Learning the Space of Deep Models

Learning the Space of Deep Models

Gianluca Berardi*, Luca De Luigi*, Samuele Salti, Luigi Di Stefano
Department of Computer Science and Engineering (DISI), University of Bologna, Italy
{gianluca.berardi3, luca.deluigi4, samuele.salti , luigi.distefano}@unibo.it
* Corresponding authors with equal contribution.

***Abstract*—Embedding of large but redundant data, such as images or text, in a hierarchy of lower-dimensional spaces is one of the key features of representation learning approaches, which nowadays provide state-of-the-art solutions to problems once believed hard or impossible to solve. In this work[1], in a plot twist with a strong meta aftertaste, we show how trained deep models are as redundant as the data they are optimized to process, and how it is therefore possible to use deep learning models to embed deep learning models. In particular, we show that it is possible to use representation learning to learn a fixed-size, low-dimensional embedding space of trained deep models and that such space can be explored by interpolation or optimization to attain ready-to-use models. We find that it is possible to learn an embedding space of multiple instances of the same architecture and of multiple architectures. We address image classification and neural representation of signals, showing how our embedding space can be learnt so as to capture the notions of performance and 3D shape, respectively. In the Multi-Architecture setting we also show how an embedding trained only on a subset of architectures can learn to generate already-trained instances of architectures it never sees instantiated at training time.**

I. Introduction

Representation learning has achieved remarkable results in embedding text, sound and images into low dimensional spaces, so as to map semantically close data into points close one to another into the learnt space. In recent years, deep learning has emerged as the most effective machinery to pursue representation learning, many scholars agreeing on representation learning laying at the very core of the deep learning paradigm. On the other hand, the success of network compression and pruning approaches [7] highlight the redundancy of parameters learned by a deep learning model, as in the Lottery Ticket Hypothesis [8], which shows that training as few parameters as 4% of those of the full network (i.e. the *winning tickets*) can attain similar or even higher performance.

Thus, we felt puzzling and worth investigating whether the parameter values of a trained deep model might be squeezed into a semantically meaningful low-dimensional latent space. Two questions arise: is it possible to train a deep learning model to learn to represent other, already trained, deep learning models? And according to which trait should two already trained models lay either close or further away in the latent space? The Lottery Ticket Hypothesis may suggest the existence of a low-dimensional key set of information that is shared by all possible sets of parameters for a predefined architecture that achieve comparable performance on a given task. Hence, it seems reasonable to conjecture that one might pursue learning of an embedding space shaped according to similarity in performance. Moreover, many recent works have demonstrated how small deep networks can be trained to fit accurately complex signals such as images[26], implicit representations of 3D surfaces [24], [22] and even radiance fields [23]. One might then be willing to embed such models into a space amenable to capture the similarity between the underlying signals.

In this paper we propose a first investigation along this new line of research. In particular, we show that it is possible to deploy a basic encoder-decoder architecture to learn a low-dimensional latent space of deep models and that such a space can be shaped so to exhibit a semantically meaningful structure. We posit that the loss to drive the learning process of our encoder-decoder architecture should entail functional similarity - rather than proximity of parameter values - between the input and output models. Accordingly, we train our architecture by knowledge distillation to drive the output model generated by the decoder to mimic the behaviour of the input model. In our study we address two settings: learning a latent space from a training set of models with the same network architecture and different parameter values as well as based on a training set comprising models with different architectures. In both settings, we show that the learnt latent space does posses a semantic structure as it is possible to sample new trained models with predictable behaviour by simple interpolation operations. Moreover, we show that in the Multi-Architecture setting a latent space trained on a set of architectures can generate already-trained models of architectures never seen instantiated at training time. Finally, we show that in both settings it is possible to train an architecture by performing latent space optimization on the low dimensional embedding space instead of optimizing directly the full set of parameters.

II. Related Work

Representations. Representation learning concerns the ability of a machine learning algorithm to transform the information contained in raw data in more accessible form. A common algorithm is the autoencoder [12], a self-supervised solution where the representation is learnt by constraining the output to reconstruct the input. Our architecture is inspired by the autoencoder but aim at producing outputs that behave akin to the input (e.g. similar performance on a certain task). In a recent meta-learning paper, LEO [25], the embedding of the weights of a single layer of a network is learnt for a few shot learning task. Task2Vec [1] learns a task embedding on different visual tasks which enables to predict similarities between them

[1]Code available at https://github.com/CVLAB-Unibo/netspace.

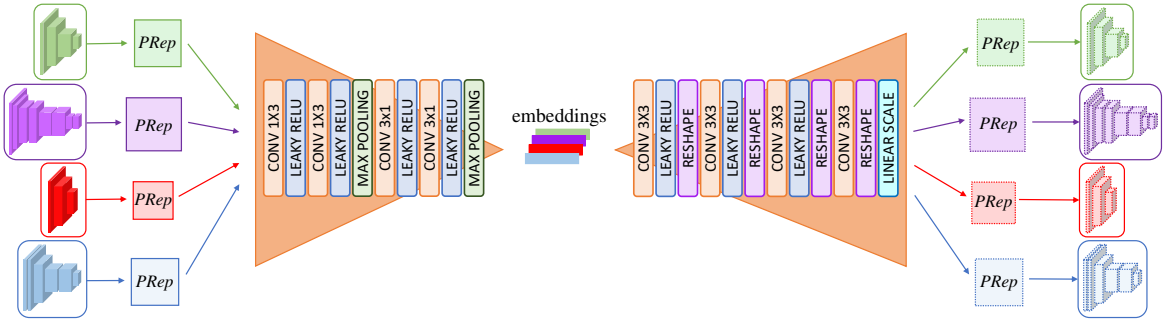


Fig. 1: Overview of the NetSpace framework.

and how well a feature extractor perform on a chosen task. Differently from all these works, we focus on learning a fixed-size embedding for diverse network architectures from which it is possible to draw ready-to-use weights for a specific task, even for networks unseen during training.

Network Parameters Prediction. Many works deploy an auxiliary network to obtain the weights of a target network. Hypernetworks [9] trained a small network (the hypernetwork) to predict weights for a large target network on a given task. The same technique has been extended and applied in many ways: transforming noise into the weights of a target network (Bayesian setting) [16], [20], adapting the weights of a target network to the current situation [2], [14], generating weights corresponding to hyperparameters [19], focusing on the acceleration of the architecture search problem [3]. Moreover, networks that generate their own weights have been proposed and analyzed [13], [6]. While these works share the use of a weights generation module with our work, our novel proposal consists in showing how to learn a fixed-size structured embedding for different architectures and navigate through this space to obtain new weights for these kinds of architectures as well as for architectures not provided as training examples.

Weight-sharing NAS. In Weight-sharing NAS, optimal architecture search occurs over the space defined by the subnets of a large network, the supernet. Commonly, subnets share weights with the supernet and they are available as ready-to-use networks after training. OFA [4] starts by training the entire supernet and progresses considering subnets of reduced size. After training, desired subnets are selected with an evolutionary algorithm. In NAT [21], many conflicting objectives are considered, training only the weights of promising subnets for every objective. While these works deal with obtaining ready-to-use networks that obey to desired characteristics, we focus on the embeddability of deep models in a latent space organized according to features of interest and on the possibility of explore such latent space by interpolation or optimization.

III. Method

Framework. In the following, we will use *architecture* to denote the structure of a deep learning model (i.e. number and kind of layers, etc.) and *instance* for an architecture featuring specific parameter values. Of course, given one architecture there can be many instances with different parameter values.

Our framework, dubbed NetSpace and shown in Fig. 1, is able to encode trained instances of different architectures into a fixed-size encoding and to decode this embedding into new instances that behave like the input ones. The parameters of each instance presented in input to our framework are stored by a simple algorithm into a PRep (parameters representation), a 2D matrix whose rows are filled one after the other with the sequence of the unrolled parameters. Further details on the PRep generation algorithm are provided in the Suppl. Material.

NetSpace encoder takes as input the PRep of an instance and produces a small fixed-size embedding, applying first horizontal and then vertical convolutions, alongside with max-pooling. It is worth pointing out that the encoder is designed to produce embeddings of the same size for any input PRep dimension.

The embedding from the encoder is then processed by NetSpace decoder, whose basic block first applies convolutions to increase the depth of the input and then reshapes the intermediate output to grow along spatial dimensions at the cost of depth. Once the required PRep resolution has been reached, an independent linear scaling is applied to every element of the predicted PRep, with weights and biases learnt during the training. We found that this is needed, in particular, for very deep models, probably because convolutions struggle to predict parameters that are close in the PRep but that belong to distant layers of the target architecture. The values of the predicted PRep are loaded into a ready-to-use instance.

The building blocks of the encoder and decoder are specified in more details in Fig. 1. In the remainder of the paper, we will use the term *target* instance to refer to the one in input to NetSpace and the term *predicted* instance to refer to that instantiated with values from the predicted PRep.

Single-Architecture Setting. In the Single-Architecture setting, NetSpace is used to learn an embedding space for the parameters of multiple instances of a single architecture. The first scenario that we consider deals with instances that exhibit different *performance* in solving the same task, such as image classification. Thus, during NetSpace training, the objective is to learn how to predict weights that match the performance of the target instances. Akin to common practice in Knowledge Distillation [11], this can be achieved by minimizing a loss term $\mathcal{L}_{pred}$ that represents the discrepancy between the outputs computed by the target instances and those computed by the

corresponding predicted instances. Formally, considering a target instance N_t and training samples x with labels y, we denote by N_p the instance predicted by NetSpace when the input instance is N_t, and by $t = N_t(x)$ and $p = N_p(x)$ the logits computed by the target and predicted instances, respectively. We then realize $\mathcal{L}_{pred}$ as in [11]:

$$\mathcal{L}_{pred} = KL(\text{softmax}(p/T),\ \text{softmax}(t/T)) \cdot T^2 \quad (1)$$

where KL denotes the Kullback–Leibler divergence averaged across the samples. As in [11], the softmax functions used in $\mathcal{L}_{pred}$ have inputs divided by a temperature term T.

A second scenario deals with networks sharing the same architecture that are trained to fit different signals. In particular, recent works [24], [22], [26], [23] have shown that it is possible to build neural representations of signals by training MLPs to regress such signals. In this scenario, each instance of the same MLP architecture is trained to represent a different signal. Given one of such instances, the objective of NetSpace is to predict weights capable of regressing the same signal. To achieve this goal, NetSpace can be trained with a loss term that directly compares the outputs of the predicted instance to those computed by the target one, thereby, also in this case, distilling the knowledge of the target instance into the predicted one. In this scenario, then, $\mathcal{L}_{pred}$ becomes simply:

$$\mathcal{L}_{pred} = MSE(y_p,\ y_t) \quad (2)$$

i.e. the Mean Squared Error (MSE) between the outputs from the predicted instance (y_p) and those from the target instance (y_t) when queried by the same inputs. In particular, in the experiments we consider MLPs trained to regress the Signed Distance Function (SDF) of a 3D shape (e.g., [24]).

We found that, in both scenarios, using a distillation loss is more effective than using a weights reconstruction loss, as the latter would aim just at mimicking on average the weights of the target instances, which we found not implying similar predictions. Furthermore, a distillation loss allows for using each target instance to create many training examples for NetSpace by simply varying the input data.

Multi-Architecture Setting. In the Multi-Architecture setting, we investigate on how to embed in a common space instances having different architectures. Thus, we consider instances trained to solve an image classification task with the best performances allowed by their architecture. NetSpace is trained to process such instances and to predict weights that reproduce their good performances. In order to ease NetSpace task, we take advantage of the complete Knowledge Distillation described in [11]. Denoting by N^* a teacher network with good performances in the task at hand and by t^* its logits for a batch of images x, we define the loss with respect to it as:

$$\mathcal{L}^*_{pred} = KL(\text{softmax}(p/T),\ \text{softmax}(t^*/T)) \cdot T^2 \quad (3)$$

Then, as in in [11], we introduce an additional term $\mathcal{L}_{task}$ which, in combination with $\mathcal{L}^*_{pred}$, defines the complete $\mathcal{L}_{kd}$:

$$\mathcal{L}_{task} = CE(\text{softmax}(p),\ y) \quad (4)$$

$$\mathcal{L}_{kd} = \alpha \cdot \mathcal{L}^*_{pred} + (1 - \alpha) \cdot \mathcal{L}_{task} \quad (5)$$

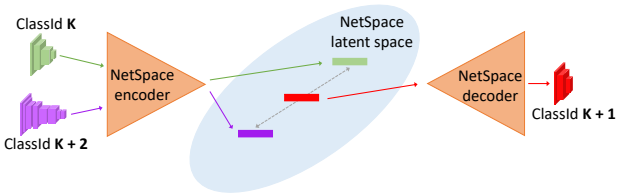


Fig. 2: An example of interpolation in the latent space learnt by NetSpace in the Multi-Architecture scenario.

where CE denotes the Cross Entropy loss averaged across the samples of the batch and α is a hyperparameter used to balance the two terms in $\mathcal{L}_{kd}$.

As far as the possibility of handling different architectures is concerned, we identify each architecture uniquely with a categorical ClassId. In this configuration, NetSpace is trained to predict an instance with the same architecture as the target. Even if this information is available at training time from the target instance itself, we would also like to explore by means of interpolation or optimization the latent space learnt by NetSpace after having trained it, without feeding input instances to the framework. Thus, we wish to be able to extract the architecture information directly from the embedding. To achieve this objective, we modify the architecture presented so far by adding a softmax classifier on top of the embedding in order to predict the ClassId of the target instance (details on this variant of the framework are reported in the Suppl. Material). Consequently, we complement the learning objective introduced in Eq. 5 with an additional $\mathcal{L}_{class}$ term. Given a target instance N_t and the embedding e produced by NetSpace encoder for it, we denote by c_t the ClassId associated to the architecture of N_t and by c_p the logits predicted by the architecture classifier from e. $\mathcal{L}_{class}$ is then defined as the Cross Entropy loss between the predicted and target ClassId:

$$\mathcal{L}_{class} = CE(\text{softmax}(c_p),\ c_t). \quad (6)$$

The initial experimental results highlighted that NetSpace was clustering the latent space according to the architecture ClassId only. We judge such organization of the embedding space as not satisfactory, as it would allow, perhaps, to sample new instances within a cluster by proximity or interpolation, but there would be no simple technique to navigate from one cluster to the others. Rather, we aim at endowing the embedding space with a structure enabling exploration along meaningful directions, i.e. directions somehow correlated to a specific characteristic, such as number of parameters or performance. Thus, a more amenable organization would consist in clusters showing up *aligned*, rather than scattered throughout the space, and possibly also *sorted* w.r.t. a given characteristic of interest.

Should such organization of the embedding space be possible, given two *boundary* embeddings (i.e. representing two instances with the smallest and the largest value of the characteristic of interest), it could be possible to move across the aligned clusters by simply interpolating the boundaries and obtain along the way representations of ready-to-use instances with increasing values

of the characteristic of interest. To further investigate along this path, we shall consider first that it is possible to assign ClassIds to a pool of architectures so as to sort them accordingly to a characteristic of interest. For instance, in our experiments, ClassIds K and $K+1$ will denote two architectures such that the latter has more parameters than the former.

Therefore, we introduce a new loss, denoted as $\mathcal{L}^\gamma$ (Interpolation Loss), whose objective is to impose the desired ordered alignment of clusters in the latent space. Given training instances belonging to boundary architectures (i.e. those with the smallest and largest ClassId), we first use NetSpace encoder to obtain their embeddings. Then, we interpolate such embeddings according to a given factor γ, and constrain the interpolated embedding to belong to the architecture whose ClassId is interpolated between the boundaries according to the same factor γ. Figure 2 presents an example of the interpolation procedure described in this paragraph.

Formally, given boundary embeddings e^A and e^B of target instances N_t^A and N_t^B with ClassIds c^A and c^B, we define the interpolated embedding $e^\gamma = (1-\gamma)\cdot e^A + \gamma \cdot e^B$. Then, considering the logits c_p^γ predicted by the ClassId classifier for e^γ and the interpolated ClassId $c_t^\gamma = (1-\gamma)\cdot c^A + \gamma \cdot c^B$, we define $\mathcal{L}_{class}^\gamma$ to impose the consistency of the interpolation factor for ClassId as:

$$\mathcal{L}_{class}^\gamma = CE(\text{softmax}(c_p^\gamma),\ c_t^\gamma). \tag{7}$$

Moreover, considering the instance N_p^γ predicted by NetSpace from e^γ, we denote by p^γ the logits predicted by such instance for a batch of images and define $\mathcal{L}_{kd}^\gamma$ as:

$$\mathcal{L}_{pred}^\gamma = KL(\text{softmax}(p^\gamma/T),\ \text{softmax}(t^*/T)) \cdot T^2 \tag{8}$$

$$\mathcal{L}_{task}^\gamma = CE(\text{softmax}(p^\gamma),\ y) \tag{9}$$

$$\mathcal{L}_{kd}^\gamma = \alpha \cdot \mathcal{L}_{pred}^\gamma + (1-\alpha)\cdot \mathcal{L}_{task}^\gamma \tag{10}$$

with the objective of distilling the teacher network N^* also in the interpolated instances. Finally, we define the total interpolation $\mathcal{L}^\gamma$ as:

$$\mathcal{L}^\gamma = \mathcal{L}_{class}^\gamma + \mathcal{L}_{kd}^\gamma \tag{11}$$

In our framework, we use $\mathcal{L}^\gamma$ with different interpolation factors γ, whose values are computed according to the number of considered architectures. More precisely, considering A architectures, γ can be computed as:

$$\gamma = \frac{i}{A-1} \quad i \in \{1, 2, ..., A-2\}. \tag{12}$$

Given a batch of instances, we compute $\mathcal{L}_{kd}$ and $\mathcal{L}_{class}$ on each of them. Then, we apply $\mathcal{L}^\gamma$, with γ values obtained from Eq. 12, on all the pairs composed of instances with the minimum and maximum ClassId. The final loss, thus, is the sum of $\mathcal{L}_{kd}$, $\mathcal{L}_{class}$ for each instance of the batch and $\mathcal{L}^\gamma$ for each pair of boundary instances.

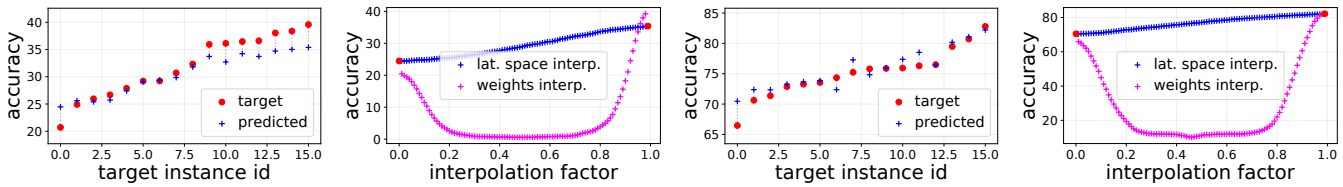
IV. Experiments

We test our framework with networks trained on image classification and 3D SDF regression.

Datasets and Architectures. For what concerns image classification, we report results on Tiny-ImageNet (TIN) [17] and CIFAR-10 [15] datasets. The target architectures for our experiments are LeNetLike, a slightly modified version of the lightweight CNN introduced in [18], VanillaCNN, a sequence of standard convolutions followed by a fully connected layer, and two variants of ResNet [10], namely ResNet8, and ResNet32. As far as 3D SDF regression is concerned, we consider MLPs trained to overfit a selection of $\sim$ 1000 *chairs* from the Shapenet dataset [5]. Each MLP has a single hidden layer with 256 nodes and uses periodic activation functions as proposed in [26]. Additional details are available in the Suppl. Material.

Single-Architecture Image Classification. As a first experiment, we test NetSpace in the Single-Architecture setting with the image classification task on CIFAR-10 and TIN. We create a dataset of 132 randomly initialized ResNet8 instances, training them for a different numbers of epochs, to collect instances with different performances. Then we randomly select 100 instances for training, 16 for validation, and 16 for testing. Fig. 3a and 3c compare the accuracy achieved on TIN and CIFAR-10 test sets by target and predicted instances. The target instances belong to the test sets and were never seen by NetSpace at training time. We can see that, beside few outliers, our framework is effective in predicting new instances that emulate the behavior of the target ones, both on CIFAR10 and on the larger and more varied TIN. It is remarkable that NetSpace is able to reconstruct an instance which follows the input one in terms of performance in spite of the huge compression it introduces. Indeed, the embedding size is a fraction of the number of parameters of the instances it can reconstruct, e.g. it is 4096 for TIN, only $\sim 3.18\%$ of the parameters of ResNet8. The key information about the behaviour of a neural net seems to live in a low dimensional space. Indeed, as shown by the visualization of PReps provided in the Suppl. Material, the predicted instance is very different from the target one: NetSpace captures the essential information to reproduce the behaviour of the target network, it does not merely learn to reproduce it.

After training, NetSpace has learnt to map target instances to fixed-sized embeddings. Thus, we can use NetSpace frozen encoder to obtain the embeddings of two anchor instances and linearly interpolate between them in order to study the representations laying in the space between the two anchors. To this aim, we decode every interpolated embedding to generate a new instance with NetSpace frozen decoder and compute the accuracy of this ready-to-use instances on the images in the test sets. As a baseline, we consider the possibility of interpolating directly the weights of the anchor instances. Results are reported in Fig. 3b and 3d for TIN and CIFAR-10, respectively. Interestingly, along the multi-dimensional line connecting the anchor embeddings we find representations corresponding to instances whose performances grow almost linearly with the interpolation factor, while interpolating directly the weights of the anchors yields random performances almost everywhere. This result suggests that the embedding space learnt by our framework can be organized to have meaningful dimensions, that are not exhibited in the instances weights space.



(a) TIN test set (b) TIN interpolation (c) CIFAR10 test set (d) CIFAR10 interpolation

Fig. 3: Single-Architecture results for ResNet8. (a) and (c): Accuracy achieved on the test set by target and predicted instances. Target instances are sorted w.r.t. their performances on the test set. (b) and (d): Accuracy achieved on the test set by instances predicted from interpolated embeddings.

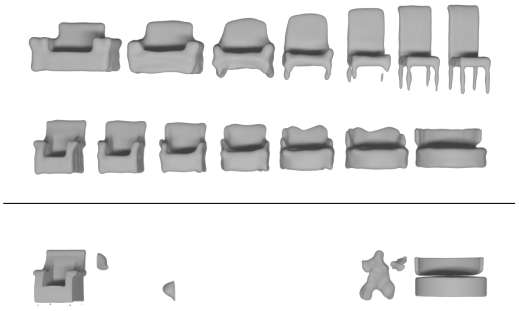


Fig. 4: Interpolation of 3D shapes. Top two rows: results obtained by interpolating NetSpace embedding space. Bottom row: the same linear interpolation applied to MLPs weights.

In fact, the loss function used in the Single-Architecture training concerns performance and our framework learns *naturally* a latent space that, at least locally, can be explored along a direction strictly correlated with performance.

Single-Architecture SDF regression. As a second Single-Architecture experiment, we train NetSpace to learn a latent space of MLPs that represent implicitly the SDF of chairs from the ShapeNet dataset. We train our framework on a dataset of ~ 1000 MLPs: each of them has been trained to overfit a different 3D shape, starting from a different random initialization. The goal of this experiment is to assess if NetSpace is capable of learning a meaningful embedding of 3D shapes, which can then be explored by linear interpolation. Thus, after training NetSpace, we obtain two anchor embeddings by processing two input MLPs with NetSpace frozen encoder. Then, we obtain new embeddings by interpolating the anchors and we predict new MLPs with NetSpace frozen decoder. The results of this experiment are reported in Fig. 4. The top two rows show interpolation results obtained from NetSpace latent space, while the bottom row presents results obtained by interpolating directly the weights of the anchor MLPs. We can notice that direct interpolation in the MLPs weights space yields catastrophic failures, while NetSpace embedding space enables smooth interpolations between the boundary shapes. This shows its ability to distill the core content of a trained model into a small-size embedding abstracting from the specific values of weights, and also its flexibility: when the loss concerns fitting of shapes, the latent space of models *naturally* organizes to have dimensions correlated with shape.

Multi-Architecture. In the Multi-Architecture setting we train NetSpace to embed four architectures: LeNetLike, VanillaCNN, ResNet8 and ResNet32. To build the dataset, we train many randomly initialized instances for each architecture, collecting multiple instances with good performances (100 for training, 16 for validation and 16 for testing). We collect in total 400 instances for training, 64 for validation and 64 for testing. We adopt a ResNet56 with high performance as the teacher network in Eq. 3 and 11 and set α to 0.9 in Eq. 5. We observe that supervision is not the same for different architectures: instances with lower performances receive a stronger signal from the $\mathcal{L}_{kd}$ and $\mathcal{L}^{\gamma}$ provides additional supervision for non-boundary architectures. We alleviate this issue modifing the training set so as to include a different number of instances for each architecture : we include 60 LeNetLike, 50 VanillaCNN, 60 ResNet8 and 100 ResNet32 instances. Fig. 5 left shows the results of this experiment: NetSpace successfully embeds instances of multiple architectures in highly compressed representations with all the information needed to a) predict correctly the architecture of the target instance and b) reconstruct an instance of such architecture whose behavior mimics that of the target one.

Multi-Architecture Embedding Interpolation. As we defined the ClassIds of architectures according to their increasing number of parameters, we expect their latent representations to be sorted w.r.t. this characteristic thanks to our interpolation loss $\mathcal{L}^{\gamma}$. In this experiment, we explore the latent space by observing the classification accuracy achieved by instances obtained when interpolating one embedding of LeNetLike and one of ResNet32, while moving with smaller steps than those defined in Eq. 12. Notably, as shown in Fig. 5 center, NetSpace learns an embedding space where architectures vary according to their number of parameters along an hyper-line. Moreover, it organized the space to place best performing embeddings for every class around the positions on which $\mathcal{L}^{\gamma}$ was computed.

Sampling of Unseen Architectures. We perform a new Multi-Architecture experiment by using a training set composed only of LeNetLike and ResNet32 instances. We collect 40 instances for LeNetLike and 80 for ResNet32, with the same balancing strategy discussed above. By not showing to NetSpace encoder any instance of VanillaCNN and ResNet8,

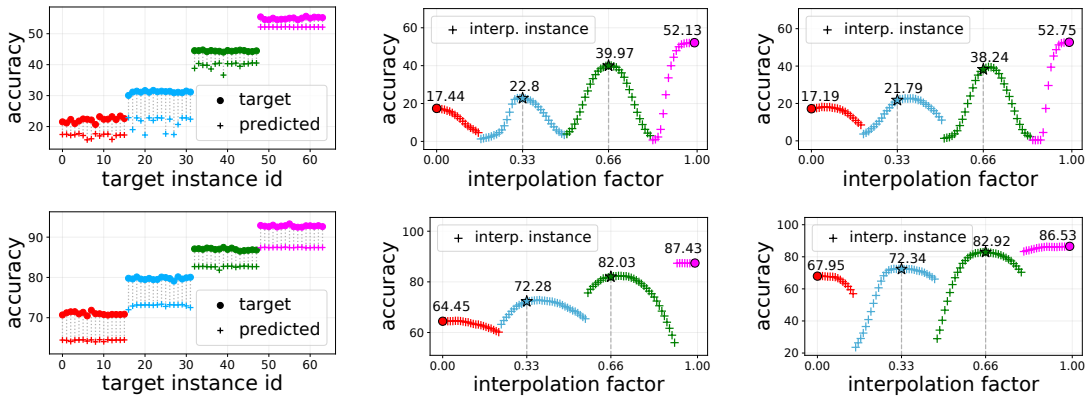


Fig. 5: Results of the Multi-Architecture setting on TIN (top) and CIFAR10 (bottom). Left: target instances from test set, instances are sorted w.r.t. their ClassId. Center: interpolation with all architectures available at training time. Right: interpolation with only variants of LeNetLike and ResNet32 seen at training time. In all figures, color represent architecture: red-LeNetLike, blue-VanillaCNN, green-ResNet8, fuchsia-ResNet32. Circles correspond to interpolation boundaries. Stars denote instances obtained with the discrete interpolation factors used in $\mathcal{L}^{\gamma}$.

TABLE I: Accuracy on TIN test set achieved with LSO.

	Single-Architecture	**Multi-Architecture**			
	ResNet8	LeNetLike	VanillaCNN	ResNet8	ResNet32
initial	25.73%	17.44%	22.89%	38.81%	52.17%
optimized	35.72%	18.55%	24.17%	42.07%	53.13%

we deny it the possibility to learn directly a portion of the embedding space dedicated to them. However, $\mathcal{L}^{\gamma}$ shapes it indirectly: for instance, given two embeddings e_{lenet} and e_{r32} of, respectively, a LeNetLike and ResNet32, it forces the embedding $e^{\gamma} = 0.33 \cdot e_{lenet} + 0.66 \cdot e_{r32}$ to represent an instance of ResNet8 (unseen during training). After training, we perform an interpolation experiment and report results in Fig. 5 right: we find that the latent space learnt by NetSpace trained on a reduced set of architectures exhibits the same properties as the space learnt by training with all of them, allowing to draw by interpolation instances with good performance of the *unseen architectures* VanillaCNN and ResNet8.

Latent Space Optimization. Here we investigate the navigation of NetSpace embedding by latent space optimization (LSO). In order to do so, we take NetSpace encoder and decoder obtained by a Single-Architecture Image classification experiment, freezing their parameters. Then, we obtain an initial latent code by embedding one ResNet8 instance from the test set with the frozen encoder. We then use the frozen decoder to perform an iterative optimization of the initial embedding. In each step of the optimization, the embedding is processed by the frozen decoder, which predicts a PRep that is loaded into a ResNet8 instance. The resulting network is then used to produce predictions on a batch of training images from TIN. We apply $\mathcal{L}_{kd}$ on these predictions using a ResNet56 as teacher network, to guide NetSpace in the search of a high performing instance in the learnt latent space. As the decoder is frozen, we compute the gradient of the loss w.r.t. the embedding, so as to explore the latent space by gradient descent.

Furthermore, we perform a similar experiment starting from NetSpace encoder and decoder taken from a Multi-Architecture training experiment. Also in this case, we use the frozen encoder to obtain initial embeddings from four instances belonging to different architectures. Then, we process each embeddings with the frozen decoder, which predicts a PRep and a ClassId. We use the predicted ClassId to select the architecture where the predicted PRep is loaded, building an instance which we use to produce predictions on a batch of training images from TIN. In addition to $\mathcal{L}_{kd}$, in this case we apply also $\mathcal{L}_{class}$ on the predicted ClassId, to guide the embedding towards the area of the latent space that corresponds to the desired class. In Tab. I we report the performances obtained by the optimizations (second row), together with the performances of the instances used to obtain the initial embeddings (first row). Remarkably, the results show that it is actually possible to improve the performances of the input instances by exploring the NetSpace embedding space via latent space optimization.

V. Conclusions and Future Work

NetSpace introduces a framework to learn the latent space of deep models. We have shown that the embedding space learnt by NetSpace can be organized according to meaningful traits and ready-to-use instances with predictable properties can be obtained by means of linear interpolation or latent space optimization. Furthermore, our experiments provide evidence that fixed-size embeddings can represent effectively instances of several different architectures. The main limitation of our work concerns lack of investigation on the potential applications of our findings dealing with embeddability of deep models. Yet, we deem it worth communicating these findings to the community as we believe they are non-obvious and valuable on their own, and may foster further research as well as identification of the most fruitful outcomes toward applications.

REFERENCES

[1] ACHILLE, A., LAM, M., TEWARI, R., RAVICHANDRAN, A., MAJI, S., FOWLKES, C. C., SOATTO, S., AND PERONA, P. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 6430–6439.

[2] BERTINETTO, L., HENRIQUES, J. F., VALMADRE, J., TORR, P., AND VEDALDI, A. Learning feed-forward one-shot learners. In *Advances in neural information processing systems* (2016), pp. 523–531.

[3] BROCK, A., LIM, T., RITCHIE, J. M., AND WESTON, N. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344* (2017).

[4] CAI, H., GAN, C., WANG, T., ZHANG, Z., AND HAN, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019).

[5] CHANG, A. X., FUNKHOUSER, T., GUIBAS, L., HANRAHAN, P., HUANG, Q., LI, Z., SAVARESE, S., SAVVA, M., SONG, S., SU, H., ET AL. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012* (2015).

[6] CHANG, O., AND LIPSON, H. Neural network quine. In *Artificial Life Conference Proceedings* (2018), MIT Press, pp. 234–241.

[7] CHOUDHARY, T., MISHRA, V., GOSWAMI, A., AND SARANGAPANI, J. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review* (2020), 1–43.

[8] FRANKLE, J., AND CARBIN, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

[9] HA, D., DAI, A., AND LE, Q. V. Hypernetworks. *arXiv preprint arXiv:1609.09106* (2016).

[10] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[11] HINTON, G., VINYALS, O., AND DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[12] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507.

[13] HYUN LEE, S., HA KIM, D., AND CHEOL SONG, B. Self-supervised knowledge distillation using singular value decomposition. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 335–350.

[14] JIA, X., DE BRABANDERE, B., TUYTELAARS, T., AND GOOL, L. V. Dynamic filter networks. In *Advances in Neural Information Processing Systems* (2016), pp. 667–675.

[15] KRIZHEVSKY, A., HINTON, G., ET AL. Learning multiple layers of features from tiny images. *Tech Report* (2009).

[16] KRUEGER, D., HUANG, C.-W., ISLAM, R., TURNER, R., LACOSTE, A., AND COURVILLE, A. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759* (2017).

[17] LE, Y., AND YANG, X. Tiny imagenet visual recognition challenge. *CS 231N 7* (2015), 7.

[18] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[19] LORRAINE, J., AND DUVENAUD, D. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419* (2018).

[20] LOUIZOS, C., AND WELLING, M. Multiplicative normalizing flows for variational bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 2218–2227.

[21] LU, Z., SREEKUMAR, G., GOODMAN, E., BANZHAF, W., DEB, K., AND BODDETI, V. N. Neural architecture transfer. *arXiv preprint arXiv:2005.05859* (2020).

[22] MESCHEDER, L., OECHSLE, M., NIEMEYER, M., NOWOZIN, S., AND GEIGER, A. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 4460–4470.

[23] MILDENHALL, B., SRINIVASAN, P. P., TANCIK, M., BARRON, J. T., RAMAMOORTHI, R., AND NG, R. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision* (2020), Springer, pp. 405–421.

[24] PARK, J. J., FLORENCE, P., STRAUB, J., NEWCOMBE, R., AND LOVEGROVE, S. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 165–174.

[25] RUSU, A. A., RAO, D., SYGNOWSKI, J., VINYALS, O., PASCANU, R., OSINDERO, S., AND HADSELL, R. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960* (2018).

[26] SITZMANN, V., MARTEL, J., BERGMAN, A., LINDELL, D., AND WETZSTEIN, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems 33* (2020).