Deep Learning on Object-Centric 3D Neural Fields

Pierluigi Zama Ramirez, Luca De Luigi, Daniele Sirocchi, Adriano Cardace, Riccardo Spezialetti, Francesco Ballerini, Samuele Salti, and Luigi Di Stefano

***Abstract*—In recent years, Neural Fields (*NF*s) have emerged as an effective tool for encoding diverse continuous signals such as images, videos, audio, and 3D shapes. When applied to 3D data, *NF*s offer a solution to the fragmentation and limitations associated with prevalent discrete representations. However, given that *NF*s are essentially neural networks, it remains unclear whether and how they can be seamlessly integrated into deep learning pipelines for solving downstream tasks. This paper addresses this research problem and introduces `nf2vec`, a framework capable of generating a compact latent representation for an input *NF* in a single inference pass. We demonstrate that `nf2vec` effectively embeds 3D objects represented by the input *NF*s and showcase how the resulting embeddings can be employed in deep learning pipelines to successfully address various tasks, all while processing exclusively *NF*s. We test this framework on several *NF*s used to represent 3D surfaces, such as unsigned/signed distance and occupancy fields. Moreover, we demonstrate the effectiveness of our approach with more complex *NF*s that encompass both geometry and appearance of 3D objects such as neural radiance fields.**

***Index Terms*—Neural fields, INR, implicit neural representations, representation learning, deep learning on neural fields, signed distance function, SDF, unsigned distance function, UDF, occupancy field, OF, neural radiance field, NeRF, 3D classification, 3D generation, 3D segmentation, 3D completion, 3D reconstruction, NeRF generation, NeRF classification.**

I. Introduction

COMPUTER vision has always been concerned with understanding the 3D world around us. One of the main challenges when dealing with 3D data is the representation strategy, which was addressed over the years by introducing various discrete representations, including voxel grids, point clouds, and meshes. Each representation has its advantages and disadvantages, especially when it comes to processing it through deep learning, leading to the development of a plethora of ad-hoc algorithms [1], [2], [3] for each coexisting representation. Hence, no standard way to store and process 3D data has yet emerged.

Recently, a new representation has been proposed, called Neural Fields [4] (*NF*s). They are continuous functions defined at all spatial coordinates, parameterized by a neural network such as a Multi-Layer Perceptron (MLP). In the context of 3D world representation, various types of *NF*s have been explored. Some of the most common *NF*s utilize the Signed/Unsigned Distance Field (*SDF*/*UDF*) [5], [6], [7], [8] and the Occupancy Field (*OF*) [9], [10] to represent the 3D surfaces or volumes of the objects in the scene. Alternatively, strategies seeking to capture both geometries and appearances often leverage the Radiance Field (*RF*), as shown in the pioneering approach NeRF [11].

Representing a 3D scene by encoding it with a continuous function parameterized as an MLP separates the memory cost of the representation from the spatial resolution. In other words, starting from the same fixed number of parameters, it is possible to reconstruct a surface with arbitrarily fine resolution or to render an image with arbitrarily high quality. Furthermore, the identical neural network architecture can be applied to learn various field functions, offering the possibility of a unified framework for representing 3D objects.

Owing to their efficacy and potential benefits, 3D *NF*s are garnering growing interest from the scientific community, as evidenced by the frequent publication of novel and impressive results [8], [12], [13], [14]. This leads us to speculate that, in the near future, *NF*s could establish themselves as a standard way to store and communicate 3D data. It is conceivable that repositories hosting digital twins of 3D objects, exclusively realized as MLPs, might become widely accessible.

The above scenario prompts an intriguing research question: can 3D *NF*s be directly processed using deep learning pipelines for solving downstream tasks, as it is commonly done with discrete representations such as point clouds or images? For instance, is it feasible to classify an object by directly processing the corresponding NeRF without rendering any image from it?

Since *NF*s are neural networks, there is no straightforward way to process them. A recent work in the field, Functa [15], fits the whole dataset with a shared network conditioned on a different embedding for each data. In this formulation, a solution could be to use such embeddings as the input for downstream tasks. Nevertheless, representing an entire dataset through a shared network poses a formidable learning challenge, as the network encounters difficulties in accurately fitting all the samples (see Section VII).

On the contrary, recent studies, including SIREN [16] and others [17], [18], [19], [20], [21], have demonstrated that it is possible to achieve high-quality reconstructions by tailoring an individual network to each input sample. This holds true even when dealing with complex 3D shapes or images. Furthermore, constructing an individual *NF* for each object is more adaptable to real-world deployment, as it does not require the availability

Manuscript received 15 December 2023; revised 23 May 2024; accepted 14 July 2024. Date of publication 17 July 2024; date of current version 5 November 2024. Recommended for acceptance by E. Ilg. *(Pierluigi Zama Ramirez, Luca De Luigi, and Daniele Sirocchi are co-first authors.) (Corresponding author: Pierluigi Zama Ramirez.)*

The authors are with the University of Bologna, 40126 Bologna, Italy (e-mail: pierluigi.zama@unibo.it; luca.deluigi4@unibo.it; adriano.cardace2@unibo.it; riccardo.spezialetti@unibo.it).

Digital Object Identifier 10.1109/TPAMI.2024.3430101

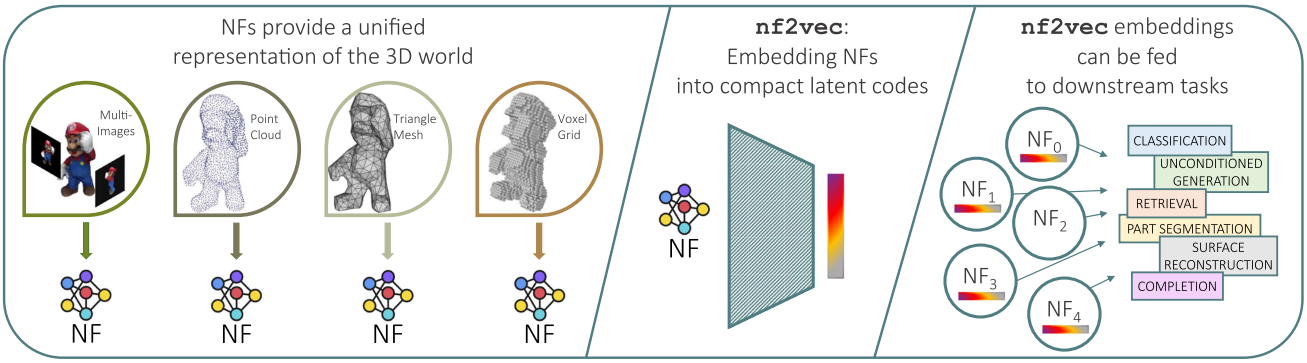


Fig. 1. *Overview of our framework. Left*: *NF*s hold the potential to provide a unified representation of the 3D world. *Center*: Our framework, dubbed `nf2vec`, produces a compact representation for an input *NF* by looking only at its weights. *Right*: `nf2vec` embeddings can be used with standard deep-learning machinery to solve various downstream tasks.

of the entire dataset to fit each individual data. The increasing popularity of such methodologies suggests that adopting the practice of fitting an individual network is likely to become commonplace in learning *NF*s.

Therefore, in the former version of this paper [22], we explored conducting downstream tasks using deep learning pipelines on 3D data represented as *individual NF*s. Recently, several methods addressing this topic have been published, such as NFN [23], NFT [23], and DWSNet [24], and all of them process individual *NF*s, supporting this paradigm.

Using *NF*s as input or output data is intrinsically non-trivial, as the MLP of a single *NF* can encompass hundreds of thousands of parameters. However, deep models inherently present a significantly redundant parameterization of the underlying function, as shown in [25], [26]. As a result, we explore whether and how an answer to the research question mentioned earlier might be identified within a representation learning framework. We present an approach that encodes individual *NF*s into compact and meaningful embeddings, making them suitable for diverse downstream tasks. We name this framework `nf2vec`, shown in Fig. 1.

Our framework has at its core an encoder designed to produce a task-agnostic embedding representing the input *NF* by processing only the *NF* weights. These embeddings can seamlessly be used in downstream deep learning pipelines as we validate for various tasks, like classification, retrieval, part segmentation, unconditioned generation, completion, and surface reconstruction. Remarkably, the last two tasks become achievable by learning a straightforward mapping between the embeddings generated using our framework, as embeddings derived from *NF*s exist in low-dimensional vector spaces, regardless of the underlying implicit function. For instance, we can learn the mapping between *NF*s of incomplete objects into *NF*s of normal ones. Then, we can complete shapes by exploiting this mapping, e.g., we can map the *NF* of an airplane with a missing wing into the *NF* of a complete airplane. Furthermore, we show that `nf2vec` learns a smooth latent space, which enables the interpolation of *NF*s representing previously unseen 3D objects.

This paper builds on our previous work [22], with revisions to the overall framework and thorough experiments on novel scenarios. Specifically, the key differences with [22] are:

- In [22], we focused solely on neural fields representing the surfaces of 3D objects. In this extended version, we also tackle the processing of neural fields capturing objects' geometry and appearance. Specifically, we extend our framework to perform deep learning tasks on NeRFs by directly processing their MLPs weights.
- The processing of MLPs parametrizing *NF*s has been investigated in works published contemporaneously or subsequently to [22]. We extend our literature review by including these recent papers, and we evaluate them to foster progress in this emerging topic and facilitate future comparisons.

Overall, the summary of our work contributions is:

- We propose and investigate the novel research problem of applying deep learning directly on individual *NF*s representing 3D objects.
- We introduce `nf2vec`, a framework designed to derive a meaningful and compact representation of an input *NF* solely by processing its weights, without the need to sample the underlying function.
- We demonstrate that a range of tasks, typically tackled with intricate frameworks tailored to specific representations, can be effectively executed using simple deep learning tools on *NF*s embedded by `nf2vec`, regardless of the signal underlying the *NF*s.
- We demonstrate the versatility of `nf2vec` by successfully applying it to neural fields that capture either the geometry alone or the combined information of both geometry and appearance of 3D objects.
- We analyze recent methods for processing *NF*s in terms of classification accuracy and representation quality. We build the first evaluation benchmark for *NF* classification.

Additional details, code, and datasets are available at https://cvlab-unibo.github.io/nf2vec.

II. Related Work

Neural Fields: Recent approaches have shown the ability of MLPs to parameterize fields representing any physical quantity of interest [4]. The works focusing on representing 3D shapes

with MLPs rely on fitting functions such as the unsigned distance [6], the signed distance [5], [7], [10], [27], [28], [29], or the occupancy [9], [30]. Among these approaches, sinusoidal representation networks (SIRENs) [16] use periodical activation functions to capture the high-frequency details of the input data. In addition to representing shapes, some of these models have been extended to encode object appearance [11], [27], [31], [32], [33], or to include temporal information [34]. Among these recent approaches, modeling the radiance field of a scene [11] has proven to be the critical factor in obtaining excellent scene representations. In our work, we employ *NF*s encoding *SDF*, *UDF*, *OF*, and *RF* as input data for deep learning pipelines.

Deep Learning on Neural Networks: Several works have explored using neural networks to process other neural networks. [35] utilizes a network's weights as input and forecasts its classification accuracy. Another approach [36] involves learning a network representation through a self-supervised learning strategy applied to the N-dimensional weight array. These representations are then employed to predict various characteristics of the input classifier. In contrast, [37], [38], [39] depict neural networks as computational graphs, subsequently processed by a Graph Neural Network (GNN). This GNN is tasked with predicting optimal parameters, adversarial examples, or branching strategies for verifying neural networks.

These works view neural networks as algorithms, primarily focusing on forecasting properties like accuracy. In contrast, some recent studies handle networks that implicitly represent 3D data, thus tackling various tasks directly from their weights, essentially treating neural networks as input/output data. Functa [15] tackles this scenario by acquiring priors across the entire dataset using a shared network and subsequently encoding each sample into a concise embedding employed for downstream discriminative and generative tasks. We note that in this formulation, each neural field is parametrized by both the shared network and the embedding. It is worth pointing out that, though not originally proposed as a framework to process neural fields, DeepSDF [5] learns dataset priors by optimizing a reconstruction objective through a shared auto-decoder network conditioned on a shape-specific embedding. Thus, the embeddings learned by DeepSDF may be used for neural processing tasks, as done in Functa.

However, shared network frameworks face several challenges, as they struggle to reconstruct the underlying signal with high fidelity and require an entire dataset to learn the neural field of an object. In response, recent approaches have shifted their focus on processing *NF*s learned on individual data, e.g., a specific object or scene. The first framework adopting this view was proposed in our previous paper version [22]. This approach leverages representation learning to condense individual *NF*s of 3D shapes into embeddings, serving as input for subsequent tasks. [40] has recently built upon this idea to learn a bidirectional mapping between image/text and NeRF latent spaces. Recognizing that MLPs exhibit weight space symmetries [41], where hidden neurons can be permuted across layers without altering the network's function, recent approaches such as DWSNet [24], NFN [42], and NFT [23] leverage these symmetries as an inductive bias to create innovative architectures tailored for MLPs. DWSNet and NFN design neural layers equivariant to the permutations inherent in MLPs. In contrast, NFT achieves permutation equivariance by removing the positional encoding from a Transformer architecture. A recent work by [43] overcomes the need to deal with MLP symmetries by proposing a Transformer-based architecture that processes *NF*s with triplanar grid features by focusing on those discrete features only.

Recently, HyperDiffusion [44] has proposed a generative diffusion approach to synthesize *NF* parameters. Like us, it employs MLPs optimized to represent individual data.

III. Learning to Represent *NF*s

This paper explores the possibility and the methodology of directly utilizing *NF*s for downstream tasks. Specifically, can we classify an object implicitly encoded in a *NF*, and if so, how? As outlined in Section I, we condense the redundant information encoded in the weights of *NF*s into latent codes by a representation learning framework. These codes can then be efficiently processed using standard deep-learning pipelines. Our framework, dubbed `nf2vec`, comprises an encoder and a decoder. In the following sections, we first provide some basic knowledge about what a 3D neural field is, then we deepen the reasons behind the architectural choices for both components and describe the representation learning protocol.

3D Neural Fields: A field is a physical quantity defined for all domain coordinates. We focus on fields describing the 3D world, thus operating on $\mathbb{R}^3$ coordinates $\mathbf{p} = (x, y, z)$. We consider the 3D fields commonly used in computer vision and graphics, such as the *SDF* [5] and *UDF* [6], which map coordinates to the signed and unsigned distance from the closest surface, the *OF* [9], which computes the occupancy probability of each position, and the *RF* [11], that outputs (r, g, b) colors and density σ for each 3D point. A field can be modeled by a function, f, parameterized by θ. Thus, for any point $\mathbf{p}$, the field is given by $f(\mathbf{p})$. If parameters θ are the weights of a neural network, f is said to be a Neural Field (*NF*) [4].

Encoder: The encoder takes as input the weights of a *NF* and produces a compact embedding that encodes all the relevant information of the input *NF*. Designing an encoder for *NF*s poses a challenge in handling weights efficiently to avoid excessive memory usage. While a straightforward solution might involve using an MLP encoder to map flattened weight vectors to desired dimensions, this approach becomes impractical for larger *NF*s. For instance, given a 4-layer 512-neurons *NF*, mapping its 800K parameters to a 1024-sized embedding space would require an encoder with roughly 800M parameters, making this approach prohibitive. Thus, we focus on developing an encoder architecture that scales gracefully with the size of the input *NF*.

Following conventional practice [16], [17], [18], [19], [20], we consider *NF*s composed of an MLP with several hidden layers, each with H nodes. The linear transformation between two consecutive hidden layers is parameterized by a matrix of weights $\mathbf{W}_i \in \mathbb{R}^{H \times H}$ and a vector of biases $\mathbf{b}_i \in \mathbb{R}^{H \times 1}$. Thus, stacking $\mathbf{W}_i$ and ${\mathbf{b}_i}^T$, the mapping between two consecutive layers can be represented by a single matrix $\mathbf{P}_i \in \mathbb{R}^{(H+1) \times H}$. Additionally, an MLP features also an input layer, parametrized

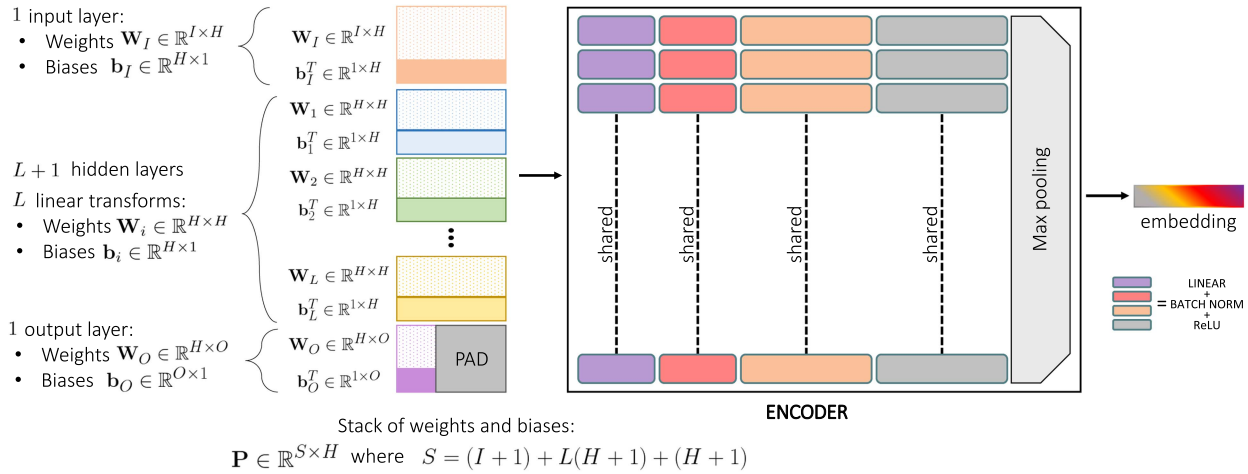


Fig. 2. *Encoder architecture. Left*: Given a *NF*, we stack its weights and biases to form a matrix $\mathbf{P}$. *Right*: The `nf2vec` encoder is a series of linear layers with batch-norms and ReLU activation functions. It processes each row of $\mathbf{P}$ independently and then aggregates all rows of one *NF* with a max-pooling to produce a compact embedding employed for downstream tasks.

TABLE I
NUM. OF PARAMETERS OF OUR VERSUS AN MLP ENCODER

NF			Our encoder	MLP encoder
Hidden dim.	#Layers	#Params	#Params	#Params
512	4	∼800K	∼3M	∼800M
512	8	∼2M	∼3M	∼2B
512	16	∼4M	∼3M	∼4B
1024	4	∼3M	∼3.5M	∼3B
1024	8	∼7M	∼3.5M	∼7.5B
1024	16	∼15M	∼3.5M	∼16B

by $\mathbf{W}_I \in \mathbb{R}^{I\times H}$ and a vector of biases $\mathbf{b}_I \in \mathbb{R}^{H\times 1}$, and an output layer, parametrized by $\mathbf{W}_O \in \mathbb{R}^{H\times O}$ and a vector of biases $\mathbf{b}_O \in \mathbb{R}^{O\times 1}$. The input and output layers can be represented by two matrices, $\mathbf{P}_I \in \mathbb{R}^{(I+1)\times H}$ and $\mathbf{P}_O \in \mathbb{R}^{(H+1)\times O}$. Considering that for a *NF* with $L+1$ hidden layers there are L linear transformations between them, we can store all the weights of the *NF* by stacking the input matrix $\mathbf{P}_I$ with all the L matrices $\mathbf{P}_i$ and with the output matrix $\mathbf{P}_O$. As $\mathbf{P}_O$ has a different number of columns (O), we pad it with zeros before stacking. The final stacked matrix $\mathbf{P}$ has dimension $S \times H$ – where $S = (I+1) + L(H+1) + (H+1)$ – and represents the input for our encoder, as shown in the left part of Fig. 2.

`nf2vec` encoder consists of a series of linear layers with batch normalization and ReLU non-linearity followed by final max pooling. At each stage, the input matrix $\mathbf{P}$ is transformed by one linear layer, processing each row independently. The final max pooling compresses all the rows into a single one, obtaining the desired embedding. An architecture overview is depicted in Fig. 2.

Our proposed architecture scales gracefully to bigger input *NF*s as supported by the analysis in Table I, that reports the parameters of our encoder *w.r.t.* those of a generic MLP encoder while varying the input *NF* dimension.

It is worth observing that the randomness involved in fitting an individual *NF* (weights initialization, data shuffling, etc.) causes the weights in the same position in the *NF* architecture not to share the same role across *NF*s. Thus, `nf2vec` encoder would have to deal with input vectors whose elements capture different information across the different data samples, making it impossible to train the framework. However, the use of a shared, pre-computed initialization has been advocated as a good practice when fitting *NF*s, e.g., to reduce training time by means of meta-learned initialization vectors, as done in MetaSDF [17] and in Functa [15], or to obtain desirable geometric properties [7]. We empirically found that following such a practice, i.e., initializing all *NF*s with the same random vector, favors the alignment of weights across *NF*s and enables the convergence of our framework. More analysis can be found in Section VIII.

Decoder: When learning to encode *NF*s, we are interested in storing the information about the represented object rather than the values of the input weights. Therefore, the adopted decoder predicts the original field values rather than reconstructing the input weights in an auto-encoder fashion. In particular, during training, we adopt an implicit decoder inspired by [5], which takes in input the embeddings produced by the encoder and a spatial coordinate $\mathbf{p}_i$ and decodes the original field values (see Fig. 3 center). We denote as $\hat{f}(\mathbf{p}_i)$ the predicted field value.

Training: We train our encoder and decoder following the input *NF*s training strategy. For instance, when dealing with *UDF*, *SDF*, and *OF* representing 3D surfaces, we supervise the framework directly using the ground truth field values computed from point clouds, voxel grids, or triangle meshes representing those surfaces. Differently, when processing NeRFs we employ volumetric rendering [11] on the radiance field values predicted by the decoder to obtain the RGB intensities of image pixels, and we supervise the framework directly with a regression loss between predicted and true RGB values.

To better understand the procedure, let us take the example where we aim to learn to represent *UDF*s. We create a set of 3D queries paired with the values of the *UDF* at those locations. The decoder takes in input the embedding produced by the encoder concatenated with the 3D coordinates of a query point and

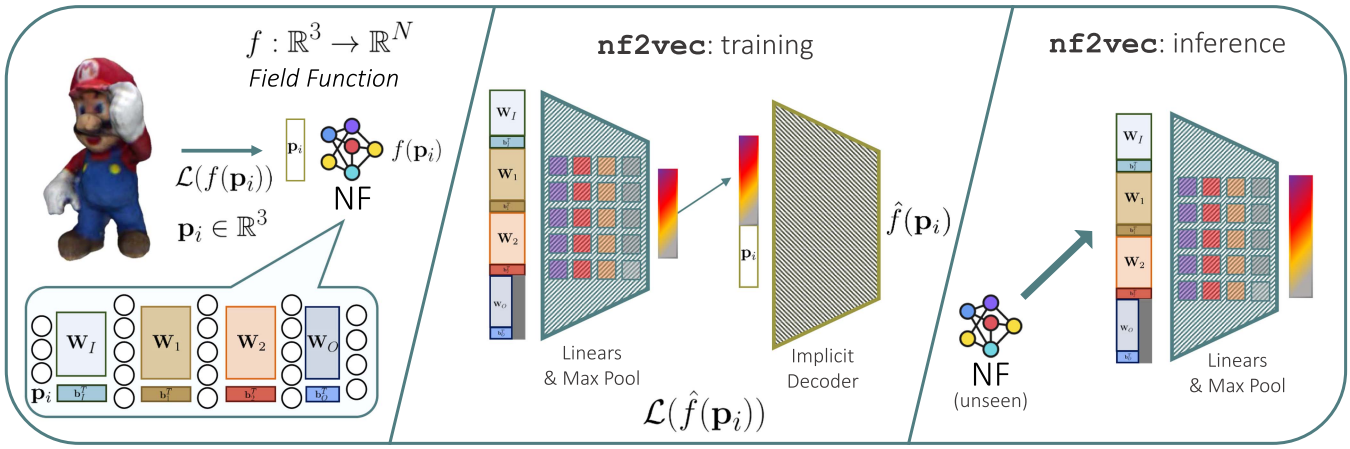


Fig. 3. *Training and inference of* nf2vec . *Left:* A Neural Field (*NF*) represents a 3D object. The *NF* is composed of an MLP that parametrizes a field function f. *Center:* nf2vec encoder is trained together with an implicit decoder. The implicit decoder processes the embedding produced by the encoder to estimate field values $\hat{f}$. We train the framework similarly to the input *NF*. *Right:* At inference time, the learned encoder can be used to obtain a compact embedding from unseen *NF*s.

estimates the *UDF* for this location. The whole encoder-decoder is supervised to minimize the discrepancy between the estimated and correct *UDF* values.

Inference: After the overall framework has been trained end to end, the frozen encoder can be used to compute embeddings of unseen *NF*s with a single forward pass (see Fig. 3 right) while the implicit decoder can be used, if needed, to reconstruct the discrete representation given an embedding. *We highlight that no discrete representations are required at inference time.*

The presented nf2vec framework shares the same high-level structure of the originally proposed approach inr2vec [22]. However, as shown in Section VI, in this extended work, we show that our framework can be applied to more complex neural fields such as NeRFs. Thus, we dub it nf2vec to emphasize its generality.

IV. Latent Space Properties

We train nf2vec on *NF*s learned from various 3D discrete representations of ShapeNet [45] surfaces. We use either *UDF* learned from point clouds, *SDF* learned from meshes, or *OF* learned from voxel grids. We also train nf2vec on NeRFs trained on multi-view renderings of ShapeNet [45] objects. The following sections explore nf2vec latent space properties. We investigate the ability to reconstruct the original discrete representation from the compact latent codes extracted with our encoder, the smoothness of the embedding space, and its structure.

Reconstruction: In Fig. 4, we compare 3D shapes reconstructed from *NF*s unseen during training with those reconstructed by the nf2vec decoder starting from the latent codes yielded by the encoder. We visualize point clouds with 8192 points, meshes reconstructed by marching cubes [46] from a grid with resolution 128^3 and voxels with resolution 64^3. Moreover, we conduct the same experiment using as input unseen NeRFs. In Fig. 5, we show the images rendered by nf2vec decoder and

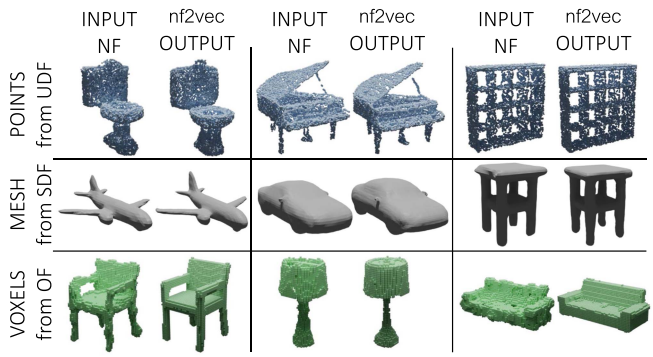


Fig. 4. nf2vec reconstructions.

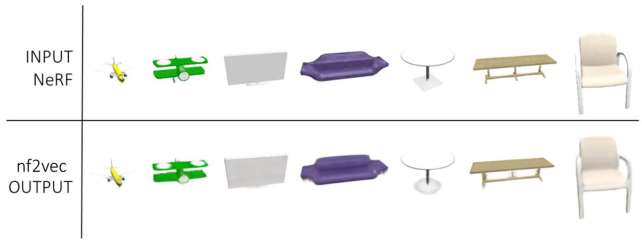


Fig. 5. nf2vec reconstructions.

those rendered from the input NeRF at 224×224 resolution. Though our embedding is dramatically more compact than the original *NF*, the reconstructed discrete data resembles those of the original input *NF*.

Interpolation: In Figs. 6 and 7, we linearly interpolate between two object embeddings produced by nf2vec. Results highlight that the learned latent spaces enable smooth interpolations between shapes represented as *NF*s. Notably, in Fig. 7, color and shapes change smoothly when interpolating two NeRF embeddings. Moreover, the *interpolated* object has an underlying 3D consistency, as visible when rendering it from different

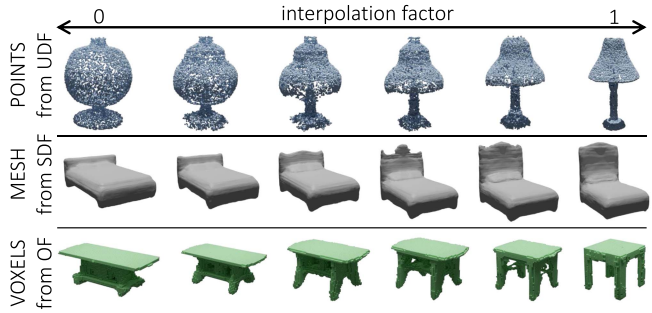


Fig. 6. `nf2vec` latent space interpolation.

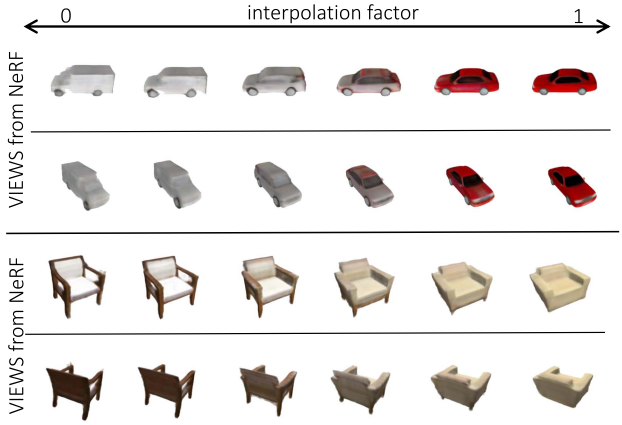


Fig. 7. `nf2vec` latent space interpolation.

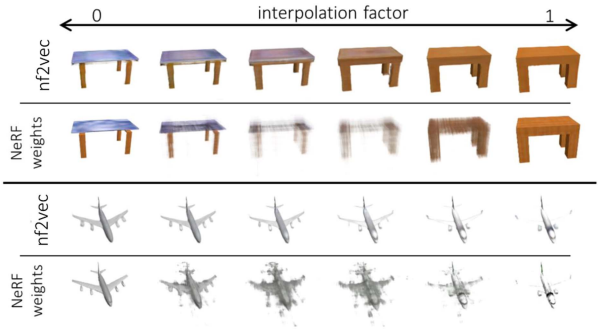


Fig. 8. `nf2vec` latent space interpolation versus NeRF weights interpolation.

camera viewpoints. Thus, these results demonstrate the capability to generate new plausible shapes and even realistic *RF*s by means of simple linear interpolation in `nf2vec` latent space.

Additionally, given two input NeRFs, we render images from networks obtained by interpolating their weights. In Fig. 8, we compare these results with those obtained from the interpolation of `nf2vec` embeddings. Notably, our interpolations exhibit superior quality. In particular, renderings obtained by averaging the weights of the two NeRFs (interpolation factor 0.5) appear blurred and lack 3D structure. In contrast, renderings produced by `nf2vec` preserve details and maintain 3D consistency. Our

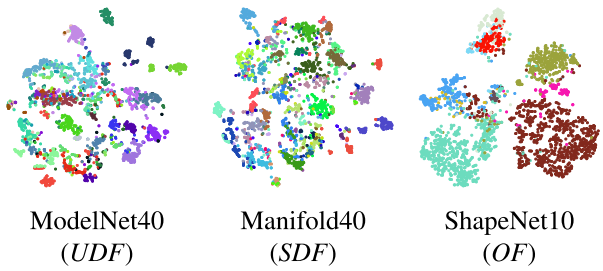


Fig. 9. *t-SNE visualizations of* `nf2vec` *latent spaces.* We plot the t-SNE components of the embeddings produced by `nf2vec` on the test sets of three datasets, ModelNet40 (left), Manifold40 (center) and Shapenet10 (right). Colors represent the different classes of the datasets.

results highlight the efficacy of our representation learning approach in transforming an initially disorganized input weight space into a well-organized latent space.

t-SNE Visualization of the Latent Space: In Fig. 9, we provide the t-SNE visualization of the embeddings produced by `nf2vec` when presented with unseen *NF* of three different datasets: ModelNet40 (*UDF* learned from point clouds), Manifold 40 (*SDF* learned from meshes), and ShapeNet10 (*OF* learned from voxel grids). During the training of our framework, the supervisory signal employed does not impose any particular constraints on the organization of the learned latent space. This lack of constraints was deliberate, as it was not necessary for our primary goal – performing downstream tasks with the generated embeddings. Nevertheless, it is intriguing to note from the t-SNE plots that our algorithm naturally arranges the embeddings in the latent space with a semantic structure, with items of the same category consistently mapped to close positions. This is evident in the colors representing different classes within the datasets under consideration.

V. Deep Learning on 3D Shapes

This section shows how several tasks dealing with 3D shapes can be tackled by working only with `nf2vec` embeddings as input and/or output.

General Settings: In all the experiments reported in this section, we convert 3D discrete representations into *NF*s featuring 4 hidden layers with 512 nodes each, using the SIREN activation function [16]. We discard the input and output layers of SIREN MLPs when processing them with `nf2vec`. This is based on the observation from the earlier version of this paper [22] that these layers do not provide information beneficial for downstream tasks when using SIREN MLPs as architecture for *NF*. We train `nf2vec` using an encoder composed of four linear layers with respectively 512, 512, 1024, and 1024 features, embeddings with 1024 values, and an implicit decoder with 5 hidden layers with 512 features. In all the experiments, the baselines are trained using standard data augmentation (random scaling and point-wise jittering), while we train both `nf2vec` and the downstream task-specific networks on datasets augmented offline with the same transformations.

TABLE II
POINT CLOUD RETRIEVAL QUANTITATIVE RESULTS

Method	Input	ModelNet40			ShapeNet10			ScanNet10		
		mAP@1	mAP@5	mAP@10	mAP@1	mAP@5	mAP@10	mAP@1	mAP@5	mAP@10
PointNet [48]	Point cloud	80.1	91.7	94.4	90.6	96.6	98.1	65.7	86.2	92.6
PointNet++ [1]	Point cloud	**85.1**	**93.9**	**96.0**	**92.2**	**97.5**	**98.6**	**71.6**	**89.3**	93.7
DGCNN [2]	Point cloud	83.2	92.7	95.1	91.0	96.7	98.2	66.1	88.0	93.1
`nf2vec`	*NF*	81.7	92.6	95.1	90.6	96.7	98.1	65.2	87.5	**94.0**

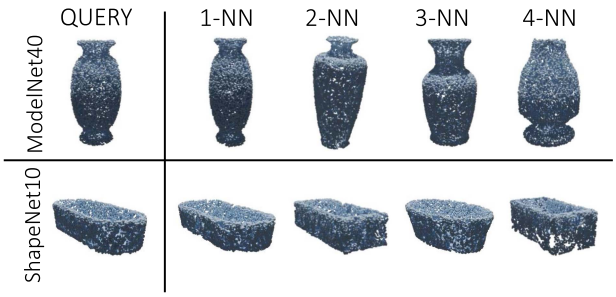


Fig. 10. *Point cloud retrieval qualitative results.* Given the `nf2vec` embedding of a query shape, we show the shapes reconstructed from the closest embeddings (L2 distance).

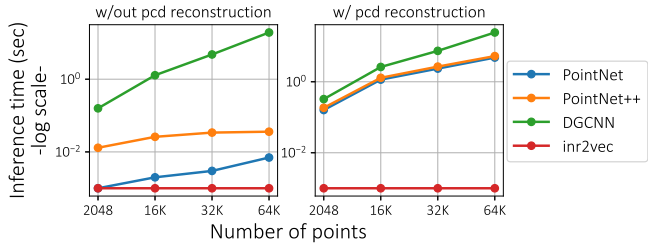


Fig. 11. *Time required to classify* NFs *encoding* **udf**. We plot the inference time of standard baselines and of our method, both considering the case in which discrete point clouds are available (left) and the one where point clouds must be reconstructed from the input *NF*s (right).

Point Cloud Retrieval: We examine the potential of using `nf2vec` embeddings for representation learning tasks, with 3D retrieval as our benchmark. We follow the procedure introduced in [45], using the euclidean distance to measure the similarity between embeddings of unseen point clouds from the test sets of ModelNet40 [47] and ShapeNet10 (a subset of 10 classes of the popular ShapeNet dataset [45]). For each embedded shape, we select its k-nearest-neighbours and compute a Precision Score comparing the classes of the query and the retrieved shapes, reporting the mean Average Precision for different k (mAP@k). Beside `nf2vec`, we consider three baselines to embed point clouds, which are obtained by training the PointNet [48], PointNet++ [1] and DGCNN [2] encoders in combination with a fully connected decoder similar to that proposed in [49] to reconstruct the input cloud. The quantitative findings in Table II reveal that `nf2vec` not only matches but sometimes exceeds the performance of other baselines, with an average gap of 1.8 mAP compared to PointNet++. Furthermore, as depicted in Fig. 10, it is evident that the retrieved shapes not only belong to the same class as the query but also exhibit similar coarse structures. These results highlight that the pretext task used to learn `nf2vec` embeddings allows encoding relevant shape information.

Shape Classification: We then address the problem of classifying point clouds, meshes, and voxel grids. We use three datasets for point clouds: ShapeNet10, ModelNet40, and ScanNet10 [52]. When dealing with meshes, we conduct our experiments on the Manifold40 dataset [3]. Finally, we use ShapeNet10 again for voxel grids, quantizing clouds to grids with resolution 64^3. Despite the different nature of the discrete representations taken into account, `nf2vec` allows us to perform shape classification on *NF*s embeddings, augmented online with E-Stitchup [53], by the very same downstream network architecture, i.e., a simple fully connected classifier consisting of three layers with 1024, 512 and 128 features. We consider as baselines well-known architectures that are optimized to work on the specific input representations of each dataset. For point clouds, we consider PointNet [48], PointNet++ [1] and DGCNN [2]. For meshes, we consider MeshWalker [50], a recent and competitive baseline that processes triangle meshes directly. As for voxel grids, we train a 3D CNN classifier that we implemented following [51] (Conv3DNet from now on). Since only the train and test splits are released for all the datasets, we created validation splits from the training sets in order to follow a proper train/val protocol for both the baselines and our method. As for the test shapes, we evaluated all the baselines on the discrete representations reconstructed from the *NF*s fitted on the original test sets, as these would be the only data available at test time in a scenario where *NF*s are used to store and communicate 3D data. The results in Table III show that `nf2vec` embeddings deliver classification accuracy close to the specialized baselines across all the considered datasets, regardless of the original discrete representation of the shapes in each dataset. Remarkably, our framework allows us to apply the same simple classification architecture to all the considered input modalities, in stark contrast with all the baselines that are highly specialized for each modality, exploit inductive biases specific to each such modality and cannot be deployed on representations different from those they were designed for. Furthermore, while presenting a gap of some accuracy points *w.r.t.* the most recent architectures, like DGCNN and MeshWalker, the simple fully connected classifier that we applied on `nf2vec` embeddings obtains scores comparable to standard baselines like PointNet and Conv3DNet.

Finally, in Fig. 11 (left), we present the baseline inference times, assuming discrete point clouds are available at test time, and compare it with that of `nf2vec`. Our framework is much

TABLE III
RESULTS ON SHAPE CLASSIFICATION ACROSS REPRESENTATIONS

Method	Input	ModelNet40	ShapeNet10	ScanNet10	Manifold40	ShapeNet10
PointNet [48]	Point cloud	88.8	94.3	72.7	–	–
PointNet++ [1]	Point cloud	89.7	**94.6**	**76.4**	–	–
DGCNN [2]	Point cloud	**89.9**	94.3	76.2	–	–
MeshWalker [50]	Mesh	–	–	–	**90.0**	–
Conv3DNet [51]	Voxels	–	–	–	–	92.1
`nf2vec`	*NF*	87.0	93.3	72.1	86.3	**93.0**

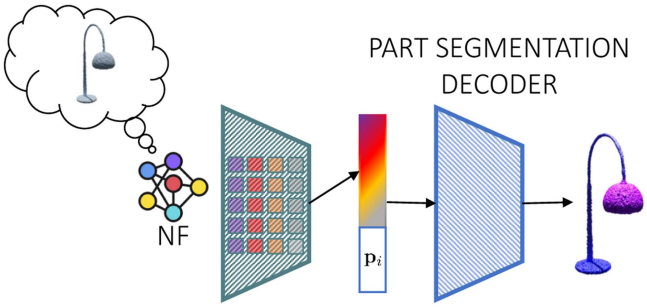


Fig. 12. *Point cloud part segmentation.* Method.

Fig. 13. *Point cloud part segmentation.* Qualitatives.

faster than competitors. Indeed, by processing directly *NF*s – where the resolution of the underlying signal is theoretically infinite – `nf2vec` can classify *NF*s representing point clouds with different numbers of points with a constant inference time of 0.001 seconds. On the contrary, the examined baselines suffer from the escalating resolution of the input point clouds. While PointNet and PointNet++ maintain a reasonable inference time even with 64K points, DGCNN experiences a significant slowdown as early as 16K points. Additionally, we emphasize that if 3D shapes are stored as *NF*s, the classification process using the designated specialized baselines would involve retrieving the original discrete representations through the extensive procedures outlined in [22]. Therefore, in Fig. 11 (right), we present the inference time of standard point cloud classification networks, including the time needed to reconstruct the discrete point cloud from the input *NF* of the underlying udf at various resolutions. Even at the coarsest resolution (2048 points), all the baselines exhibit an inference time that is one order of magnitude higher than the time required to classify the `nf2vec` embeddings directly. As the resolution of the reconstructed clouds increases, the inference time of the baselines becomes prohibitively high, while `nf2vec`, not reliant on explicit clouds, maintains a constant inference time of 0.001 seconds.

Point Cloud Part Segmentation: The classification and retrieval tasks explore the potential of utilizing `nf2vec` embeddings as a global representation of the input shapes. In contrast, in this section, we focus on point cloud part segmentation to examine if `nf2vec` embeddings also retain local shape properties. Part segmentation aims to predict a semantic (i.e., part) label for each point of a given cloud. We tackle this problem by training a decoder similar to that used to train our framework (see Fig. 12). Such decoder is fed with the `nf2vec` embedding of the *NF* representing the input cloud, concatenated with the coordinate of a 3D query, and it is trained to predict the label of the query point. We train it, as well as PointNet, PointNet++, and DGCNN, on the ShapeNet Part Segmentation dataset [54] with point clouds of 2048 points, with the same train/val/test as in the classification task. The outcomes presented in Table IV demonstrate the potential of accomplishing a local discriminative task, such as part segmentation, by using the task-agnostic embeddings generated by `nf2vec`. In doing so, the performance achieved is notably close to that of dedicated architectures designed for this specific task. Additionally, in Fig. 13, we show point clouds reconstructed at 100K points from the input *NF*s and segmented with high precision thanks to our formulation based on a semantic decoder conditioned by the `nf2vec` embedding.

Shape Generation: So far, we have validated that *NF* can be used as input in standard deep learning machinery thanks to `nf2vec`. In this section, we focus on the task of shape generation in an adversarial setting to investigate whether the compact representations produced by our framework can also be adopted as a medium for the output of generative deep learning pipelines. We employ a Latent-GAN [55] to generate embeddings resembling those produced by `nf2vec` from random noise, as illustrated in Fig. 14. Generated embeddings can be decoded into discrete representations using the implicit decoder from the `nf2vec` training. As our framework is agnostic towards the original discrete representation of shapes used to learn the *NF*s, we can train Latent-GANs with embeddings representing point clouds or meshes based on the same identical protocol and architecture (two simple fully connected networks as generator and discriminator). For point clouds, we train a Latent-GAN on the *chair* class of ShapeNet10, while we use models of cars provided by [9] when dealing with

TABLE IV
PART SEGMENTATION QUANTITATIVE RESULTS

Method	Input	instance mIoU	class mIoU	airplane	bag	cap	car	chair	earphone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skateboard	table
PointNet [48]	Point cloud	83.1	78.96	81.3	76.9	79.6	71.4	89.4	67.0	91.2	80.5	80.0	95.1	66.3	91.3	80.6	57.8	73.6	81.5
PointNet++ [1]	Point cloud	**84.9**	**82.73**	**82.2**	**88.8**	**84.0**	**76.0**	**90.4**	80.6	**91.8**	84.9	**84.4**	94.9	**72.2**	**94.7**	**81.3**	61.1	**74.1**	**82.3**
DGCNN [2]	Point cloud	83.6	80.86	80.7	84.3	82.8	74.8	89.0	**81.2**	90.1	**86.4**	84.0	**95.4**	59.3	92.8	77.8	**62.5**	71.6	81.1
nf2vec	*NF*	81.3	76.91	80.2	76.2	70.3	70.1	88.0	65.0	90.6	82.1	77.4	94.4	61.4	92.7	79.0	56.2	68.6	78.5

We report the IoU for each class, the mean IoU over all the classes (class mIoU) and the mean IoU over all the instances (instance mIoU).

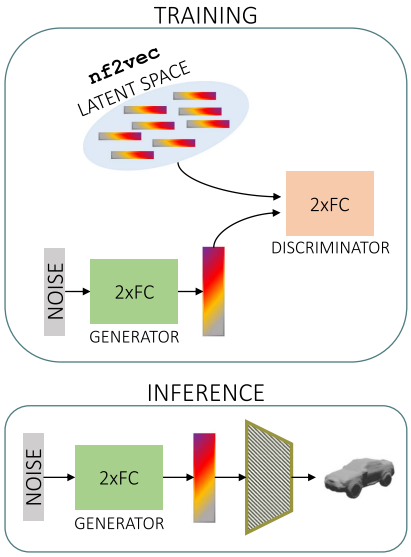


Fig. 14. *Learning to generate shapes from* nf2vec *latent space.* Method.

meshes. In Fig. 15, we report some examples generated with the above procedure, comparing them with SP-GAN [56] for what concerns point clouds and Occupancy Networks [9] (VAE formulation) for meshes. Shapes generated by our Latent-GAN, trained exclusively on nf2vec embeddings, look quite similar to those from the considered baselines regarding diversity and richness of details. Furthermore, by generating embeddings representing *NF*s, our method allows point cloud sampling at any arbitrary resolution (e.g., 8192 points in Fig. 15). In contrast, SP-GAN needs a new training for each desired resolution, as the number of generated points must be predetermined during training.

Learning a Mapping Between nf2vec *Embedding Spaces:* We have shown that nf2vec embeddings can be employed as a proxy of *NF*s as input to deep learning pipelines and that they can also be obtained as output of generative frameworks. In this section, we advance our exploration by examining the potential of learning a mapping between two distinct latent spaces generated by our framework for two separate datasets of *NF*s. This involves developing a *transfer* function specifically designed to operate on nf2vec embeddings as both input and output data. Such transfer function can be realized by a simple MLP that maps the input embedding into the output one and is trained with standard MSE loss (see Fig. 16). As nf2vec generates compact embeddings of the same dimension regardless of the input *NF* modality, the transfer function described here can be applied seamlessly to a great variety of tasks, usually tackled with ad-hoc frameworks tailored to specific input/output modalities. In particular, We explore two tasks. First, on the dataset presented in [57], we address point cloud completion by learning a mapping from nf2vec embeddings of *NF*s that represent incomplete clouds to embeddings associated with complete clouds. Then, we tackle the task of surface reconstruction on ShapeNet cars, training the transfer function to map nf2vec embeddings representing point clouds into embeddings that can be decoded into meshes. As we note from Figs. 17 and 18, in both tasks, the transfer function can learn an effective mapping between nf2vec latent spaces. Indeed, by processing exclusively *NF* embeddings, we can obtain output shapes that are highly compatible with the input ones whilst preserving their distinctive details, like the pointy wing of the airplane in Fig. 17 or the flap of the first car in Fig. 18.

VI. DEEP LEARNING ON NERFS

In this section, our focus shifts to processing *NF*s encoding both geometry and appearance of objects, i.e., NeRFs. The goal is to illustrate the efficacy of nf2vec in addressing various downstream tasks related to 3D objects implicitly represented by NeRFs.

General Settings: In all experiments detailed within this section, we learn NeRFs from images using an MLP comprising three hidden layers with 64 nodes each. We utilize the ReLU activation function between all layers except the final layer, which computes the density and RGB values without any activation function. NeRFs take as input the frequency encoding of the 3D coordinates as in [11]. NeRFs are trained using an L_1 loss between predicted and estimated RGB pixel intensities, weighting background pixels less than foreground pixels (0.8 foreground versus 0.2 background). We use the NeRF formulation without the view direction in input. When training nf2vec, we adhere to the same encoder and decoder architectures previously described in Section V. The key distinction lies in the implicit decoder, wherein the dimensions of its layers are doubled. Throughout these experiments, both baseline models and nf2vec undergo training with offline data augmentation on the 3D shapes used to generate renderings for training the models. This augmentation

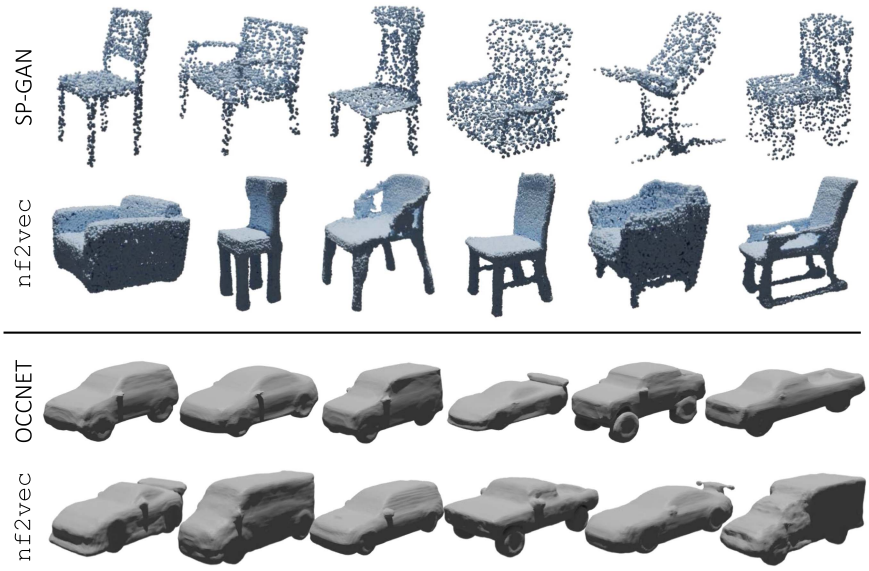


Fig. 15. *Learning to generate shapes from* nf2vec *latent space.* Qualitative results.

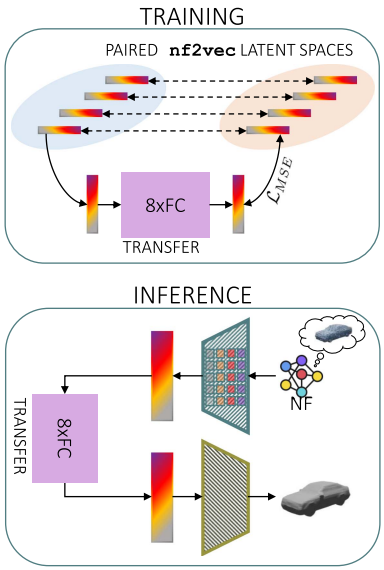


Fig. 16. *Learning a mapping between* nf2vec *latent spaces.* Method.

involves implementing random deformations and introducing random color changes to each component of the original 3D shapes.

NeRF Retrieval. We first investigate the quality of nf2vec embeddings with a retrieval task as done in Section V. Given an embedding of an input NeRF, we extract the classes of its k-nearest neighbor within the embedding latent space and compare them to the input NeRF's class.

We also implement two baseline approaches: the single-view and multi-view baselines. Both strategies rely on a *ResNet50* [58] backbone pre-trained on ImageNet [59]. We extract feature vectors with *ResNet50* from each image. Given a single image for the single-view baseline or 9 images for the multi-view

TABLE V
NERF RETRIEVAL QUANTITATIVE RESULTS

Method	#Views	mAP@1	mAP@5	mAP@10
nf2vec	-	72.38	**91.89**	**95.96**
ResNet50 [58] single-view	1	74.65	91.52	95.10
ResNet50 [58] multi-view	9	**82.74**	91.66	93.79

baseline, we find the k-nearest neighbors in the *ResNet50* feature space. We compare the classes of query and retrieved objects and compute the mAP for different k as done in Table II. In the case of the multi-view baseline, we retrieve the nearest neighbors for each of the 9 input images. Then, we select the class with the highest frequency to calculate the mAP. All these experiments are conducted on embeddings of unseen NeRFs from the test set of ShapeNetRender [60]. The quantitative results in Table V indicate that nf2vec yields comparable performance to the baselines and, in certain instances, outperforms them. Moreover, Fig. 19 shows that the selected neighbors exhibit similar structures and colors. There is, however, no way to prioritize one property over the other, a limitation discussed in Section IX.

NeRF Classification: This section investigates the task of predicting the category of an object represented by a NeRF. In this scenario, only NeRFs would be available as input data.

Our approach processes nf2vec embeddings with the same classification architecture as already deployed for shapes, a three-layer MLP with 1024, 512, and 128 neurons each. We highlight that the embeddings are obtained by processing the NeRFs weights without sampling the underlying *RF*.

As the discrete representations used to learn NeRFs are a set of images depicting the same object, selecting a proper baseline is not straightforward. In our experiment, we choose ResNet50 [58] as the baseline classifier. The network predicts the class for a given input image. Given this architecture, akin

Fig. 17. *Learning a mapping between* `nf2vec` *latent spaces.* Point cloud completion.

Fig. 18. *Learning a mapping between* `nf2vec` *latent spaces.* Surface reconstruction.

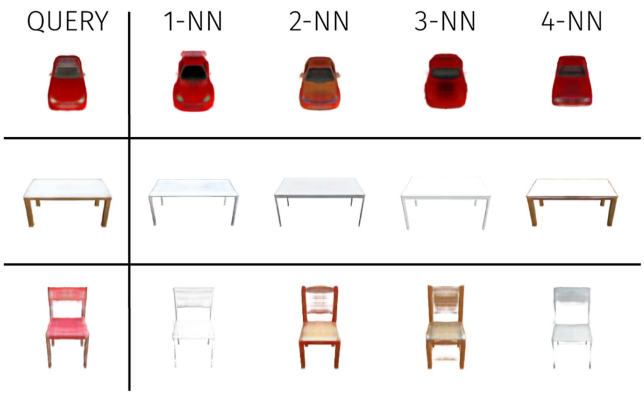


Fig. 19. *NeRF retrieval qualitative results.* Given the `nf2vec` embedding of a query NeRF, we show the renderings reconstructed from the closest embeddings (L2 distance).

TABLE VI
RESULTS ON NERF CLASSIFICATION

Method	#Views	Accuracy (%)
`nf2vec`	-	87.28
ResNet50 [58] Single-view	1	86.88
ResNet50 [58] Multi-view	9	**93.28**

TABLE VII
NERFS CLASSIFICATION INFERENCE TIME

	Time (ms)			
Method	Encoding	Rendering	Classification	Total
`nf2vec`	0.75	-	0.28	**1.03**
Resnet50 [58] Single-View	-	4.63	6.21	10.84
Resnet50 [58] Multi-View	-	41.67	55.89	97.56

All times are in milliseconds. For each method, we report the time needed for each pipeline step and the total time. Encoding: `nf2vec` encoding of *NF* weights. Rendering: obtaining images from NeRFs. Classification: the time required by the classifier.

to the retrieval experiment, we propose two types of baseline approaches, single-view and multi-view. In the former, we train the network on a single rendering for each NeRF obtained from the same fixed pose, while for the latter, we employ 9 renderings for each NeRF from different viewpoints. At test time, regarding the single-view approach, we test the network on images rendered from unseen NeRFs employing the same training pose. Concerning the multi-view baseline, we render 9 images from the training viewpoints for each unseen NeRF, obtaining 9 distinct predictions per object, that we aggregate by taking the class predicted with the highest frequency.

We report the accuracy results on ShapeNetRender [60] in Table VI. Moreover, we also report the time required to classify NeRFs in Table VII, highlighting the impact of each of the main pipeline steps, such as the `nf2vec` encoding time for our approach and the time to render images for the baseline approaches. We point out that we rely on a fast implementation of NeRFs provided by the NerfAcc [61] library. We note that the classifier directly leveraging *nerf2vec* embeddings achieves a slightly better score than the single-view baseline while being worse than the multi-view one. However, if we analyze the total inference times, our approach is two orders of magnitude faster than the baselines (1.03 ms our versus 97.56 ms ResNet50 multi-view). This holds true even excluding the rendering time and looking only at the classification time (third column of Table VII): ResNet50 classifiers processing 224×224 images are remarkably slower than our full pipeline, taking only 1.03 ms.

NeRF Generation: We experiment here with the task of generating `nf2vec` embeddings with the same approach depicted in Fig. 14. We trained multiple adversarial networks, one for

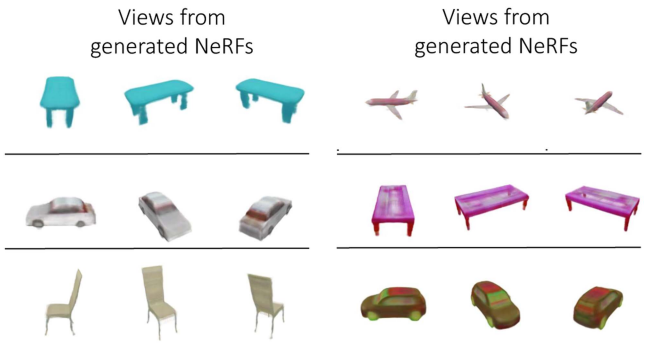


Fig. 20. *Learning to generate NeRFs from* nf2vec *latent space.* Qualitative results.

Fig. 21. *Learning a mapping between* nf2vec *latent spaces. UDF* to NeRF.

each class, utilizing the ShapeNetRender dataset [60]. After the embedding generation, we can decode them into discrete representations using the same implicit decoder employed to train the framework. In Fig. 20, we show images rendered from NeRFs generated through this process. The renderings have a good level of realism and diversity. Notably, the 3D consistency of images obtained from different viewpoints is preserved. These results show that generating novel NeRFs in the form of nf2vec embeddings is possible.

Learning a Mapping Between Embedding Spaces: We explore here whether it is possible to learn a *transfer* network that maps nf2vec embeddings of *UDF*s to nf2vec embeddings of NeRFs using the methodology described in Fig. 16. We highlight that it is a non-trivial task, as the network has to hallucinate the appearance of the input shape without modifying the underlying 3D structure.

We run this experiment on ShapeNetRender [60], using the rendered images to learn NeRFs and the corresponding 3D models to learn *UDF* neural fields. Then, we train nf2vec and then the *transfer* network on *NF*s of the training set. Finally, we test the *transfer* network on unseen test *NF*, obtaining for each *UDF* latent code the corresponding mapped nf2vec embedding. We report qualitative results in Fig. 21, showing the point clouds obtained from the input *UDF* in the first row, and the renderings obtained by feeding the mapped embedding

TABLE VIII
PROPERTIES OF INPUT NFS USED BY RECENT METHODS PROCESSING NEURAL FIELDS

Method	Type	#Params	Mesh Reconstruction	
			CD (mm) ↓	F-score (%) ↑
DeepSDF [5]	Shared MLP + Embedding	2400K	6.6	25.1
Functa [15]	Shared MLP + Modulation	7091K	2.85	21.3
DWS [24]	Individual SIREN MLP	**800K**	**0.26**	**69.7**
nf2vec	Individual SIREN MLP	**800K**	**0.26**	**69.7**

Mesh reconstruction results on Manifold40 test set.

to the implicit decoder of nf2vec in the last three rows. We can appreciate that the mapped NeRF preserves the original shape geometry, enabling the rendering of realistic images from various viewpoints. Notably, the renderings exhibit plausible diverse colors associated with different object parts, as can be seen for the glasses and the wheels of the blue car in the second column of Fig. 21.

VII. COMPARISON WITH RECENT APPROACHES

As outlined in Section I, several contemporary works addressing the problem of processing neural fields have been proposed recently. For all methods, the goal is to perform deep learning tasks such as classification using as input data a *NF*, i.e., data represented with continuous functions. We can divide these methods into two categories. Those relying on a shared network and those focusing on individual *NF*s. In the former case, referred to here as *Shared*, the *NF* is defined as a shared network trained on all training samples, plus a distinct vector representing each object. Typically this vector is processed to perform downstream tasks. This is the case of Functa [15] and DeepSDF [5]. In the latter case, denoted as *Individual*, the *NF* is typically an MLP trained on a single object or scene. In this scenario, the MLP weights are processed directly to perform the downstream tasks. This is the case of our framework, nf2vec, as well as NFN [42], NFT [23], and DWS [24].

In this section, we investigate the characteristics of each category of techniques, showing that *Shared* frameworks are problematic, as they cannot reconstruct the underlying signal with high fidelity and need a whole dataset to learn the neural field of an object. Moreover, we build the first benchmark of *NF* classification by comparing recent approaches in this area.

Representation Quality: We first investigate the representation quality of *Shared* approaches compared to *Individual* ones. Specifically, we compare the reconstructions of explicit meshes from *SDF* neural fields with ground truth meshes on the Manifold40 test set. We report the quantitative comparisons in Table VIII, using two metrics: the Chamfer Distance (CD) as defined in [49], and the F-Score as defined in [62]. We note that we use the SIREN MLP described in Section V to represent *SDF* with *Individual* frameworks. In the first two rows, we note that *Shared* methods achieve poor reconstruction performance. Indeed, we believe that representing a whole dataset with a shared network is a difficult learning task, and the network struggles to fit accurately the totality of the samples. *Individual* methods instead do not suffer from this problem and achieve very good reconstruction performance. Moreover, we believe that

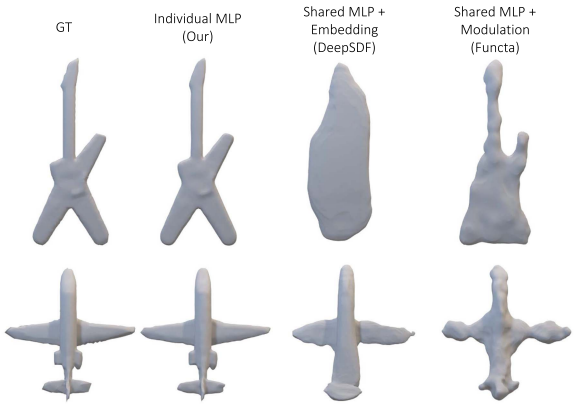


Fig. 22. Reconstruction comparison for Manifold40 meshes obtained from *SDF*.

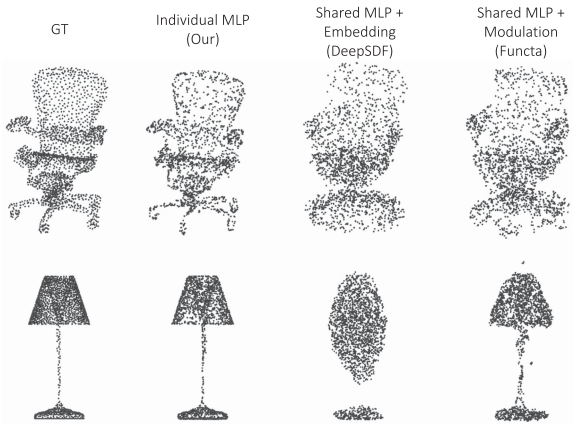


Fig. 23. Reconstruction comparison for ModelNet40 point clouds obtained from *UDF*.

the approaches based on *Shared* networks struggle to represent unseen samples the further they are from the training distribution. Hence, in the foreseen scenario where *NF*s become a standard representation for 3D data hosted in public repositories, leveraging on a single shared network may imply the need to frequently retrain the model upon uploading new samples, which, in turn, would change the embeddings of all the previously stored data. On the contrary, uploading the repository with a new object would not cause any sort of issue with individual *NF*s, where one learns a network for each data point. Finally, we also provide a qualitative perspective of the aforementioned problem in Figs. 22 and 23. The visualizations confirm the results of Table VIII, with shared network frameworks struggling to represent properly the ground-truth shapes, while individual *NF*s enable high-fidelity reconstructions. We note that the quality of our DeepSDF reconstructions, where a single model is trained on the whole dataset, is inferior to the one reported in [5], where instead a different auto-decoder is trained for each class. This approach is not applicable in our case, as it would require to know in advance the class label of each shape in order to choose the right auto-decoder.

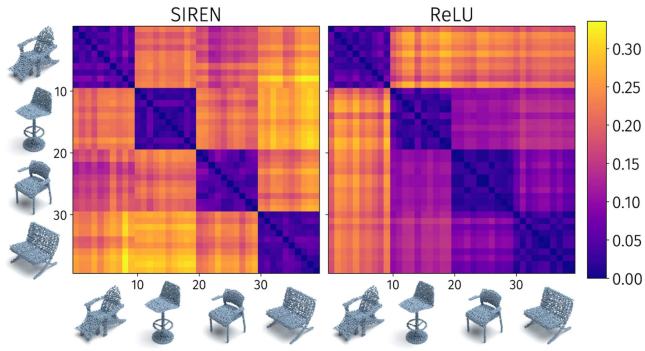


Fig. 24. *L2 distances between* `nf2vec` *embeddings.* For each shape, we fit 10 *NF*s starting from the same weight initialization (40 *NF*s in total). We then plot the L2 distances between the embeddings obtained by `nf2vec` for such *NF*s.

TABLE IX
CLASSIFICATION ACCURACY OF RECENT METHODS PROCESSING NEURAL FIELDS

Method	Type	Input	ModelNet40 (*UDF*)	Manifold40 (*SDF*)
DeepSDF [5]	Shared	Embedding	41.2	64.9
Functa [15]	Shared	Modulation	**87.3**	85.9
DWS [24]	Individual	MLP weights	71.6	69.9
`nf2vec`	Individual	MLP weights	87.0	**86.3**

We believe that these results highlight that frameworks based on a single shared network cannot be used as a medium to represent objects as *NF*s, because of their limited representation power when dealing with large and varied datasets and because of their difficulty in representing new shapes not available at training time.

Classification Accuracy: We compare recent methods in the *NF*s classification task. The goal is to predict the category of objects represented within the input *NF*s without recreating the discrete signals. Specifically, we test all methods on *UDF* obtained from point clouds of ModelNet40 [47] and on *SDF* learned from meshes of Manifold40 [3]. We compare `nf2vec` with other frameworks designed to process *NF*s realized as Individual MLPs, such as DWSNet [24]. These methods process the MLP weights of individual *NF*s, implemented as SIREN networks [16]. The MLPs in our benchmark are initialized using the same random seed (see Section VIII for more details). We also tried to run other recent *Individual* approaches, such as NFN [42] and NFT [23], but the training did not converge, probably because the SIREN MLPs used in our experiments are much larger than those used in their paper. Moreover, we compare with *Shared* frameworks where neural fields are realized by a shared network and a small latent vector or modulation, i.e., DeepSDF [5] and Functa [15]. Whenever possible, we use the official code released by the authors to run the experiments.

As we can see from results reported in Table IX, Functa and `nf2vec` achieve the best results with a large margin over other competitors, with the former slightly more accurate on *UDF*s while the latter on *SDF*s. However, as explained in the previous section, since our method does not rely on shared networks, it

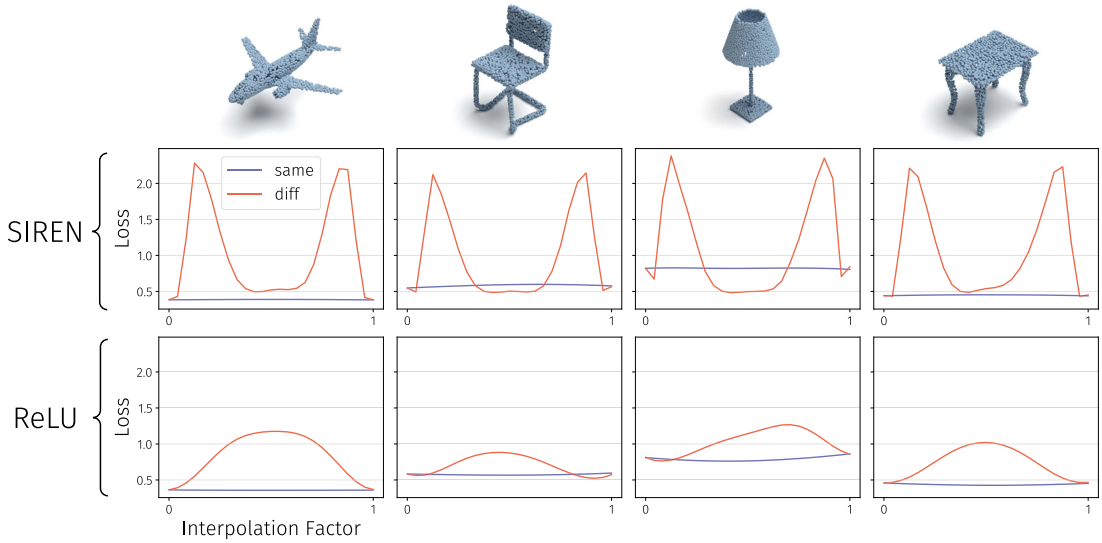


Fig. 25. *Linear mode connectivity study.* Each plot shows the variation of the loss function over the same batch of points when interpolating between two *NF*s representing the same shape. The red line describes the interpolation between *NF*s initialized differently, whereas the blue line shows the interpolation between *NF*s initialized with the same random vector.

does not sacrifice the representation quality of *NF*s and it is more suitable for real-world applications.

VIII. Using the Same Initialization for *NF*s

The need to align the multitude of *NF*s that can approximate a given shape is a challenging research problem that has to be dealt with when using *NF*s as input data. We empirically found that fixing the weights initialization to a shared random vector across *NF*s is a viable and simple solution to this problem.

We report here an experiment to assess if the order of data or other sources of randomness arising while fitting *NF*s do affect the repeatability of the embeddings computed by `nf2vec`. We fitted 10 *NF*s on the same discrete shape for 4 different chairs, i.e., 40 *NF*s in total. Then, we embed all of them with the pretrained `nf2vec` encoder and compute the L2 distance between all pairs of embeddings. The block structure of the resulting distance matrix (see Fig. 24) highlights how, under the assumption of shared initialization, `nf2vec` is repeatable across multiple fittings.

Seeking for a proof with a stronger theoretical foundation, we turn our attention to the recent work *git re-basin* [63], where authors show that the loss landscape of neural networks contains (nearly) a single basin after accounting for all possible permutation symmetries of hidden units. The intuition behind this finding is that, given two neural networks that were trained with equivalent architectures but different random initializations, data orders, and potentially different hyperparameters or datasets, it is possible to find a permutation of the network's weights such that when linearly interpolating between their weights, all intermediate models enjoy performance similar to them – a phenomenon denoted as *linear mode connectivity*.

Intrigued by this finding, we conducted a study to assess whether initializing *NF*s with the same random vector, which we found to be key to `nf2vec` convergence, also leads to linear mode connectivity. Thus, given one shape, we fitted it with two different *NF*s, and then we interpolated linearly their weights, observing at each interpolation step the loss value obtained by the *interpolated NF* on the same batch of points. We repeated the experiment twice for each shape, once initializing the *NF*s with different random vectors and once with the same random vector.

The results of this experiment are reported for four different shapes in Fig. 25. It is possible to note that, as shown by the blue curves, when interpolating between *NF*s obtained from the same weights initialization, the loss value at each interpolation step is nearly identical to those of the boundary *NF*s. On the contrary, the red curves highlight how there is no linear mode connectivity at all between *NF*s obtained from different weights initializations.

[63] also proposes different algorithms to estimate the permutation needed to obtain linear mode connectivity between two networks. We applied the algorithm proposed in their paper in Section III-B (*Matching Weights*) to our *NF*s and observed the resulting permutations. Remarkably, when applied to *NF*s obtained from the same weights initialization, the retrieved permutations are identity matrices, both when the target *NF*s represent the same shape and when they represent different ones. Instead, the permutations obtained for *NF*s obtained from different initializations are far from being identity matrices.

All these results favor the hypothesis that our technique of initializing *NF*s with the same random vector leads to linear mode connectivity between different *NF*s. We believe that the possibility of performing meaningful linear interpolation between the weights occupying the same positions across different *NF*s can be interpreted by considering corresponding weights as carrying out the same role in terms of feature detection units, explaining why the `nf2vec` encoder succeeds in processing the weights of our *NF*s.

TABLE X
POINT CLOUD CLASSIFICATION FROM UDF WITH nf2vec

Method	Test accuracy (%)
nf2vec SIREN	87.0
nf2vec ReLU	**88.1**

Comparison of nf2vec trained on SIRENs [16] vs ReLU MLPs on ModelNet40.

The experiments in this section were conducted on *NF* with sine and ReLU activation functions, as those are the activations used throughout this paper. To further validate the applicability of our method to SIRENs and ReLU *NF*s, we show in Table X the comparable results obtained by classifying nf2vec embeddings of *NF*s with different activation functions.

IX. LIMITATIONS

We point out three main limitations of our approach: i) although *NF*s capture continuous geometric cues, in some cases deep learning on nf2vec embeddings achieve results inferior to state-of-the-art solutions that work on specific discrete representations; ii) there is no obvious way to perform online data augmentation on shapes represented as *NF*s by directly altering their weights; iii) when processing NeRFs, appearance and geometry are intrinsically entangled in our framework, both in NeRFs as an architecture, where a single MLP predicts both color and density, and in the method we use to process them, which does not take any measure aimed at disentangling geometric and texture information. As a result, nf2vec could not be applied in scenarios where one needs to control how much appearance and geometry independently affect the outcome. We plan to investigate these shortcomings in future works.

X. CONCLUDING REMARKS

We have shown that it is possible to apply deep learning on individual *NF*s representing 3D shapes and object-centric radiance fields. Our approach leverages a task-agnostic encoder which embeds *NF*s into compact and meaningful latent codes without accessing the underlying function. We have shown that these embeddings can be fed to standard deep-learning machinery to solve various tasks effectively. Moreover, we have introduced the first benchmark for the task of *NF* classification, showing that our proposal obtains the best score (on par with Functa [15]) while preserving the ability to reconstruct the input dataset with high quality.

In the future, we plan to go beyond *NF*s of 3D objects by applying nf2vec to *NF*s encoding 3D scenes and other input modalities, like images or audio. We will also investigate weight-space symmetries [64] as a different path to favor the alignment of weights across *NF*s, possibly while exploring the effect of activation functions other than sine and ReLU.

We reckon that our work may foster the adoption of *NF*s as a unified 3D representation, overcoming the current fragmentation of 3D structures and processing architectures.

REFERENCES

[1] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5099–5108.

[2] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 1–12, 2019.

[3] S.-M. Hu et al., "Subdivision-based mesh convolution networks," *ACM Trans. Graph.*, vol. 41, no. 3, pp. 1–16, 2022.

[4] Y. Xie et al., "Neural fields in visual computing and beyond," 2021, *arXiv:2111.11426*.

[5] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 165–174.

[6] J. Chibane et al., "Neural unsigned distance fields for implicit function learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 21 638–21 652.

[7] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman, "Implicit geometric regularization for learning shapes," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3789–3799.

[8] T. Takikawa et al., "Neural geometric level of detail: Real-time rendering with implicit 3D shapes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 11 358–11 367.

[9] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3D reconstruction in function space," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4460–4470.

[10] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, "Convolutional occupancy networks," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2020, pp. 523–540.

[11] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2020, pp. 405–421.

[12] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, Jul. 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530127

[13] J. N. P. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, and G. Wetzstein, "ACORN: Adaptive coordinate networks for neural scene representation," *ACM Trans. Graph.*, vol. 40, no. 4, 2021, Art. no. 58.

[14] H.-T. D. Liu, F. Williams, A. Jacobson, S. Fidler, and O. Litany, "Learning smooth neural functions via Lipschitz regularization," 2022, *arXiv:2202.08345*.

[15] E. Dupont, H. Kim, S. A. Eslami, D. J. Rezende, and D. Rosenbaum, "From data to functa: Your data point is a function and you can treat it like one," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 5694–5725.

[16] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 7462–7473.

[17] V. Sitzmann, E. Chan, R. Tucker, N. Snavely, and G. Wetzstein, "MetaSDF: Meta-learning signed distance functions," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 10 136–10 147.

[18] E. Dupont, A. Goliński, M. Alizadeh, Y. W. Teh, and A. Doucet, "COIN: Compression with implicit neural representations," 2021, *arXiv:2103.03123*.

[19] Y. Strümpler, J. Postels, R. Yang, L. Van Gool, and F. Tombari, "Implicit neural representations for image compression," 2021, *arXiv:2112.04267*.

[20] Y. Zhang, T. van Rozendaal, J. Brehmer, M. Nagel, and T. Cohen, "Implicit neural video compression," 2021, *arXiv:2112.11312*.

[21] M. Tancik et al., "Fourier features let networks learn high frequency functions in low dimensional domains," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 7537–7547.

[22] L. De Luigi, A. Cardace, R. Spezialetti, P. Z. Ramirez, S. Salti, and L. Di Stefano, "Deep learning on implicit neural representations of shapes," in *Proc. Int. Conf. Learn. Representations*, 2023, pp. 1–39.

[23] A. Zhou et al., "Neural functional transformers," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2023, Art. no. 3387.

[24] A. Navon, A. Shamsian, I. Achituve, E. Fetaya, G. Chechik, and H. Maron, "Equivariant architectures for learning in deep weight spaces," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 25790–25816.

[25] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018, *arXiv:1803.03635*.

[26] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artif. Intell. Rev.*, vol. 53, no. 7, pp. 5113–5155, 2020.
[27] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene representation networks: Continuous 3D-structure-aware neural scene representations," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 1119–1130.
[28] C. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, and T. Funkhouser, "Local implicit grid representations for 3D scenes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 6001–6010.
[29] M. Atzmon and Y. Lipman, "SAL: Sign agnostic learning of shapes from raw data," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2565–2574.
[30] Z. Chen and H. Zhang, "Learning implicit fields for generative shape modeling," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5939–5948.
[31] S. Saito, Z. Huang, R. Natsume, S. Morishima, A. Kanazawa, and H. Li, "PIFU: Pixel-aligned implicit function for high-resolution clothed human digitization," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 2304–2314.
[32] M. Oechsle, L. Mescheder, M. Niemeyer, T. Strauss, and A. Geiger, "Texture fields: Learning texture representations in function space," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 4531–4540.
[33] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, "Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 3504–3515.
[34] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, "Occupancy flow: 4D reconstruction by learning particle dynamics," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 5379–5389.
[35] T. Unterthiner, D. Keysers, S. Gelly, O. Bousquet, and I. O. Tolstikhin, "Predicting neural network accuracy from weights," 2020, *arXiv:2002.11448*.
[36] K. Schürholt, D. Kostadinov, and D. Borth, "Self-supervised representation learning on neural network weights for model characteristic prediction," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 16481–16493. [Online]. Available: https://openreview.net/forum?id=F1D8buayXQT
[37] B. Knyazev, M. Drozdzal, G. W. Taylor, and A. Romero, "Parameter prediction for unseen deep architectures," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 29433–29448. [Online]. Available: https://openreview.net/forum?id=vqHak8NLk25
[38] F. Jaeckle and M. P. Kumar, "Generating adversarial examples with graph neural networks," in *Proc. 37th Conf. Uncertainty Artif. Intell.*, 2021, pp. 1556–1564.
[39] J. Lu and M. P. Kumar, "Neural network branching for neural network verification," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1evfa4tPB
[40] F. Ballerini, P. Zama Ramirez, R. Mirabella, S. Salti, and L. Di Stefano, "Connecting NeRFs, images, and text," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops*, 2024, pp. 866–876.
[41] R. Hecht-Nielsen, "On the algebraic structure of feedforward network weight spaces," in *Advanced Neural Computers*. Amsterdam, The Netherlands: Elsevier, 1990, pp. 129–135.
[42] A. Zhou et al., "Permutation equivariant neural functionals," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2023, Art. no. 1085.
[43] A. Cardace, P. Zama Ramirez, F. Ballerini, A. Zhou, S. Salti, and L. Di Stefano, "Neural processing of tri-plane hybrid neural fields," in *Proc. 12th Int. Conf. Learn. Representations*, 2024, pp. 1–27.
[44] Z. Erkoç, F. Ma, Q. Shan, M. Nießner, and A. Dai, "HyperDiffusion: Generating implicit neural fields with weight-space diffusion," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2023, pp. 14 300–14 310.
[45] A. X. Chang et al., "ShapeNet: An information-rich 3D model repository," 2015, *arXiv:1512.03012*.
[46] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, 1987.
[47] Z. Wu et al., "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1912–1920.
[48] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 652–660.
[49] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3D object reconstruction from a single image," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 605–613.
[50] A. Lahav and A. Tal, "MeshWalker: Deep mesh understanding by random walks," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–13, 2020.
[51] D. Maturana and S. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 922–928.
[52] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3D reconstructions of indoor scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5828–5839.
[53] C. R. Wolfe and K. T. Lundgaard, "E-stitchup: Data augmentation for pre-trained embeddings," 2019, *arXiv:1912.00772*.
[54] L. Yi et al., "A scalable active framework for region annotation in 3D shape collections," *ACM Trans. Graph.*, vol. 35, 2016, Art. no. 210.
[55] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3D point clouds," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 40–49.
[56] R. Li, X. Li, K.-H. Hui, and C.-W. Fu, "SP-GAN: Sphere-guided 3D shape generation and manipulation," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–12, 2021.
[57] L. Pan et al., "Variational relational point completion network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 8524–8533.
[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[59] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," 2015, *arXiv:1409.0575*.
[60] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann, "DISN: Deep implicit surface network for high-quality single-view 3D reconstruction," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., 2019, Art. no. 45. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/39059724f73a9969845dfe4146c5660e-Paper.pdf
[61] R. Li, H. Gao, M. Tancik, and A. Kanazawa, "NerfAcc: Efficient sampling accelerates NeRFs," 2023, *arXiv:2305.04966*.
[62] M. Tatarchenko, S. R. Richter, R. Ranftl, Z. Li, V. Koltun, and T. Brox, "What do single-view 3D reconstruction networks learn?," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 3405–3414.
[63] S. K. Ainsworth, J. Hayase, and S. Srinivasa, "Git re-basin: Merging models modulo permutation symmetries," 2022, *arXiv:2209.04836*.
[64] R. Entezari, H. Sedghi, O. Saukh, and B. Neyshabur, "The role of permutation invariance in linear mode connectivity of neural networks," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–24.

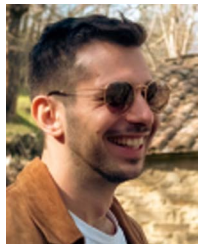
Pierluigi Zama Ramirez received the PhD degree in computer science and engineering, in 2021. He has been a research Intern with Google for six months and is currently a post-doc with the University of Bologna. He co-authored more than 20 publications on computer vision research topics, such as semantic segmentation, depth estimation, optical flow, domain adaptation, virtual reality, and 3D computer vision.

Luca De Luigi received the PhD degree in computer science and engineering, in 2023. He is currently a computer vision engineer with eyecan.ai. His research focuses on deep learning for computer vision problems, especially in 3D geometry and neural fields.

Daniele Sirocchi received the master's degree in artificial intelligence, in 2023. He currently works as a software engineer, mainly focused on creating applications that blend various technologies, encompassing a wide spectrum of technological domains. His research interests primarily revolve around machine and deep learning for computer vision tasks, where he extensively engages with neural fields.

Adriano Cardace is currently working toward the PhD degree with the Computer Vision Laboratory (CVLab), University of Bologna. He has been a research intern with Mitsubishi Electric Research Laboratories (MERL) and authored several research papers covering a range of subjects, including semantic segmentation, domain adaptation, and neural fields.

Riccardo Spezialetti received the PhD degree in computer science and engineering from the University of Bologna, in 2020. After two years as a post-doc researcher with the Department of Computer Science and Engineering, University of Bologna, he recently joined eyecan.ai as a computer vision engineer. His research interest concerns machine/deep learning for 3D computer vision problems.



Francesco Ballerini received the master's degree in artificial intelligence from the University of Bologna, in 2023, where he is currently working toward the PhD degree in computer science and engineering. His main research interests include the study of neural architectures aimed at processing neural fields, specifically those representing 3D data, with a focus on permutation symmetries and the effect of initialization on the input neural fields themselves.

Samuele Salti is currently an associate professor with the Department of Computer Science and Engineering (DISI), University of Bologna, Italy. His main research interest is computer vision, mainly 3D computer vision and machine/deep learning applied to computer vision problems. He has co-authored more than 60 publications and eight international patents. In 2020, he co-founded the start-up eyecan.ai.

Luigi Di Stefano received the PhD degree in electronic engineering and computer science from the University of Bologna, in 1994. He is a full professor with the Department of Computer Science and Engineering, University of Bologna, where he founded and led the Computer Vision Laboratory (CVLab). His research interests include image processing, computer vision, and machine/deep learning. He is the author of more than 150 papers and several patents. He has been a scientific consultant for major computer vision and machine learning companies. He is a member of the IEEE Computer Society and the IAPR-IC.

Open Access funding provided by 'Alma Mater Studiorum - Università di Bologna' within the CRUI CARE Agreement