does not differ greatly (Table 3.2, column w). To get a better intuition in the distribution of models in

weight space, we compute the pairwise $\ell_2(\mathbf{w_k}, \mathbf{w_l}) = \frac{\|\mathbf{w_k}-\mathbf{w_l}\|_2^2}{1/N\sum_{n=1}^{N}\|\mathbf{w_n}\|_2^2}$ and cosine distance $cos(\mathbf{w_k}, \mathbf{w_l}) = 1 - \frac{\mathbf{w_l}^T\mathbf{w_k}}{\|\mathbf{w_k}\|_2^2\|\mathbf{w_l}\|_2^2}$, and investigate their distribution. Here, too, varying the hyperparameters introduces higher amounts of diversity, while changing or fixing the seeds does not affect the weight diversity much. As these values are computed at the end of model training, repeated starting points due to fixed seeds appear not to reduce weight diversity significantly. In a more hands-off approach, we compute 2d reductions of the weight over all epochs using UMAP [120]. In the 2d reductions (see Figure 3.3), the zoos varying in seed only show little to no structure. Zoos with hyperparameter changes and random seeds are similarly unstructured. Zoos with varying hyperparameters and fixed seeds show clear clusters with models of the same initialization method and activation function. These findings are further supported by the predictability of the initialization method and activation function (Table 3.3). The structures are unsurprising considering that the activation function is very influential in shaping the loss surface, while the initialization method and the seed determine the starting point on it. Depending on the downstream task, this property can be desirable or should be avoided, which is why we provide both configurations.

Model Property Prediction As a set of benchmark results on the proposed model zoos and to further evaluate the zoos, we use linear models to predict hyperparameters or performance values of the individual models. As features, we use the model weights $\mathbf{w}$ or per-layer quintiles of the weights $s(\mathbf{w})$ as in [159]. Linear models are used to evaluate the properties of the dataset and the quality of the features. We report these results in Table 3.3. The layer-wise weight statistics ($s(\mathbf{w})$) have generally higher predictive performance than the raw weights $\mathbf{w}$. In particular, $s(\mathbf{w})$ are not affected by using fixed or random seeds and thus generalize well to unseen seeds. For the ResNet-18 zoos, $\mathbf{w}$ becomes too large to be used as a feature and is therefore omitted. Across all zoos, the accuracy as well as the hyperparameters can be predicted very accurately. The generalization gap and epoch appear to be more difficult to predict. These findings hold for all zoos, regardless of the different architectures, model sizes, task complexity, and performance range. $\mathbf{w}$ can be used to predict the initialization method and activation function to very high accuracy, if the seeds are fixed. The performance drops drastically if seeds are varied. These results confirm our expectation of diversity in weight space induced by fixing or varying seed. These results show i) that the model weights of our zoos contain rich information on their properties; ii) confirm the notions of diversity that were design goals for the zoos; and iii) leave room for improvements on the more difficult properties to predict, in particular the generalization gap.

Table 3.3: Benchmark results for predicting model properties from the weights ($\mathbf{w}$) and layer-wise weight statistics ($s(\mathbf{w})$) using linear models. We report the prediction R^2 for accuracy, generalization gap (GGap), epoch, learning rate (LR) and dropout (Drop), and prediction accuracy for initialization method (Init) and activation function (Act). Values reported in %, higher values are better.

			Accuracy		GGap		Epoch		Init		Act	
Dataset	Architecture	Config	$\mathbf{w}$	$s(\mathbf{w})$	$\mathbf{w}$	$s(\mathbf{w})$	$\mathbf{w}$	$s(\mathbf{w})$	$\mathbf{w}$	$s(\mathbf{w})$	$\mathbf{w}$	$s(\mathbf{w})$
MNIST	CNN (s)	Seed	92.3	98.7	2.1	68.8	87.2	97.8	n/a	n/a	n/a	n/a
	CNN (s)	Hyp-10-r	-11.2	69.4	-49.8	13.7	-95.5	14.3	42.6	77.6	45.5	78.5
	CNN (s)	Hyp-10-f	66.5	70.1	5.4	12.5	-4.8	14.5	94.3	79.8	81.2	76.8
F-MNIST	CNN (s)	Seed	87.5	97.2	20.9	60.5	89.1	97.1	n/a	n/a	n/a	n/a
	CNN (s)	Hyp-10-r	8.7	76.9	-47.5	13.7	-70.1	18.9	48.4	81.5	47.9	79.6
	CNN (s)	Hyp-10-f	62.4	75.6	3.9	12.6	-2.0	17.0	95.4	81.6	84.6	77.7
SVHN	CNN (s)	Seed	91.0	98.6	-42.8	65.9	66.9	92.5	n/a	n/a	n/a	n/a
	CNN (s)	Hyp-10-r	-8.6	90.3	-55.3	27.6	-30.5	11.1	38.2	58.5	55.7	72.3
	CNN (s)	Hyp-10-f	64.2	89.9	17.5	27.4	-0.1	11.1	67.3	58.2	76.1	73.6
USPS	CNN (s)	Seed	92.5	98.7	44.3	71.8	86.0	98.4	n/a	n/a	n/a	n/a
	CNN (s)	Hyp-10-r	-11.5	70.3	-35.2	13.6	-75.7	21.3	49.2	88.8	43.7	66.2
	CNN (s)	Hyp-10-f	73.2	70.8	10.8	14.7	18.9	23.0	96.3	88.1	74.5	72.7
CIFAR10	CNN (s)	Seed	75.3	96.0	27.0	90.2	68.6	91.1	n/a	n/a	n/a	n/a
	CNN (s)	Hyp-10-r	50.1	88.0	-4.3	40.5	-2.7	34.2	34.0	50.5	71.5	80.9
	CNN (s)	Hyp-10-f	67.0	87.9	38.2	42.9	27.0	31.8	72.0	52.2	75.6	80.0
CIFAR10	CNN (l)	Seed	83.6	98.2	33.4	92.9	86.5	95.7	n/a	n/a	n/a	n/a
	CNN (l)	Hyp-10-r	32.6	90.5	-0.9	47	-10.5	35.5	41.6	51.6	69.1	83.1
	CNN (l)	Hyp-10-f	64.5	91.4	30.4	40.7	29.8	35.3	74.5	54.9	77.7	86.0
STL	CNN (s)	Seed	17.8	91.2	2.0	30.2	45.3	95.0	n/a	n/a	n/a	n/a
	CNN (s)	Hyp-10-r	-8.7	77.1	-44.0	9.3	-68.8	19.1	41.3	93.9	46.3	66.8
	CNN (s)	Hyp-10-f	76.1	76.5	6.7	10.7	21.2	22.4	98.1	91.3	78.1	62.6
STL	CNN (l)	Seed	-112	94.2	2.8	37.3	5.6	98.7	n/a	n/a	n/a	n/a
	CNN (l)	Hyp-10-r	-79.6	74.1	-118	10.7	-106	18.8	43.8	90.4	49.4	68.3
	CNN (l)	Hyp-10-f	84.1	77.7	10.4	11.7	14.6	19.1	97.8	92.8	78.8	68.0
CIFAR10	ResNet-18	Seed	–.-	96.8	–.-	76.7	–.-	99.6	n/a	n/a	n/a	n/a
CIFAR100	ResNet-18	Seed	–.-	97.4	–.-	95.4	–.-	99.9	n/a	n/a	n/a	n/a
t-ImageNet	ResNet-18	Seed	–.-	96.1	–.-	87.5	–.-	99.9	n/a	n/a	n/a	n/a

3.5 Potential use cases & Applications

While populations of NNs have been used in previous work, they still are relatively novel as a dataset. As use cases for such datasets may not be obvious, this section presents potential use cases and applications. For all use cases, we collect related work that uses model populations. Here, the zoos may be used as data or to evaluate the methods. For some of the use cases, the analysis above provides support. Lastly, we suggest ideas for future work that we hope can inspire the community to make use of the model zoos.

3.5.1 Model Analysis

The analysis of trained models is an important and difficult step in the machine learning pipeline. Commonly, models are applied on hold-out test sets, which may contain difficult cases with specific properties [101]. Other approaches identify subsections of input data that are relevant for a specific output [84, 177, 189]. A third group of methods compares the activations of models, e.g. the cka method used in Sec. 3.4 to measure diversity [91].

Populations of models have been used to identify commonalities in model weights, activations, or graph structure which are predictive for model properties. Some methods use the weights, weight statistics or eigenvalues of the weight matrices as features to predict a model's accuracy or hyper-parameters [43, 116, 159]. Recently, [147] have learned self-supervised representation of the weights and demonstrate their usefulness for predicting model properties. Other publications use activations to approximate intermediate margins [80, 172] or graph connectivity features [26] to predict the generalization gap or test accuracy. Standardized, diverse model zoos may facilitate the development of new methods, or be used as evaluation datasets for existing model analysis, interpretability, or comparison methods.

Previous work as well as the experiment results in Sec 3.4 indicate that even more complex model properties might be predicted from the weights. By studying populations of models, in-depth diagnostics of models, such as whether a model learned a specific bias, may be based on the weights or topology of models. Lastly, model properties as well as the weights may be used to derive a model 'identity' along the training trajectory, to allow for NN versioning.

3.5.2 Learning Dynamics

Analyzing and utilizing the learning dynamics of models has been a useful practice. For example, early stopping [48], which determines when to end training at minimal generalization error based on a cross-validation set has become standard in machine learning practice.

More recently, methods have exploited zoos of models. Population-based training [78] evaluates the performance of model candidates in a population, and decides which of the candidates to pursue further and which to give up. HyperBand evaluates performance metrics for groups of models to optimize hyperparameters [104, 105]. Research in Neural Architecture Search was greatly simplified by the NASBench dataset family [175], which contains performance metrics for varying hyperparameter choices. Our model zoos extend these datasets by adding models including their weights at states throughout training, which may open new doors for new approaches.

The accuracy distribution of our model zoos becomes relatively broad if hyperparameters are varied (Figure 3.2). For early stopping or population-based methods, identifying a good range of hyperparameters to try, and then identifying those candidates that will perform best towards the end of training, is a challenging and relevant task. Our model zoos may be used to develop and evaluate methods to that end. Beyond that, diverse model zoos offer the opportunity to make further steps of understanding and exploiting the learning dynamics of models, i.e., by studying the regularities of generalizing and overfitting models. The shape and curvature of training trajectories may contain rich information on the state of model training. Such information could be used to monitor model training or adjust hyperparameters to achieve better results. The sparsified model zoos add several potential use cases. They may be used to study the sparsification performance on a population level, study emerging patterns of populations of sparse models, or the relation of full models and their sparse counterparts.

3.5.3 Representation Learning

NN models have grown in recent years and with them the dimensionality of their parameter space. Empirically, it is more effective to train large models to high performance and distill them in a second step, than to directly train the small models [70, 110]. This and other related problems raise interesting questions. What are useful regularities in NN weights? How can the weight space be navigated in a more efficient way?

Recent work has attempted to learn lower dimensional representations of the weights of NNs [61, 88, 142, 147, 148, 149, 179]. Such representations can reveal the latent structure of NN weights. Other approaches identify subspaces in the weight space that relate to high performance or generalization [6, 112, 167]. In [147], representations learned on model zoos achieve higher performance in predicting model properties than weights or weight statistics. [88] proposes a method to learn from a population of diverse neural architectures to generate weights for unseen architectures in a single forward pass.

Our model zoos can be either a dataset to train representations on as in [147] or [6], or as a common dataset to validate such methods. Learned representations may bring a better understanding of the weight space and thus help to reduce the computational cost and improve the performance of NNs.

3.5.4 Generating New Models

In conventional machine learning, models are randomly initialized and then trained on data. As that procedure may require large amounts of data and computational resources, fine-tuning and transfer learning are more efficient training approaches that re-

use already trained models for a different task or dataset [46, 176]. Other publications have extended the concept of transfer learning from a one-to-one setup to many-to-one setups [108, 153]. Both approaches attempt to combine learned knowledge from several source models into a single target model. Most recently, [148, 149] have generated unseen NN models with desirable properties from representations learned on model zoos. The generated models were able to outperform random initialization and pretraining in transfer-learning regimes. In [136], a transformer is trained on a population of models with diffusion to generate model weights.

All these approaches require suitable and diverse models to be available. Further, the exact properties of models suitable for generative use, transfer learning, or ensembles are still in discussion [46]. Population-based transfer learning methods such as zoo-tuning [153], knowledge flow [108] or model-zoo [140] have been demonstrated on populations with only a few models. Populations for these methods ideally are as diverse as possible, so that they provide different features. Investigating the models in the proposed zoos may help identify models that lend themselves to transfer learning or ensembling.

3.6 Conclusion

To enable the investigation of populations of neural network models, we release a novel dataset of model zoos with this work. These model zoos contain systematically generated and diverse populations of 50'360 neural network models comprised of 3'844'360 collective model states. The released model zoos come with a comprehensive analysis and initial benchmarks for multiple downstream tasks and invite further work in the direction of the following use cases: (i) model analysis, (ii) learning dynamics, (iii) representation learning and (iv) model generation.

Appendix

3.A Model Zoo Generation Details

In our model zoos, we use three architectures. Two of them rely on a general CNN architecture, the third is a common ResNet-18[65]. For the first two architectures, use the general CNN architecture in two sizes, detailed in Table 3.A.1. By varying different generating factors listed in Table 3.1, we create a grid of configurations, where each node represents a model. Each node is instantiated as a model and trained with the exact same training protocol. We chose the hyperparameters with diversity in mind. The ranges for each of the generating factors are chosen such that they can lead to functioning models with a corresponding set of other generating factors. Nonetheless, that leads to some nodes with uncommon and less-than-promising configurations.

The code to generate the models can be found on www.modelzoos.cc. With that code, the model zoos can be replicated, changed, or extended. We trained our model zoos on CPU nodes with up to 64 CPUs. Training a zoo takes between 3h (small models, small configuration, and small dataset) and 3 days (large models, large configuration, and large dataset). Overall, the generation of the zoos took around 30'000 CPU hours.

3.B Data Management and Accessibility of Model Zoos

Data Management and Documentation: To ensure that every zoo is reproducible, expandable, and understandable, we document each zoo. For each zoo, a Readme file is generated, displaying basic information about the zoo. The exact search pattern and the training protocol used to train the zoo are saved in a machine-readable JSON file. To make the zoos expandable, the dataset used to train the zoo and a file describing the model architecture are included. The model class definition in pytorch is included with the zoo. Each model is saved along with a JSON file containing its exact hyperparameter combination. A second JSON file contains the the performance metrics during training.

Table 3.A.1: CNN architecture details for the models in model zoos.

Layer	**Component**	**CNN small**	**CNN large**
Conv 1	input channels	1 or 3	3
	output channels	8	16
	kernel size	5	3
	stride	1	1
	padding	0	0
Max Pooling	kernel size	2	2
Activation			
Conv 2	input channels	8	16
	output channels	6	32
	kernel size	5	3
	stride	1	1
	padding	0	0
Max Pooling	kernel size	2	2
Activation			
Conv 3	input channels	6	32
	output channels	4	15
	kernel size	2	3
	stride	1	1
	padding	0	0
Activation			
Linear 1	input channels	36	60
	output channels	20	20
Activation			
Linear 2	input channels	20	20
	output channels	10	10
Total Parameters		2464 or 2864	10853

Model checkpoints are saved for every epoch. To enable further training of the models in the zoo, a checkpoint recording the optimizer state is saved for the final epoch of each model. All data can be found on the model zoo website as well as directly from Zenodo.

Accessibility: We ensure the technical accessibility of the data by hosting it on Zenodo, where the data will be hosted for at least 20 years. Further, we take steps to reduce access barriers by providing code for data loading and preprocessing. With that, we reduce the friction associated with analyzing the raw zoo files. Further, it improves consistency by reducing errors associated with extracting information from the zoo. To

that end, we provide a PyTorch dataset class encapsulating all model zoos for easy and quick access within the PyTorch framework. A Tensorflow counterpart will follow. All code can be found on the model zoo website as well as a code repository on github. To ensure conceptional accessibility, we include detailed insights, visualizations, and the analysis of the model zoo (Sec. 3.4) with each zoo. Mode details can be found on the dataset website www.modelzoos.cc.

3.C Dataset Documentation and Intended Uses

The main dataset documentation can be found at www.modelzoos.cc and is detailed in the paper in Section 3.3.4. There, we provide links to the zoos, which are hosted on Zenodo as well as analysis of the zoos. In the future, the analysis will be systematically extended. The documentation includes code to reproduce, adapt, or extend the zoos, code to reproduce the benchmark results, as well as code to load and preprocess the datasets. Dataset Metadata and DOIs are automatically provided by Zenodo, which also guarantees the long-term availability of the data. Files are stored as `zip`, `json` and `pt` (pytorch) files. All libraries to read and use the files are common and open source. We provide the code necessary to read and interpret the data.

The datasets are synthetic and intended to investigate populations of neural network models, i.e., to develop or evaluate model analysis methods, progress the understanding of learning dynamics, serve as datasets for representation learning on neural network models, or as a basis for new model generation methods. More information regarding the usage is given in the paper.

3.D Author Statement

The dataset is publicly available under www.modelzoos.cc and licensed under the Creative Commons Attribution 4.0 International license (CC-BY 4.0). The authors state that they bear responsibility under the CC-BY 4.0 license.

3.E Hosting, Licensing, and Maintenance Plan

The dataset is publicly available under www.modelzoos.cc and licensed under the Creative Commons Attribution 4.0 International license (CC-BY 4.0). The landing page contains documentation, code, and references to the datasets, as detailed in the paper in Section 3.3.4. The datasets are hosted on Zenodo, to ensure (i) long-term

availability (at least 20 years), (ii) automatic searchable dataset metadata, (iii) DOIs for the dataset, and (iv) dataset versioning. The authors will maintain the datasets, but invite the community to engage. Code to recreate, correct, adapt, or extend the datasets is provided, and s.t. maintenance can be taken over by the community at need. The github repository allows the community to discuss, interact, add, or change code.

Chapter 4

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

Abstract

Self-supervised learning approaches proved to be particularly valuable for extracting task-relevant, useful, and information-preserving representations. Neural Networks (NNs) are widely applied, yet their weight space is still not fully understood. Therefore, we propose self-supervised representation learning on the weights of populations of trained NNs from which we predict NN characteristics. 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 helps to faithfully recover useful NN model properties. We show that learned representations for trained NNs outperform prior work for the tasks of hyper-parameter, test accuracy, and generalization gap prediction and transfer to out-of-distribution settings.

This work was accepted for publication at NeurIPS 2021 [147]

4.1 Introduction

Self-supervised and unsupervised representation learning approaches offer task-relevant, function-serving, and information-preserving data representations [5, 20]. Such methods proved to be particularly valuable in helping to gain insights, uncover latent factors, find grouping, or unveil intrinsic structure within the data. Bengio et al. [4, 5], LeCun et al. [101], Lee et al. [102], Vincent et al. [162] used them for data exploration, dimensionality reduction, reconstruction, and prediction tasks in image, video, audio, and text data.

Only recently, researchers scratched the surface on related studies for populations of trained NNs (referred to as *model zoos* [43, 120, 159]). Insights about populations of trained NNs weights and biases might help gain a better understanding of the fundamental characteristics that make NN successful, their learning dynamics, and their performance. Thus, we could improve NN hyper-parameters selection, perform test accuracy prediction without seeing the test data, estimate the generalization gap, and give a more reliable out-of-distribution estimate [26, 43, 73, 80, 159, 172]. However, the weight space of NNs is high-dimensional and, does not offer a straightforward and intuitive understanding of its latent factors. The space of trained NN is not well understood and little is known about the most valuable characteristic of trained NN models or how to obtain useful representations for them.

Most of the recent efforts in this direction focused on manually finding a set of features, carefully designing summaries and measures with discriminative and predictive power over a) NN activation responses triggered by data, or b) trained NN weights and biases. Raghu et al. [139], Morcos et al. [126] and Kornblith et al. [91] compared NN models by correlating their activations. Dinh et al. [39] link model properties to characteristics of the loss surface around the minimum solution. Such methods provide insights about the learning process and have their merits, but they are expensive to compute, and can only compare two NN models at a time. Jiang et al. [80] proposed a measure based on the concept of margin distribution, manually summarised over data points and across NN layers. Corneanu et al. [26] used connectivity patterns in the NN activation, and computed topological summaries. Corneanu et al. [26], Jiang et al. [80] showed that such measures correlate with the generalization gap, but estimating them requires data and computation. Eilertsen et al. [43] predicted NN hyper-parameters using weight statistics or 1D convolutional neural networks (CNN) directly from the weights. Unterthiner et al. [159] proposed layer-wise statistics derived from the NN models to predict the test accuracy of the NN model. Unfortunately, such features are limited by design as they disregard the order of weights and neglect the structural relations among weights.

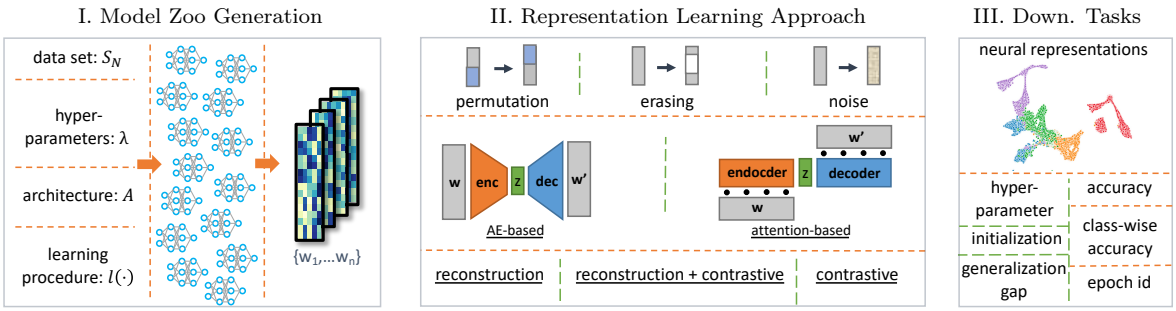


Figure 4.1.1: An overview of the proposed self-supervised representation learning approach. I. Populations of trained NNs form model zoos. NNs weights are vectorized. II. Neural representations are learned from the model zoos. III. Neural representations are evaluated on downstream tasks.

In this paper, we introduce a novel approach. Instead of using hand-crafted features, or summaries, we propose self-supervised representation learning on populations of trained NN weights and biases and predict the NN properties from its learned representation. We hypothesize that learned representations over populations of NNs can faithfully unveil properties about the NNs and the data on which they were trained. It is reasonable to assume that a self-supervised method can find latent variables common for diverse populations of NNs given sufficient capacity and efficient training of the learning model. Also, similarly to image, video, or audio data, a self-supervised approach could capture the relevant correlations in the populations of trained NNs with very little inductive bias while robust to variations and having task-agnostic utility [100]. We adapt and examine different NN architectures focusing on an attention-based module to model associations between the different weights in a single NN. We apply self-supervised learning with reconstruction [56] and contrastive losses [20]. So that we can preserve the distinctive information about trained NNs and compactly relate to their common properties without relying on supervisory feedback from weak labels. To enable efficient training, we further propose novel augmentation methods. We introduce structure-preserving permutations as augmentations, which take into account the structural symmetries within the NN weights that we find necessary and crucial for learning. We also adapt erasing [183] and noise [56] as augmentations for NN weights. We give an overview of our learning approach in Figure 4.1.1.

To validate our hypothesis, we perform extensive numerical experiments over different populations of trained NN models. To do so, we use publicly available as well as generate and publish NN model zoos. In contrast to [43, 159], our zoos contain models with different initialization points and diverse configurations, and include dense number of check-points, *i.e.*, model versions for its evolving points during training. Our ablation study confirms that the various factors for generating a population of trained NNs

Figure 4.1.2: 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 4.1.3: 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.

play a vital role in how and which properties are recoverable for trained NNs [159]. 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.

Similarly, as in the common representation learning setups [5], we use linear probing [59] and evaluate multiple downstream tasks to see the potential of our approach. We find that not only learned NNs but also their *neural representations*[1] contain the latent footprint of their training data [145]. We show that our task-agnostic neural representations have high utility on tasks like hyper-parameters, test accuracy, and generalization gap prediction. Furthermore, we demonstrate improved performance compared to the state-of-the-art on the previously mentioned tasks and outlay the advantages in an out-of-distribution prediction.

4.2 Model Zoos and Augmentations

Model Zoo. We denote by $\mathcal{D}$ 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 with A the specific NN architecture and with $l(.)$ the learning procedure. Training under different prescribed configurations $\{\mathcal{D}, \lambda, A, l(.)\}$ results in a population of NNs which we refer to as *model zoo*. We convert the weights and biases of all NNs in the model zoo into a vectorized form. In the resulting set $\mathcal{W} = \{\mathbf{w}_1, ..., \mathbf{w}_M\}$, $\mathbf{w}_i$ denotes the flattened vector of dimension N, representing the weights and biases for one trained NN model.

[1] By neural representation, we refer to a learned representation from a population of NNs given a model zoo.

Augmentations. In many machine learning tasks, data augmentation helps to learn robust models, both by increasing the number of training samples and preventing overfitting of strong features [152]. We propose three methods to augment individual instances of our model zoos to support our self-supervised representation learning approach.

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 [10]. 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

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

where

$$\begin{aligned}\hat{\mathbf{W}}^{l+1} &= \mathbf{W}^{l+1}(\mathbf{P}^{l})^{\mathrm{T}},\\ \hat{\mathbf{W}}^{l} &= \mathbf{P}^{l}\mathbf{W}^{l},\\ \hat{\mathbf{b}}^{l} &= \mathbf{P}^{l}\mathbf{b}^{l}\end{aligned}$$

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[2]. The permutation can be applied also to kernels of convolution layers. While the *permutation augmentation* differs significantly from existing augmentation techniques, flips for images are similar, but specific instances from the set of possible permutations in the image domain. 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 [37]. We adopt an approach of *random erasing* of sections in the vectorized forms of trained NN weights. Similarly, as in [183], we apply the erasing augmentation with a probability p to an area that is randomly chosen with 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 [56]. We note that noise and erasing as weight augmentations previously were not explored in this domain.

[2]Details, formal statements and proofs can be found in the Appendix

4.3 Neural Representation Learning

Commonly, self-supervised representation learning employs a discriminative or generative approach Grill et al. [59]. We follow a discriminative modeling principle such as contrastive learning and borrow ideas from generative approaches such as auto-encoding to learn representations in a self-supervised fashion. We adopt an encoder-decoder setup where we use different losses.

Architectures and Self-Supervised Losses. 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 $\hat{\mathbf{w}}_i = h_\psi(\mathbf{z}_i) = h_\psi(g_\theta(\mathbf{w}_i))$. To learn the encoder and decoder parameters, we use a loss $\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_c$ composed of MSE reconstruction term $\mathcal{L}_{MSE} = \frac{1}{M}\sum_{i=1}^{M} \|\mathbf{w}_i - h_\psi(g_\theta(\mathbf{w}_i))\|_2^2$ and the NT_Xent loss Chen et al. [20] term $\mathcal{L}_c$. We denote this architecture with its loss as E_cD. In contrastive learning, many methods prevented mode collapse by using negative samples. Our loss $\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_c$, 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

$$\mathcal{L}_{c+} = \sum_i -\log\left(\exp(\text{sim}(\mathbf{z}_i^j, \mathbf{z}_i^k)/\tau\right),$$

where each $\mathbf{z}_i$ is randomly augmented twice and forms the two *views* $\mathbf{z}_i^j$ and $\mathbf{z}_i^k$, while $\text{sim}(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}_i^T\mathbf{z}_j / ||\mathbf{z}_i|| ||\mathbf{z}_j||$ is the cosine similarity. We denote the encoder-decoder with the loss $\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_{c+}$ as E_{c+}D. We also experiment with encoder-decoder and a reconstruction loss $\mathcal{L} = \mathcal{L}_{MSE}$ and denote it as ED, and consider only an encoder with a contrastive loss $\mathcal{L} = \mathcal{L}_c$ and denote it as E_c. In all of the architectures, we embed the neural representation $\mathbf{z}_i$ in a low dimensional space, $L < N$. We would like to point out that Eilertsen et al. [43] and Unterthiner et al. [159] manually selected statistics and summaries. In our approach, $\mathbf{z}_i$ can also be seen as a summary, which is learned to compactly extract the relevant and meaningful information from the weights.

Attention Module. Our encoder and decoder pairs are symmetrical and of the same type. We apply fully connected feed-forward networks (FFN) as baselines. As there is no intuition on good inductive biases in the weight space, we further use multi-head self-attention modules (Att) [160] as an architecture with very little inductive bias. In the multi-head self-attention module, we apply learned position encodings to preserve structural information [40]. We propose two methods to encode the weights into a

sequence (Figure 4.1.2). 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 4.1.3). 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 Devlin et al. [37], Zhong et al. [183], we add a learned token to the input sequence that we dub *compression token*. After passing the input sequence through 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. The compression token aggregates information while the input sequence passes through the transformer. Its dimension is directly tied to the dimension of the value tokens and so its capacity affects the overall memory consumption.

Downstream Tasks. We use linear probing [59] 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, F-Score [56] prediction (F_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 [170]. We also evaluate for hyper-parameters prediction tasks, like the activation function (Act), optimizer (Opt), and initialization method (Init). Here, we train a linear perceptron by minimizing a cross-entropy loss [56] and measure the prediction accuracy.

4.4 Empirical Evaluation

4.4.1 Model Zoos

Publicly Available Model Zoos. Unterthiner et al. [159] introduced model zoos of CNNs with 4970 parameters trained on MNIST [99], Fashion-MNIST [171], CIFAR10 [93] and SVHN [129], 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						
	Rec.	Eph	Acc	F_{C0}	F_{C1}	F_{C2}	F_{C3}
E_c	-	40.3	42.4	38.2	27.3	46.5	33.4
ED	93.5	**94.4**	81.2	59.8	63.5	55.6	59.2
E_cD	76.8	92.1	**85.7**	**66.0**	**70.0**	**66.3**	**81.7**
E_{c+}D	**94.3**	91.7	71.6	52.3	58.9	42.5	54.2

Table 4.4.1: Ablation results over self-supervised learning losses. All models are implemented with attention-based reference architecture. All values are given in %.

	TETRIS-SEED						
	Rec.	Eph	Acc	F_{C_0}	F_{C_1}	F_{C_2}	F_{C_3}
FF	11.7	73.8	69.6	51.5	52.2	52.4	59.4
Att_W	64.1	93.6	79.6	63.5	**70.0**	53.7	65.8
Att_{W+t}	20.8	56.9	52.8	44.5	46.9	43.8	41.8
Att_N	**82.2**	87.3	80.5	63.5	65.1	63.0	66.5
Att_{N+t}	76.8	**92.1**	**85.7**	**66.0**	**70.0**	**66.3**	**81.7**

Table 4.4.2: 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.

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, we first create 4x4 grey-scaled image data set that we call *tetris* by using four tetris shapes.

	TETRIS-SEED							
	-	P	E	N	P,E	P,N	E,N	P,E,N
ED	79.4	**93.8**	75.1	79.7	93.5	93.8	75.3	93.5
E_cD	38.3	48.2	53.9	19.7	79.1	52.6	42.9	81.1
E_{c+}D	**81.0**	**93.8**	**83.2**	**82.0**	**94.3**	**94.0**	**83.8**	**94.3**

Table 4.4.3: R^2 reconstruction score given in % in the augmentation ablation for different attention-based architectures. We use: permutation (P), erasing (E), and noise (N) augmentation.

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 100 different seeds. Out of the 3600 models in total, we have successfully trained 2900 for 75 epochs - the remainder 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 [3].

[3]Full details on the generation of the zoos can be found in the Appendix

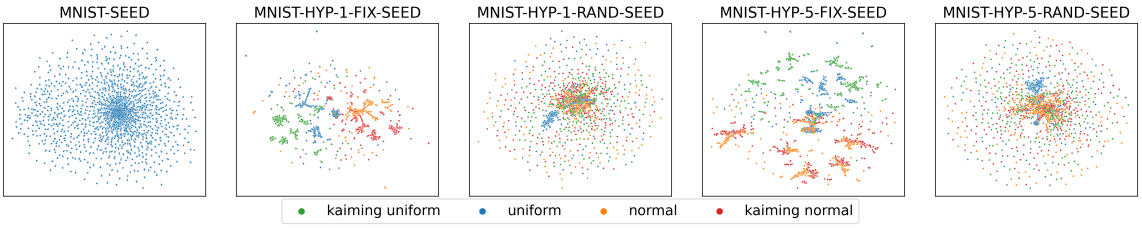


Figure 4.4.1: UMAP dimensionality reduction for NN model zoos are created with different generating factors. Colors represent initialization methods. Compare numerical results in Table 4.4.4.

Model Zoo Generating Factors. Prior work discusses the impact of random seeds on the properties of model zoos. While Yak et al. [172] use multiple random seeds for the same hyper-parameter configuration, Unterthiner et al. [159] 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 [3]. `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 [159]). To decouple the hyper-parameter configuration from one specific seed, `MNIST-HYP-1-RAND-SEED` draws 1 random seed 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 fixed and random seeds, respectively.

4.4.2 Training and Testing Setup

Architectures. We evaluate our approach under different types of architectures, including E_c, ED, E_cD and $E_{c+}D$, see Section 4.3. As encoders E and decoders D, we experimented with the architectures introduced in Section 4.3. 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. 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.

	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}	78.5	90.9	78.5	86.5	81.4
R_{tes}	59.5	79.0	57.9	78.8	63.7

	W	s(W)	E_{c+}	D	W	s(W)	E_{c+}	D	W	s(W)	E_{c+}	D	W	s(W)	E_{c+}	D
EPH	54.9	**97.7**	95.9		03.1	06.2	**06.8**		-67	**04.2**	-14		01.1	**12.4**	03.4		-20	**07.8**	-03.	
ACC	82.8	**98.4**	98.1		07.7	74.4	**87.0**		$<\kappa$	**56.6**	55.5		21.5	66.1	**78.7**		$<\kappa$	**69.1**	58.1	
GGAP	$<\kappa$	**45.3**	40.4		-72	35.8	**49.4**		$<\kappa$	**24.7**	01.8		$<\kappa$	33.9	**35.7**		$<\kappa$	**41.5**	10.8	
INIT	–	–	–		**88.5**	67.3	85.6		39.2	**58.2**	44.8		**82.5**	61.3	76.9		37.6	**55.4**	40.1	

Table 4.4.4: R^2 score in %. Results about the impact of the generating factors for the model zoos. 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 architecture after 1750 epochs. $\kappa = -170$.

Neural Representation Learning and Downstream Tasks. We apply the proposed data augmentation methods for representation learning (see Section 4.2). We run our representation learning algorithms for up to 2500 epochs, using the adam optimizer [86], 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 85% of the model zoos for training and 15% for testing. 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 a quality metric for self-supervised learning, we track the reconstruction R^2 on the test split of the zoo. As a proxy for how much useful information is contained in the neural representation, we evaluate on downstream tasks as described in Section 4.3. To ensure numerical stability of the solution to the linear probing, we apply Tikhonov regularization [157] with regularization parameter α in the range $[1e\text{-}5, 1e3]$ (we choose α by cross-validating over the R^2 score for the training 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 adam optimizer [86] with a learning rate of $1e$-4 and weight-decay of $1e$-6. The linear probing is applied to the same train-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 [159]. We investigate how well the linear probing estimator computed over neural representations generalizes 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 a performance metric, which is a measure of rank correlation. It measures the ordinal association between two measured quantities [85].

Baselines, Computing Infrastructure and Run Time. As a baseline, we use the model zoos (W). In addition, we also consider linear PCA (PCA_l), cosine similarity PCA (PCA_c), radial basis kernel PCA (PCA_r), and UMAP (U_m) [120]. We further compare to layer-wise weight-statistics (mean, var, quintiles) s(W) as in [159]. Similar features are used in [43]. As computing hardware, we use half of the available resources from NVIDIA DGX2 station with 3.3GHz CPU and 1.5TB RAM memory, which 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.

	TETRIS-SEED						
	W	PCA_l	PCA_c	PCA_r	U_m	s(W)	E_cD
Eph	59.8	49.3	78.5	93.7	65.7	**96.4**	95.2
Acc	53.5	28.7	76.5	75.7	76.0	86.6	**87.0**
F_{C0}	30.6	11.9	54.8	50.7	50.3	62.2	**67.3**
F_{C1}	45.3	31.2	50.7	59.9	39.2	62.1	**68.2**
F_{C2}	47.3	12.5	42.6	40.3	46.9	60.9	**66.4**
F_{C3}	57.8	47.3	**75.2**	73.5	72.9	71.8	60.3

	TETRIS-HYP						
	W	PCA_l	PCA_c	PCA_r	U_m	s(W)	E_cD
Rec	–	79.7	43.4	-30	–	–	**91.0**
Eph	02.8	02.9	01.9	13.2	0.*	16.2	**17.2**
Acc	21.5	22.1	25.6	32.4	0.*	48.5	**60.8**
LR	00.0	00.0	00.2	44.0	00.0	**53.1**	49.0
Act	75.0	73.5	79.6	80.4	46.7	73.8	**87.6**
Init	38.0	36.9	36.1	36.5	29.6	48.1	**48.6**

Table 4.4.5: R^2 given in %. **Left**. Reconstruction, epoch, accuracy, and class-wise F-scores prediction. **Right**. Epoch, accuracy, learning rate, seed, activation function, and initialization prediction.

4.4.3 Results

Augmentation Ablation. To evaluate the impact of the proposed augmentations (Section 4.2) for our representation learning method, we present an ablation analysis, in which we measure R^2 for ED, E_cD, and E_{c+}D[4]. We use 120 permutations, a probability of 0.5 for erasing the weights, and zero-mean noise with a standard deviation 0.05 (see Table 4.4.3). 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 a few training epochs. Erasing further improves test performance and allows for extended training without 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 4.4.1. 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. 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 that embedding all weights of one neuron (or convolutional kernel) to one token combined with compression

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

	MNIST-HYP			FASHION-HYP			CIFAR10-HYP			SVHN-HYP		
	W	S(W)	$E_{c+}D$	W	S(W)	$E_{c+}D$	W	S(W)	$E_{c+}D$	W	S(W)	$E_{c+}D$
EPH	21.6	32.1	**33.4**	21.0	32.8	**40.7**	18.7	29.4	**35.6**	14.6	37.2	**39.7**
ACC	72.7	81.9	**90.0**	66.4	79.4	**90.2**	73.5	83.2	**90.9**	79.5	82.1	**89.9**
GGAP	15.3	24.8	**32.9**	41.0	42.1	**53.7**	32.7	41.1	**55.4**	30.2	38.0	**48.9**
LR	08.4	32.5	**36.4**	29.4	35.6	**43.0**	20.8	30.5	**40.2**	20.3	33.0	**40.0**
ℓ_2-REG	08.4	15.6	**18.1**	06.9	16.4	**24.0**	02.0	13.6	**24.6**	04.8	12.4	**18.2**
DROP	23.3	18.8	**34.2**	21.9	21.9	**39.1**	10.2	15.7	**28.2**	04.4	14.0	**20.8**
TF	-1.2	07.4	**11.1**	-0.3	07.5	**13.0**	02.2	06.6	**17.0**	-3.9	08.3	**13.0**
ACT	**89.0**	83.7	86.9	**90.3**	83.4	87.8	**88.8**	80.3	86.1	**87.7**	79.7	84.0
INIT	**94.5**	72.8	93.1	**95.7**	77.5	94.9	**93.4**	75.3	92.8	**91.0**	72.8	88.4
OPT	**76.8**	67.0	72.1	**79.8**	68.8	76.0	**74.0**	67.6	72.1	**72.0**	69.4	69.9

Table 4.4.6: **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.

tokens (Att_{Nt}) shows the overall best performance and scales to larger architectures. Compression tokens achieve higher performance. 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. On the large zoos, compression tokens could not be applied successfully. High-capacity compression tokens required overall high token dimensions, which exceeded the available memory.

Self-Supervised Learning Ablation. In Table 4.4.2, we evaluate the usefulness of the self-supervised learning tasks (Section 4.3). The application of purely contrastive loss in E_c, appears not to provide useful representations for the 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 performance on the downstream tasks. The variation $E_{c+}D$, has the highest reconstruction accuracy. The addition of a contrastive loss helps to pronounce distinctiveness so that the neural representations are both better at reconstructing the NN weights, as well as revealing properties about the NNs. When analyzing the impact of the compression ratio N/L in E_cD, we found that very high compression ratios hurt performance. E_cD could not be trained to satisfactory results on the medium and large model zoos. We therefore apply E_cD on the small, and $E_{c+}D$ on the medium and large model zoos, which is less demanding and shows superior performance to ED on those zoos.

Zoo Generating Factors Ablation. Figure 4.4.1 visualizes the weight of the zoos, which contain models of all epochs[5]. In table 4.4.4, we report numerical properties. Only changing the seeds appears to result in homogeneous development with a very high correlation between $s(W)$ and the properties of the samples in the zoo. Varying the hyper-parameters reduces the correlation. With fixed seeds, visually, 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 function determine the shape of the loss surface, while the seed and initialization method decide the starting point. Such clustering appears to facilitate representation learning, with high R^2 for reconstruction and high performance on downstream tasks. We observe similar properties in the zoos of [159], see Appendix. Initializing models with random seeds visually and numerically disperses the clusters. 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. On the other hand, the results show that sharing the same hyper-parameter configuration across five seeds helps the representation learning and NN property prediction. Thus, we conclude that changing only the seeds results in models with very similar evolution during learning while using one seed shared across models might create shortcuts in the weight space. Zoos that vary both appear to be the most diverse and hardest for learning and NN property prediction.

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 with multiple baselines and s(W). The results are shown in Tables 4.4.4, 4.4.5 and and 4.4.6. 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 small and large zoos. On the medium-sized zoos, neural representations are on par with s(W). Our approach demonstrates that we can faithfully recover useful information. On the `TETRIS-SEED`, and `MNIST-SEED` model zoos, s(W) achieves high R^2 scores on all downstream tasks (see Table 4.4.5 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 Acc, F_{C0-C3} and are competitive on Eph. On the `TETRIS-SEED` model zoo, we outperform the baseline methods. On

[5]Further visualizations can be found in the Appendix

	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	.35	.20	.10	**.32**	.13	**.24**	.19	-.02	-.05	**.05**
FASHION-HYP	-.03	**.07**	.04	.54	.48	**.55**	.06	**.14**	-.03	.08	.09	**.20**
SVHN-HYP	-.03	**.16**	**.16**	-.03	**.21**	.05	.44	.35	**.46**	-.02	.07	**.08**
CIFAR10-HYP	.11	.10	**.12**	.35	.35	**.45**	.14	.17	**.19**	**.41**	.30	**.41**

Table 4.4.7: 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.

TETRIS-HYP, the overall performance of all methods is the lowest across the downstream tasks compared to the other zoos (see Table 4.4.5 right). 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. They achieve a higher R^2 score compared to W on the prediction of continuous hyper-parameters. On the categorical hyper-parameters, like activation, initialization method, and optimizer, the weight space achieves the highest R^2 scores, see Table 4.4.6. We explain this with the small amount of variation in the model zoo, see Section 4.4.1, which allows to separate these properties in weight space.

Out-of-Distribution Prediction. Table 4.4.7 shows the out-of-distribution results for generalization gap prediction[6], which is a very challenging task. Neural representations outperform the baselines for Kendall's τ measure in the majority of the results. Surprisingly, W and $s(W)$ in this setup have similar performance across the used model zoos. Our approach has better scores in the 7 out of 12 results (the off-diagonal elements in Table 4.4.7) and has 1 tie in the comparisons with W and $s(W)$. 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.

[6]In the Appendix, we also give results for other tasks, including epoch id and test accuracy prediction

4.5 Related Work

There is ample research evaluating the structures of NNs by visualizing activations, *e.g.* [178], 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. Laakso and Cottrell [94] compared the activations of NNs by a measure of "sameness". Li et al. [106] computed correlations between the activations of different nets. Wang et al. [166] tried to match the subspaces of the activation spaces in different networks [82], which showed to be unreliable. Kornblith et al. [91], Morcos et al. [126], Raghu et al. [139] applied correlation metrics to NN activations in order to study the learning dynamics. Jia et al. [79] approximated the space of DNN activations with a convex hull. Jiang et al. [80] also used activations to approximate the margin distribution and predict the generalization gap. Corneanu et al. [26] proposed persistent homology by using connectivity patterns in the NN activation, and computed 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. Only recently, two publications attempted to exploit populations of NNs. Instead of classifying a classifier for hyper-parameters prediction [43], we learn a general-purpose neural representations in self-supervised fashion. Unterthiner et al. [159] proposed layer-wise statistics derived from the NN models to predict the test accuracy of NN model. In our work, we model associations between the different weights in NN using an attention-based module. This helps us to learn representations that compactly extract relevant and meaningful information while considering the correlations between the weights in an NN.

4.6 Conclusions

In this work, we present a novel approach to learning neural representations from the weights of neural networks. To that end, we proposed new 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.

Appendix

4.A Permutation Augmentation

In this appendix section, we give the full derivation about the permutation equivalence in the proposed permutation augmentation (Section 4.2 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.

4.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

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

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 [144]. We express the error vector at layer l as

$$\delta^l = \nabla_{\mathbf{n}^l}\mathcal{L}, \tag{4.A.2}$$

and further use it to express the gradients as

$$\begin{aligned} \nabla_{\mathbf{W}^l} \mathcal{L} &= \delta^l (\mathbf{a}^{l-1})^{\mathrm{T}}, \\ \nabla_{\mathbf{b}^l} \mathcal{L} &= \delta^l. \end{aligned} \tag{4.A.3}$$

The output layer's error is simply given by

$$\delta^L = \nabla_{\mathbf{a}^L} \mathcal{L} \odot \sigma'(\mathbf{n}^L), \tag{4.A.4}$$

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

$$\delta^l = (\mathbf{W}^{l+1})^{\mathrm{T}} \delta^{l+1} \odot \sigma'(\mathbf{n}^l), \qquad l \in \{1, \cdots, L-1\}. \tag{4.A.5}$$

A usual parameter update takes on the form

$$(\mathbf{W}^l)_{\text{new}} = \mathbf{W}^l - \beta \nabla_{\mathbf{W}^l} \mathcal{L}, \tag{4.A.6}$$

where β is a positive learning rate.

4.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

$$\begin{aligned} \mathbf{n}^{l+1} &= \mathbf{W}^{l+1}\, \mathbf{a}^l + \mathbf{b}^{l+1} \\ &= \mathbf{W}^{l+1}\, \sigma(\mathbf{n}^l) + \mathbf{b}^{l+1} \\ &= \mathbf{W}^{l+1}\, \sigma(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l) + \mathbf{b}^{l+1}. \end{aligned} \tag{4.A.7}$$

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

$$\mathbf{P}^l \sigma(\mathbf{n}^l) = \sigma(\mathbf{P}^l \mathbf{n}^l), \tag{4.A.8}$$

which implies that we can write

$$\begin{aligned}\mathbf{n}^{l+1} &= \mathbf{W}^{l+1}\,\mathbf{I}\,\sigma(\mathbf{W}^{l}\,\mathbf{a}^{l-1}+\mathbf{b}^{l})+\mathbf{b}^{l+1}\\ &= \mathbf{W}^{l+1}\,(\mathbf{P}^{l})^{\mathrm{T}}\,\mathbf{P}^{l}\,\sigma(\mathbf{W}^{l}\,\mathbf{a}^{l-1}+\mathbf{b}^{l})+\mathbf{b}^{l+1}\\ &= \mathbf{W}^{l+1}\,(\mathbf{P}^{l})^{\mathrm{T}}\,\sigma(\mathbf{P}^{l}\,\mathbf{W}^{l}\,\mathbf{a}^{l-1}+\mathbf{P}^{l}\,\mathbf{b}^{l})+\mathbf{b}^{l+1}\\ &= \hat{\mathbf{W}}^{l+1}\,\sigma(\hat{\mathbf{W}}^{l}\,\mathbf{a}^{l-1}+\hat{\mathbf{b}}^{l})+\mathbf{b}^{l+1},\end{aligned} \tag{4.A.9}$$

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, $\forall l \in \{1,\cdots,L-1\}$, (4.A.9) 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

$$\begin{aligned}(\mathbf{P}^{l}\mathbf{W}^{l})_{\text{new}} =& \mathbf{P}^{l}\mathbf{W}^{l} - \alpha\mathbf{P}^{l}\nabla_{\mathbf{W}^{l}}\mathcal{L}\\ =& \mathbf{P}^{l}\mathbf{W}^{l} - \alpha\mathbf{P}^{l}\delta^{l}(\mathbf{a}^{l-1})^{\mathrm{T}}\\ =& \mathbf{P}^{l}\mathbf{W}^{l} - \alpha\mathbf{P}^{l}\left[(\mathbf{W}^{l+1})^{\mathrm{T}}\delta^{l+1}\odot\sigma'(\mathbf{n}^{l})\right]\,(\mathbf{a}^{l-1})^{\mathrm{T}}\\ =& \mathbf{P}^{l}\mathbf{W}^{l} - \alpha\left[(\mathbf{W}^{l+1}\mathbf{P}^{\mathrm{T}})^{\mathrm{T}}\delta^{l+1}\odot\sigma'(\mathbf{P}^{l}\mathbf{n}^{l})\right]\,(\mathbf{a}^{l-1})^{\mathrm{T}}\\ =& \mathbf{P}^{l}\mathbf{W}^{l} - \alpha[(\mathbf{W}^{l+1}(\mathbf{P}^{l})^{\mathrm{T}})^{\mathrm{T}}\delta^{l+1}\odot\sigma'(\mathbf{P}^{l}\mathbf{W}^{l}\mathbf{a}^{l-1}+\mathbf{P}^{l}\mathbf{b}^{l})](\mathbf{a}^{l-1})^{\mathrm{T}}.\end{aligned} \tag{4.A.10}$$

If we apply a permutation $\mathbf{P}^{l}$ to our update rule (equation 4.A.6) at any layer except the last, then using the above, we can express the gradient-based update as

$$(\hat{\mathbf{W}}^{l})_{\text{new}} = \hat{\mathbf{W}}^{l} - \alpha\left[(\hat{\mathbf{W}}^{l+1})^{\mathrm{T}}\delta^{l+1}\odot\sigma'(\hat{\mathbf{W}}^{l}\mathbf{a}^{l-1}+\hat{\mathbf{b}}^{l})\right]\,(\mathbf{a}^{l-1})^{\mathrm{T}}\square \tag{4.A.11}$$

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 through 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

$$\mathbf{a}_o = \mathbf{b}_o + \mathbf{K}_o \star \mathbf{x}, o \in \{1, \cdots, O\}, \tag{4.A.12}$$

where $\star$ denotes the discrete convolution operation.

Note that in contrast to the whole set of permutation matrices $\mathbf{P}^l$ (which were 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 to the permutation equivalence for the 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 first 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

$$\mathbf{a}_o = \mathbf{b}_o + \mathbf{K}_o \star \mathbf{x} = \quad \mathbf{b}_o + \mathbf{R}_o \mathbf{x}, \tag{4.A.13}$$

where $\mathbf{R}_o$ is the convolution matrix. We build the matrix $\mathbf{R}_o$ in this alternative form (4.A.13) 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 channel 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.*,

$$\mathbf{a}_{s_1} = \begin{bmatrix} \mathbf{a}_1 \\ . \\ \mathbf{a}_O \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ . \\ \mathbf{b}_O \end{bmatrix} + \begin{bmatrix} \mathbf{R}_1 \\ . \\ \mathbf{R}_O \end{bmatrix} \mathbf{x} = \mathbf{b}_f + \mathbf{R}_f \mathbf{x}, \tag{4.A.14}$$

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.*,

$$\begin{bmatrix} \mathbf{a}_{s_1} \\ . \\ \mathbf{a}_{s_s} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_f \\ . \\ \mathbf{b}_f \end{bmatrix} + \begin{bmatrix} \mathbf{R}_f & \mathbf{0} & ... & \mathbf{0} \\ . & . & . & . \\ \mathbf{0} & ... & \mathbf{0} & \mathbf{R}_f \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ . \\ \mathbf{x}_S \end{bmatrix} = \begin{bmatrix} \mathbf{b}_f \\ . \\ \mathbf{b}_f \end{bmatrix} + \mathbf{R} \begin{bmatrix} \mathbf{x}_1 \\ . \\ \mathbf{x}_S \end{bmatrix}, \tag{4.A.15}$$

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

$$\mathbf{a} = \mathbf{b} + \mathbf{R} \begin{bmatrix} \mathbf{x}_1 \\ . \\ \mathbf{x}_S \end{bmatrix}, \tag{4.A.16}$$

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 an equivalent form with equation (4.A.7), 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

$$\frac{\partial f(\mathbf{R})}{\partial R_{ij}} = \sum_k \sum_l \frac{\partial f(\mathbf{R})}{\partial R_{kl}} \frac{\partial R_{kl}}{\partial R_{ij}} = Tr \left[\left[\frac{\partial f(\mathbf{R})}{\partial \mathbf{R}} \right]^T \frac{\partial \mathbf{R}}{\partial R_{ij}} \right]. \tag{4.A.17}$$

Replacing $f()$ by $\mathcal{L}$ in the above and using the update rule equation (4.A.6) gives us the update equation for R_{ij}. Using similar argumentation and derivation that leads to equation (4.A.11) concludes the proof for permutation equivalence for a convolutional layer $\square$

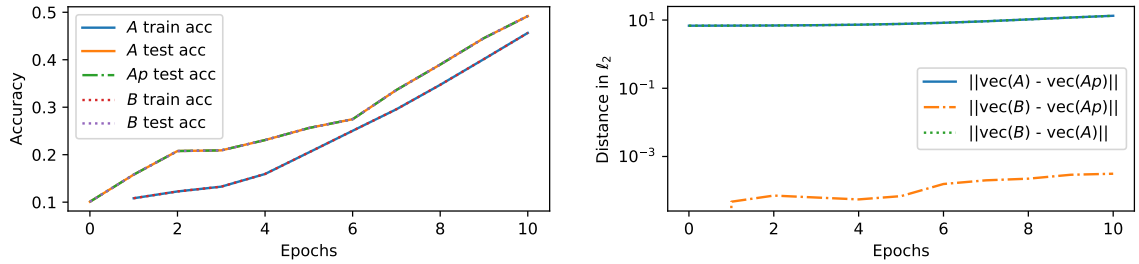


Figure 4.A.1: Empirical Evaluation of Permutation Equivalence. **Left:** Accuracies of orignal model A, permuted model Ap and model B trained from Ap's initialization. All models are indistinguishable in their accuracies. **Right:** pairwise distances of vectorized weights over epochs of models A, Ap 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.

Empirical Evaluation. We empirically confirm the permutation equivalence (see Figure 4.A.1). We begin by comparing the accuracies of permuted models (Figure 4.A.1 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 backward 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 backward pass, too.

To track how models develop in weight space, we compute the mutual ℓ_2 distances between the vectorized weights (Figure 4.A.1 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 between Ap and B is small, confirming the backward 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 changes the representation of a NN model, but not its mapping, *e.g.*, scaling of subsequent layers with piece-wise linear activation functions [39]. 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.

4.B Self-Supervised Loss Additional Details

Below, we give additional details about the losses in our self-supervised representation learning approach.

Encoder with Contrastive Loss. **($\mathbf{E}_c$)**. We use an encoder $g_\theta(\mathbf{w}_i)$ with parameters θ, where $\mathbf{z}_i = g_\theta(\mathbf{w}_i)$ denotes the neural representation with dimension of L. To learn the encoder parameters, we use the NT_Xent loss [20]. For a batch of M_B model weights, each sample $\mathbf{w}_i$ is randomly augmented twice and propagated through the encoder, which results in two *views* $\mathbf{z}_{i,v_1} = \mathbf{z}_i$ and $\mathbf{z}_{i,v_2}$ (positive pair). Given a batch of positive pairs $\mathbf{z}_i$ and $\mathbf{z}_{i,v_2}$ the loss is defined as

$$\mathcal{L}_E = \mathcal{L}_c = \sum_i -\log \frac{\exp\left(\text{sim}(\mathbf{z}_i, \mathbf{z}_{i,v_2})/\tau\right)}{\sum_{j\neq i} \exp\left(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau\right)},$$

where $\text{sim}(\mathbf{z}_i, \mathbf{z}_{i,v_2}) = \mathbf{z}_i^T \mathbf{z}_{i,v_2} / ||\mathbf{z}_i|| ||\mathbf{z}_{i,v_2}||$ is the cosine similarity and τ is the temperature parameter. The encoded negative samples $\mathbf{z}_j$ for each positive pair $(\mathbf{z}_i, \mathbf{z}_{i,v_2})$ are indexed by $j \neq i$. The configuration is denoted as ED.

Encoder-Decoder with Reconstruction Loss. **(ED)**. We denote the encoder as $g_\theta(\mathbf{w}_i)$ with parameters θ, while $\mathbf{z}_i = g_\theta(\mathbf{w}_i)$ denotes the neural representation with dimension L. We denote the decoder as $h_\psi(\mathbf{z}_i)$ with parameters ψ, while $\hat{\mathbf{w}}_i = h_\psi(\mathbf{z}_i) = h_\psi(g_\theta(\mathbf{w}_i))$ denotes the reconstructed NN weights. To learn the encoder and decoder parameters, we use the common MSE reconstruction loss

$$\mathcal{L}_{ED} = \mathcal{L}_{MSE} = \frac{1}{M} \sum_{i=1}^{M} \|\mathbf{w}_i - h_\psi(g_\theta(\mathbf{w}_i))\|_2^2.$$

The configuration is denoted as ED.

Encoder-Decoder with Reconstruction and Contrastive Loss. **($\mathbf{E}_c\mathbf{D}$)**. As above, this architecture also consists of an encoder and decoder. To learn its parameters, we use a reconstruction loss $\mathcal{L}_{MSE}$ and a contrastive loss defined on the representations $\mathbf{z}_i$, *i.e.*,

$$\mathcal{L}_{E_cD} = \mathcal{L}_{MSE} + \beta \mathcal{L}_c,$$

where $\mathcal{L}_{MSE}$ and $\mathcal{L}_c$ are defined as above and β is a Lagrangian parameter. We denote this configuration as E_cD.

Encoder-Decoder with Reconstruction and Modified Contrastive Loss. ($\mathbf{E}_{c+}\mathbf{D}$). As the reconstruction loss prevents mode collapse, which negative samples are commonly used for in contrastive learning, we further experiment with applying the contrastive loss only on positive pairs. Then, $\mathcal{L}_c$ simplifies to

$$\mathcal{L}_{c+} = \sum_i -\log \exp\left(\text{sim}(\mathbf{z}_i, \mathbf{z}_{i,v_2})/\tau\right)$$

and the resulting loss is

$$\mathcal{L}_{E_{c+}D} = \mathcal{L}_{MSE} + \beta\mathcal{L}_{c+}.$$

We denote this configuration as E_{c+}D.

A Note on the Composite Losses. We point out that in both composite losses $\mathcal{L}_{E_cD} = \mathcal{L}_{MSE} + \beta\mathcal{L}_c$ and $\mathcal{L}_{E_{c+}D} = \mathcal{L}_{MSE} + \beta\mathcal{L}_{c+}$, we use a Lagrangian parameter $\beta \geq 0$. It determines the strength of influence of each of the components in the loss. Low-valued $\beta > 0$ puts focus on the reconstruction, while high-valued $\beta > 0$ puts focus on the distinctiveness between the neural representations. Therefore, we experimented with different $\beta > 0$, and found out that β has to be chosen such that it represents a balanced trade-off between reconstruction and preservation of distinctiveness. In particular, we found that $\beta = 1$ works well in our experiments.

4.C Downstream Tasks Additional Details

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

4.C.1 Downstream Tasks Problem Formulation

We use linear probing as a proxy to evaluate the utility of the learned neural representations, similar to [59].

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 [170] 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 [56] 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), and initialization method (Init), we train a linear perceptron by minimizing a cross-entropy loss [56]. Here, instead of R^2 score, we measure the prediction accuracy.

4.C.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 [56] 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), and nonlinear thresholding function (TF), we solve the linear regression problem ($Q1$). Similarly to the previous task, our targets for the task of hyperparameters prediction are the actual hyperparameters values.

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

For categorical hyper-parameters, like activation function (Act), optimizer (Opt), and initialization method (Init), instead of regression loss ($Q1$), we train a linear perception by minimizing a cross-entropy loss [56]. Here, we also define the targets as detailed above. The only difference here is that the targets have discrete categorical values.

4.D Model Zoos Details

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 checkpoints
[159]	$\surd$	$\surd$	$\times$	$\times$
[43]	$\surd$	$\times$	$\surd$	$\times$
proposed zoos	$\surd$	$\surd$	$\surd$	$\surd$

Table 4.D.1: Overview of the characteristics for the model zoos proposed and used (existing) in this work.

In Table 4.D.1, 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.

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 4.D.2: Architecture configurations and modes of variation of our model zoos.

Figure 4.D.1: Visualization of samples representing the four basic shapes in our Tetris data set.

In Table 4.D.1 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 4.D.2 we provide the architecture configurations and exact modes of variation of our model zoos.

4.D.1 Zoos Generation Using Tetris Data

As a toy example, we first create 4x4 grey-scaled image data set that we call *Tetris* by using four tetris shapes. In Figure 4.D.1 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 an 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 4.D.3). We give an illustration of the used FFN architecture in Figure 4.D.4.

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 remainder crashed and are disregarded. So in total, this zoo contains $2900 \times 75 = 217500$ trained NN weights and biases.

4.D.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 4.D.4. We give an illustration of the used CNN architecture in Figure 4.D.5. 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 [159]). The `MNIST-HYP-1-RAND-SEED` model zoo contains CNN models, where per each model we draw and use 1 random seed and different hyper-parameter configurations.

We generate `MNIST-HYP-5-FIX-SEED` ensuring that for each hyper-parameter configuration, we add 5 models that share 5 fixed seeds. We build `MNIST-HYP-5-RAND-SEED` such that for each hyper-parameter configuration 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 4.D.2 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 that 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 methods and activation functions 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 benefit neural representation learning.

In Figure 4.D.3, 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.

4.D.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 zoo 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.

	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 4.D.3: FFN Architecture Details. CH-IN describes the number of input channels, CH-OUT the number of output channels.

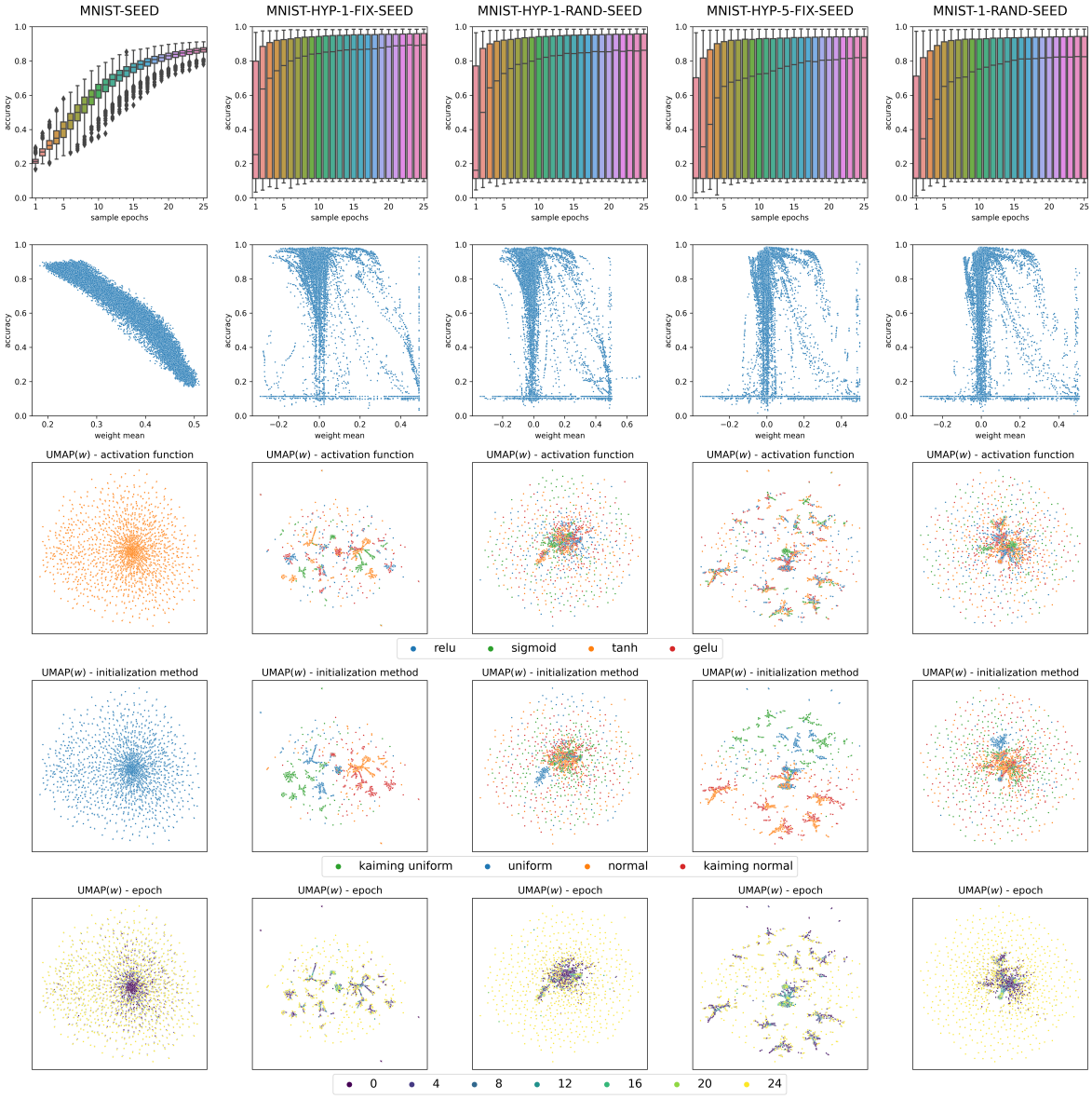


Figure 4.D.2: Visualization on the properties for the `MNIST-SEED`, `MNIST-HYP-1-FIX-SEED`, `MNIST-HYP-1-RANDOM-SEED`, `MNIST-HYP-5-FIX-SEED` and `MNIST-HYP-5-RANDOM-SEED` zoos. **Row One**. Boxplot of NNs accuracy over the epoch ids. **Row Two**. NNs accuracy plotted over the mean of the NNs weights 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 colored by activation function, initialization method, and sample epoch. Zoos with fixed seeds contain visible clusters of NNs that share the same initialization method or activation function. Zoos with varying hyperparameters and random seeds do not contain such clear clusters.

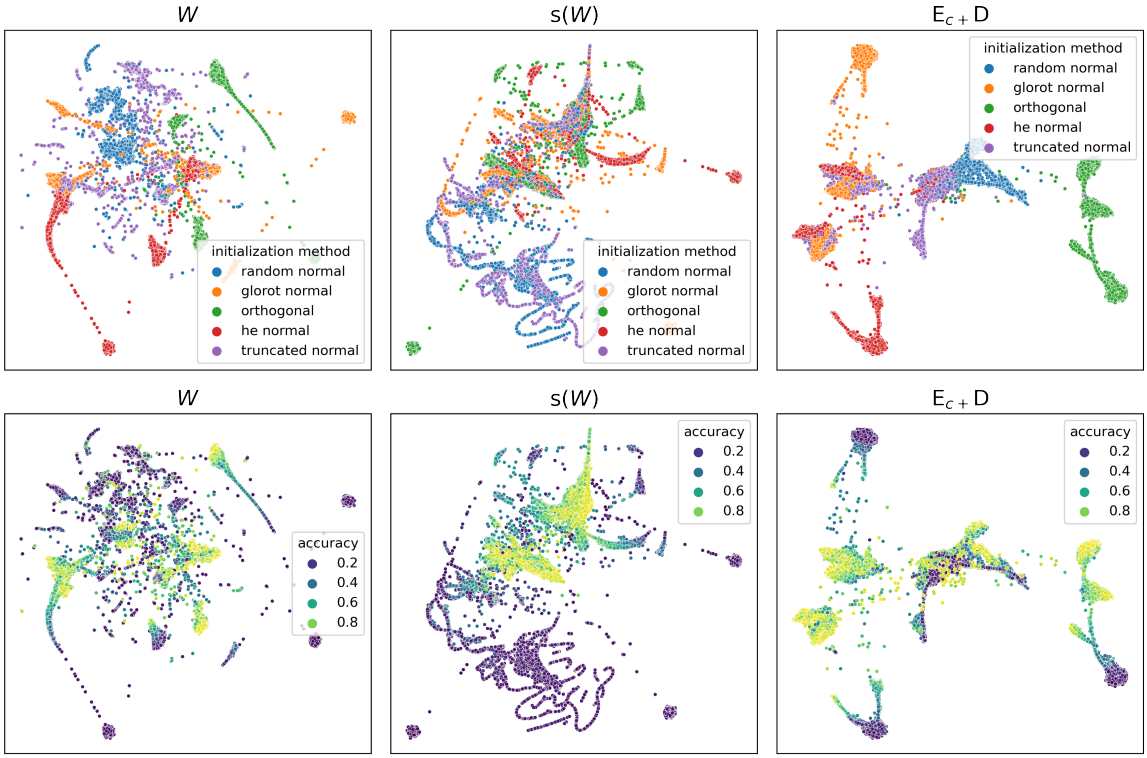


Figure 4.D.3: UMAP dimensionality reduction of the weight space (left), weight statistics (middle) and learned neural representations (right) for the MNIST-HYP zoo [159]. The initialization methods for the trained NN weights are already visually separable to a high degree in weight space, which carries 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 appear to cluster the models according to their initialization methods and within the clusters help to preserve high accuracy.

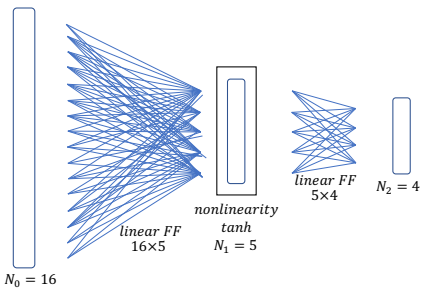


Figure 4.D.4: A diagram for the feed-forward architecture of the NNs in the `TETRIS-SEED` and `TETRIS-HYP` zoos.

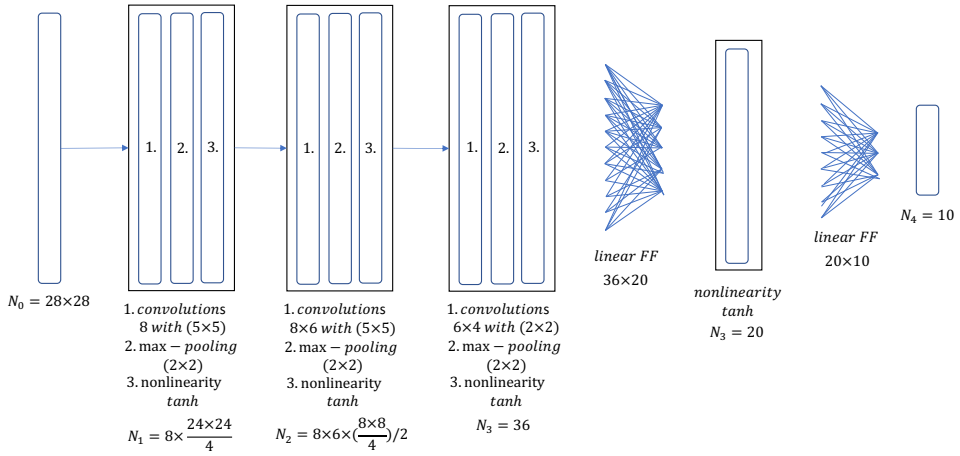


Figure 4.D.5: A diagram for the CNN architecture of the NNs in the `MNIST` zoos.

	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 4.D.4: 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.

4.E Additional Results

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

4.E.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 $E_c D$ (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 [86], 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 85% of the model zoos for training and 15% for testing. 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 a quality metric for self-supervised learning, we track the reconstruction R^2 on the test split of the zoo.

Results. As Table 4.E.1 shows, all NN architectures decrease in performance, as the compression ratio increases. The purely contrastive setup E_c generally learns embeddings that are useful for the downstream tasks but suffer from higher 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 the most under compression. $c = 3$ yields a neural representation with very low performance in downstream tasks. Higher compression ratios perform better on the downstream tasks but don't manage high reconstruction R^2. We interpret this as a sign that the combination of losses requires high-capacity bottlenecks. If the capacity is insufficient, the two objectives can't be both satisfied ($c = 3$). The training then prioritizes one of the two loss components ($c = 5, 10$).

Encoder with contrative loss E_c

c	Rec	Eph	Acc	F_{C0}	F_{C1}	F_{C2}	F_{C3}
2	–	82.0	77.5	56.1	59.3	63.6	67.4
3	–	84.4	74.5	53.8	56.7	53.9	71.9
5	–	75.4	67.1	45.7	56.7	52.2	59.8

Encoder and decoder with reconstruction loss ED

c	Rec	Eph	Acc	F_{C0}	F_{C1}	F_{C2}	F_{C3}
2	91.9	92.0	81.0	58.0	62.3	57.7	58.3
3	87.9	91.0	79.6	57.2	59.0	50.2	54.9
5	88.8	82.1	75.7	53.1	51.5	45.2	57.4

Encoder and decoder with reconstruction and contrastive loss E_cD

c	Rec	Eph	Acc	F_{C0}	F_{C1}	F_{C2}	F_{C3}
2	90.4	95.2	87.0	67.3	68.0	66.4	60.3
3	74.4	19.1	19.2	24.2	17.9	33.8	10.8
5	26.9	88.8	73.7	51.9	57.3	47.9	49.6
10	16.4	57.5	36.1	21.6	39.5	25.1	35.5

Table 4.E.1: 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 %.

4.E.2 NN Model Characteristics Prediction on FASHION-SEED

Due to space limitations, here in Figure 4.E.2, we present the results on the `FASHION-SEED` together with the results on `MNIST-SEED`. The experimental setup is the 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 seed 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 zoo.

4.E.3 In-distribution and Out-of-distribution Prediction

In Figures 4.E.1, 4.E.2 and 4.E.3 we show in-distribution and out-of-distribution comparative results for test accuracy, epoch id, and generalization gap prediction using the `MNIST-HYP` zoo.

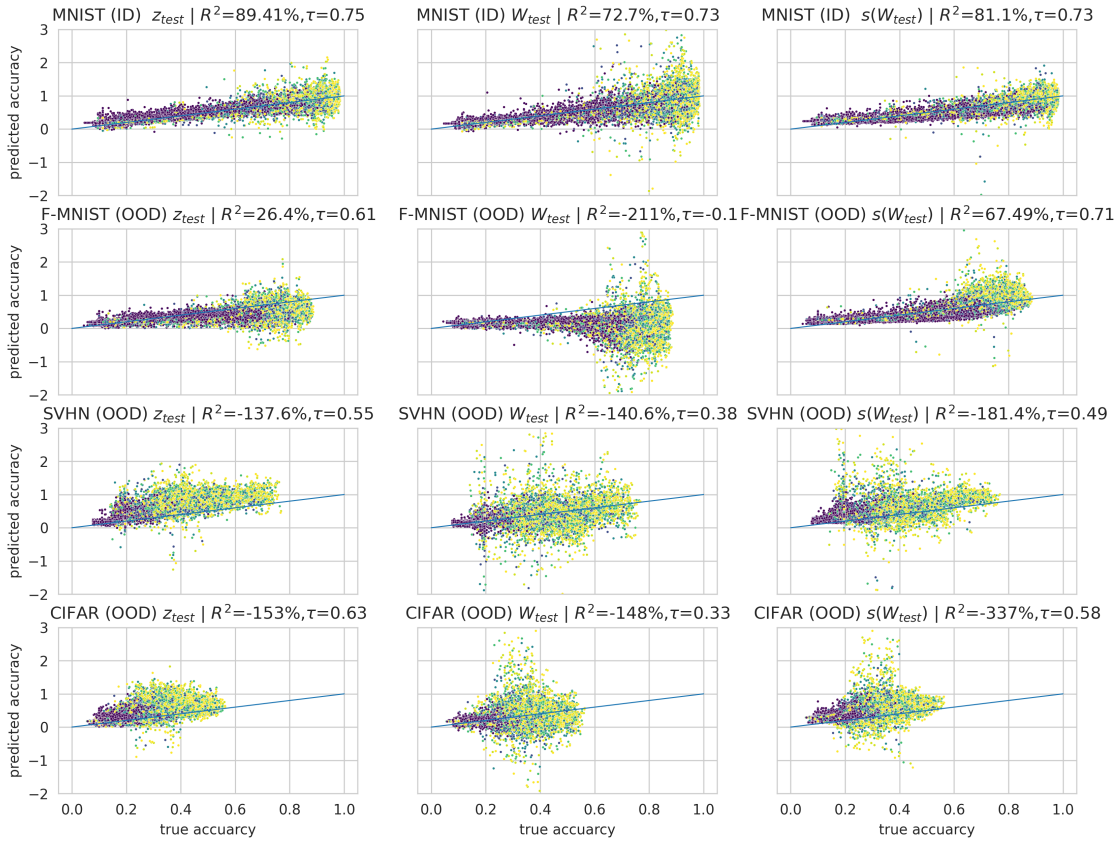
In the majority of the results for accuracy and generalization gap prediction, our

	MNIST-SEED			FASHION-SEED		
	W	S(W)	$E_{c+}D$	W	S(W)	$E_{c+}D$
EPH	54.9	**97.7**	95.9	51.8	**98.0**	95.7
ACC	82.8	**98.1**	97.5	57.6	**97.5**	96.1
GGAP	82.8	**98.5**	98.1	57.6	**97.5**	96.1

Table 4.E.2: R^2 score in % for epoch, accuracy and generalization gap.

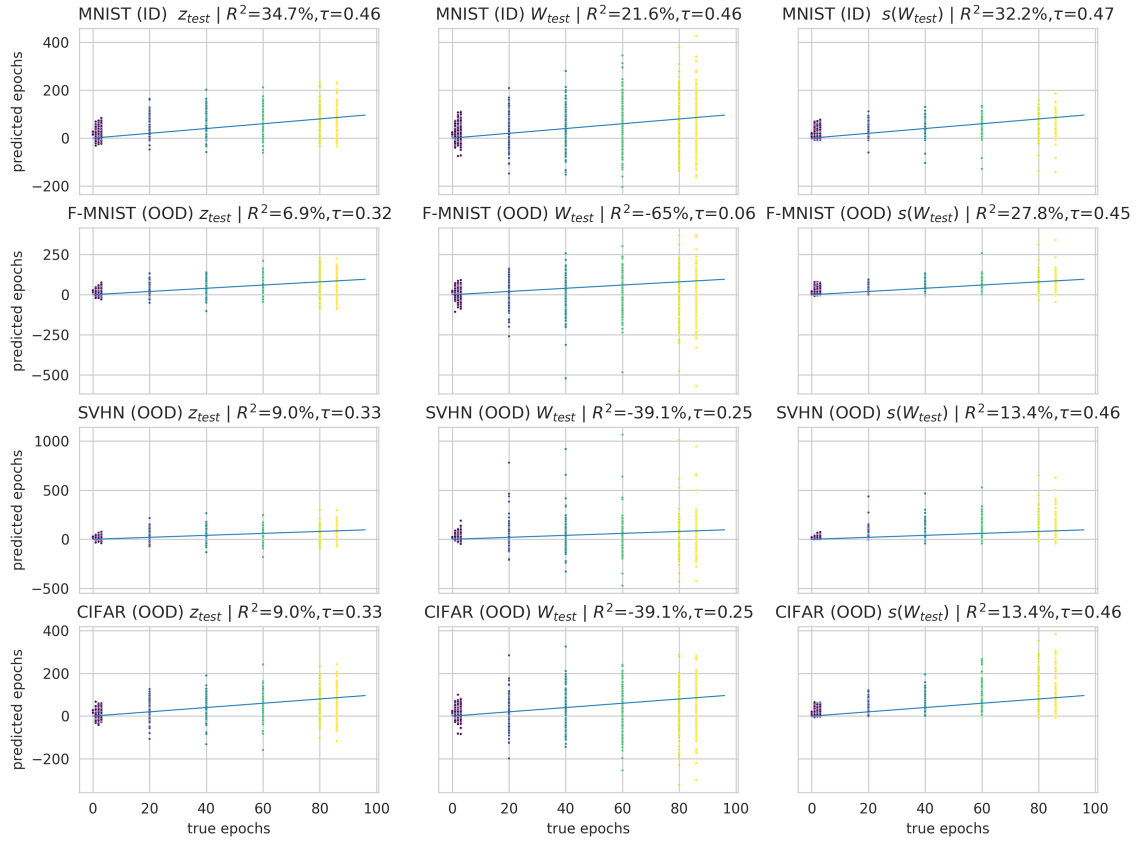
learned representations have higher R^2 and Kendall's τ score. Also, in the baseline methods the distribution of predicted target values is more dispersed compared to the true target values. On the epoch id prediction we have comparable results but with lower score, we attribute this to the fact that the zoos contain sparse checkpoints 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.

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 4.E.4. 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 noticeable 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.



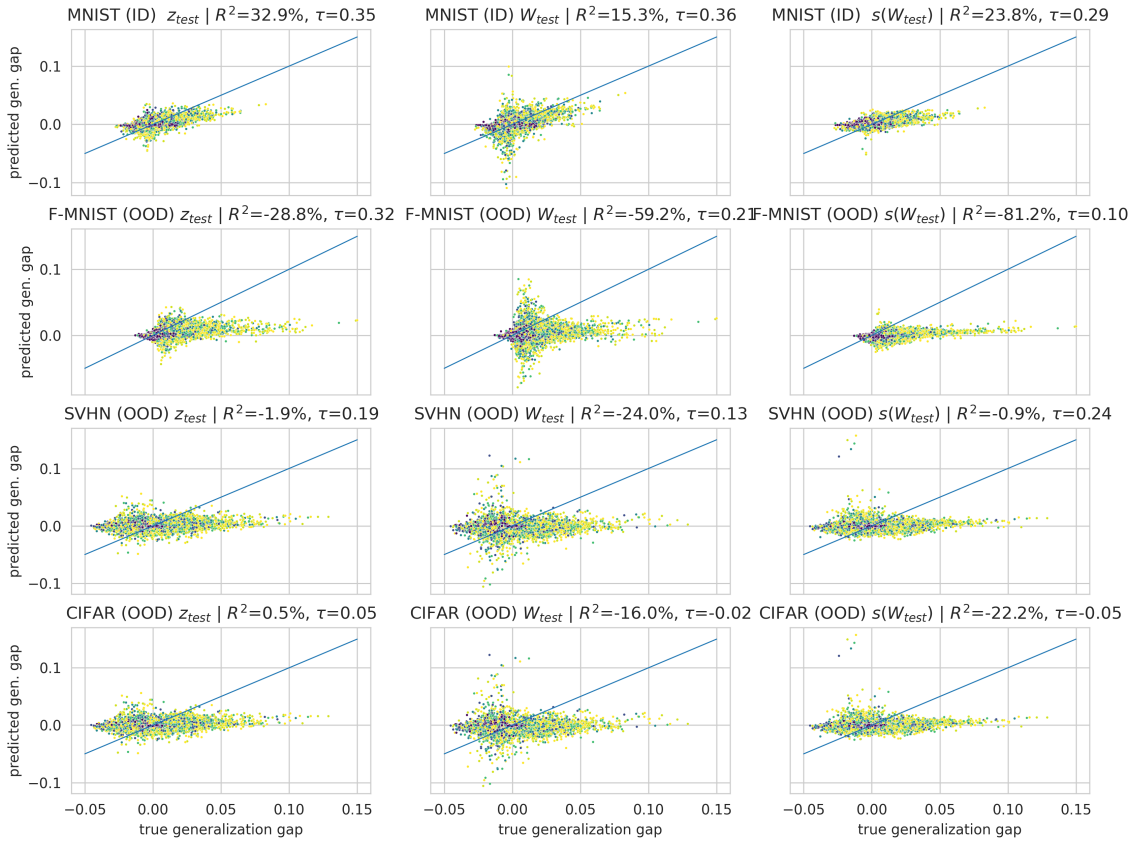
	MNIST-HYP			FASHION-HYP			SVHN-HYP			CIFAR10-HYP		
	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$
MNIST-HYP (τ)	.73	.73	**.75**	-.08	**.71**	.61	.38	.49	**.55**	.33	.58	**.63**
MNIST-HYP (R^2)	72.7	81.1	**89.4**	-211	**67**	26	-140	-180	**-137**	**-148**	-337	-153

Figure 4.E.1: 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.



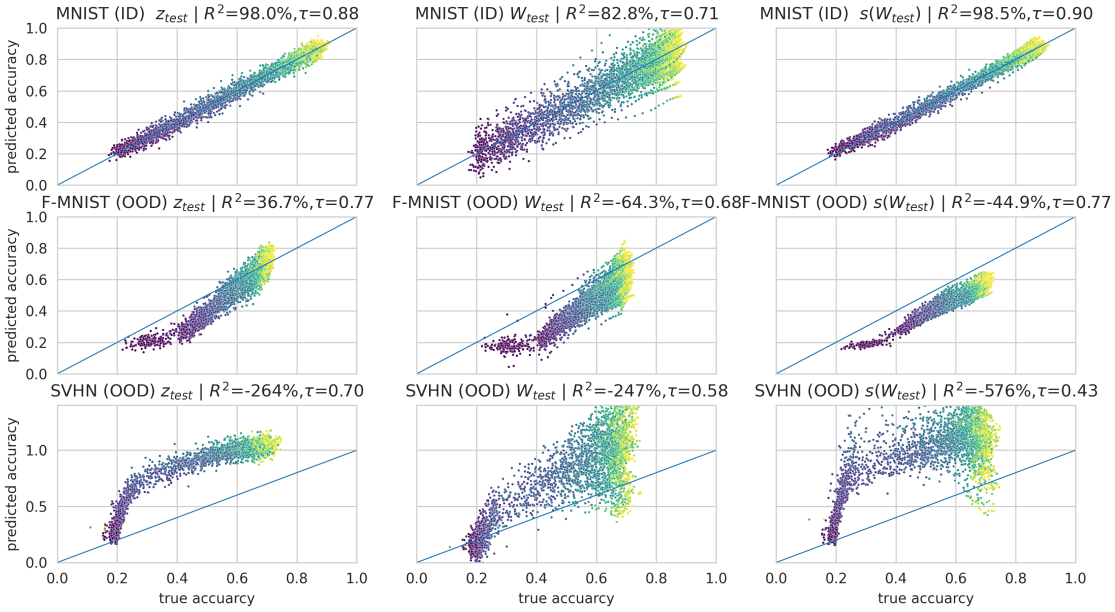
	MNIST-HYP			FASHION-HYP			SVHN-HYP			CIFAR10-HYP		
	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$	W	s(W)	$E_{c+}D$
MNIST-HYP (τ)	.46	**.47**	.46	.06	**.45**	.32	.25	**.46**	.33	.18	**.41**	.16
MNIST-HYP (R^2)	21.6	32.2	**34.7**	-64.9	**27.8**	6.9	-39.1	**13.4**	9.	-21.9	**19.2**	-13.

Figure 4.E.2: 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 4.E.3: 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)	.71	**.90**	.88	.68	**.77**	**.77**	.58	.43	**.70**
MNIST-SEED (R^2)	82.8	**98.5**	**98.0**	-64.3	-44.9	**36.7**	**-247**	-576	-264

Figure 4.E.4: 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.

Chapter 5

Hyper-Representations as Generative Models: Sampling Unseen Neural Network Weights

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 of outperforming 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.

This work was accepted for publication at NeurIPS 2022 [148]

5.1 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 [118, 147, 159]. According to [147], NNs populate a low dimensional manifold, which can be learned with an autoencoder via self-supervised learning directly from the model parameters (weights and biases) without access to the original image data and labels. This so-called *hyper-representation* has been demonstrated to be useful for predicting several model properties such as accuracy, hyperparameters, or architecture configurations.

However, [147] 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 initialization, (ii) ensemble sampling, and (iii) transfer learning. Our results demonstrate its capability to outperform previous hyper-representation learning and conventional baselines.

Previous work on generating model weights proposed (Graph) HyperNetworks [61, 88, 179], Bayesian HyperNetworks [36], HyperGANs [142] and HyperTransformers [182] 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.

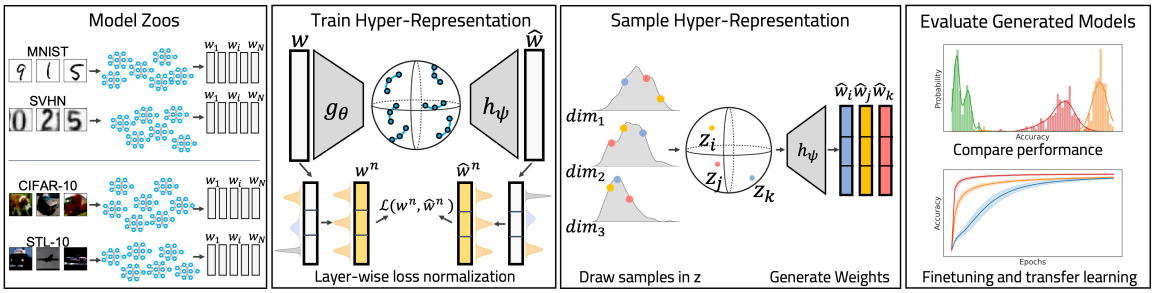


Figure 5.1.1: Outline of our approach: Model zoos are trained on image classification tasks. Hyper-representations are trained with self-supervised learning on the weights of the model zoos using layer-wise loss normalization in the reconstruction loss. We sample new embeddings in hyper-representation space and decode to weights. Generated models perform significantly better than random initialization or models sampled from baseline hyper-representations. Sampled models achieve high performance fine-tuned and transfer learned on new datasets.

The results suggest our approach (Figure 5.1.1) 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 are available at `https://github.com/HSG-AIML/NeurIPS_2022-Generative_Hyper_Representations`.

5.2 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* [147]. In [147] and this paper, a model zoo consists of models trained on the same task such as CIFAR-10 image classification [92]. 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 block [160]. The samples from model zoos are understood as sequences of convolutional or fully connected neurons. Each of the neurons is encoded as a token embedding and concatenated to form a sequence. The sequence is passed through several layers of multi-head self-attention. Afterward, 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 are added.

Training is done in a multi-objective 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 [147]). We can write the latter in a layer-wise way to facilitate our discussion in § 5.3.1:

$$\mathcal{L}_{MSE} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{l=1}^{L} ||\hat{\mathbf{w}}_i^{(l)} - \mathbf{w}_i^{(l)}||_2^2, \tag{5.2.1}$$

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.

5.3 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.

5.3.1 Layer-Wise Loss Normalization

We observed that hyper-representations as proposed by [147] 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.

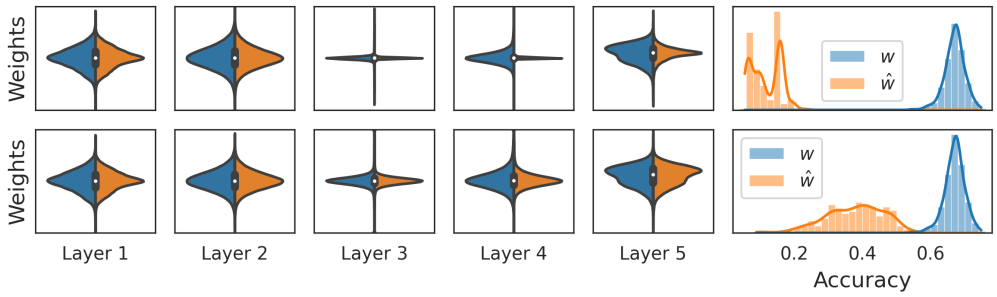


Figure 5.3.1: Comparison of the distributions of SVHN zoo weights $\mathbf{w}$ (blue) and reconstructed weights $\hat{\mathbf{w}}$ (orange) as well as their test accuracy on the SVHN test set. **Top:** Baseline hyper-representation as proposed by [147], the weights of layers 3, 4 collapse to the mean. 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 (5.2.1), 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 leads to undesired effects. Layers with broader distributions and large-magnitude weights are reconstructed well, while layers with narrow distributions and small-magnitude weights are disregarded. The latter layers can become a weak link in the reconstructed models, causing performance to drop significantly down to random guessing. The top row of Figure 5.3.1 shows an example of a baseline hyper-representation learned on the zoo of SVHN models [129]. Common initialization schemes [54, 64] 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 [88]. 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:

$$\mathcal{L}_{\bar{MSE}} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{l=1}^{L} \left\| \frac{\hat{\mathbf{w}}_i^{(l)} - \mu_l}{\sigma_l} - \frac{\mathbf{w}_i^{(l)} - \mu_l}{\sigma_l} \right\|_2^2 = \frac{1}{MN} \sum_{i=1}^{M} \sum_{l=1}^{L} \frac{\|\hat{\mathbf{w}}_i^{(l)} - \mathbf{w}_i^{(l)}\|_2^2}{\sigma_l^2}. \quad (5.3.1)$$

5.3.2 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 [87] instead of the autoencoder of [147]. While training VAEs on common domains such as images has become well-understood and feasible, in our relatively novel weight domain, we found it problematic (see details in Appendix 5.E). Other generative 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 [60, 109]. These methods assume the representation space to have strong regularities. The hyper-representation space learned by the autoencoder of [147] is already regularized by dropout regularization applied to the encoder and decoder as in [53]. 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 5.4.1 in § 5.4).

Given the smoothness and robustness of the learned hyper-representation space, we follow [53, 60, 109] 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 5.3.1), 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 5.4.3). Finally, as baseline and sanity check we explore sampling uniformly in representation space S_U and sampling in low-probability regions S_C.

Uniform S_U

As a naive baseline, we draw samples uniformly in hyper-representation space (bounded by tanh, § 5.2) 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 5.D.1 and 5.D.2 in Appendix 5.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.

Density estimation S_{KDE} and counterfactual sampling S_C

The dimensionality D of hyper-representations $\mathbf{z}$ in [147], 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 a 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.

Neighbor sampling $S_{\mathbf{Neigh}}$

Sampling neighbors of anchor samples $\{\mathbf{z}_i\}$ could be a simple and effective sampling strategy, but due to the high sparsity of the hyper-representation space, this strategy results in poor-quality samples. We therefore propose to use a neighborhood-based dimensionality reduction function $k : \mathbb{R}^D \to \mathbb{R}^d$ that maps $\mathbf{z}_i$ to low-dimensional embeddings $\mathbf{n}_i \in \mathbb{R}^d$ where sampling is facilitated. The assumption is that due to the low dimensionality of $\mathbb{R}^d$ (we choose $d = 3$) there will be fewer low-probability regions so that uniform sampling in $\mathbb{R}^d$ can be effective. Specifically, given low-dimensional embeddings $\mathbf{n}_i = k(\mathbf{z}_i)$, we sample $\mathbf{n}^*$ uniformly from the cube: $\mathbf{n}^* \sim U(min(\mathbf{n}), max(\mathbf{n}))$. Samples $\mathbf{n}^*$ are then mapped back to hyper-representations $\mathbf{z}^* = k^{-1}(\mathbf{n}^*)$. To preserve the neighborhood topology of $\mathbb{R}^D$ in $\mathbb{R}^d$ and enable mapping back to $\mathbb{R}^D$, we choose k to be an approximate inverse neighborhood-based dimensionality reduction function based on UMAP [120].

Latent space GAN $S_{\mathbf{GAN}}$

A common choice for generative representation learning is generative adversarial networks (GANs) [55]. While training a GAN directly to generate weights is a promising yet challenging avenue for future research [142], we found the GAN framework to work reasonably well when trained on the hyper-representations. This idea follows [60, 109] that showed improved training stability and efficiency compared to training GANs on inputs directly. We train a generator $G : \mathbb{R}^d \to \mathbb{R}^D$ 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 5.E.

5.4 Experiments

5.4.1 Experimental Setup

We train and evaluate our approaches on four image classification datasets: MNIST [98], SVHN [129], CIFAR-10 [92], STL-10 [25]. For each dataset, there is a model zoo that we use to train an autoencoder following [147].

Model zoos: In practice, there are already many available model zoos, e.g., on Hugging Face or GitHub, that can be used for hyper-representation learning and sampling. Unfortunately, these zoos are not systematically constructed and require further effort to mine

and evaluate. Therefore, in order to control the experiment design, ensure feasibility and reproducibility, we generate novel or use the model zoos of [147, 150] created in a systematic way. With controlled experiments, we aim to develop and evaluate inductive biases and methods to train and utilize hyper-representation, which can be scaled up efficiently to large-scale and non-systematically constructed zoos later. For 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 [147]. 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 [54, 64]. Each zoo is split into 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 [147]. Here the models have already achieved high performance, but have not fully converged. The development in the remaining epochs of each model is treated as hold-out data to compare against. We use the MNIST and SVHN zoos from [147] and based on them create the CIFAR-10 and STL-10 zoos. Details on the zoos can be found in Appendix 5.A.

Experimental details: We train separate hyper-representations on each of the model zoos. Images and labels are not used to train the hyper-representations (see § 5.2). Using the proposed sampling methods (§ 5.3.2), 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, then all populations are evaluated and fine-tuned on a different target dataset. Full details on training, including infrastructure and compute are detailed in the Appendix 5.B.

Baselines: As the first baseline, we consider the autoencoder of [147], which is the same as ours but without the proposed layer-wise loss-normalization (LWLN, § 5.3.1). 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 [21, 38, 90].

Reproducibility, reliability and comparability: We compare populations of at least 50 models to evaluate each method reliably. We report standard deviation in Tables 5.4.1-5.4.2 and statistical significance, effect size and 95% confidence interval in Appendix 5.F. To ensure fairness and comparability, all methods share training hyperparameters. Fine-tuning uses the hyperparameters of the target domain.

5.4.2 Results

In the following, we first analyze the learned hyper-representations further justifying our sampling methods and assumptions made in § 5.3.2. We then confirm the effectiveness of our approach for model initialization without and with fine-tuning in the in-dataset and transfer learning settings.

Hyper-Representations are Robust and Smooth

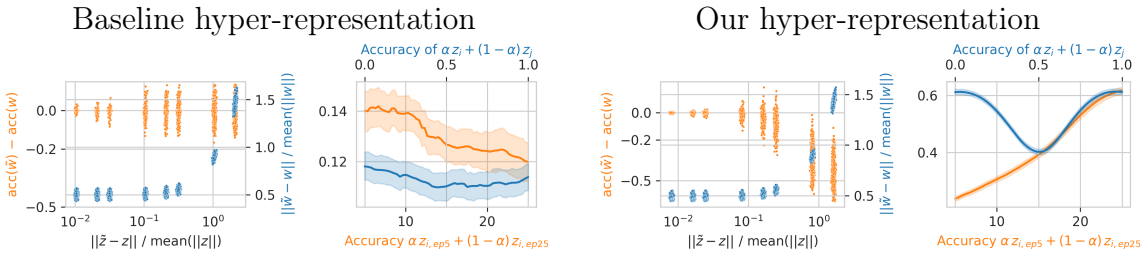


Figure 5.4.1: **(a,c):** Robustness of hyper-representations. For both baseline and our hyper-representation, relatively large levels of relative noise >10% are necessary to degrade the test accuracy (orange) or reconstruction (blue); see the text for further discussion. **(b,d):** Interpolations along model trajectories (orange) and between $\mathbf{z}$ of different models (blue) show the smoothness of our hyper-representation.

We evaluate the robustness and smoothness of the hyper-representation space with two experiments on the SVHN zoo. 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 SVHN classification task. We found that both the baseline as well as our hyper-representations are robust to noise as large levels of relative noise >10% are required to affect performance (Figure 5.4.1, a,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 the models' accuracies

on the classification task. For our model, we found remarkably smooth development of accuracy along the interpolation in both schemes (Figure 5.4.1, d). The lack of fluctuations along and between trajectories supports both local and global notions of smoothness in hyper-representation space.

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 5.4.1, b), while robustness can be misleading since the accuracy even without adding noise is already low (Figure 5.4.1, a). Therefore, LWLN together with regularizations added to the autoencoder allows for learning robust and smooth hyper-representation. This property makes sampling from that representation more meaningful as we show next.

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 25 epochs (Figure 5.4.2). 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.

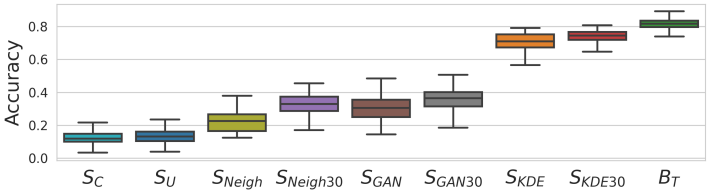


Figure 5.4.2: MNIST results of sampled weights (no fine-tuning) compared to training from scratch with SGD (B_T).

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 5.4.1 (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 5.3.1 top to bottom). Second, by fixing the weak links, the reconstructed models perform significantly better (see Appendix 5.C for more results).

In-dataset fine-tuning: When fine-tuning, our S_{KDE30} and baseline B_{KDE30} appear to gradually converge to similar performance (Table 5.4.1). While unfortunate, this result aligns well with previous findings that longer training and enough data make initialization less important [66, 123, 141].

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.

Table 5.4.1: Mean and std of test 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

Sampling ensembles: We found that a potentially 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 5.4.3). In contrast, ensembles generated using our S_{KDE30} gracefully improve with the ensemble

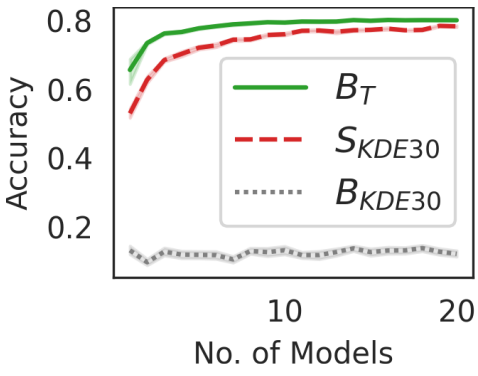


Figure 5.4.3: Generated ensembles evaluated on SVHN. Test accuracy is averaged over 15 ensembles of randomly chosen models.

size outperforming single B_T models and almost matching B_T ensembles with enough models in the ensembles. Remarkably, 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 that hyper-representations learned with LWLN generate models that are not only performant but also diverse. Although generating ensembles requires learning hyper-representation and model zoo first, we assume that in the future such a hyper-representation can be trained once and reused in unseen scenarios as we tentatively explore below (see results in Table 5.4.3 and the discussion therein).

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 5.4.1). We therefore explore the question, if reconstructed models develop in weight space in the same direction as their original, or find a different solution. On SVHN, we found that the reconstructed models ($\hat{\mathbf{w}}$) after one epoch of fine-tuning perform similar to their originals ($\mathbf{w}$) and slightly outperform from there on (Figure 5.4.4, left). At the same time, pairs of original and reconstructed models move further apart and become less aligned in weight space (Figure 5.4.4, 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. Combining this with the ensembling results in Figure 5.4.3, hyper-representations do not collapse to a single solution but decode to diverse and useful weights.

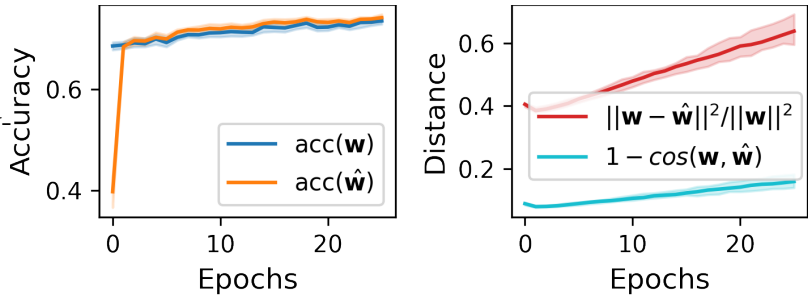


Figure 5.4.4: Progression of test accuracy (left) and distance (right) between weights during fine-tuning on SVHN; $\mathbf{w}$ – initialization with the weights trained using SGD for 25 epochs; $\hat{\mathbf{w}}$ – initialization with reconstructed weights.

Sampling Initializations for Transfer Learning

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 5.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 5.4.2: Transfer-learning results (mean and standard deviation of the test 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 § 5.4.3.

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

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 compare favorably to B_F (Figure 5.4.5, Table 5.4.2). In the STL-10 to CIFAR-10 experiment, all populations appear to saturate with only small differences in their performances (Table 5.4.2). 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 5.F). Interestingly and other than in related work [122], 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.

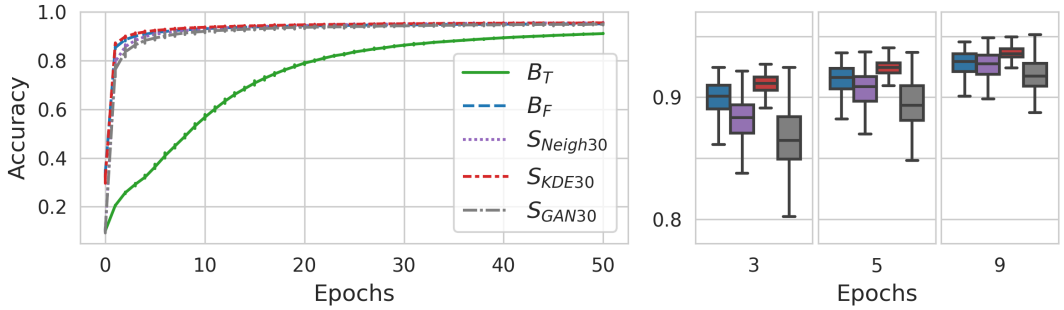


Figure 5.4.5: SVHN to MNIST transfer learning experiment: test accuracy over epochs. Our sampling methods outperform the baselines after the first epoch. **Left:** epochs from 0 to 50. **Right**: epochs from 3 to 9, where B_T is significantly lower than 80% and thus is not visible.

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 the 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 [16]. Our results in Table 5.4.3 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 promising, as the hyper-representations were trained on single-dataset zoos.

Table 5.4.3: Test accuracy (%) of models generated conditioned on the models of unseen zoos.

Training zoo	**Conditioning unseen**	**Mean / max 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**

Sampling Initializations for Unseen Architectures

Generalization to unseen large architectures with complex connectivity (ResNet, MobileNet, and EfficientNet) is a very interesting and ambitious research problem. As a step towards that goal, we perform experiments in which we attempted to use our hyper-representation beyond the same simple architecture. Surprisingly, our results indicate the promise of leveraging the hyper-representation for more diverse architectures and settings. Further experiments investigating the cross-architecture generalization capabilities of hyper-representations can be found in Appendix 5.D.

Setup: With this experiment, we aim to verify if it is possible to adapt our approach to architectures not seen during training, e.g., with skip connections and/or with more layers. We follow the transfer-learning setup of § 5.4.2 and use an existing MNIST hyper-representation to sample weights as initialization for training on SVHN. However, we now also vary the architecture. While the decoder outputs a fixed-sized vector of weights, we can assign these weights to new architectures by either making sure that the new architecture still has the same number of parameters or by initializing randomly the extra parameters introduced. Specifically, we create three cases: (1) we add ResNet-style skip connections [65] (1x1 conv) to the convolutional layers (3-conv + res-skip), (2) re-distribute the weights to smaller four convolutional layers (4-conv), (3) re-distribute to smaller four convolutional layers and add identity skip connections (4-conv + id.-skip).

Results: Surprisingly, despite training our hyper- representation on the models of the same architecture, generated weights for all three cases outperform random initialization and converge significantly faster across all the variations (Table 5.4.4). In all the variations even just after 5 epochs, the models with generated weights are better than training the baseline for 50 epochs. In the 3-conv + res-skip experiments, some models in both populations did not learn, which leads to high standard deviation. Further analysis is required to explain the gains of our approach in this challenging setup. To extend and scale up our method further, future work could combine it with the methods of growing networks [19, 163], so that some layers are generated while some are initialized in a sophisticated way to preserve the functional form of the network.

Table 5.4.4: Test accuracy (%) on SVHN of populations with generated weights compared to models trained from scratch B_T. Best results for each epoch and dataset are bolded. `r. i.` indicates random initialization, `gen.` denotes weights generated with our (S_{KDE30}).

Initialization	Epoch 1	Epoch 5	Epoch 50
3-conv (r. i.) + res-skip (r.i.)	18.9±1.6	31.4±17	50.6±28
3-conv (gen.) + res-skip (r.i.)	**34.5±14**	**60.5±21**	**68.0±21**
4-conv (r.i.)	19.2±1.0	19.2±0.9	55.2±11
4-conv (gen.)	**44.0±4.5**	**57.8±3.5**	**67.6±1.9**
4-conv + id.-skip (r.i.)	18.9±1.0	19.6±1.7	56.4±7.9
4-conv + id.-skip (gen.)	**48.0±4.0**	**59.9±2.5**	**66.4±1.7**

5.4.3 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 small scale as in [147]. While on MNIST and SVHN, the architectures of such model zoos allowed us to achieve high performance, on CIFAR-10 and STL-10, the performance of all populations is limited by the low capacity of the model zoo's 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 5.F for details. 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. Larger model architectures might mitigate this behavior. Corresponding model zoos have recently been made available in [150] to tackle this issue[1].

5.5 Related Work

HyperNetworks: Recently, 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 [9, 61, 88, 179]. HyperNetworks have also been extended to meta-learning by conditioning weight generation on data [143, 182]. Closely related to our work, HyperGANs [142] can sample model weights by combining the hypernetworks and the GAN framework. Similarly, [36] 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 [147]. Our methods build on that work to allow for better reconstruction and sampling. [35] showed that given a few parameters of a network, the remaining values 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. Very recently, [136] used diffusion on a population of models to generate model weights for the original task via prompting.

Transfer Learning: Transfer learning via fine-tuning aims at re-using models and their learned knowledge from a source to a target task [21, 38, 90, 122, 176]. Transfer learning models make training less expensive, boost performance, or allow training on

[1] www.modelzoos.cc

datasets with very few samples and have been applied on a wide range of domains [188]. The common transfer learning methods however only consider transferring from a single model, and so disregard the large variety of pre-trained models and the potential benefit of combining them.

Knowledge distillation: Our work is related to [108, 153, 165] that allows 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 [108, 153]. Knowledge distillation however requires the source models at training as in [108] and at inference as in[153] thus increasing memory cost. Further, the learned knowledge cannot be shared between different target models. **Learnable initialization** [31, 187] 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.

5.6 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 of outperforming both models trained from scratch, as well as pre-trained and fine-tuned models. Populations of sampled models, even for some unseen architectures, 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.

Appendix

5.A Model Zoo Details

The model zoos are generated following the method of [147, 150] An overview of the model zoos is given in Table 5.A.1. All model zoos share one general CNN architecture, outlined in Table 5.A.2. The hyperparameter choices for each of the populations are listed in Table 5.A.3. The hyperparameters are chosen to generate zoos with smooth, continuous development and spread in performance.

Table 5.A.1: Model zoo overview.

Zoo	**Channels**	**Parameters**	**Population Size**
MNIST	1	2464	1000
SVHN	1	2464	1000
CIFAR-10	3	2864	1000
STL-10	3	2864	1000

Table 5.A.2: CNN architecture details for the models in model zoos.

Layer	**Component**	**Value**
Conv 1	input channels	1/3
	output channels	8
	kernel size	5
	stride	1
	padding	0
Max Pooling	kernel size	2
Activation	tanh / gelu	
Conv 2	input channels	8
	output channels	6
	kernel size	5
	stride	1
	padding	0
Max Pooling	kernel size	2
Activation	tanh / gelu	
Conv 3	input channels	6
	output channels	4
	kernel size	2
	stride	1
	padding	0
Activation	tanh / gelu	
Linear 1	input channels	36
	output channels	20
Activation	tanh / gelu	
Linear 2	input channels	20
	output channels	10

5.B Hyper-Representation Architecture and Training Details

Hyper-representations are learned with an autoencoder based on multi-head self-attention. The architecture is outlined in Figure 5.B.1. Convolutional and fully connected neurons are embedded to token embeddings of dimension d_{token}. Learned position encodings are added to provide relational information. A learned compression token (CLS) is appended to the sequence of token embeddings. The sequence of token embeddings is passed to N_{layers} layers of multi-head self-attention with N_{heads} heads

Table 5.A.3: Hyperparameter choices for the model zoos.

Model Zoo	**Hyperparameter**	**Value**
MNIST	input channels	1
	activation	tanh
	weight decay	0
	learning rate	3e-4
	initialization	uniform
	optimizer	Adam
	seed	[1-1000]
SVHN	input channels	1
	activation	tanh
	weight decay	0
	learning rate	3e-3
	initialization	uniform
	optimizer	adam
	seed	[1-1000]
CIFAR-10	input channels	3
	activation	gelu
	weight decay	1e-2
	learning rate	1e-4
	initialization	kaiming-uniform
	optimizer	adam
	seed	[1-1000]
STL-10	input channels	3
	activation	tanh
	weight decay	1e-3
	learning rate	1e-4
	initialization a	kaiming-uniform
	optimizer	adam
	seed	[1-1000]

with hidden embedding dimension d_{hidden}. The CLS token is compressed to the bottleneck of dimension d_z with an MLP or a linear layer. For the decoder, an MLP or a linear layer maps the bottleneck to a sequence of token embeddings. The sequence is passed through another stack of multi-head self-attention, which is symmetric to the encoder. Debedders map the token embeddings back to convolutional and fully connected neurons. The reconstruction and contrastive loss are balanced with a parameter β. The contrastive loss is computed on the embeddings $\mathbf{z}$ mapped through a projection head $\bar{\mathbf{z}} = p(\mathbf{z}$, where p is a learned MLP with four layers with 400 neurons each and $\bar{\mathbf{z}}$ has 50 dimensions. In Table 5.B.1, the exact hyper-parameters for each of the hyper-representations are listed to reproduce our results.

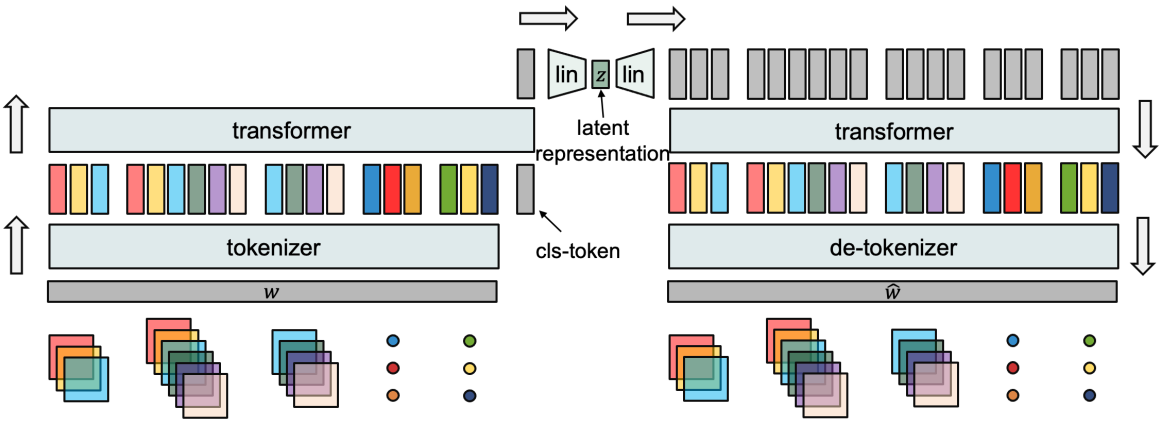


Figure 5.B.1: Schematic of the auto-encoder architecture to learn hyper-representations.

Table 5.B.1: Hyper-representation architecture and training details.

	MNIST	SVHN	CIFAR-10	STL-10
		Architecture		
d_{inpot}	2464	2464	2864	2864
d_{token}	972	1680	1488	1632
d_{hidden}	1140	1800	1164	1680
N_{layers}	2	4	2	4
N_{heads}	12	12	12	24
d_z	700	1000	700	700
Compression	linear	linear	linear	linear
		Training		
Optimizer	Adam	Adam	Adam	Adam
Learning rate	0.0001	0.0001	0.0001	0.0001
Dropout	0.1	0.1	0.1	0.1
Weight Decay	1e-09	1e-09	1e-09	1e-09
β	0.977	0.920	0.950	0.950
training epochs	1750	1750	500	2000
batch size	500	250	200	200

5.C Evaluation of Layer-Wise Loss Normalization

To evaluate layer-wise loss normalization, we compare two hyper-representations with comparable reconstruction. Both have a $R^2 = 1 - \frac{mse(\hat{\mathbf{w}},\mathbf{w})}{mse(\mathbf{w_{mean}},\mathbf{w})}$ as a measure of the explained variance of around 70%. One is trained with the baseline hyper-representation MSE, the other with layer-wise-normalization. Figures 5.C.1 and 5.C.2 show the distribution of weights per layer before and after reconstruction, as well as the accuracy

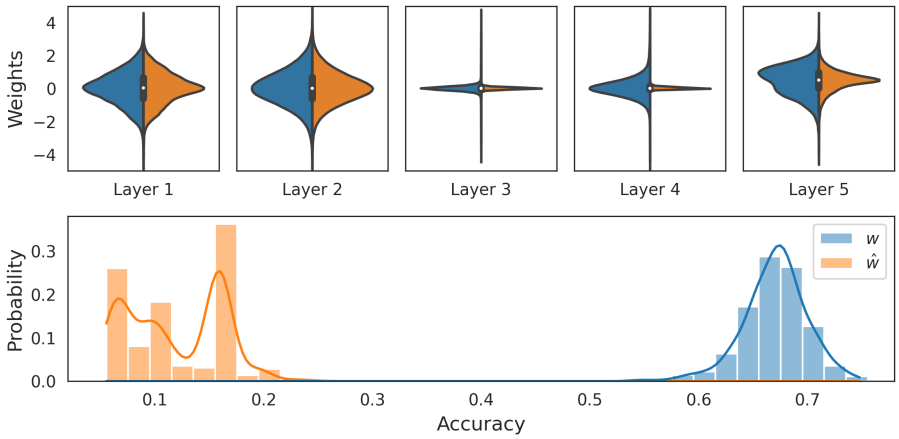


Figure 5.C.1: **Top:** Weight distribution per layer (1-5) of the SVHN test set before w and after reconstruction $\hat{w}$ with the baseline hyper-representation training loss. Layers 3 and 4 have small weight distributions, therefore adding little penalty to the MSE, and are consequently poorly reconstructed. **Bottom:** Accuracy distribution of the same population before and after reconstruction. The badly reconstructed layers (top) cause the reconstructed models to perform around random guessing.

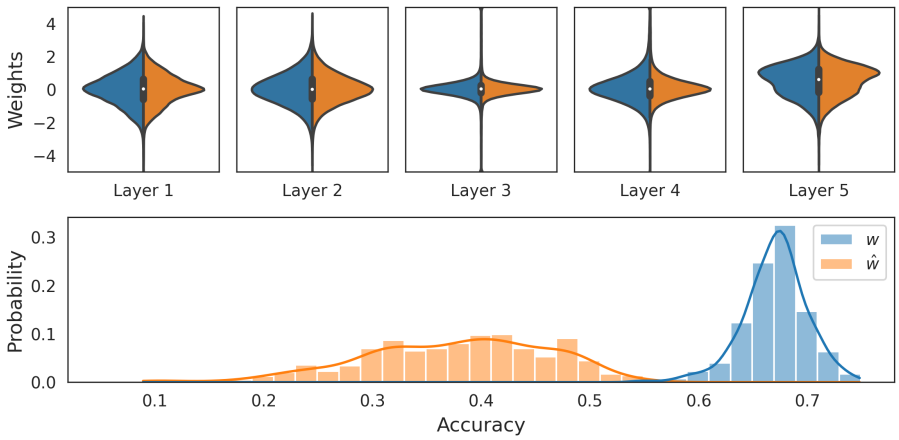


Figure 5.C.2: **Top:** Weight distribution per layer (1-5) of the SVHN test set before w and after reconstruction $\hat{w}$ with layer-wise loss normalization. The distributions of all layers are more similar, the reconstruction is equally distributed across the layers. **Bottom:** Accuracy distribution of the same population before and after reconstruction. The normalization fixes the catastrophic failure of the models. The remaining loss in accuracy can be explained by the remaining reconstruction error.

distribution of both populations on the SVHN image test set. With the baseline learning scheme in Figure 5.C.1, the distributions in layers 3 and 4 do not match. In these layers, the original weight distribution is smaller, and so there is only a small error even if the reconstructions predict the mean. These layers become a weak link of the reconstructed models and cause performance around random guessing. With layer-wise

loss normalization in Figure 5.C.2, the weight distribution between the layers becomes more similar. As a consequence, the reconstruction error is more evenly distributed across the layers, there are no single layers that aren't reconstructed at all. This appears to allow information to flow forward through the model and significantly improves the performance of reconstructed models. We find layer-wise-normalization necessary to reconstruct or sample functional models across all populations, where the weights are unevenly distributed.

5.D Hyper-Representation Analysis

In this section, we detail the analysis of hyper-representations. We begin with their geometry, followed by the distributions of individual dimensions of hyper-representations, and finally investigate robustness and smoothness.

Embeddings in Hyper-Representation Space Populate a Hyper-Sphere We analyze the geometry of hyper-representations $\mathbf{z}$. The space of hyper-representations is bounded to a high dimensional box by a tanh activation. Surprisingly, hyper-representations do not populate the entire space, but sections on a shell of a high-dimensional sphere. Figure 5.D.1 shows the distribution of the norm of the embeddings of the MNIST zoo. All embeddings are distributed on a small band between lengths 10 and 12, therefore they must populate the shell of a hyper-sphere. In Figure 5.D.2 we investigate pairwise cosine distances between the embeddings of the MNIST zoo. The majority of the embeddings populate the region between 0.6 and 0.8. The outliers around 1.0 are embeddings of the same model at different epochs. This indicates that models are not entirely orthogonal, but mutually equally far apart, populating a section of the shell of the hyper-sphere. While hyper-spheres are commonly found in embeddings of contrastive learning [81], in our experiments hyper-spheres form even without a contrastive loss. Properties of the models embedded on that hyper-sphere can be predicted from hyper-representations, therefore the topology on the sphere appears to encode model properties.

Distributions of Dimensions of Embeddings in Hyper-Representation Encode Properties Previous work showed that linear probing from hyper-representations accurately predicts i.e. model accuracy. In these linear probes, the individual z dimensions each linearly contribute to accuracy predictions. This allows us to investigate z dimensions independently. Figure 5.D.3 shows examples for the distribution of selected individual dimensions of hyper-representations $\mathbf{z}$. On the left is the distribution of the

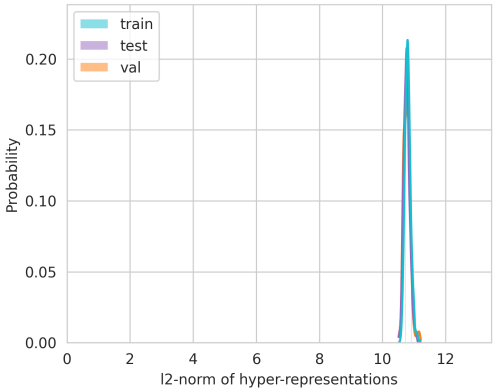


Figure 5.D.1: Distributions of ℓ_2 norm of hyper-representations $\mathbf{z}$ of the MNIST zoo.

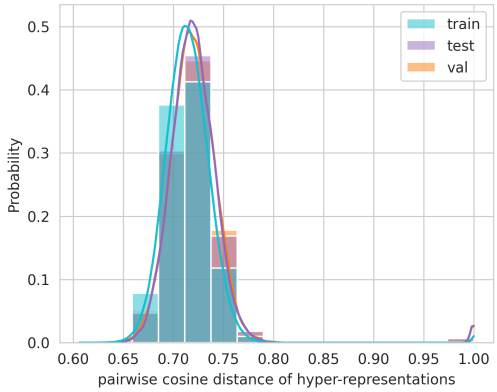


Figure 5.D.2: Distributions of pairwise cosine distance of hyper-representations $\mathbf{z}$ of the MNIST zoo.

entire population, and on the right of the top 30 % performing models. The individual dimensions show different types of distributions, with different modes. Most have a zero mean and span 3/4 of the available range, but some collapse to either -1 or 1. Further, the distributions also differ in at least some dimension between the entire population, and the better-performing split of the population.

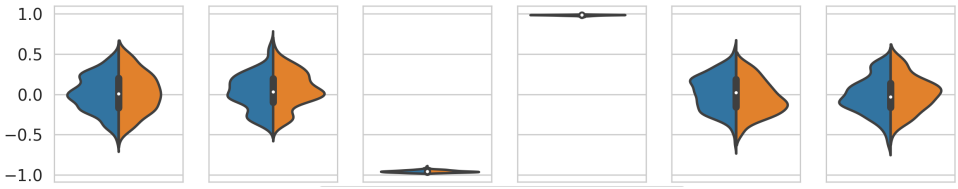


Figure 5.D.3: Distributions of individual dimensions of hyper-representations $\mathbf{z}$ of the MNIST zoo. In blue is the distribution of all samples, in orange the subset of the 30 % best samples.

Generalization Capabilities of Hyper-Representations to Diverse Model Zoos There are certain architectural changes such as adding/removing/changing pooling layers and nonlinearity that do not change the number of parameters (the dimensionality of the input/output required by our approach). These changes as well as changes in hyperparameters used to train models in a zoo may drastically alter the distribution of weights and pose a challenge to the proposed approach. Modern neural networks (ResNet, MobileNet, EfficientNet, etc.) are often trained with very different hyperparameters. With the experiment below, we investigate the generalization capabilities of hyper-representations to such changes, which might be important for modern large-scale settings as well.

Setup: We experimentally evaluate the generalizability of the proposed approach on models trained with a different choice of nonlinearity or other hyperparameters with two experiments (a and b). To that end, in addition to the original SVHN test zoo (zoo 1), we use two more diverse SVHN zoos (zoo 2 and zoo 3). In zoo 2, in addition to random seed, models differ in the activation (tanh, relu, gelu, sigmoid), l2-regularization (0, 0.001, 0.1), and dropout (0,0.3,0.5). In zoo 3 (extending zoo 2), we increase the diversity further by additionally varying the initialization method (uniform, normal, kaiming-uniform, kaiming-normal) and the learning rate (0.0001, 0.001, 0.01).

Table 5.D.1: Generalizability of hyper-representations towards more diverse model zoo configurations (measured as the reconstruction score, higher is better).

Training zoo	**Test zoo 1 original**	**Test zoo 2 vary activation**	**Test zoo 3 vary hyperparameters**
Original	81.9%	45.7%	38.9%
Diverse (zoo 3)	25.8%	89.1%	75.6%

Experiment (a): We first evaluate our original encoder-decoder trained on a model zoo varying in random seed only. For evaluation, we pass the test splits of zoo 2 and zoo 3 through the encoder-decoder. We measure the reconstruction R^2 score of the original encoder-decoder on the diverse test zoos.

Results: Our results (Table 5.D.1) indicate that our original encoder-decoder can still encode and decode weights even in such a challenging setting, although there is an expected drop in performance.

Experiment (a): We next evaluate if hyper-representations can be trained on diverse zoos. For this experiment, we train a hyper-representation on the train split of zoo 3. With this, we aim to show that training hyper-representations on diverse zoos improve generalization capabilities further.

Results: Our results show that training on diverse zoos is a much more difficult task to optimize, hence the reconstruction on the original zoo degrades. It nonetheless improves the reconstruction results on the test split of the diverse zoos 2 and 3. This indicates that varying seeds and hyperparameters may be different aspects of complexity that need to be considered.

5.E Sampling Methods

VAE

A common extension of the autoencoder of [147] to enable sampling from its latent representation is to make the autoencoder variational [87]. In our experiments, VAEs could not be trained to reconstruct model weights without unweighting the KL-divergence to insignificance essentially making it deterministic as in [147]. Empirically, embeddings in hyper-representations are mapped on the shell of a sphere (see Section 5.D) and leave the inside of the sphere entirely empty. On the other hand, a Gaussian prior allocates most of the probability mass near the center of the sphere. It therefore appears plausible that the two may be incompatible. That issue of non-compatible priors is well known. [53] find that regularizing embeddings and decoder yields equally smooth representation spaces as VAEs without restrictions to specific priors. During training of hyper-representations, both encoder and decoder are regularized with a small ℓ_2 penalty. Further, dropout is applied throughout the autoencoder, which serves as another regularizer and adds blurriness to the embeddings. The combination of dropout, the erasing augmentation and the contrastive loss further regularizes the hyper-representation space. In all our sampling methods, we draw samples from probability distributions, which effectively disconnects the drawn samples from training embeddings.

Latent Space GAN Details

The generator and discriminator of our GAN consist of four fully-connected layers interleaved with ReLU nonlinearities. The same architecture and training hyperparameters are used for all experiments. The generator's input is a Gaussian noise $\mathbf{n}^*$ of dimensionality $d = 16$, the hidden dimensionalities are 128, 256, and 512, and the output dimensionality is equal to the hyper-representation length D. The discriminator's input is D-dimensional, the hidden dimensionalities are 1024, 512, and 256, and the output dimensionality is a scalar denoting either a real or fake sample. The discriminator is regularized with Spectral Norm [124]. The discriminator and generator are trained for 1000 epochs and batch size 32 using Adam with a two-time-scale update rule [69]: learning rate is 1e-4 for the generator and 2e-4 for the discriminator.

5.F Full Experiment Results

Table 5.F.1: Accuracy of sampled models: median and 95% confidence intervals. On the main diagonal are in-dataset experiments, otherwise transfer-learning from source to target. Bold numbers highlight the best source-to-target results. N/A denotes cases, in which the bootstrapped CI on the median could not be computed.

Population	Source	Target	
		MNIST	SVHN
B_T		91.1 [91.1, 91.2]	72.3 [72.0, 72.4]
B_F	MNIST	91.2 [91.0, 91.3]	76.2 [75.8, 76.5]
S_{KDE}		92.3 [92.1, 92.8]	76.7 [76.2, 77.0]
S_{KDE30}		93.1 [92.9, 93.4]	77.2 [76.8, 77.6]
S_{Neigh}		93.4 [93.2, 93.5]	76.8 [76.4, 77.1]
S_{Neigh30}		**94.0 [93.8, 94.1]**	**77.0 [76.3, 77.4]**
S_{GAN}		93.5 [93.3, 93.6]	76.9 [76.6, 77.6]
S_{GAN30}		93.9 [93.5, 93.9]	76.5 [76.3, 76.8]
B_F	SVHN	95.1 [95.0, 95.3]	73.2 [72.8, 73.4]
S_{KDE}		95.1 N/A	73.0 [72.6, 73.3]
S_{KDE30}		95.5 N/A	74.2 [73.9, 74.5]
S_{Neigh}		**97.2 [97.0, 97.3]**	**78.1 [77.9, 78.2]**
S_{Neigh30}		95.5 [95.4, 95.7]	76.5 [76.3, 76.7]
S_{GAN}		94.3 [94.1, 94.6]	74.5 [74.0, 74.9]
S_{GAN30}		94.9 [94.8, 95.1]	75.3 [75.0, 75.6

Table 5.F.2: Mann-Whitney U test of Samples S vs Baselines B: p-value and CLES (Common Language Effect Size). p-values indicate the probability of the samples of two groups originating from the same distribution. CLES=0.5 indicates no effect, CLES=1.0 a strong positive, CLES=0.0 a strong negative effect. As the results indicate, both proposed sampling methods are almost always statistically significantly better than the two baselines. Further, their effect is often very strong.

Population Pairs	Source	Target	
		MNIST	SVHN
S_{KDE} vs. B_T	MNIST	**2.1e-18** | **0.8701**	**5.2e-27** | **0.9551**
S_{KDE} vs. B_F		**0.0e+00** | **0.8639**	**1.1e-01** | **0.5920**
S_{KDE30} vs. B_T		**7.0e-27** | **0.9539**	**2.5e-29** | **0.9754**
S_{KDE30} vs. B_F		**6.9e-22** | **0.9545**	**1.7e-04** | **0.7180**
S_{Neigh} vs. B_T		**1.5e-30** | **0.9857**	**6.6e-31** | **0.9888**
S_{Neigh} vs. B_F		**4.5e-25** | **0.9889**	**5.2e-03** | **0.6622**
S_{Neigh30} vs. B_T		**1.7e-35** | **0.9987**	**1.3e-29** | **0.9778**
S_{Neigh30} vs. B_F		**3.1e-28** | **0.9994**	**1.4e-02** | **0.6426**
S_{GAN} vs. B_T		**7.6e-31** | **0.9883**	**8.0e-25** | **0.9351**
S_{GAN} vs. B_F		**3.0e-25** | **0.9907**	**7.8e-03** | **0.6546**
S_{GAN30} vs. B_T		**1.1e-31** | **0.9953**	**2.1e-26** | **0.9496**
S_{GAN30} vs. B_F		**6.8e-26** | **0.9973**	**4.9e-02** | **0.6144**
S_{KDE} vs. B_T	SVHN	**6.1e-79** | **0.9943**	**1.1e-04** | **0.6006**
S_{KDE} vs. B_F		7.8e-01 | 0.4904	3.8e-01 | 0.4704
S_{KDE30} vs. B_T		**1.7e-82** | **1.0000**	**1.6e-30** | **0.7985**
S_{KDE30} vs. B_F		**0.0e+00** | **0.7292**	**3.0e-08** | **0.6850**
S_{Neigh} vs. B_T		**2.9e-78** | **0.9867**	**8.6e-80** | **0.9916**
S_{Neigh} vs. B_F		**2.8e-44** | **0.9661**	**1.8e-47** | **0.9833**
S_{Neigh30} vs. B_T		**1.7e-82** | **1.0000**	**4.7e-76** | **0.9797**
S_{Neigh30} vs. B_F		**8.2e-08** | **0.6791**	**1.7e-42** | **0.9563**
S_{GAN} vs. B_T		**1.2e-31** | **0.9948**	**0.0e+00** | **0.8140**
S_{GAN} vs. B_F		1.5e-07 | 0.2517	**7.5e-06** | **0.7118**
S_{GAN30} vs. B_T		**4.2e-32** | **0.9987**	**6.7e-22** | **0.9067**
S_{GAN30} vs. B_F		3.6e-01 | 0.4565	**0.0e+00** | **0.8335**

Table 5.F.3: Accuracy of sampled models: median and 95% confidence intervals. On the main diagonal are in-dataset experiments, otherwise transfer-learning from source to target. Bold numbers highlight the best source-to-target results. N/A denotes cases, in which the bootstrapped CI on the median could not be computed.

Population	Source	Target	
		CIFAR-10	STL-10
B_T		49.0 [48.9, 49.0]	39.0 [38.9, 39.1]
B_F	CIFAR-10	48.6 [48.3, 48.7]	**42.8 [42.5, 42.9]**
S_{KDE}		48.3 [48.1, 48.4]	40.7 [40.3, 40.9]
S_{KDE30}		**48.7 [48.4, 48.8]**	41.3 [40.9, 41.5]
S_{Neigh}		45.6 [44.9, 46.0]	36.7 [35.8, 37.4]
S_{Neigh30}		46.2 [45.8, 46.4]	37.9 [37.3, 38.2]
S_{GAN}		46.0 N/A	38.6 [38.1, 39.0]
S_{GAN30}		47.0 [46.5, 47.2]	38.6 [38.2, 39.1]
B_F	STL-10	**49.3 [49.0, 49.4]**	**39.5 [38.9, 39.7]**
S_{KDE}		48.6 [48.4, 48.9]	37.3 [37.0, 37.8]
S_{KDE30}		48.8 [48.4, 49.2]	38.3 [37.9, 38.4]
S_{Neigh}		10.0 N/A	28.3 [26.8, 29.1]
S_{Neigh30}		49.0 [48.5, 49.1]	37.8 [37.6, 38.2]
S_{GAN}		49.0 [48.6, 49.4]	38.5 [37.9, 38.9]
S_{GAN30}		48.8 [48.5, 49.1]	37.9 N/A

Table 5.F.4: Mann-Whitney U test of Samples S vs Baselines B: p-value and CLES (Common Language Effect Size). p-values indicate the probability of the samples of two groups originating from the same distribution. CLES=0.5 indicates no effect, CLES=1.0 a strong positive, CLES=0.0 a strong negative effect.

Population Pairs	Source	Target	
		CIFAR-10	STL-10
S_{KDE} vs. B_T	CIFAR-10	1.5e-06 | 0.2966	**7.4e-19 | 0.8750**
S_{KDE} vs. B_F		3.7e-02 | 0.4014	1.7e-18 | 0.0849
S_{KDE30} vs. B_T		3.6e-02 | 0.4114	**4.8e-25 | 0.9371**
S_{KDE30} vs. B_F		**2.9e-01 | 0.5498**	0.0e+00 | 0.1266
S_{Neigh} vs. B_T		5.7e-28 | 0.0364	7.4e-18 | 0.1359
S_{Neigh} vs. B_F		3.1e-22 | 0.0413	7.1e-26 | 0.0024
S_{Neigh30} vs. B_T		3.5e-25 | 0.0616	2.0e-07 | 0.2800
S_{Neigh30} vs. B_F		2.2e-19 | 0.0741	3.0e-25 | 0.0089
S_{GAN} vs. B_T		6.6e-25 | 0.0642	6.6e-02 | 0.4223
S_{GAN} vs. B_F		2.8e-19 | 0.0754	1.0e-24 | 0.0145
S_{GAN30} vs. B_T		2.1e-21 | 0.0983	1.1e-02 | 0.3928
S_{GAN30} vs. B_F		8.8e-16 | 0.1195	2.7e-25 | 0.0084
S_{KDE} vs. B_T	STL-10	1.3e-01 | 0.4362	0.0e+00 | 0.1730
S_{KDE} vs. B_F		6.9e-04 | 0.3028	6.0e-10 | 0.1404
S_{KDE30} vs. B_T		6.1e-01 | 0.4783	1.2e-06 | 0.2948
S_{KDE30} vs. B_F		1.1e-02 | 0.3528	9.1e-06 | 0.2424
S_{Neigh} vs. B_T		2.9e-32 | 0.0000	3.0e-32 | 0.0000
S_{Neigh} vs. B_F		3.3e-20 | 0.0000	7.1e-18 | 0.0000
S_{Neigh30} vs. B_T		1.0e+00 | 0.5000	4.3e-09 | 0.2517
S_{Neigh30} vs. B_F		2.1e-02 | 0.3654	5.4e-07 | 0.2090
S_{GAN} vs. B_T		3.2e-01 | 0.5418	2.0e-04 | 0.3427
S_{GAN} vs. B_F		2.7e-01 | 0.4360	2.4e-04 | 0.2864
S_{GAN30} vs. B_T		6.2e-01 | 0.4788	5.4e-07 | 0.2880
S_{GAN30} vs. B_F		1.2e-02 | 0.3532	4.6e-06 | 0.2340

Chapter 6

Towards Scalable and Versatile Hyper-Representation Learning

Abstract

Learning representations of well-trained neural network models holds the promise to provide an understanding of the inner workings of those models, potentially shedding light on their robustness, safety, and other aspects. However, previous work faced limitations when processing larger networks or were task-specific to either discriminative or generative tasks. This paper introduces `SANE`, which overcomes these challenges by learning a task-agnostic representation of neural networks that is not only scalable to much larger model sizes but also shows capabilities beyond a single task. Our method extends the idea of *hyper-representations* towards sequential processing of subsets of neural network weights, thus allowing one to embed a potentially large neural network as a set of tokens into the learned representation space. This technique reveals global model information across layer-wise components, and it can sequentially generate unseen neural network models, an aspect previously unattainable with previous *hyper-representation* learning methods. We evaluate `SANE` on multiple downstream tasks across multiple models zoos, representing six computer vision datasets. Our findings demonstrate that `SANE` not only matches but also exceeds state-of-the-art performance on several weight representation learning benchmarks, particularly in initialization and transfer learning tasks for larger models like ResNets.

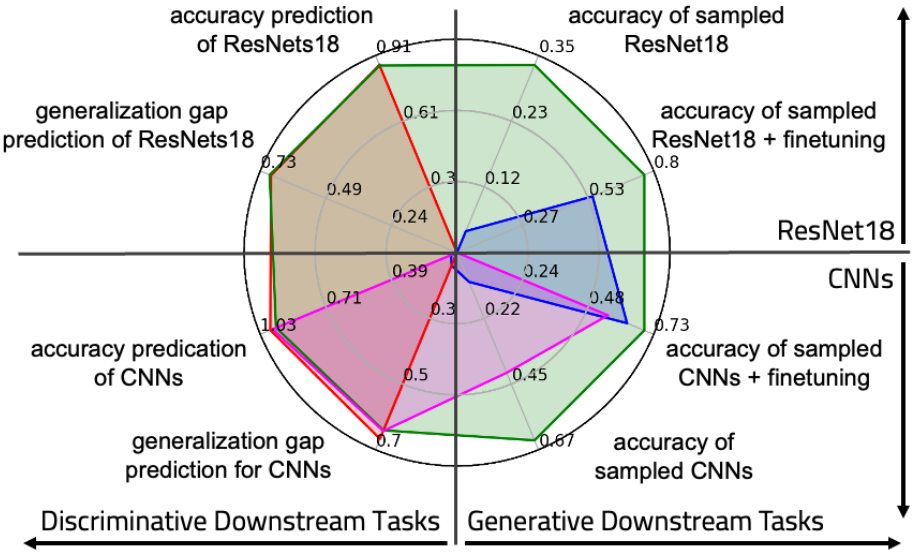


Figure 6.1.1: Aggregated results of 56 experiments showing **(left:)** four discriminative downstream tasks in R^2, and (**right:**) four generative downstream tasks in accuracy, each evaluated on (**bottom:**) CNNs model zoos trained on 4 datasets (NMIST, SVHN, CIFAR-10, STL) and (**top:**) ResNet18 model zoos trained on three datasets (CIFAR-10, CIFAR-100, Tiny-ImageNet). The colors indicate performance of `Blue:` training from scratch baseline, `Orange:` weight statistics from Unterthiner et al. [159], `Purple:` trained *hyper-representations* from Schürholt et al. [147, 148], and `Green:` `SANE` (ours). It can be seen that while some methods are performing well on their specific tasks, or are restricted by the size of the underlying models, `SANE` can deliver excellent performance on all tasks and model sizes.

6.1 Introduction

The exploration of the "weight space" of neural network (NN) models, i.e., the high-dimensional space spanned by the model parameters of a population of trained NNs, allows us to gain insights into the inner workings of those models.

In the discriminative context, works aim to link weight space properties to properties such as model quality, generalization gap, or hyperparameters, using either the margin distribution [80, 172], or graph topology features [26], or eigenvalue decompositions of weight matrices [113, 116, 117, 119]. Some works learn classifiers to map between statistics of weights and model properties [43, 159], or learn lower-dimensional manifolds to infer NN model properties [147].

In the generative context, methods have been proposed to generate model weights using (Graph) HyperNetworks [61, 88, 179], Bayesian HyperNetworks [36], HyperGANs [142], and HyperTransformers [182]. These approaches have been used for tasks such as neural architecture search, model compression, ensembling, transfer learning, and meta-learning. They have in common that they derive their learning signal from the

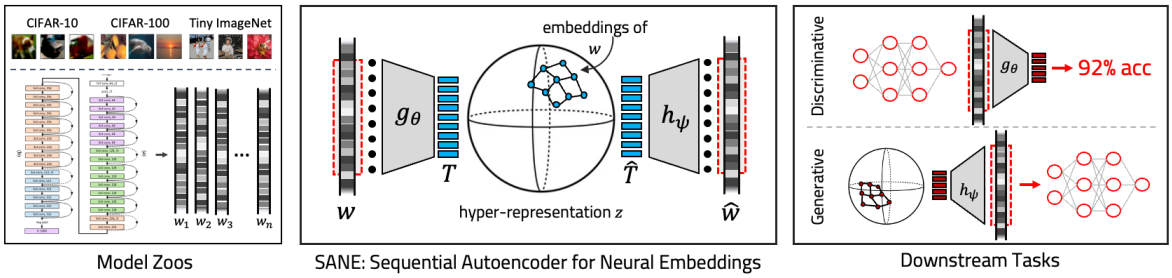


Figure 6.1.2: Given model zoos trained on different classification tasks, SANE trains *hyper-representations* on sequences of layers or subset of weights of these model zoos. Once trained, SANE can be used for multiple downstream tasks, either only using the encoder for discriminative tasks such as the prediction of model accuracy, or the decoder for generative tasks such as sampling of unseen models.

underlying (typically image) dataset. In contrast to these methods, so-called *hyper-representations* [148] learn a lower-dimensional representation directly from the weight space without the need to have access to the e.g., underlying image dataset, to sample unseen NN models from that latent representation.

In this paper, we present Sequential Autoencoder for Neural Embeddings (SANE), an approach to learn task-agnostic representations of NN weight spaces capable of embedding individual NN models into latent to perform the above-mentioned discriminative or generative downstream tasks. Our approach builds upon the idea of *hyper-representations* [147, 148], which learn a lower-dimensional representation z from a population of NN models. This is accomplished by auto-encoding their flattened weight vectors w_i through a transformer architecture, with the bottleneck acting as a lower-dimensional embedding z_i of each NN model. While the *hyper-representation* method promises to be useful for discriminative and generative tasks, until now, separate *hyper-representations* had to be trained specifically for either discriminative or generative tasks. Additionally, it has a major shortcoming: the underlying encoder-decoder model has to embed the entire flattened weight vectors w_i at once into the learned lower-dimensional representation z. This drastically limits the size of NNs that can be embedded. SANE addresses these limitations by decomposing the entire weight vector w_i into layers or smaller subsets and sequentially processes these. Instead of encoding the entire NN model by one embedding, SANE encodes a potentially very large NN as multiple embeddings. The change from processing the entire flattened weight vector to subsets of weights is motivated by Martin and Mahoney [115, 117], who showed that global model information is preserved in the layer-wise components of NNs. An illustration of the approach can be found in Fig. 6.1.2.

To evaluate SANE, first, we analyze how NN embeddings encoded by SANE behave in

comparison to Martin and Mahoney [115, 117] quality measures. We show that some of these quality metrics as derived from the matrices of the analyzed NN models show similar characteristics as the embeddings produced by SANE. This holds not only for held-out NN models of the model zoo used for training SANE but also for NN models of other model zoos we consider out-of-distribution, given the model architectures and underlying image datasets. Second, we demonstrate that SANE can learn *hyper-representations* from much larger NN models, rendering them applicable to real-world problems. In particular, the models in the ResNet model zoo used for training are three orders of magnitudes larger than all model zoos used for *hyper-representation* learning in previous works. While previous *hyper-representation* learning methods were structurally constrained to encode the entire NN model at once, SANE is scalable, given its sequential approach to encode layers or subsets of weights into hyper-representation embeddings. It is thus potentially applicable to NN models far larger than ResNets. Finally, we evaluate SANE on both discriminative and generative downstream tasks. For discriminate tasks, we evaluate on four model zoos by linear-probing for properties of the underlying NN models. For generative tasks, we evaluate on three model zoos by sampling unseen model weights for initialization and transfer learning.

We provide an aggregated overview of our results in Fig. 6.1.1. For very small CNN models (as evaluated on MNIST, SVHN, CIFAR-10, and STL, which we include for comparison with prior work) SANE performs as well as previous state-of-the-art (SOTA) for discriminative tasks. For generative downstream tasks SANE outperforms SOTA by 25% in accuracy for initialization on the same task and 17% in accuracy for finetuning to new tasks. For larger models such as ResNets (as evaluated on CIFAR-10, CIFAR-100, Tiny-ImageNet and were beyond the capabilities of prior work), we show results comparable to baselines for discriminative downstream tasks and report outperformance to baselines for generative downstream tasks by 31% for initialization and 28% for finetuning to new tasks. Additionally, we show that we can sample unseen models by prompting SANE with different architectures than it used for training. These sampled models can outperform models trained from scratch on the prompted architecture.

6.2 Methods

Hyper-representations learn an encoder-decoder model on the weights of NNs [147]:

$$\mathbf{z} = g_\theta(\mathbf{W}) \tag{6.2.1}$$

$$\widehat{\mathbf{W}} = h_\psi(\mathbf{z}), \tag{6.2.2}$$

where g_θ is the encoder which maps the flattened weights $\mathbf{W}$ to embeddings $\mathbf{z}$, and h_ψ decodes back to reconstructed weights $\widehat{\mathbf{W}}$. Even though previous work realized both encoder and decoder with transformer backbones, the weight vector had to be of fixed size, and models are represented in a global embedding space [147, 148].*Hyper-representations* are trained with a reconstruction loss $\mathcal{L}_{rec} = \|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2$ and contrastive guidance loss $\mathcal{L}_c = NTXent(p_\phi(\mathbf{z_i}), p_\phi(\mathbf{z_j}))$, where p_ϕ is a projection head. Schürholt et al. [147] proposed weight permutation, noise, and masking as augmentations to generate views i, j of the same model.

Existing *hyper-representation* methods have two major limitations: i) using the full weight vector to compute global model embeddings becomes infeasible for larger models; and ii) can only embed models that share the architecture with the original model zoo. Our `SANE` method addresses both. To make models more digestible for pretraining and inference, we propose to express models as sequences of token vectors. To address i), `SANE` learns per-token embeddings, which are trained on subsequences of the full base model sequence. This way, the memory and compute load are decoupled from the base model size. By decoupling the tokenization from the representation learning, we also address ii). The models in the model zoo set can have varying architectures, as long as they are expressed as a sequence with the same token-vector size. The transformer backbone and per-token embeddings also allow to change the length of the sequence during or after training. Below, we provide technical details on `SANE`. We first provide details on pretraining `SANE`, computing model embeddings, and sampling models; and we then introduce *aligning*, *haloing*, and *bn-conditioning*.

`Sequential Autoencoder for Neural Embeddings` To tokenize weights, we reshape the weights $\mathbf{W}_{raw} \in \mathbb{R}^{c_{out} \times c_1 \times \cdots \times c_{in}}$, to 2d matrices $\mathbf{W} \in \mathbb{R}^{c_{out} \times c_r}$, where c_{out} are the outgoing channels, and where c_r the remaining, flattened dimensions. We then slice the weights row-wise, along the outgoing channel. Using global token size d_t, we split the slices into multiple parts if $c_r > d_t$ and zero-pad to fill up to d_t. For weights $\mathbf{W}_l$ of layer l, this gives us tokens $\mathbf{T}_l \in \mathbb{R}^{n_l \times d_t}$, where $n_l = c_{out,l}\, ceil(\frac{c_r}{d_t})$. Since all tokens $\mathbf{T}_l$ share the same token size, the tokens of layer $l = 1...L$ can be concatenated to get the model

token sequence $\mathbf{T} \in \mathbb{R}^{N \times d_t}$. To indicate the position of a token in the sequence, we use a 3-dimensional position $\mathbf{P}_n = [n, l, k]$, where $n \in [1, N]$ indicates the global position in the sequence, $l \in [1, L]$ indicate the layer index, and $k \in [1, K(l)]$ is the position of the token within the layer.

Out of the full token sequence $\mathbf{T}$ and positions $\mathbf{P} \in \mathbb{N}^{N \times 3}$, we take a random consecutive sub-sequence $\mathbf{T}_{s,n} = \mathbf{T}_{n,\ldots,n+ws}$ with positions $\mathbf{P}_{s,n} = \mathbf{P}_{n,\ldots,n+ws}$ of length ws. We call these sub-sequences windows and the length of the sub-sequence the window size ws.

For SANE on windows of tokens, we extend Eqs. 6.2.1 and 6.2.2 to encode and decode token windows as

$$\mathbf{z}_{s,n} = g_\theta(\mathbf{T}_{s,n}, \mathbf{P}_{s,n}) \tag{6.2.3}$$

$$\widehat{\mathbf{T}}_{s,n} = h_\psi(\mathbf{z}_{s,n}, \mathbf{P}_{s,n}), \tag{6.2.4}$$

where $\mathbf{z}_{s,n} \in \mathbb{R}^{ws \times d_z}$ is the per-token latent representation of the window. In contrast to *hyper-representations* Eqs. 6.2.1 and 6.2.2 which operate on the full flattened weights of a model, SANE encodes sub-sequences of tokenized models. For simplicity, we apply linear mapping to and from bottleneck, to reduce tokens from d_t to d_z.

We adapt the composite training loss of *hyper-representations*, $\mathcal{L} = (1-\gamma)\mathcal{L}_{rec} + \gamma\mathcal{L}_c$, for sequences as:

$$\mathcal{L}_{rec} = \|\mathbf{M}_{s,n} \odot \left(\mathbf{T}_{s,n} - \widehat{\mathbf{T}}_{s,n}\right)\|_2^2 \tag{6.2.5}$$

$$\mathcal{L}_c = NTXent(p_\phi(\mathbf{z_{s,n,i}}), p_\phi(\mathbf{z_{s,n,j}})). \tag{6.2.6}$$

Here, the mask $\mathbf{M}_{s,n}$ indicates signal with 1 and padding with 0, to ensure that the loss is only computed on actual weights. The contrastive guidance loss uses the augmented views i, j and projection head p_ϕ.

The pretraining procedure is detailed in Algorithm 1. We preprocess model weights by standardizing weights per layer and aligning all models to a reference model, see *Model Alignment* below. As in previous work [136, 147], the encoder and decoder are realized as transformer blocks. Training on the full sequence would memory-limit the base-model size by its sequence length. Training the encoder and decoder on windows instead of the full model sequence decouples the memory requirement from the base model's full sequence length. The window size can be used to balance GPU memory load and the amount of context information. Notably, since we disentangle the tokenization from the representation learning model, SANE also allows to embed sequences of models

Algorithm 1 SANE pretraining

Input: population of models
i: standardize models weights
ii: align models to one common reference model
iii: tokenize models to tokens $\mathbf{T}$, positions $\mathbf{P}$, masks $\mathbf{M}$
iv: draw k windows per model: $\mathbf{T}_{s,n}$, $\mathbf{P}_{s,n}$, $\mathbf{M}_{s,n}$
v: train on $\mathcal{L}_{train}$ until convergence of $\mathcal{L}_{val}$

with varying architectures, as long as their token size is the same. To prevent potential overfitting to specific window positions, we propose to sample windows from each model sequence multiple times randomly.

Computing SANE Model Embeddings. SANE can be used to analyse models in embedding space, e.g., by using embeddings as features to predict properties such as accuracy, or to identify other model quality metrics. In contrast to *hyper-representations*, SANE can embed different model sizes and architectures in the same embedding space. To embed any model, we begin by preprocessing weights by standardizing per layer and aligning models to a pre-defined reference model (see *Model Alignment* below). Subsequently, the preprocessed models are tokenized as described above. For short model sequences, the embedding sequences can be directly computed as $\mathbf{z} = g_\theta(\mathbf{T}, \mathbf{P})$. For larger models, the token sequences are too long to embed as one. We therefore employ *haloing* (see below) to encode the entire sequence as coherent subsequences. Algorithm 2 summarizes the embedding computation. To compare different models in embedding

Algorithm 2 SANE model embedding computation

Input: population of models
i: preprocessing: standardize and align model weights
ii: tokenize models: $\mathbf{T}$, positions $\mathbf{P}$, property y
iii: split model sequences to consecutive chunks $\mathbf{T}_{hs,n}, \mathbf{P}_{hs,n}$
iv: compute embeddings $\mathbf{z}_{hs,n} = g_\theta(\mathbf{T}_{hs,n}, \mathbf{P}_{hs,n})$
v: stitch model embeddings $\mathbf{z}$ together from chunks $\mathbf{z}_{hs,n}$

space, we aggregate the sequences of token embeddings. To that end, we understand the token sequence of one model to form a surface in embedding space and choose to represent that surface by its center of gravity. That is, we take the mean of all tokens along the embedding dimension as $\bar{\mathbf{z}} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{z}_n)$. That results in one vector in embedding space per model. Of course, one could use other aggregation methods with SANE.

Sampling Models with Few Prompt Examples. Sampling models from SANE promises to transfer knowledge from pretraining populations to new models with different architectures. Given pre-trained encoders g_θ and decoder h_ψ, the challenge is to identify the distribution $\mathcal{P}$ in latent space which contains the targeted properties. To approximate that distribution, previous work used a large number of well-trained models [136, 148]. However, increasing the size of the sampled models makes pretraining a large number of high-performance models exceedingly expensive. Instead of using expensive high-performance models to model $\mathcal{P}$ directly, we propose to find a rough estimate of $\mathcal{P}$, sample broadly, and refine $\mathcal{P}$ using the signal from the sampled models. Using E prompt examples $\mathbf{W}^e$ we compute the token sequence $\mathbf{T}^e$ and corresponding embedding sequence $\mathbf{z}^e = g_\theta(\mathbf{T}^e, \mathbf{P})$. Following previous work, we use a KDE to model the distribution $\mathcal{P}$ per token as $\mathcal{P}_{e\in E}(\mathbf{z}_n^e)$ [148]. We then draw k new samples per token as:

$$\mathbf{z}_n^k \sim \mathcal{P}_{e\in E}(\mathbf{z}_n^e). \tag{6.2.7}$$

We reconstruct the sampled embeddings to weight tokens $\mathbf{T}^k = h_\psi(\mathbf{z}^k, \mathbf{P})$ and ultimately weights $\mathbf{W}^k$. Sampling tokens can be done cheaply, decoding and evaluating the weights using some performance metric involves only forward passes and is likewise cheap. Therefore, one can draw a large amount of samples and keep only the top m models, according to the performance metric. We call this method *subsampling*. The process can be refined iteratively, by re-using the embeddings $\mathbf{z}^k$ of the best models as new prompt examples, to adjust the sampling distribution to best fit the needs of the performance metric. We call this sampling method *bootstrapped*. By only requiring a rough version of $\mathcal{P}$ and refining with the target signal, our sampling strategy reduces requirements on prompt examples s.t. only need very few and slightly trained prompt examples are necessary. The overall sampling method is outlined in Algorithm 3. It makes use of *model alignment*, *haloing*, and *batch-norm conditioning* which are detailed below. In addition to the compute efficiency, these sampling methods learn the distribution of targeted models in embedding space. Further, they are not bound to the distribution of prompt examples but can find the distribution that best satisfies the target performance metric, independent of the prompt examples.

Growing sample model size poses several additional challenges, three of which we address with the following methods. We evaluate these methods in Appendix 6.A.

Algorithm 3 Sampling models with `SANE`

Input: model prompt examples
i: tokenize prompt examples: tokens $\mathbf{T}$, positions $\mathbf{P}$
ii: embed prompt examples $\mathbf{z}^e$ following Alg. **??**
for $i_{boot} = 1$ **to** *bootstrap iterations* **do**
  iii: draw k samples $\mathbf{z}_n^k \sim \mathcal{P}_{e \in E}(\mathbf{z}_n^e)$
  iv: decode to tokens $\mathbf{T^k} = h_\psi(\mathbf{z}^k)$
  v: apply batch-norm conditioning
  vi: compute target metric and keep best m models
  if *bootstrap iterations* > 1 **then**
    vii: $\mathbf{z}^e = \mathbf{z}^k for\ k \in m$
  end if
end for

Model Alignment. Symmetries in the weight space of NN complicate representation learning of the weights. The number of symmetries grows fast with model size [10]. To make representation learning easier, we reduced all training models to a unique, canonical basis of a reference model. With reference model A we align model B by finding the permutation $\pi = argmin_\pi \|\text{vec}(\Theta(A)) - \text{vec}(\Theta(B))\|^2$, where $\Theta(A)$ are the parameters of model A [1]. We fix the same reference model across all dataset splits and use the last epoch of each model to determine the permutation for that model.

Haloing. The sequential decomposition of `SANE` decouples the pretraining sequence length from downstream task sequence lengths. Since the memory load at inference is considerably lower, the sequences at inference can be longer. However, full model sequences may still not fit in memory and have to be processed in slices. To ensure consistency between the slices, we add context around the content windows. With added context halo before and after the content window, we get $\mathbf{T}_{hs,n} = \mathbf{T}_{n-h,..,n,...,n+ws,n+ws+h}$ Similar to approaches in computer vision [161], this context halo is added for the pass through encoder and decoder, but disregarded after.

Batch-Norm Conditioning. In most current NN models, some parameters like batch-norm weights are updated during forward passes instead of with gradients. Since that makes them structurally different, we exclude these parameters from representation learning and sampling with `SANE`. Nonetheless, these parameters need to be instantiated for sampled models to work well. For model sampling methods we therefore propose to condition batch-norm parameters by doing a few forward passes with some target data. Importantly, this process does not update the learned weights of the model. It serves to align the batch norm statistics with the model's weights.

6.3 Training SANE

We pretrain SANE following Alg. 1 on several populations of trained NN models, from the model zoo dataset [150]. We use zoos of small models to compare with previous work, as well as zoos containing larger ResNet-18 models. All zoos are split into training, validation, and test splits $70:15:15$.

- **Smaller CNN zoos.** The MNIST and SVHN zoos contain LeNet-style models with 3 convolution and 2 dense layers and only $\sim 2.5k$ parameters. The slightly larger CIFAR-10 and STL-10 zoos use the same architecture with wider layers and $\sim 12k$ parameters.
- **Larger ResNet zoos.** We also use the CIFAR-10, CIFAR-100, and Tiny-Imagenet zoos containing ResNet-18 models [150] with $\sim 12M$ parameters to evaluate scalability to large models.

Pretraining. We train SANE using Alg. 1. As augmentations, we use noise and permutation. The permutation is computed relative to the aligned model, the aligned model serves as one view, and a permuted version as the second view.

Implementation Details. To maintain diversity within each batch, we select only a single window from each model. Loading, preprocessing, and augmenting the entire sample only to use approximately 1% of it is infeasible. To address this, we leverage FFCV [97] to compile datasets consisting of sliced and permuted windows of models. Each model is super-sampled for approximately full coverage within the training set, considering the ratio of window length to sequence length. For the ResNet zoos, we include 140 models per zoo, a number that remains manageable in terms of memory and storage. We train for 50 epochs using a OneCycle learning rate scheduler [155]. Seeds are recorded to ensure reproducibility. We build SANE in PyTorch [135], using automatic mixed precision and flash attention [29] to enhance performance. We use ray.tune [107] for hyperparameter optimization. Code to reproduce the experiments will be made available upon publication.

6.4 Embedding Analysis

In this section, we analyze the embeddings of SANE and compare to the weight-analysis methods *WeightWatcher* (WW) [119]. We focus on three aspects: i) global relation between accuracy and embeddings; ii) the trend of embeddings over layer index, as in [119]; and iii) the identification of training phases as in [116, 117].

To analyze weights, we focus on two WW metrics which in previous work reveal model performance as well as internal model composition (*correlation flow*); the log spectral norm $\log(\|\mathbf{W}\|_\infty^2)$ and weighted α, the coefficient of the power law fitted to the empirical spectral density [119]. These two metrics describe different aspects of the eigenvalue distribution. To get a similar signal on the internal dependency of weight matrices, we compute per-layer scalars $\hat{z}_l$ as the spread of the tokens of one layer in hyper-representation space, i.e., their standard deviation.

$$\hat{z}_l = std_t(\mathbf{z}_m^t) \tag{6.4.1}$$

$$\mathbf{z}_m^t = g(\mathbf{W}_m^t), \tag{6.4.2}$$

where g is the hyper-rep encoder, $\mathbf{z}_m^t$ are the stacked tokens t of layer m, and $\mathbf{W}_m^t$ is the weight-slice t of layer m.

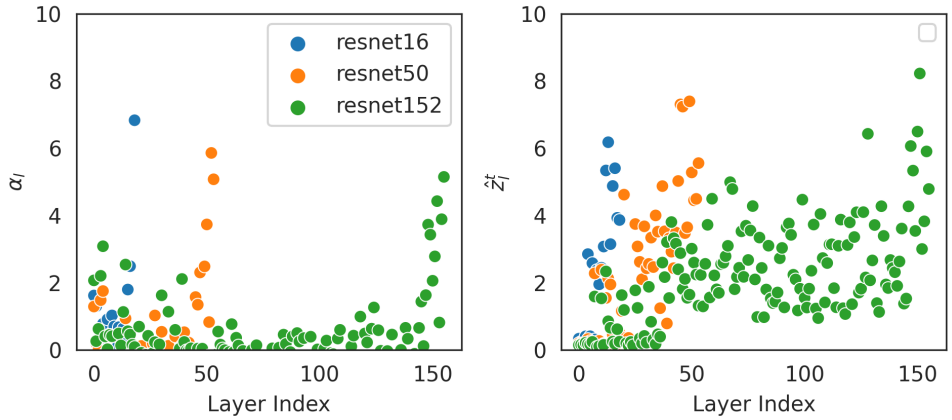


Figure 6.4.1: Comparison between WeightWatcher (WW) features (left) and SANE (right). Features over layer index for ResNets from pytorchcv of different sizes.

To compare WW metrics to SANE, we pretrain SANE on a Tiny-Imagenet ResNet-18 zoo and compute the two metrics on ResNets and VGGs of different sizes trained on ImageNet from pytorchcv [151]. On both ResNets in Figure 6.4.1, 6.C.3 and VGGs in Figure 6.C.2, the WW metrics and our embeddings show similar global trends. On

ResNets, our embeddings and WW have low values at early layers and a sharp increase at the end. However, our embeddings add an additional step for intermediate layers, which may indicate that SANE is sensitive to a higher degree of variation in these layers which previous work found by comparing activations [91]. In a second experiment, we aggregate the layer-wise embeddings $\hat{z}_l$ to evaluate relations to model accuracy in Figures **??**, **??** and **??**, similar to previous work [119]. On models from pytorchcv and the Tiny-ImagNet model zoo from [150], the WW features and SANE embeddings both show strong correlations to model accuracy. However, while the WW metrics are negatively correlated to accuracy, our embeddings are positively correlated to accuracy. The reason for that may lie in the additional 'step' in Figure 6.4.1. That is, larger models with more layers generally have higher performance. As Figure 6.4.1 shows, more layers add very small values reducing the global average for WW metrics. For our embeddings, deeper models have more layers with higher $\hat{z}_l$ values, due to the afore-mentioned step. This increases the global model average with growing model size. Lastly, we compare the eigenvalue spectrum to embeddings. Previous work identified distinct shapes at different training phases or with varying training hyperparameters [116, 117]. While we can replicate the distributions of the eigenvalues, the distributions of our embeddings only show the change from early phases of training to the heavy-tailed distribution, see Figure 6.C.1. In summary, the embedding analysis indicates that SANE represents aspects of model quality globally and on a layer level.

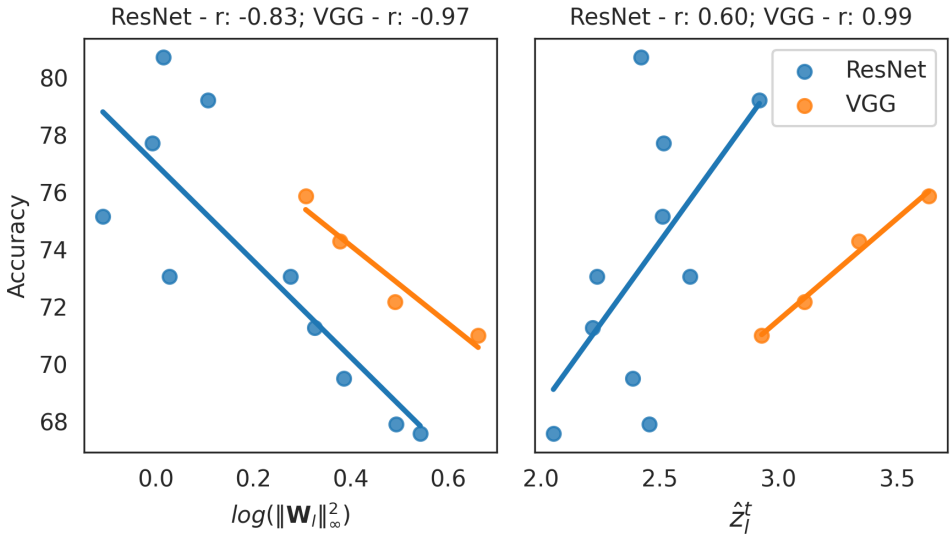


Figure 6.4.2: Comparison between WeightWatcher features (left) and SANE (right). Accuracy over model features for ResNets and VGGs from pytorchcv of different sizes. Although SANE is pretrained in a self-supervised fashion, it preserves the linear relation of a globally-aggregated embedding to model accuracy.

6.5 Empirical Performance

In this section, we describe the general performance SANE.

6.5.1 Predicting Model Properties

Here, we evaluate SANE for discriminative downstream tasks as a proxy for encoded model qualities. Specifically, we investigate whether SANE matches the predictive performance of *hyper-representations* on small CNN models (Table 6.5.1) and whether similar performance can be achieved on ResNet-18 models (Table 6.5.2). To that end, we compute model embeddings $\bar{\mathbf{z}}$ as outlined in Alg. 2 and compare against flattened weights W and weight statistics $s(W)$. Following the experimental setup of [43, 147, 159], we compute embeddings using the three methods and linear probe for test accuracy (Acc), epoch (Ep), and generalization gap (Ggap). We again use trained models from the modelzoo repository [150], with the same train, test, val splits as above.

Table 6.5.1: Property prediction on populations of small CNNs used in previous work [147]. We report the regression R^2 on the test set prediction test accuracy Acc., epoch Ep. and generalization gap Ggap for linear probing with model weights W, model weights statistics $s(W)$ or SANE embeddings as inputs.

	MNIST			SVHN			CIFAR-10 (CNN)		
	W	s(W)	SANE	W	s(W)	SANE	W	s(W)	SANE
Acc.	0.965	**0.987**	0.978	0.910	0.985	**0.991**	-7.580	**0.965**	0.885
Ep.	0.953	**0.974**	0.958	0.833	**0.953**	0.930	0.636	**0.923**	0.771
Ggap	0.246	0.393	**0.402**	0.479	0.711	**0.760**	0.324	**0.909**	0.772

SANE matches baselines on small models. The results of linear probing on small CNNs in Tables 6.5.1 and 6.D.1 confirm the performance of W (low) and $s(W)$ (very high) of previous work. SANE embeddings show comparably high performance to the $s(W)$ and previous *hyper-representations*. Sequential decomposition and representation learning as well as using the center of gravity does not appear to negatively affect the information contained in SANE embeddings. Additional experiments in App. 6.D compare to previous work and confirm these findings. Sequential decomposition and representation learning as well as using the center of gravity does not significantly reduce the information contained in SANE embeddings.

SANE performance prediction scales to ResNets. Both $s(W)$ and SANE embeddings show similarly high performance on populations of ResNet-18s, see Table 6.5.2. On ResNet-18s, using the full weights W for linear probing is infeasible due to the size of

Table 6.5.2: Property prediction on ResNet-18 model zoos of [150]. We report the regression R^2 on the test set prediction test accuracy `Acc.`, epoch `Ep.` and generalization gap `Ggap` for linear probing with model weights statistics $s(W)$ or `SANE` embeddings as inputs.

	CIFAR-10		CIFAR-100		Tiny-Imagenet	
	s(W)	SANE	s(W)	SANE	s(W)	SANE
Acc.	0.880	0.879	0.923	0.922	0.802	0.795
Ep.	0.999	0.999	0.999	0.992	0.999	0.980
Ggap	0.490	0.512	0.882	0.879	0.704	0.699

the flattened weights. `SANE` matches the high performance of $s(W)$. The results show that sequential *hyper-representation* are capable of scaling to ResNet-18 models. Further, the aggregation even of long sequences (50k tokens) preserves meaningful information on model performance, which indicates the feasibility of applications like model diagnostics or targeted sampling.

6.5.2 Generating Models

Here, we evaluate `SANE` for sampling model weights. We generate weights following Alg. 3 and test them in fine-tuning, transfer learning, and how it generalizes to new tasks and architectures. In the following paragraphs, we begin with experiments on small CNN models from the modelzoo repository to compare with previous work (Tables 6.5.3, 6.E.1. Subsequently, we evaluate `SANE` for sampling ResNet-18 models for finetuning and transfer learning (Tables 6.5.4, 6.E.2. Lastly, we evaluate sampling for new tasks and new architectures using only few prompt examples (Figure 6.5.1 and Tables 6.5.5, 6.E.3,6.E.4, 6.E.5).

We pretrain `SANE` on models from the first half of the training epochs with Alg. 1, and keep the remaining epochs (26-50) as holdout to compare against, following the experimental setup of [148]. We sample using Alg. 3 and use models from the last epoch in the pretraining set (epoch 25) as prompt examples. We compare sampled models using prompt examples against training from scratch, as well as fine-tuning from the prompt examples. We denote subsampling with $\texttt{SANE}_{SUB}$ and iteratively updating the distribution $\mathcal{P}$ as $\texttt{SANE}_{BOOT}$. To evaluate the impact of the sampling method, we also combine `SANE` with the $KDE30$ sampling approach that uses high-quality prompt examples [148]. We also evaluate sampling without prompt examples by bootstrapping off of a Gaussian prior $\mathcal{P}$, denoted as $\texttt{SANE}_{GAUSS}$. We compare against training from scratch, as well as fine-tuning from the prompt examples.

Table 6.5.3: Model generation on CNN model populations fine-tuned on the same task. We compare training from scratch with S_{KDE30} from [148], SANE combined with the $KDE30$ sampling method, and our SANE subsampled. Each of the sampled populations is fine-tuned over 25 epochs.

Ep.	Method	MNIST	SVHN	CIFAR-10	STL
0	tr. fr. scratch	∼10 /%	∼10 /%	∼10 /%	∼10 /%
	S_{KDE30}	68.6±6.7	54.5±5.9	*n/a*	*n/a*
	SANE $_{KDE30}$	84.8±0.8	70.7±1.4	56.3±0.5	39.2±0.8
	SANE $_{SUB}$	**86.7±0.8**	**72.3±1.6**	**57.9±0.2**	**43.5±1.0**
	SANE $_{GAUSS}$	20.8±0.1	21.6±0.5	19.3±0.2	17.5±1.5
1	tr. fr. scratch	20.6±1.6	19.4±0.6	37.2±1.4	21.3±1.6
	S_{KDE30}	83.7±1.3	69.9±1.6	*n/a*	*n/a*
	SANE $_{KDE30}$	85.5±0.8	71.3±1.4	58.2±0.2	43.5±0.7
	SANE $_{SUB}$	**87.5±0.6**	**73.3±1.4**	**59.1±0.3**	**44.3±1.0**
	SANE $_{GAUSS}$	61.3±3.1	24.1±4.4	27.2±0.3	22.4±1.0
5	tr. fr. scratch	36.7±5.2	23.5±4.7	48.5±1.0	31.6±4.2
	S_{KDE30}	**92.4±0.7**	57.3±12.4	*n/a*	*n/a*
	SANE $_{KDE30}$	87.5±0.7	72.2±1.2	58.8±0.4	45.2±0.6
	SANE $_{SUB}$	89.0±0.4	**73.6±1.5**	**59.6±0.3**	**45.3±0.9**
	SANE $_{GAUSS}$	83.4±0.8	35.6±8.9	43.3±0.3	34.2±0.7
25	tr. fr. scratch	83.3±2.6	66.7±8.5	57.2±0.8	44.0±1.0
	S_{KDE30}	**93.0±0.7**	74.2±1.4	*n/a*	*n/a*
	SANE $_{KDE30}$	92.0±0.3	74.7±0.8	60.2±0.6	**48.4±0.5**
	SANE $_{SUB}$	92.3±0.4	**75.1±1.0**	**61.2±0.1**	48.0±0.4
	SANE $_{GAUSS}$	**94.2±0.4**	54.2±17.6	52.2±0.6	43.5±0.5
50	tr. fr. scratch	91.1±2.6	70.7±8.8	61.5±0.7	47.4±0.9

Sampling High-Performing CNNs Zero-Shot. We begin with finetuning and transfer learning experiments on small CNNs from the modelzoo dataset to validate that the sequential decomposition for pretraining and sampling does not hurt performance. The results of these experiments show dramatically improved performance zero-shot for fine-tuning and transfer learning over previous hyper-representations, see Tables 6.5.3 and 6.E.1. At epoch 0, SANE improves over previous *hyper-representations* S_{KDE30} by almost 20%. The effect becomes smaller during fine-tuning. Nonetheless SANE consistently outperforms training from scratch with a higher epoch budget, often by several percentage points. This demonstrates on small CNNs that sequential pretraining and sampling of SANE improves performance, particularly zero shot. That indicates the potential for scenarios with little labelled data.

Table 6.5.4: Model generation on ResNet-18 model populations fine-tuned on the same task. We compare sampled models at different epochs with models trained from scratch.

Epoch	Method	CIFAR-10	CIFAR-100	Tiny-Imagenet
0	tr. fr. scratch	∼10 /%	∼1 /%	∼0.5 /%
	SANE $_{KDE30}$	64.8±2.0	19.8±2.5	8.4±0.9
	SANE $_{SUB}$	68.1±0.7	19.8±1.3	11.1±0.5
	SANE $_{BOOT}$	**68.6±1.2**	**20.4±1.3**	**11.7±0.5**
1	tr. fr. scratch	43.7±1.3	17.5±0.7	13.8±0.8
	SANE $_{KDE30}$	82.4±0.9	59.0±1.3	46.7±0.8
	SANE $_{SUB}$	**83.6±1.5**	**60.8±0.8**	**47.4±1.0**
	SANE $_{BOOT}$	82.8±1.4	60.2±0.5	47.2±0.8
5	tr. fr. scratch	64.4±2.9	36.5±2.0	31.1±1.6
	SANE $_{KDE30}$	**85.9±0.6**	56.2±1.7	45.6±1.4
	SANE $_{SUB}$	85.4±1.3	**56.7±1.6**	45.7±0.8
	SANE $_{BOOT}$	85.4±0.7	56.4±1.2	**49.1±1.7**
10	tr. fr. scratch	76.5±2.7	49.0±2.0	39.9±2.2
	SANE $_{KDE30}$	91.4±0.1	**72.9±0.2**	**64.2±0.3**
	SANE $_{SUB}$	**91.6±0.2**	**72.9±0.1**	64.0±0.2
	SANE $_{BOOT}$	91.6±0.2	72.8±0.1	64.1±0.2
25	tr. fr. scratch	85.5±1.5	56.5±2.0	43.3±1.9
50	tr. fr. scratch	92.14±0.2	70.7±0.4	57.3±0.6
60	tr. fr. scratch	*n/a*	74.2±0.3	63.9±0.5

SANE Sequential Sampling Scales to ResNets. To evaluate how well sampling with SANE scales to larger models, we continue with experiments on ResNet-18s. The results of these experiments Tables 6.5.4 and 6.E.2 show that despite the long sequences, the sampled ResNet models perform well above random initialization. E.g., sampled ResNet-18s achieve 68.1% on CIFAR-10 without any fine-tuning (Table 6.5.4). These models are at least three orders of magnitude larger than models in *hyper-representations*, and are computationally infeasible for these methods since they operate on full models at once [148], hence we cannot compare to *hyper-representations* here. As before, the performance difference to random initialization becomes smaller during fine-tuning. Similar to our experiments on CNNs, sampled ResNet-18s achieve competitive performance or even outperform training from scratch with a considerably smaller computational budget.[1] Transferred to a new task, sampled models outperform training from scratch and match fine-tuning from prompt examples (Table 6.E.2). Interestingly, subsampling and bootstrapping appears to work well when there is a useful signal to start with, i.e., on easier tasks

[1]The base population is trained with a one-cycle learning rate scheduler. To avoid any bias, we adopt the same scheduler, but train for only 10 epochs, which affects direct comparability.

that are similar to the pretraining distribution. This suggests that the sampling distributions are not ideal, and may require a better fit, more samples, or iterative adjustment to fit new datasets zero-shot. Nonetheless, even the relatively naive sampling methods can successfully sample competitive models, even at the scale of ResNet-sized architectures. This shows that our sequential sampling works even for long sequences of tokens.

Subsampling Improves Performance Previous work requires high-quality prompt examples to target specific properties [148]. Our sampling methods drop these requirements and use prompt examples only to model a prior. We therefore compare SANE with S_{KDE30} from [148] to SANE. Further, we compare the $KDE30$ sampling method with our subsampling approach on SANE. On datasets where published results are available, using $KDE30$ with SANE improves performance over previously published results with S_{KDE30}, see Table 6.5.3 MNIST and SVHN results, e.g. epoch 0. We credit that to the better reconstruction quality of pre-training with SANE. Further, our sampling methods improve performance over S_{KDE30}. We compare SANE+ S_{KDE30} with SANE+ subsampling and SANE+ bootstrapping, for example in Table 6.5.4 on CIFAR-10 at epoch 0 from 64.8% to 68.1%, or on Tiny Imagenet from 8.4% to 11.1%. Using bootstrapping to adjust $\mathcal{P}$ iteratively further improves the sampled models slightly. It even allows to replace prompt examples with a Gaussian prior $\mathcal{P}$. The results of SANE$_{GAUSS}$ show high performance after fine-tuning, even the highest overall on MNIST. These results show that our sampling methods not only drop requirements for the prompt examples but even improve the performance of the sampled models.

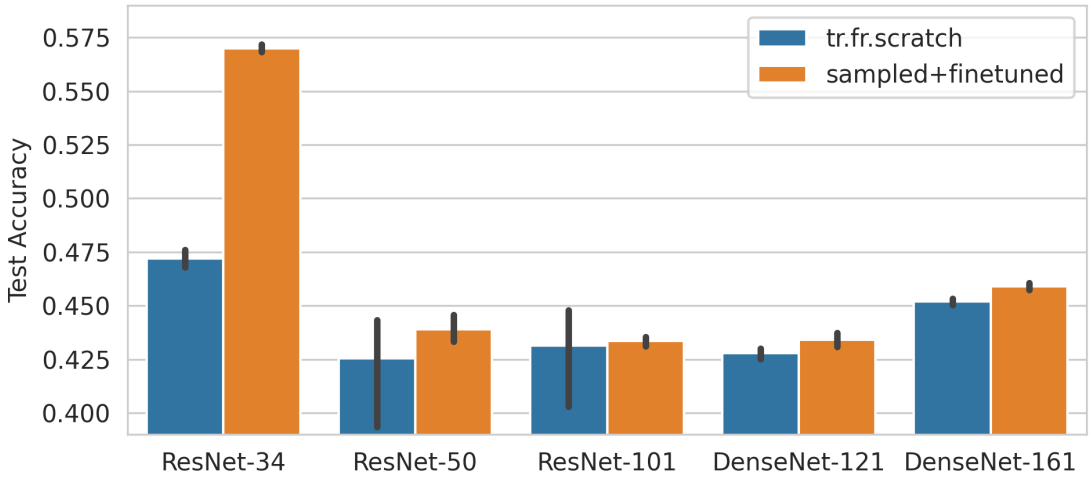


Figure 6.5.1: Comparison between sampled models and random initialization trained for 5 epochs Tiny-Imagenet. Different architectures are sampled from SANE pretrained on a ResNet-18 CIFAR-100 zoo. Although both models and task are changed, sampled models perform better.

Few-Shot Model Sampling Transfers to New Tasks and Architectures. Lastly, we explore whether sampling models using SANE generalize beyond the original task and architecture with very few prompt examples. Such transfers are out of reach of previous *hyper-representations*, which are bound to a fixed number of weights. SANE on the other hand represents models of different sizes or architectures simply as sequences of different lengths, which can vary between pretraining and sampling. Since we use the prompt examples only to roughly model the sampling distribution, we need only a few (1-5) prompt examples which are trained for only a few epochs (1-5). That way, sampling for new architectures and/or tasks can become very efficient. We test that idea in three experiments: (i) *changing the tasks* between pretraining and prompt-examples from CIFAR-100 to Tiny-Imagenet (Table 6.5.5); (ii) *changing the architecture* between pretraining and prompt-examples from ResNet-18 to ResNet-34 (Table 6.E.3); and (iii) *changing both task and architecture* from ResNet-18 on CIFAR-100 to ResNet-34 on Tiny-Imagenet (Figure 6.5.1 and Table 6.E.4).

In all three experiments, using target prompt examples improves over random initialization as well as previous transfer experiments. This indicates that SANE representations contain useful features even for new architectures or tasks. The sampled models outperform the prompt examples and training from scratch, considerably in earlier epochs, and preserve a performance advantage throughout fine-tining.

Table 6.5.5: Sampling ResNet-18 models for Tiny-Imagenet. SANE was pre-trained on CIFAR-100, 15 samples are drawn using subsampling, and 5 prompt examples are taken from the Tiny-Imagenet ResNet-18 zoo at epoch 25 with a mean accuracy of 43%.

ResNet-18 CIFAR100 to TinyImagnet		
Ep.	Method	Acc TI
0	tr. fr. scratch	0.5±0.0
	SANE	0.6±0.0
1	tr. fr. scratch	10.4±2.2
	SANE	**39.4±1.5**
2	tr. fr. scratch	28.5±0.9
	SANE	**61.0±0.2**
2	SANE Ens.	64.0

Sampling for a new task (Table 6.5.5), the sampled models outperform the prompt examples after just two epochs of fine-tuning, which indicates that transfer-learning using SANE is an efficient alternative. Sampling from ResNet-18 to ResNet-34 for the same task (Table 6.E.3) shows likewise improved performance over training from scratch, which indicates that the learned representation generalizes to larger architectures as well. Sampling for new tasks and different architecture (Figure 6.5.1 and Table 6.E.4) combines the previous experiments and confirms their results. Sampled models outperform training from scratch by a considerable margin. Figure 6.5.1 indicates that with increasing distance from the pretraining architecture for SANE, the performance gain of sampled models decreases, e.g. with

increasing ResNet size. Additionally, since sampling models using SANE is cheap and lends itself to ensembling, we investigate the diversity of sampled models in Sec. 6.E in the Appendix. Taken together, the experiments show that SANE learns representations that can generalize beyond the pretraining task and architecture, and can efficiently be sampled for both new tasks and architectures.

6.6 Related Work

Representation learning in the space of Neural Network weights has become a growing field recently. Several methods with different approaches to deal with weight spaces have been proposed to predict model properties such as accuracy [2, 43, 159, 180] or learn the encoded concepts [3, 33]. Other work investigates the structure of trained weights on a fundamental level, using their eigen or singular value decompositions to identify training phases or predict properties [113, 114, 116, 117, 121, 173]. Taking an optimization perspective, other work has investigated the uniqueness of the basis of trained neural networks [1, 14]. Other work identifies subspaces of weights that are relevant, which motivates our work [6, 49, 112, 167]. The mode connectivity of trained models has been investigated to improve understanding of how to train models [41, 50, 131].

A different line of work trains models to generate weights for target models, such as HyperNetworks [61, 88, 130, 179], with a recurrent backbone [164] as learned initialization [31] or for meta learning [47, 127, 182]. While the last category uses data to get learning signals, another line of work learns representations of the weights directly. Hyper-Representations train an encoder-decoder architecture using reconstruction of the weights, with contrastive guidance and has been proposed to predict model properties [147] or generate new models [148, 149]. While previous work was limited to small models of fixed length, this paper proposes methods to decouple the representation learner size from the base model. Related approaches use convolutional auto-encoders [7] or diffusion on the weights [136].

6.7 Conclusion

In this work, we propose SANE to learn task-agnostic representations of models. SANE decouples model tokenization from *hyper-representation* learning and can generalize to larger or smaller models of different architectures. We analyze SANE embeddings and find they reveal model quality metrics. Experimental evaluations show that i) SANE embeddings are predictive of model performance and ii) sampling models with SANE achieve high performance and generalize to larger models and new architectures. Further, we propose sampling methods that reduce quality and quantity requirements for prompt examples and allow the targeting of new model distributions.

Limitations

In this paper, we use homogeneous zoos with one architecture. This simplifies alignment for pre-training, but more importantly levels the playing field for model generation. Since SANE can train on varying architectures and model sizes, the model population requirement for pre-training is significantly relaxed. A sufficient number of models are available on public model hubs. Further, our sampling method requires access to prompt examples, to have an informed prior from which to sample. For small models, bootstrapping from a Gaussian finds the targeted distribution. For large models, that approach is too expensive for now, which is why we rely on prompt examples. Lastly, in this paper, we perform experiments only on computer vision tasks. This is a choice to simplify the experiment setup and evaluation.

Appendix

6.A Ablation Studies

In this section, we perform ablation studies to assess the effectiveness of the methods proposed above: model alignment to simplify the learning task, inference window size to improve inference quality, haloing, and batch-norm conditioning to increase sample quality.

Impact of Model Alignment. Model alignment intuitively reduces training complexity by mapping all models to the same subspace. To evaluate its impact, we conduct training experiments with the same configuration on datasets with and without aligned models. In the dataset with aligned models, we use either the aligned form or 5 random permutations for the two views for both reconstruction and contrastive learning. As shown in Table 6.A.1, the results show two effects. First, alignment through git re-basin simplifies the learning task and contributes to improved generalization, both training and test losses are reduced by more than 50%. Second, anchoring at least one of the views to the aligned form does further reduce the training loss, but does not improve generalization.

Table 6.A.1: Impact of alignment ablation and permutation on reconstruction loss.

Sample Permutations			$\mathcal{L}_{rec}$	
Aligned	View 1	View 2	Train	Test
No	Perm.	Perm.	0.304	0.167
Yes	Perm.	Perm.	0.148	0.082
Yes	Align	Perm.	0.107	0.082
Yes	Align	Align	0.072	0.082

Window Size Ablation. The sequential decomposition of `SANE` allows to pretrain not on the full model sequence, but on subsequences. The choice of the lenght of the subsequence, the window size, is as a critical parameter that balances computational load and context. We used a window of 256 for pretraining for most of our experiments.

Here, we study the influence of the window size on reconstruction error, exploring values ranging from 32 to 2048. Our experiments did not reveal substantial impact of

smaller windows on pretraining loss or sampling performance. This seems to either suggest that a window size as large as 2048 may still be insufficient on ResNets to capture enough context. Alternatively, it may suggest that the underlying assumption that context matters may not entirely hold up.

However, we did observe an important impact on the relationship between training and inference window sizes. During inference, memory load is significantly lower. Inference allows much larger window sizes, up to the entire length of the ResNet sequence. However, departing from the training window size appears to introduce interference, which affects the reconstruction error (Figure 6.A.1).

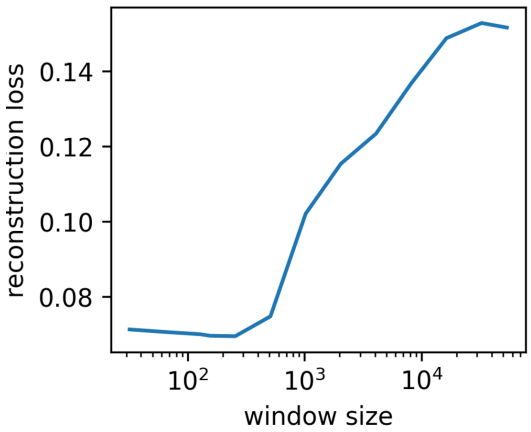


Figure 6.A.1: `SANE` reconstruction loss over number of tokens within a window. The loss is lowest around the training window-size of 256 tokens, longer sequences up to the full models sequence length of 50k tokens cause interference and double the reconstruction error.

Halo and BN conditioning. Haloing and batch-norm conditioning aim at reducing noise in model sampling, see Section 6.2. To assess their impact on sampling performance, we conduct an in-domain experiment using `SANE` trained on CIFAR-10 ResNet-18s, using prompt examples from the train set and fine-tuning on CIFAR-10. We compare with *naïve* sampling without Haloing and BN conditioning. The results in Table **??** show the significant improvements achieved by both haloing and batch-norm conditioning. From random guessing of naïve sampling, combining both improve to around 65%. Since both methods aim at reducing noise for zero shot sampling, their effect is largest then and diminishes somewhat during finetuning. Both methods not only improve zero-shot sampling per se, but make the sampled models provide enough signal to facilitate subsampling or bootstrapping strategies.

6.B SANE Architecture Details

In Table 6.B.1, we provide additional information on the training hyper-parameters for SANE on populations of small CNNs as well as ResNet18s. These values are the stable mean across all experiments, exact values can vary from population to population. Full experiment configurations are documented in the code.

Table 6.B.1: Architecture Details for SANE

Hyper-Parameter	CNNs	ResNet-18
tokensize	289	288
sequence lenght	∼50	∼50k
window size	32	256, 512
d_model	1024	2048
latent_dim	128	128
transformer layers	4	8
transformer heads	4,8	4,8

6.C SANE Embedding Analysis - Additional Results

This section contains additional results on SANE embedding analysis. In Figure 6.C.1, we compare the eigenvalue distribution for different models with SANE embeddings. Replicating the experiment setup from [116, 117], we train MiniAlexNet models on CIFAR-10 varying only the batch size. With smaller batch size and longer training duration, the eigenvalue distribution transitions from random with very few spikes, over a bulk with many spikes, to heavy tailed. The embeddings of SANE appear to also become more heavy tailed, but do not seem to pick up on the change from few to many spikes.

Figures 6.C.2 and 6.C.2 compare SANE with different WeightWatcher metrics on VGGs from pytorchcv [151] and the ResNet-18 zoo from the modelzoo dataset [150].

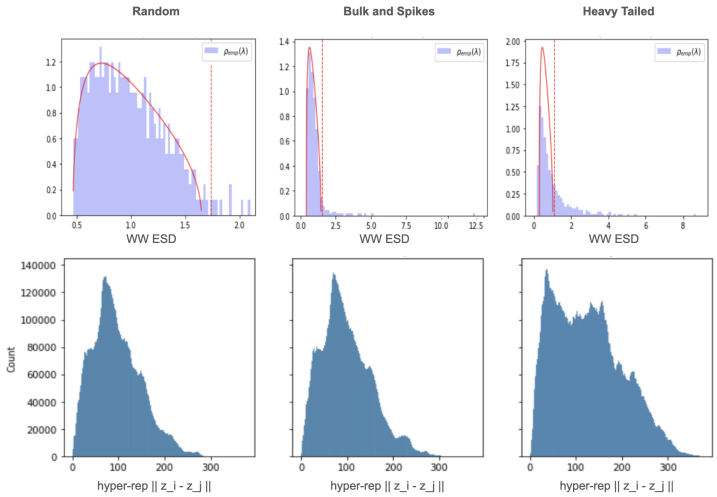


Figure 6.C.1: Comparison between WeightWatcher features (top) and `SANE` (bottom). [116] identify different phases in the eigenvalue spectrum of trained weight matrices. We replicate the experiment setup and find ESDs similar to *random* (top left), *bulk and spikes* (top middle) and *heavy tailed* (top right). We compare these against pairwise distances of `SANE` embeddings of the same layer. While the distributions have a different shape, it appears to become more heavy tailed going from *random* to *heavy tailed*.

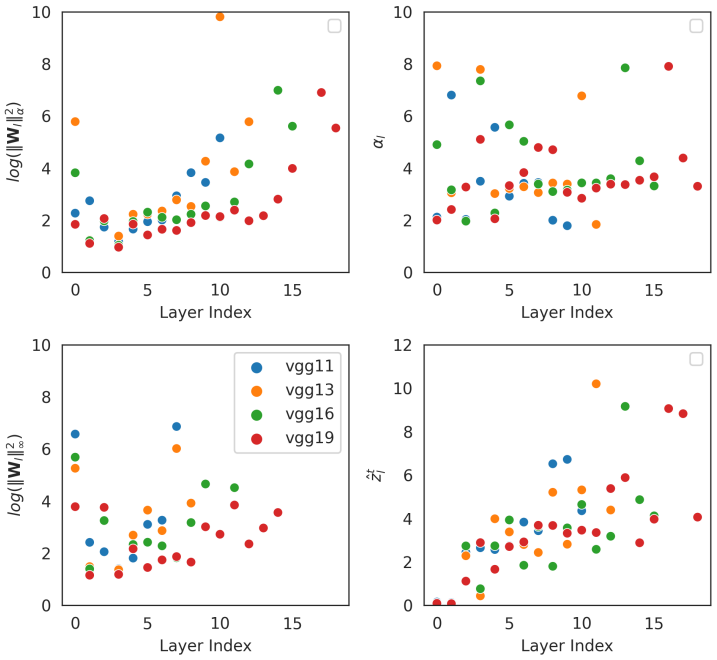


Figure 6.C.2: Comparison between different WeightWatcher (WW) features (left) and `SANE` (right). Features over layer index for VGGs from pytorchcv of different sizes. `SANE` shows similar trends to WW, low values at early layers and a sharp increase at the end.

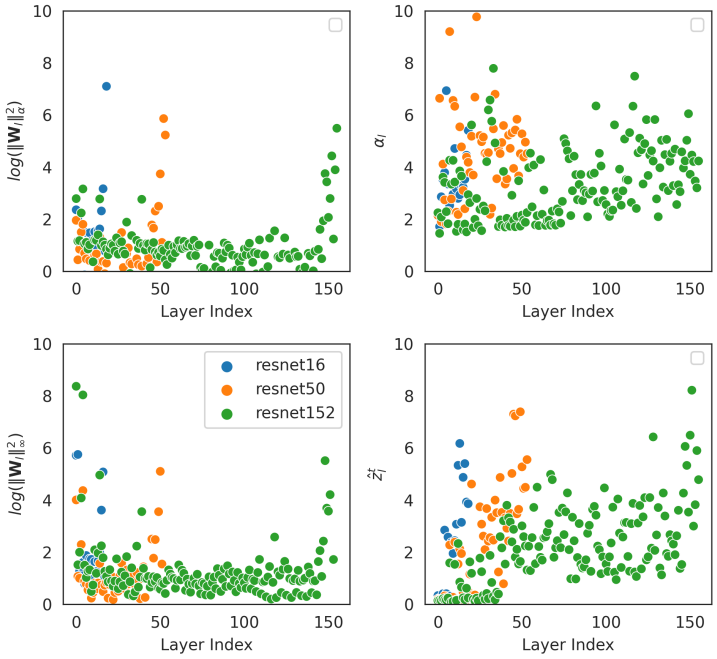


Figure 6.C.3: Comparison between different WeightWatcher (WW) features (left) and SANE (right). Features over layer index for Resnets from pytorchcv of different sizes. SANE shows similar trends to WW, low values at early layers and a sharp increase at the end.

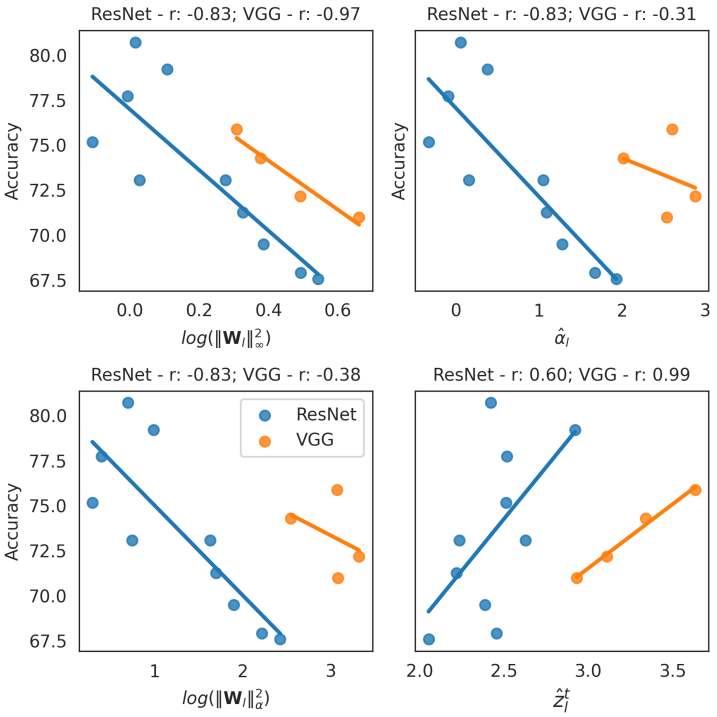


Figure 6.C.4: Comparison between WeightWatcher features (left) and SANE (right). Accuracy over model features for Resnets and VGGs from pytorchcv of different sizes. SANE shows similar trends to WW, low values at early layers and a sharp increase at the end.

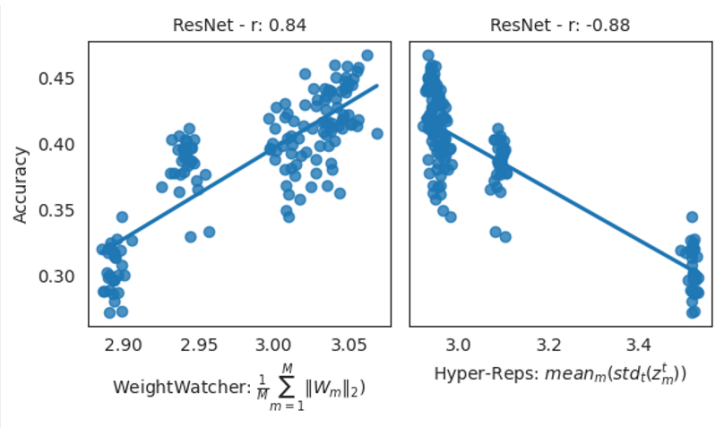


Figure 6.C.5: Comparison between WeightWatcher features (left) and `SANE` (right). Accuracy over model features for ResNets from the ResNet model zoo. Although `SANE` is pretrained in a self-supervised fashsion, it preserves the linear relation of a globally-aggregated embedding to model accuracy.

6.D Model Property Prediction - Additional Results

In this section we provide additional details for Section 6.5.1. Table 6.D.1 shows full results for populations of small CNNs.

Table 6.D.1: Property prediction on populations of small CNNs.

	MNIST			SVHN			CIFAR-10 (CNN)			STL		
	W	s(W)	SANE	W	s(W)	SANE	W	s(W)	SANE	W	s(W)	SANE
ACC	0.965	**0.987**	0.978	0.910	0.985	**0.991**	-7.580	**0.965**	0.885	-18.818	**0.919**	0.305
Epoch	0.953	**0.974**	0.958	0.833	**0.953**	0.930	0.636	**0.923**	0.771	-1.926	**0.977**	0.344
Ggap	0.246	0.393	**0.402**	0.479	0.711	**0.760**	0.324	**0.909**	0.772	-0.617	**0.858**	0.307

Comparison to Previous Work

Here, we compare SANE with previous work to disseminate the information contained in model embeddings. The experiment setup in this paper is designed around the ResNets, and therefore uses sparse epochs for computational efficiency. For consistency, we use the same setup for the CNN zoos as well. The exact numbers are therefore not directly comparable to Schürholt et al. [147]. To provide as much context as possible, we approach the comparison from two angles:

(1) **Direct comparison to the published results:** to contextualize, we use the (deterministic) results of weight statistics $s(W)$ to adjust for the differences in setup. We mark the results for $s(W)$ from Schürholt et al. [147] as $s(W)_{pp}$ and compare to their $E_{c+}D$ where possible.

(2) **Approximation of the effect of global embeddings:** previous work used global model embeddings, which we approximate by using the full model embedding sequence. We therefore compare SANE + aggregated tokens (as proposed in the submission) to SANE + full model sequence (similar to Schürholt et al. [147]).

The results in Tables 6.D.2, 6.D.3 and 6.D.4 allow the following conclusions:

(1) **SANE matches the performance of previous work:** The only data available for direct comparison is the MNIST zoo. Here, both in direct comparison and in relation to s(W) cross-relating our results with published numbers, SANE matches published performance of $E_c + D$. On other zoos, $E_c + D$ had similar performance to $s(W)$. We likewise find SANE embeddings to have similar performance to $s(W)$ in our experiments.

(2) **SANE + full sequence improves downstream task performance over the SANE + aggregated sequence:** That indicates that SANE + full sequence contains more information for model prediction. However, both Schürholt et al. [147] and SANE with full sequence have the disadvantage that they do not scale. With growing models, the representation learner of Schürholt et al. [147] and the input to the linear probe of SANE + full sequence grow accordingly. SANE + aggregated sequence does lose some information on small models, but scales gracefully to large models and remains competitive.

Table 6.D.2: Property Prediction comparison to previous work on the MNIST-CNN model zoo. We compare our linear probing results from weights W, layer-wise quintiles $s(W)$, embeddings from SANE either aggregated into one embedding or using the full sequence to results previously published in Schürholt et al. [147]. We mark their results for $s(W)$ as $s(W)_{pp}$. Since the experimental setup is not the same, the numbers of $s(W)$ do not match.

	W	$s(W)$	SANE aggregated	SANE full sequence	$s(W)_{pp}$	$E_{c+}D$
ACC	0.965	**0.987**	0.978	**0.987**	0.977	0.973
Epoch	0.953	0.974	0.958	**0.975**	0.987	0.989
Ggap	0.246	0.393	0.402	**0.461**	0.662	0.667

Table 6.D.3: Property Prediction comparison to previous work on the SVHN-CNN model zoo. We compare our linear probing results from weights W, layer-wise quintiles $s(W)$, to embeddings from SANE either aggregated into one embedding or using the full sequence. For this zoo, previous results are not available.

	W	$s(W)$	SANE aggregated	SANE full sequence	$s(W)_{pp}$	$E_{c+}D$
ACC	0.910	**0.985**	0.991	**0.993**	n/a	n/a
Epoch	0.833	**0.953**	0.930	**0.943**	n/a	n/a
Ggap	0.479	0.711	0.760	**0.77**	n/a	n/a

Table 6.D.4: Property Prediction comparison to previous work on the CIFAR-CNN(m) model zoo. We compare our linear probing results from weights W, layer-wise quintiles $s(W)$, to embeddings from SANE either aggregated into one embedding or using the full sequence. For this zoo, previous results are not available.

	W	$s(W)$	SANE aggregated	SANE full sequence	$s(W)_{pp}$	$E_{c+}D$
ACC	-7.580	**0.965**	0.885	**0.947**	n/a	n/a
Epoch	0.636	**0.923**	0.771	**0.879**	n/a	n/a
Ggap	0.324	**0.909**	0.772	**0.811**	n/a	n/a

6.E Model Generation - Additional Results

This section contains additional results from model sampling experiments, extending Section 6.5.2. In Table 6.E.1, we show results on small CNNs transferring to a new task. Similarly, Table 6.E.2 shows results on ResNet-18 models for task transfers. Lastly, Tables 6.E.3, 6.E.4 and 6.E.5 contain additional results for transferring from ResNet-18 CIFAR-100 to ResNet34 and/or Tiny-Imagenet.

Table 6.E.1: Model generation on CNN model populations transfer learned on a new task. We compare sampled models at different epochs with models trained from scratch and models fine-tuned from the anchor samples.

Method	SVHN to MNIST			CIFAR-10 to STL-10		
	Epoch 0	Epoch 1	Epoch 25	Epoch 0	Epoch 1	Epoch 25
tr.fr.scratch	∼10 /%	20.6+-1.6	83.3+-2.6	∼10 /%	21.3+-1.6	44.0+-1.0
pretrained	29.1+-7.2	84.1+-2.6	94.2+-0.7	16.2+-2.3	24.8+-0.8	49.0+-0.9
S_{KDE30}	31.8+-5.6	86.9+-1.4	95.5+-0.4	n/a	n/a	n/a
SANE $_{KDE30}$	**40.2+-4.8**	86.7+-1.6	94.8+-0.4	15.5+-2.3	24.9+-1.6	49.2+-0.5
SANE $_{SUB}$.	37.9+-2.8	**88.2+-0.5**	**95.6+-0.3**	**17.4+-1.4**	**25.6+-1.7**	**49.8+-0.6**

Diversity of sampled models

An interesting question is whether sampling SANE generates versions of the same model. To test that, we evaluate the diversity of samples generated with only a few few-shot examples by combining the models to ensembles. The improvements of the ensembles over the individual models demonstrate their diversity. This indicates that given very few, early-stage prompt examples, sampling hyper-representations improves learning speed and performance in otherwise equal settings. Additionally, we conducted experiments with varying numbers of prompt examples, revealing that increasing the number of prompt examples enhances both performance and diversity. Nonetheless, even a single prompt example trained for just 2 epochs contains sufficient information to generate model samples that surpass those derived from random initialization, see Table 6.E.5.

Table 6.E.2: Model generation on ResNet-18 model populations transferred to a new task. We compare sampled models at different transfer learning epochs with models trained from scratch and models fine-tuned from the same anchor samples.

Epoch	Method	CIFAR-10 to CIFAR-100	CIFAR-100 to Tiny-Imagenet	Tiny-Imagenet to CIFAR-100
	tr. fr. scratch	∼1 /%	∼0.5 /%	∼1 /%
	Finetuned	1.0+-0.3	0.5+-0.0	1.1+-0.2
0	SANE $_{KDE30}$	1.0+-0.3	0.5+-0.1	1.0+-0.2
	SANE $_{SUB}$	1.0+-0.3	0.6+-0.0	1.1+-0.2
	SANE $_{BOOT}$	1.1+-0.2	0.5+-0.0	0.9+-0.2
	tr. fr. scratch	17.5+-0.7	13.8+-0.8	17.5+-0.7
	Finetuned	27.5+-1.3	25.7+-0.5	51.7+-0.5
1	SANE $_{KDE30}$	26.8+-1.4	21.5+-0.9	40.2+-1.0
	SANE $_{SUB}$	26.4+-1.9	21.5+-1.0	40.63+-1.3
	SANE $_{BOOT}$	25.7_01.9	21.7+-1.0	40.9+-0.8
	tr. fr. scratch	36.5+-2.0	31.1+-1.6	36.5+-2.0
	Finetuned	45.7+-1.0	36.3+-2.5	52.6+-1.3
5	SANE $_{KDE30}$	44.5+-2.0	36.3+-1.2	47.2+-3.3
	SANE $_{SUB}$	45.6+-1.2	35.8+-1.4	49.8+-2.3
	SANE $_{BOOT}$	43.3+-2.4	**37.3+2.0**	50.2+-3.4
	tr. fr. scratch	53.3+-2.0	38.5+-1.9	53.3+-2.0
	Finetuned	71.9+-0.1	63.4+-0.2	**73.9+-0.3**
15	SANE $_{KDE30}$	71.8+-0.3	**63.6+-0.2**	73.4+-0.2
	SANE $_{SUB}$	72.0+-0.2	**63.6+-0.3**	73.5+-0.2
	SANE $_{BOOT}$	71.9+-0.3	63.4+-0.1	73.7+-0.3
25	tr. fr. scratch	56.5+-2.0	43.3+-1.9	56.5+-2.0
50	tr. fr. scratch	70.7+-0.4	57.3+-0.6	70.7+-0.4
60	tr. fr. scratch	74.2+-0.3	63.9+-0.5	74.2+-0.3

Table 6.E.3: Few-shot model generation for a new task: Sampling ResNet-34 models for CIFAR-100. SANE was pretrained on CIFAR-100 ResNet-18s, 5 samples are drawn using subsampling. To get prompt examples, we train 3 ResNet-34 models on CIFAR-100 for 2 epochs to a mean accuracy of 26 %.

CIFAR100 ResNet-18 to ResNet-34			
Epoch	Method	5 Epochs	15 Epochs
0	tr. fr. Scratch	1.0±0.1	1.0±0.1
	SANE	**1.6±0.3**	**1.6±0.3**
1	tr. fr. Scratch	12.4±1.0	12.9±0.8
	SANE	**16.8±0.7**	**23.1±0.3**
5	tr. fr. Scratch	49.5±0.6	36.2±1.7
	SANE	**51.9±0.6**	**37.8±1.4**
15	tr. fr. scratch		68.8±0.4
	SANE		**69.3±0.3**
	SANE Ens.	53.5	71.3

Table 6.E.4: Few-shot model generation for a new task and architecture: SANE trained on CIFAR-100 ResNet-18s used to generate ResNet-34s for Tiny-Imagenet. 5 samples are drawn using subsampling. To get prompt examples, we train 3 ResNet-34 models on Tiny-Imagenet for 2 epochs to a mean accuracy of 28.5 %.

ResNet-18 CIFAR100 to ResNet-34 Tiny-Imagenet			
Epoch	Method	5 epochs	15 epochs
0	tr. fr. Scratch	0.5±0.0	0.5±0.0
	SANE	0.5±0.1	0.6±0.2
1	tr. fr. Scratch	10.5±1.4	11.9±1.9
	SANE	**13.3±0.5**	**18.5±0.7**
5	tr. fr. Scratch	47.2±0.7	31.1±1.7
	SANE	**50.6±0.3**	**31.6±0.6**
15	tr. fr. Scratch		61.9±0.3
	SANE		**62.7±0.3**
	SANE Ens.	52	65.1

Table 6.E.5: Sampling ResNet-34 models for CIFAR-100. SANE was pretrained on CIFAR-100 ResNet-18s, 5 samples are drawn using subsampling. To get prompt examples, we train a single ResNet-34 model on CIFAR-100 for 2 epochs to an accuracy of 26 %.

CIFAR100 ResNet-18 to ResNet-34			
Epoch	Method	5 Epochs	15 Epochs
0	tr. fr. Scratch	1.0+-0.1	1.0+-0.1
	SANE	**1.5+-0.2**	**1.6+-0.1**
1	tr. fr. Scratch	12.4+-1.0	12.9+-0.8
	SANE	**16.9+-0.7**	**19.4+-0.2**
5	tr. fr. Scratch	49.5+-0.6	36.2+-1.7
	SANE	**51.5+-0.3**	**38.6+-1.6**
15	tr. fr. scratch		68.8+-0.4
	SANE		**69.1+-0.1**
Ensemble	SANE	51.8	70.2

Bibliography

[1] Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git Re-Basin: Merging Models modulo Permutation Symmetries, September 2022.

[2] Bruno Andreis, Soro Bedionita, and Sung Ju Hwang. Set-based Neural Network Encoding, May 2023.

[3] Maor Ashkenazi, Zohar Rimon, Ron Vainshtein, Shir Levi, Elad Richardson, Pinchas Mintz, and Eran Treister. NeRN – Learning Neural Representations for Neural Networks, December 2022.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *arXiv:1206.5538 [cs]*, April 2014.

[5] Yoshua Bengio, Dong-Hyun Lee, Jörg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *CoRR*, abs/1502.04156, 2015.

[6] Gregory W. Benton, Wesley J. Maddox, Sanae Lotfi, and Andrew Gordon Wilson. Loss Surface Simplexes for Mode Connecting Volumes and Fast Ensembling. In *PMLR*, 2021.

[7] Gianluca Berardi, Luca De Luigi, Samuele Salti, and Luigi Di Stefano. Learning the Space of Deep Models, June 2022.

[8] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[9] 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.

[10] Christopher M Bishop. *Pattern Recognition and Machine Learning*. springer, 2006.

[11] Avrim Blum and Ronald L Rivest. Training a 3-Node Neural Network is NP-Complete. In *NIPS*, page 8, 1988.

[12] Avrim Blum, John Hopcroft, and Ravindran Kannan. Foundations of Data Science. 2018.

[13] Tim Brooks, Bill Peebles, Connor Homes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Wing Yin Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. 2024. URL `https://openai.com/research/video-generation-models-as-world-simulators`.

[14] Davis Brown, Nikhil Vyas, and Yamini Bansal. On Privileged and Convergent Bases in Neural Network Representations, July 2023.

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020.

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

[17] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Dataset Distillation by Matching Training Trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 10, 2022.

[18] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative Pretraining From Pixels. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1691–1703. PMLR, November 2020.

[19] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2Net: Accelerating Learning via Knowledge Transfer. In *International Conference on Learning Representations (ICLR)*, April 2016. doi: 10.48550/arXiv.1511.05641.

[20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *arXiv:2002.05709 [cs, stat]*, June 2020.

[21] 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.

[22] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Qiuyi Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards Learning Universal Hyperparameter Optimizers with Transformers, October 2022.

[23] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Qiuyi Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards Learning Universal Hyperparameter Optimizers with Transformers, October 2022.

[24] Rajas Chitale, Ankit Vaidya, Aditya Kane, and Archana Ghotkar. Task Arithmetic with LoRA for Continual Learning. https://arxiv.org/abs/2311.02428v1, November 2023.

[25] 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.

[26] 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.

[27] Piotr Dabkowski and Yarin Gal. Real Time Image Saliency for Black Box Classifiers. *arXiv:1705.07857 [stat]*, May 2017.

[28] Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. Scalable Deep Generative Modeling for Sparse Graphs. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2302–2312. PMLR, November 2020.

[29] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, June 2022.

[30] Yann N Dauphin and Samuel Schoenholz. Metainit: Initializing learning by learning to initialize. *Advances in Neural Information Processing Systems*, 32, 2019.

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

[32] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, page 9, 2014.

[33] Luca De Luigi, Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. Deep Learning on Implicit Neural Representations of Shapes, February 2023.

[34] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. page 8.

[35] Misha Denil, Babak Shakibi, Laurent Dinh, and Marc'Aurelio Ranzato. Predicting Parameters in Deep Learning. In *Neural Information Processing Systems (NeurIPS)*, page 9, 2013.

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

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://arxiv.org/abs/1810.04805v2, October 2018.

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

[39] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp Minima Can Generalize For Deep Nets. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1019–1028. PMLR, July 2017.

[40] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*, October 2020.

[41] Felix Draxler, K. Veschgini, M. Salmhofer, and F. Hamprecht. Essentially No Barriers in Neural Network Energy Landscape. In *International Conference on Machine Learning*, March 2018.

[42] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An Embodied Multimodal Language Model, March 2023.

[43] 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.

[44] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A Survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. ISSN 1533-7928.

[45] Li Fei Fei, Jia Deng, Minh Do, Hao Su, and Kai Li. Construction and Analysis of a Large Scale Image Ontology. page 1.

[46] Yunzhen Feng, Runtian Zhai, Di He, Liwei Wang, and Bin Dong. Transferred Discrepancy: Quantifying the Difference Between Representations. *arXiv:2007.12446 [cs, stat]*, July 2020.

[47] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1126–1135. PMLR, July 2017.

[48] William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6(6):771–783, January 1993. ISSN 0893-6080. doi: 10.1016/S0893-6080(05)80122-4.

[49] Stanislav Fort and Stanislaw Jastrzebski. Large Scale Structure of Neural Network Loss Landscapes. *arXiv:1906.04724 [cs, stat]*, June 2019.

[50] Jonathan Frankle, G. Dziugaite, Daniel M. Roy, and Michael Carbin. Linear Mode Connectivity and the Lottery Ticket Hypothesis. *ArXiv*, December 2019.

[51] T. Garipov, Pavel Izmailov, Dmitrii Podoprikhin, D. Vetrov, and A. Wilson. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. *ArXiv*, February 2018.

[52] Paul Gavrikov and Janis Keuper. CNN Filter DB: An Empirical Investigation of Trained Convolutional Filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 11, 2022.

[53] Partha Ghosh, Mehdi S. M. Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From Variational to Deterministic Autoencoders. In *arXiv:1903.12436 [Cs, Stat]*, May 2020.

[54] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR*, page 8, 2010.

[55] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Conference on Neural Information Processing Systems (NeurIPS)*, page 9, 2014.

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

[57] Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. Qualitatively characterizing neural network optimization problems. *arXiv:1412.6544 [cs, stat]*, May 2015.

[58] J. Elisenda Grigsby, Kathryn Lindsey, and David Rolnick. Hidden symmetries of ReLU networks, June 2023.

[59] 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.

[60] 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.

[61] David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. In *arXiv:1609.09106 [Cs]*, 2016.

[62] Boris Hanin and David Rolnick. How to Start Training: The Effect of Initialization and Architecture. *arXiv:1803.01719 [cs, stat]*, November 2018.

[63] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, October 1990. ISSN 1939-3539. doi: 10.1109/34.58871.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *arXiv:1502.01852 [Cs]*, 2015.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[66] 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.

[67] Robert Hecht-Nielsen. ON THE ALGEBRAIC STRUCTURE OF FEEDFORWARD NETWORK WEIGHT SPACES. In Rolf Eckmiller, editor, *Advanced Neural Computers*, pages 129–135. North-Holland, Amsterdam, January 1990. ISBN 978-0-444-88400-8. doi: 10.1016/B978-0-444-88400-8.50019-4.

[68] Vincent Herrmann, Francesco Faccio, and Jürgen Schmidhuber. Learning Useful Representations of Recurrent Neural Network Weight Matrices, March 2024.

[69] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[70] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks, January 2021.

[71] Dominik Honegger, Konstantin Schürholt, and Damian Borth. Sparsified Model Zoo Twins: Investigating Populations of Sparsified Neural Network Models. In *ICLR 2023 Workshop on Sparsity in Neural Networks*. arXiv, April 2023. doi: 10.48550/arXiv.2304.13718.

[72] Dominik Honegger, Konstantin Schürholt, Linus Scheibenreif, and Damian Borth. Eurosat Model Zoo: A Dataset and Benchmark on Populations of Neural Networks and Its Sparsified Model Twins. In *IGARSS 2023 - 2023 IEEE International Geoscience and Remote Sensing Symposium*, pages 888–891, July 2023. doi: 10.1109/IGARSS52108.2023.10283060.

[73] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Z. Kira. Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10948–10957, 2020.

[74] J.J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, May 1994. ISSN 1939-3539. doi: 10.1109/34.291440.

[75] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer International Publishing, Cham, 2019. ISBN 978-3-030-05317-8 978-3-030-05318-5. doi: 10.1007/978-3-030-05318-5.

[76] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing Models with Task Arithmetic, December 2022.

[77] Grammarly Inc. Grammarly: Free AI Writing Assistance, 2024.

[78] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population Based Training of Neural Networks. *arXiv:1711.09846 [cs]*, November 2017.

[79] 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.

[80] 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.

[81] 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.

[82] Jeremiah Johnson. Subspace Match Probably Does Not Accurately Assess the Similarity of Learned Representations. *arXiv:1901.00884 [cs, math, stat]*, January 2019.

[83] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-03819-2.

[84] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and Understanding Recurrent Networks. *arXiv:1506.02078 [cs]*, June 2015.

[85] M. G. KENDALL. A NEW MEASURE OF RANK CORRELATION. *Biometrika*, 30(1-2):81–93, June 1938. ISSN 0006-3444. doi: 10.1093/biomet/30.1-2.81.

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

[87] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2013.

[88] Boris Knyazev, Michal Drozdzal, Graham W. Taylor, and Adriana Romero-Soriano. Parameter Prediction for Unseen Deep Architectures. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[89] Boris Knyazev, Doha Hwang, and Simon Lacoste-Julien. Can We Scale Transformers to Predict Parameters of Diverse ImageNet Models? In *arXiv.Org*, March 2023.

[90] 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.

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

[92] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. page 60, 2009.

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

[94] 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.

[95] Lauro Langosco, Neel Alex, William Baker, David Quarel, Herbie Bradley, and David Krueger. Detecting Backdoors with Meta-Models. In *NeurIPS 2023 Workshop on Backdoors in Deep Learning - The Good, the Bad, and the Ugly*, October 2023.

[96] Ya Le and Xuan Yang. Tiny ImageNet Visual Recognition Challenge. page 6.

[97] Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. FFCV: Accelerating Training by Removing Data Bottleneck. In *Computer Vision and Pattern Recognition (CVPR)*, 2023.

[98] 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.

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

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

[101] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14539.

[102] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *ECML/PKDD (1)*, volume 9284 of *Lecture Notes in Computer Science*, pages 498–515. Springer, 2015.

[103] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. In *NIPS*, page 11, 2018.

[104] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A System for Massively Parallel Hyperparameter Tuning. In *Proceedings of the 3 Rd MLSys Conference*, Austin, TX, USA, 2020. arXiv.

[105] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research (JMLR)*, 18, June 2018.

[106] 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.

[107] 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:1807.05118 [cs, stat]*, July 2018.

[108] Iou-Jen Liu, Jian Peng, and Alexander G. Schwing. Knowledge Flow: Improve Upon Your Teachers. In *International Conference on Learning Representations (ICLR)*, April 2019.

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

[110] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do We Actually Need Dense Over-Parameterization? In-Time Over-Parameterization in Sparse Training. In *International Conference on Machine Learning*, pages 6989–7000. PMLR, July 2021.

[111] James Lucas, Juhan Bae, Michael R Zhang, Stanislav Fort, Richard Zemel, and Roger Grosse. Analyzing Monotonic Linear Interpolation in Neural Network Loss Landscapes. page 12.

[112] James R. Lucas, Juhan Bae, Michael R. Zhang, Stanislav Fort, Richard Zemel, and Roger B. Grosse. On Monotonic Linear Interpolation of Neural Network Parameters. In *International Conference on Machine Learning*, pages 7168–7179. PMLR, July 2021.

[113] C. H. Martin and M. W. Mahoney. Heavy-tailed Universality predicts trends in test accuracies for very large pre-trained deep neural networks. In *Proceedings of the 20th SIAM International Conference on Data Mining*, 2020.

[114] C. H. Martin, T. S. Peng, and M. W. Mahoney. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*, 12 (4122):1–13, 2021.

[115] Charles H. Martin and Michael W. Mahoney. Rethinking generalization requires revisiting old ideas: Statistical mechanics approaches and complex learning behavior, February 2019.

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

[117] Charles H. Martin and Michael W. Mahoney. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *The Journal of Machine Learning Research*, 22(1):165:7479–165:7551, January 2021. ISSN 1532-4435.

[118] 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.

[119] 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):4122, July 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-24025-8.

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

[121] Dan Meller and Nicolas Berkouk. Singular Value Representation: A New Graph Perspective On Neural Networks, February 2023.

[122] 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.

[123] Dmytro Mishkin and Jiri Matas. All you need is a good init. In *International Conference on Learning Representations (ICLR)*. arXiv, 2016. doi: 10.48550/arXiv.1511.06422.

[124] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[125] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational Dropout Sparsifies Deep Neural Networks. In *International Conference on Machine Learning (ICML)*, page 10, 2017.

[126] 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.

[127] Elvis Nava, Seijin Kobayashi, Yifei Yin, Robert K. Katzschmann, and Benjamin F. Grewe. Meta-Learning via Classifier(-free) Diffusion Guidance. October 2022.

[128] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant Architectures for Learning in Deep Weight Spaces, January 2023.

[129] 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.

[130] Phuoc Nguyen, Truyen Tran, Sunil Gupta, Santu Rana, and Hieu-Chi Dam. HyperVAE: A Minimum Description Length Variational Hyper-Encoding Network. page 7, 2019.

[131] Quynh N. Nguyen. On Connected Sublevel Sets in Deep Learning. In *International Conference on Machine Learning*, January 2019.

[132] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth. *arXiv:2010.15327 [cs]*, October 2020.

[133] OpenAI. ChatGPT, 2024.

[134] OpenAI. ChatGPT Model Versions, 2024.

[135] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, page 12, 2019.

[136] William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A. Efros, and Jitendra Malik. Learning to Learn with Generative Models of Neural Network Checkpoints, September 2022.

[137] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. 2018.

[138] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust Speech Recognition via Large-Scale Weak Supervision, December 2022.

[139] 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.

[140] Rahul Ramesh and Pratik Chaudhari. Model Zoo: A Growing "Brain" That Learns Continually. In *International Conference on Learning Representations ICLR*, 2022.

[141] 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.

[142] 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.

[143] 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.

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

[145] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Radioactive data: tracing through training. In *International Conference on Machine Learning*, pages 8326–8335. PMLR, 2020.

[146] Konstantin Schürholt and Damian Borth. An Investigation of the Weight Space to Monitor the Training Progress of Neural Networks, March 2021.

[147] Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-Supervised Representation Learning on Neural Network Weights for Model Characteristic Prediction. In *Conference on Neural Information Processing Systems (NeurIPS)*, volume 35, 2021.

[148] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i-Nieto, and Damian Borth. Hyper-Representations as Generative Models: Sampling Unseen Neural Network Weights. In *Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS)*, September 2022.

[149] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i-Nieto, and Damian Borth. Hyper-Representations for Pre-Training and Transfer Learning. In *First Workshop of Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, 2022.

[150] Konstantin Schürholt, Diyar Taskiran, Boris Knyazev, Xavier Giró-i-Nieto, and Damian Borth. Model Zoos: A Dataset of Diverse Populations of Neural Network Models. In *Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, September 2022.

[151] Oleg Sémery. Osmr/imgclsmob, January 2024.

[152] 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`.

[153] 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 (ICML)*, page 12, 2021.

[154] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961.

[155] Leslie N. Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates, May 2018.

[156] Felipe Petroski Such, Vashisht Madhavan, Rosanne Liu, Rui Wang, Pablo Samuel Castro, Yulun Li, Jiale Zhi, Ludwig Schubert, Marc G. Bellemare, Jeff Clune, and Joel Lehman. An Atari Model Zoo for Analyzing, Visualizing, and Comparing Deep Reinforcement Learning Agents, May 2019.

[157] 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.

[158] Hugo Touvron, Louis Martin, and Kevin Stone. Llama 2: Open Foundation and Fine-Tuned Chat Models.

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

[160] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *arXiv:1706.03762 [Cs]*, December 2017.

[161] Ashish Vaswani, Prajit Ramachandran, Aravind Srinivas, Niki Parmar, Blake Hechtman, and Jonathon Shlens. Scaling Local Self-Attention for Parameter Efficient Visual Backbones. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12894–12904, 2021.

[162] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435.

[163] Jiayun Wang, Yubei Chen, Stella X. Yu, Brian Cheung, and Yann LeCun. Recurrent Parameter Generators, July 2021.

[164] Jiayun Wang, Yubei Chen, Stella X. Yu, Brian Cheung, and Yann LeCun. Compact and Optimal Deep Learning with Recurrent Parameter Generators. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3889–3899, Waikoloa, HI, USA, January 2023. IEEE. ISBN 978-1-66549-346-8. doi: 10.1109/WACV56688.2023.00389.

[165] 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.

[166] 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, November 2018.

[167] Mitchell Wortsman, Maxwell C. Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning Neural Network Subspaces. In *International Conference on Machine Learning*, pages 11217–11227. PMLR, July 2021.

[168] Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, 2022.

[169] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo-Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

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

[171] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms, September 2017.

[172] Scott Yak, Javier Gonzalvo, and Hanna Mazzawi. Towards Task and Architecture-Independent Generalization Gap Predictors. *arXiv:1906.01550 [cs, stat]*, June 2019.

[173] Y. Yang, R. Theisen, L. Hodgkinson, J. E. Gonzalez, K. Ramchandran, C. H. Martin, and M. W. Mahoney. Evaluating natural language processing models with generalization metrics that do not need access to any training or testing data. Technical Report Preprint: arXiv:2202.02842, 2022.

[174] Yaoqing Yang, Liam Hodgkinson, Ryan Theisen, Joe Zou, Joseph E Gonzalez, Kannan Ramchandran, and Michael W Mahoney. Taxonomizing local versus global structure in neural network loss landscapes. In *Advances in Neural Information Processing Systems*, volume 34, pages 18722–18733. Curran Associates, Inc., 2021.

[175] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *PMLR*, volume 97. arXiv, 2019.

[176] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Neural Information Processing Systems (NeurIPS)*, November 2014.

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

[178] 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.

[179] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph HyperNetworks for Neural Architecture Search. In *International Conference on Learning Representations (ICLR)*, 2019.

[180] David W Zhang, Miltiadis Kofinas, Yan Zhang, Yunlu Chen, Gertjan J Burghouts, and Cees G M Snoek. Neural Networks Are Graphs!Graph Neural Networks for Equivariant Processing of Neural Networks. July 2023.

[181] Tianping Zhang, Shaowen Wang, Shuicheng Yan, Jian Li, and Qian Liu. Generative Table Pre-training Empowers Models for Tabular Prediction, May 2023.

[182] Andrey Zhmoginov, Mark Sandler, and Max Vladymyrov. HyperTransformer: Model Generation for Supervised and Semi-Supervised Few-Shot Learning. In *International Conference on Machine Learning (ICML)*, January 2022.

[183] 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.

[184] Allan Zhou, Kaien Yang, Kaylee Burns, Yiding Jiang, Samuel Sokota, J. Zico Kolter, and Chelsea Finn. Permutation Equivariant Neural Functionals, February 2023.

[185] Allan Zhou, Chelsea Finn, and James Harrison. Universal Neural Functionals, February 2024.

[186] Wen-Yang Zhou, Guo-Wei Yang, and Shi-Min Hu. Jittor-GAN: A fast-training generative adversarial network model zoo based on Jittor. *Computational Visual Media*, 7(1):153–157, March 2021. ISSN 2096-0433, 2096-0662. doi: 10.1007/s41095-021-0203-2.

[187] 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.

[188] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning. In *Proceedings of IEEE*, 2020.

[189] Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. Visualizing Deep Neural Network Decisions: Prediction Difference Analysis. *arXiv:1702.04595 [cs]*, February 2017.

Applied AI Software

Throughout the preparation of this thesis, the *Generative Pretrained Transformer (GPT)* 4 models by OpenAI [133], specifically version 'gpt-4-1106-preview' with 128k tokens context and training data up to April 2023 [134], served as an auxiliary tool for spell checking, and reviewing language and grammar. Additionally, the *Grammarly* software [77] was used to assist in the proofreading and refinement of the thesis. The inclusion of *GPT-4* and *Grammarly* aimed to elevate the linguistic quality of the manuscript, facilitating a clearer and more coherent presentation of the research.