

# Supplementary Material for 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.

## I. CLASSID CLASSIFIER

In Sec. 3 (*Multi-Architecture Setting*) of the main paper we introduce the need to extend NetSpace architecture to the Multi-Architecture setting. In this setting, in fact, we ask our framework to extract the ClassId of the predicted instances from the embeddings. To achieve this goal, we extend the architecture of our framework with a softmax classifier, which takes in input the embeddings generated by NetSpace encoder and is trained to predict the correct ClassId with  $\mathcal{L}_{class}$ . The classifier is a lightweight neural network, composed of one convolutional layer and one fully connected layer, interleaved by the LeakyReLU activation function. The outputs of the classifier are transformed into probabilities by the softmax function. Fig. 1 presents an overview of the version of NetSpace used in the Multi-Architecture case, including the ClassId classifier.

## II. PARAMETERS REPRESENTATION (PREP)

A PRep is a 2D tensor where we store all the parameters of an instance of a neural network by means of a simple algorithm exemplified in Fig. 2.

We designed our framework fixing PReps to be rectangular matrices with high width/height ratio. Additionally, to favor easy implementation, the height and the width of a PRep are fixed to be multiple of 4 and 8, respectively. Considering an architecture with  $P$  parameters and having fixed the width of the PRep to a number  $W$  (divisible by 8), the minimum necessary height of the PRep can be computed as  $\lceil \frac{P}{W} \rceil$ , adding a padding of  $4 - (\lceil \frac{P}{W} \rceil \bmod 4)$  rows, if needed, to fulfill the divisibility by 4.

Given an instance of a neural network and a chosen PRep size, the algorithm to produce the instance PRep is straightforward: parameters from all the layers of the instance are copied in the matrix in sequence one row after the other and final zero-padding is added as needed to match the required size.

## III. NETWORK ARCHITECTURES

**Image Classification.** Fig. 3 and 4 present the architecture of the neural networks used in our experiments dealing with image classification, i.e. LeNetLike [6], VanillaCNN and ResNet8/32/56 [2]. Each convolutional layer is presented in the form CONV (in  $I$ , out  $O$ , k  $K$ , s  $S$ , p  $P$ ), where  $I$ ,  $O$ ,  $K$ ,  $S$  and  $P$  represent input channels, number of filters, kernel size, stride

and padding, respectively. Fully Connected layers, instead, are reported in the form FC (in  $I$ , out  $O$ ), where  $I$  and  $O$  stand for input and output units, respectively. Average pooling is shown as Average Pooling (k  $K$ , s  $S$ ), where  $K$  is kernel size and  $S$  is stride. Finally, CONV  $1 \times 1$  represents a convolutional layer with kernel size 1 used to adapt feature maps in residual connections.

**3D SDF Regression.** As far as 3D SDF regression is concerned, we use simple Multilayer Perceptrons (MLPs) composed of a single hidden layer with 256 nodes. Following [8], we use a periodic activation function between the input layer and the hidden layer and between the hidden layer and the output layer. No activation function is applied instead on the final outputs.

## IV. IMAGE CLASSIFICATION: EXPERIMENT DETAILS

**Images Datasets.** To test our general framework in image classification, we make use of the CIFAR-10 [4] and Tiny-ImageNet [5] datasets. CIFAR-10 is composed of 60K  $32 \times 32$  colour images, 50K for training and 10K for test, categorized in 10 classes. For our experiments we obtain a validation set by splitting the training set in 40K images for training and 10K for validation, while we keep unchanged the test set. Tiny-ImageNet consists of 100K colour images with resolution  $64 \times 64$ , categorized in 200 different classes. We split the training set in 80K images for training and 20K images for validation and use the 10K images of the provided validation set for testing. With both datasets, we follow a standard data augmentation regime [9] and use a batch size of 128.

**Nets Datasets.** In the Single-Architecture setting, we train the instances in input to NetSpace with the Adam optimizer [3] and constant learning rate set to 0.0001 for a number of epochs that vary from 1 to 600 epochs, obtaining instances with weights and performances varying smoothly across the training iterations. Then we use 100 instances for training, 16 for validation and 16 for test. In the Multi-Architecture setting, we aim at embedding only instances with high performances. Accordingly, we found it more effective to train models with the SGD optimizer for 300 epochs and setting momentum and weight decay to 0.9 and  $5e-4$ , respectively. We set the initial learning rate to 0.05 and decay it by 0.1 at epochs 150, 180 and 210. As discussed in the main paper in Sec. 4, for the Multi-Architecture setting we define a training dataset with 60 LeNetLike instances, 50 VanillaCNN instances, 60

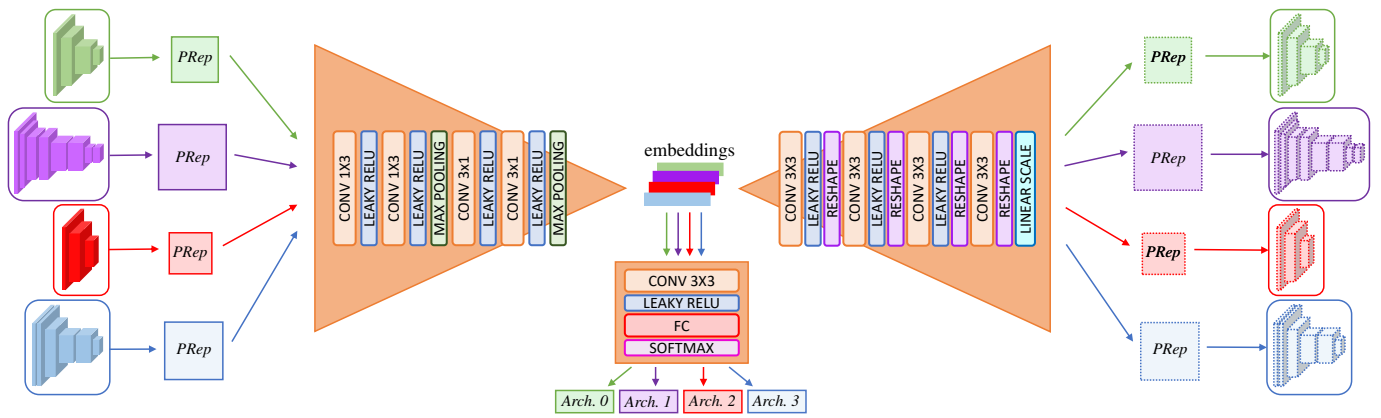


Fig. 1: Overview of NetSpace architecture used in the Multi-Architecture experiments.

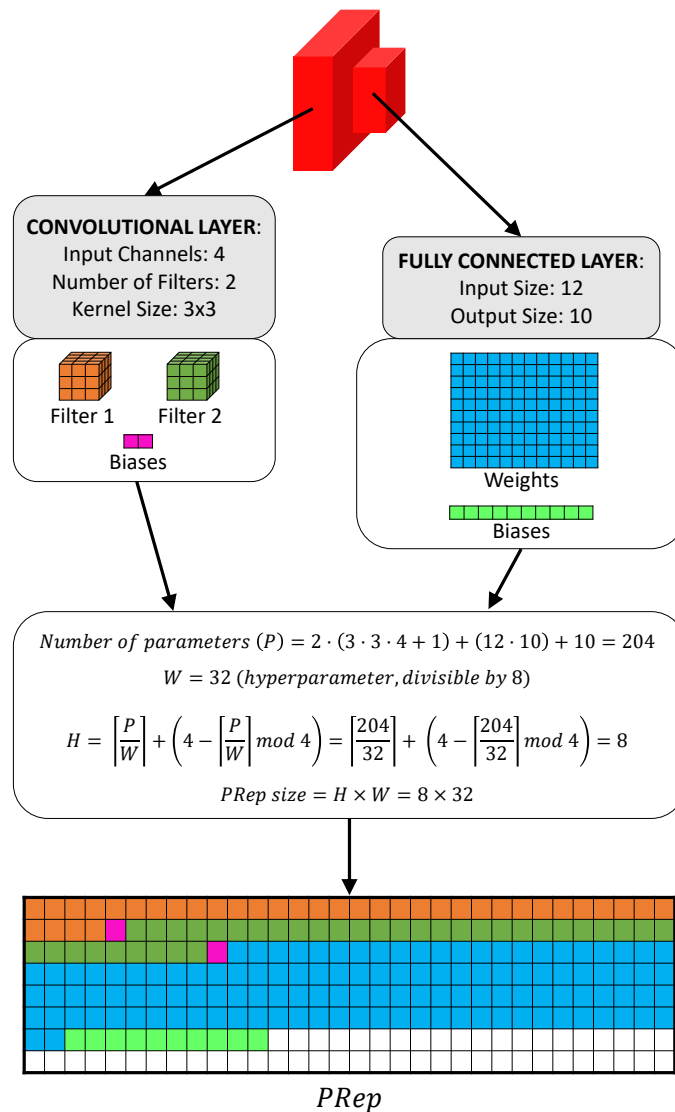


Fig. 2: Algorithm to compute the PRep of a given instance. We consider here a toy architecture made out of one convolutional layer and one fully connected layer and we fix the PRep width to 32. White cells represent padding with constant value 0.

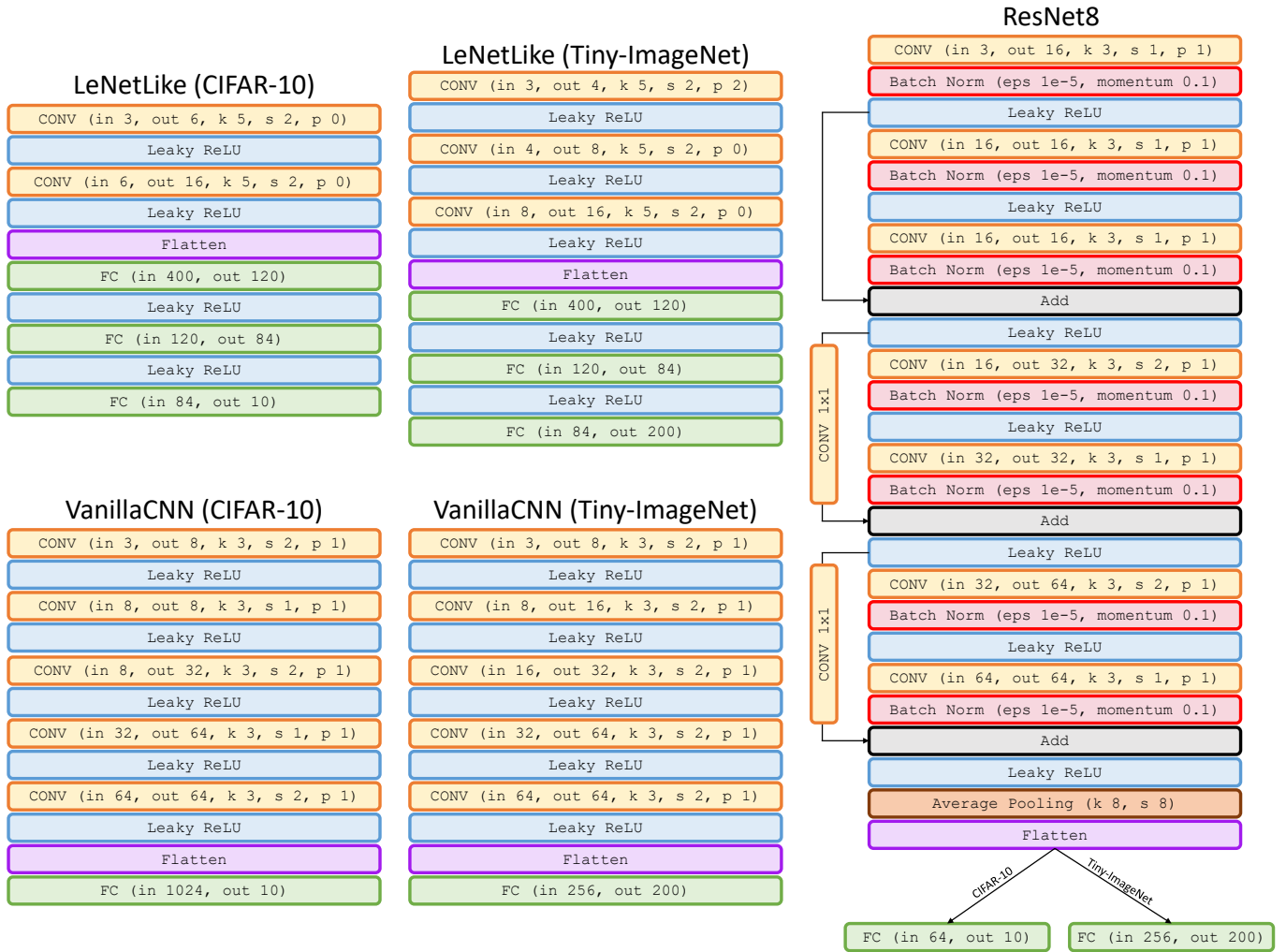


Fig. 3: Architectures used in our experiments dealing with image classification. Top left: LeNetLike, bottom left: VanillaCNN, right: ResNet8.

ResNet8 instances and 100 ResNet32 instances, while for the experiment "Sampling of Unseen Architectures" we use a training set composed of 40 LeNetLike instances and 80 ResNet32 instances. In both Multi-Architecture settings, the validation and test sets are composed of 16 instances of the architectures available during training. In Tab. I we report the accuracy achieved by the trained models on the CIFAR-10 and the Tiny-ImageNet test sets, alongside with the number of parameters of each architecture.

**Framework Training.** To train NetSpace in the Single-Architecture and in the Multi-Architecture settings, we use the Adam optimizer and a learning rate value of 0.0001, we train for around 1K epochs, and then we use the model with the highest performance on the validation set. The temperature term used in  $\mathcal{L}_{kd}$  and in  $\mathcal{L}^\gamma$  is set to 4, while the size of the meta-batch of instances is set to 8 and to 2 in the Single-Architecture and in the Multi-Architecture settings, respectively. The Latent Space Optimization experiments are conducted by training NetSpace with the Adam optimizer and constant learning rate

0.0001, stopping the training when the accuracy achieved by the optimized network doesn't show any additional improvement.

**Computational Time and Resources.** For our experiments we used several Nvidia RTX 3090 GPUs. Training the networks to populate the datasets requires approximately 600 GPU hours, while training NetSpace with Tiny-ImageNet requires around 84 GPU hours in the Multi-Architecture setting and around 48 GPU hours in the Single-Architecture setting. Training over CIFAR-10, instead, requires less time, with less than 48 and 36 GPU hours respectively in the Multi-Architecture and in the Single-Architecture settings. Finally, the Latent Space Optimization experiments require few GPU hours, but it's possible to observe good improvements over the initial performance already after few minutes of training.

## V. 3D SDF REGRESSION: EXPERIMENT DETAILS

**3D Shape Dataset.** To test NetSpace with networks dealing with 3D SDF regression, we use the ShapeNet dataset [1]. In particular, we use ShapeNetCore, a subset of the full ShapeNet

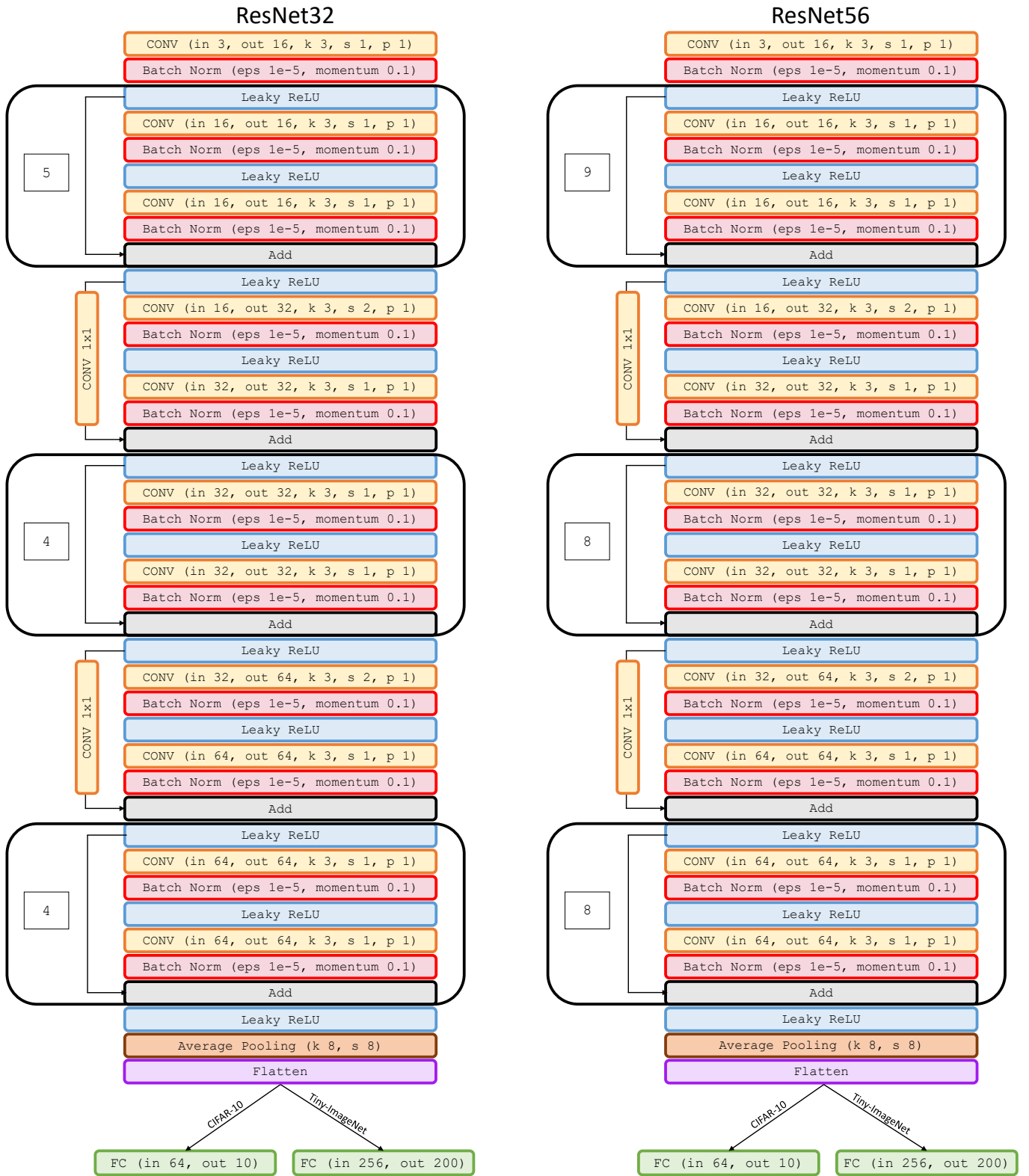


Fig. 4: Architectures used in our experiments dealing with image classification. Left: ResNet32, right: ResNet56.

TABLE I: Models used in our experiments. We show classification accuracies on the CIFAR-10 (top) and Tiny-ImageNet (bottom) test sets alongside the number of parameters for each architecture. For the Single-Architecture setting (Adam optimizer), we report the accuracy achieved by the best performing network between the trained ones, while for the Multi-Architecture setting (SGD optimizer), we report the average accuracy of the networks that compose the training set.

CIFAR-10				
Net	ClassId	Acc. (Adam)	Acc. (SGD)	# Params
LeNetLike	0	-	70.87 %	62,006
VanillaCNN	1	-	79.77 %	68,818
ResNet8	2	82.79 %	87.13 %	78,042
ResNet32	3	-	92.70 %	466,906
ResNet56	-	-	92.85 %	855,770

Tiny-ImageNet				
Net	ClassId	Acc. (Adam)	Acc. (SGD)	# Params
LeNetLike	0	-	22.13 %	79,612
VanillaCNN	1	-	31.32 %	112,856
ResNet8	2	40.51 %	44.38 %	128,792
ResNet32	3	-	54.81 %	517,656
ResNet56	-	-	56.93 %	906,520

dataset which covers 55 common object categories with about 51,300 unique 3D models. We conduct our experiments on a small subset of ShapeNetCore, consisting of 1000 3D objects from the *chair* category.

**MLP Dataset.** The MLP dataset used in our experiments contains 1000 MLP. Each of them is obtained by training a randomly initialized MLP to fit the SDF of a single *chair*, whose ground-truth is computed with the code provided with [7]. The fitting procedure consists in 10,000 gradient descent steps: at each step the MLP is queried on 20,000 random 3D coordinates and is asked to regress the SDF value for each of them. Then, the parameters of the MLP are optimized by Adam [3], using as loss function the mean squared error between the predictions and the ground-truth. The learning rate is initially set to 0.0001 and multiplied by 0.9 every 1000 steps.

**Framework Training.** NetSpace is trained on the 1000 MLPs with the protocol described in Sec. 3 of the main paper, using Adam [3] with learning rate set to 0.0001. During training, we evaluate the performance of NetSpace by comparing directly the predictions of the output MLPs with those of the input MLPs: by querying input and output MLPs with the same random coordinates, we can compute the percentage of predictions of the output MLPs that are sufficiently close to the values predicted by the input MLPs. We monitor this metric and stop the training when it reaches the value 0.8. The results reported in the main paper are obtained by training for 4000 epochs.

**Computational Time and Resources.** We adopted Nvidia RTX 3090 GPUs also in the experiments involving SDF regression. The creation of the MLP dataset requires approximately 20 GPU hours, while training NetSpace requires around 48 GPU hours.

## VI. FUSING BATCH NORM AND CONVOLUTIONS

To be able to process architectures including batch norm layers, e.g. ResNet in our experiments, without changing the PRep structure, we decided to fuse batch norm layers with convolutional layers, which is always possible for a trained model since batch norm becomes a frozen affine transformation at test time. Therefore, when processing ResNet instances in our experiments, we first prepared a dataset of instances trained with batch norm to achieve the best performances and then, by the process described below, we transformed such instances into equivalent ones featuring only plain convolutional layers without batch norm.

If we consider a feature map  $F$  with shape  $C \times H \times W$ , at inference time its batch normalized version  $\widehat{F}$  is obtained by computing at each spatial location  $i, j$ :

$$\begin{pmatrix} \widehat{F}_{1,i,j} \\ \widehat{F}_{2,i,j} \\ \vdots \\ \widehat{F}_{C,i,j} \end{pmatrix} = W_{BN} \cdot \begin{pmatrix} F_{1,i,j} \\ F_{2,i,j} \\ \vdots \\ F_{C,i,j} \end{pmatrix} + b_{BN} \quad (1)$$

with

$$W_{BN} = \begin{pmatrix} \frac{\gamma_1}{\sqrt{\widehat{\sigma}_1^2 + \epsilon}} & & & \\ & \frac{\gamma_2}{\sqrt{\widehat{\sigma}_2^2 + \epsilon}} & & \\ & & \ddots & \\ & & & \frac{\gamma_C}{\sqrt{\widehat{\sigma}_C^2 + \epsilon}} \end{pmatrix} \quad (2)$$

$$b_{BN} = \begin{pmatrix} \beta_1 - \gamma_1 \frac{\widehat{\mu}_1}{\sqrt{\widehat{\sigma}_1^2 + \epsilon}} \\ \beta_2 - \gamma_2 \frac{\widehat{\mu}_2}{\sqrt{\widehat{\sigma}_2^2 + \epsilon}} \\ \vdots \\ \beta_C - \gamma_C \frac{\widehat{\mu}_C}{\sqrt{\widehat{\sigma}_C^2 + \epsilon}} \end{pmatrix} \quad (3)$$

where  $\widehat{\mu}_c$ ,  $\widehat{\sigma}_c^2$ ,  $\beta_c$  and  $\gamma_c$  ( $c = 1, 2, \dots, C$ ) are respectively the mean, variance and batch norm parameters computed during training for the channel  $c$  of the feature map. From this formulation, we can see that batch norm can be implemented as a  $1 \times 1$  convolution and therefore, when batch norm comes after another convolution as in ResNet, we can fuse these two convolutions into a single one.

We can express a convolutional layer with kernel size  $k$  processing the  $C_{prev} \times k \times k$  volume at the spatial location  $(i, j)$  of a feature map  $F_{prev}$  with  $C_{prev}$  channels to produce the feature map  $\widetilde{F}$  with  $C$  output channels as an affine transformation

$$\widetilde{f}_{i,j} = W_{conv} f_{i,j} + b_{conv}, \quad (4)$$

where  $W_{conv} \in \mathbb{R}^{C \times (C_{prev} k^2)}$ ,  $b_{conv} \in \mathbb{R}^C$  and  $f_{i,j}$  represents the area of size  $C_{prev} \times k \times k$  around cell  $(i, j)$  reshaped as a  $(C_{prev} k^2)$ -dimensional vector.

If the batch norm defined by  $W_{BN} \in \mathbb{R}^{C \times C}$  and  $b_{BN} \in \mathbb{R}^C$  presented in Eq. 2 and 3 comes after such convolutional layer,

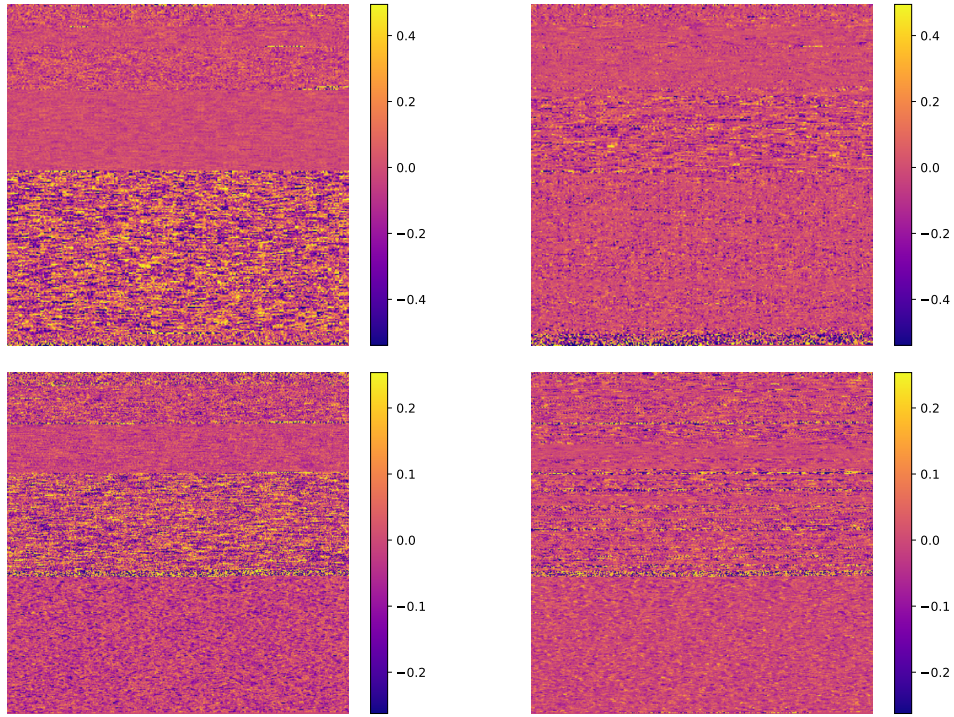


Fig. 5: Visualization of the PRep of a ResNet8 instance in the Single-Architecture setting (Image classification): top, CIFAR-10, bottom, Tiny-ImageNet. The target PRep on the left is given in input to NetSpace, that produces the predicted PRep on the right.

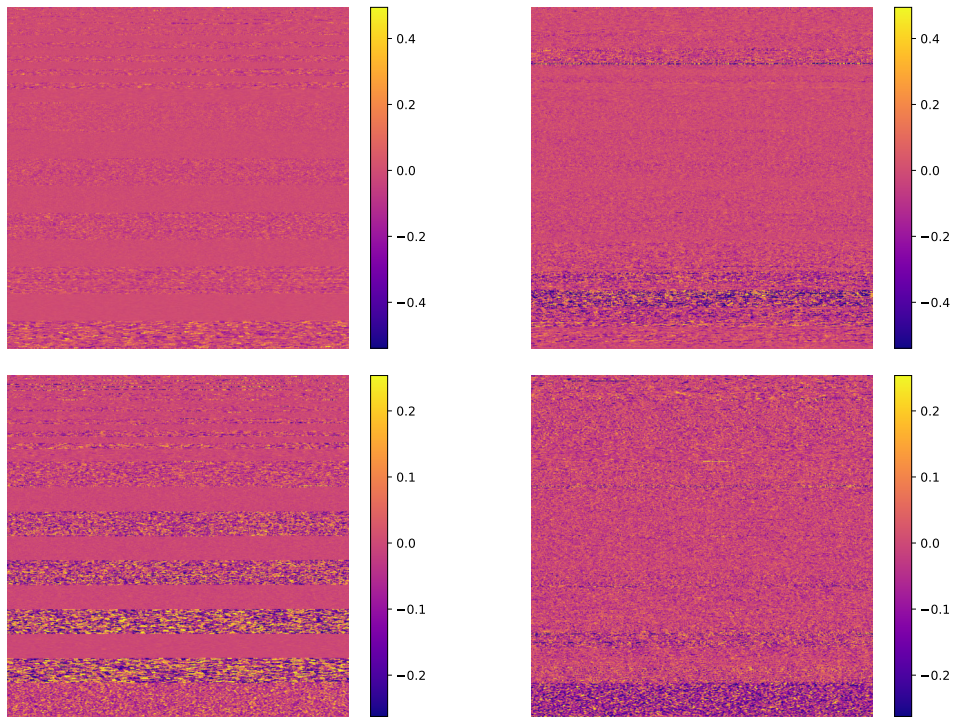


Fig. 6: Visualization of the PRep of a ResNet32 instance in the Multi-Architecture setting: top, CIFAR-10, bottom, Tiny-ImageNet. The target PRep on the left is given in input to NetSpace, that produces the predicted PRep on the right.

the normalized values  $\hat{f}_{i,j} \in \mathbb{R}^C$  at cell  $(i, j)$  of its output feature map can be computed as

$$\begin{aligned}\hat{f}_{i,j} &= W_{BN} \tilde{f}_{i,j} + b_{BN} \\ &= W_{BN} (W_{conv} f_{i,j} + b_{conv}) + b_{BN}.\end{aligned}\quad (5)$$

Hence, it is possible to replace every convolutional layer (with weights  $W_{conv}$  and  $b_{conv}$ ) followed by a batch norm layer (whose weights can be shaped in  $W_{BN}$  and  $b_{BN}$  as described above) with a single convolutional layer whose parameters  $W$  and  $b$  can be computed as:

$$W = W_{BN} W_{conv} \quad (6)$$

$$b = W_{BN} b_{conv} + b_{BN} \quad (7)$$

## VII. VISUALIZING NETWORKS AS IMAGES

As in our framework network instances are represented as PRep tensors, they can be visualized as images. Thus, in this section we highlight some properties of NetSpace by visualizing input and output instances as images. In particular, following the ResNet8 Single-Architecture (Image classification) and ResNet32 Multi-Architecture trainings, we take some instances from the test set and obtain their PReps along with those of the corresponding instances predicted by NetSpace. As explained in Sec. II of this document, such representations are 2D matrices, that we reshaped to obtain images of form factors amenable to clear visualization. Then, as shown in Fig. 5 and 6, we represent parameters according to a standard colormap.

From these results we can highlight some interesting properties about our framework. Firstly, we observe that PReps predicted by NetSpace are significantly different w.r.t. the input ones: as we did not use any reconstruction loss in the learning objective, NetSpace learnt to predict instances which behave like the input ones but that are different in terms of parameter values. Secondly, we can see that PReps corresponding to different instances can indeed be visualized as different images, which suggests that, perhaps, in future work these images may be used as proxies for neural networks instances, so that training or fine-tuning or distillation may be realized by learning to generate images.

## REFERENCES

- [1] 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).
- [2] 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.
- [3] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. *ICLR* (2015).
- [4] KRIZHEVSKY, A., HINTON, G., ET AL. Learning multiple layers of features from tiny images. *Tech Report* (2009).
- [5] LE, Y., AND YANG, X. Tiny imagenet visual recognition challenge. *CS 231N 7* (2015), 7.
- [6] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.
- [7] PARK, J. J., FLORENCE, P., STRAUB, J., NEWCOMBE, R., AND LOVE-GROVE, 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.

- [8] 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).
- [9] TIAN, Y., KRISHNAN, D., AND ISOLA, P. Contrastive representation distillation. *arXiv preprint arXiv:1910.10699* (2019).