ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Machine Learning for Computer Vision

NERF2VEC: DEEP LEARNING ON NEURAL RADIANCE FIELDS

CANDIDATE

Daniele Sirocchi

SUPERVISOR

Prof. Samuele Salti

CO-SUPERVISORS

Dr. Luca De Luigi

Dr. Pierluigi Zama Ramirez

Academic year 2022-2023

Session 3rd

AI is the new electricity.

Andrew Ng

Abstract

Virtualization of 3D world remains a challenge, as a standardized technique has yet to emerge. Neural Radiance Fields (NeRFs), a recent and promising approach, have attracted a lot of excitement due to their speed and quality reconstruction capabilities. Thus, NeRFs are poised to shape the future of 3D world modeling. However, a question arises about the potential use of NeRFs as input and output data for other algorithms due to their neural network nature. Additionally, since they have been introduced recently, there is no publicly available large dataset of NeRFs suitable for training deep learning models. Hence, in the initial phase of this thesis, a new and rigorously organized dataset of NeRFs was assembled. This dataset served as the bedrock upon which the subsequent research was built. Besides, it brings considerable value to the wider research community, paving the way for future advancements in the field. The following stride involved the development of *nerf2vec*, a new framework designed to learn embeddings that serve as compressed representations of input NeRFs. This endeavor highlighted the capacity of these embeddings to faithfully represent the underlying NeRFs, maintaining high reconstruction quality. Moreover, this work showcased the direct applicability of these embeddings within deep neural architectures, tackling tasks like classification, retrieval, embeddings interpolation, and adversarial generation. It achieved noteworthy results comparable to state-of-the-art methods, while optimizing resource usage and eliminating the need for costly machinery. To the best of our knowledge, this is the first work that introduces these approaches for NeRFs and makes a significant contribution to the adoption of NeRFs as a standard way to represent 3D scenes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

From the beginning of computer vision, the world around us has consistently been portrayed through 2D images arranged in a two-dimensional grid of pixels. Nevertheless, the real world is inherently 3D, prompting the development of various techniques aimed at capturing its complexity through more intricate representations. Among these representations, voxels, point clouds, and meshes are commonly employed today. Unlike 2D images, which benefit from a multitude of neural architectures optimized for efficient use in various deep learning tasks, these 3D representations present unique challenges, including the need for specialized machinery and substantial computational resources required for processing them.

A significant advancement in the field is exemplified by Implicit Neural Representations (INRs). INRs possess the ability to model a continuous function that implicitly represents a signal of interest using a basic Multi-Layer Perceptron (MLP). They are adept at encoding 3D objects by fitting various functions, including the signed distance function, unsigned distance function, and occupancy fields. A key advantage of INRs is that they decouple the memory cost of the representation from the spatial resolution. This means that any resolution can potentially be reconstructed from the fixed number of parameters within the MLP.

As extensively demonstrated in the *inr2vec* paper [41], the utilization of INRs has proven highly effective for addressing deep learning tasks by directly processing their weights within neural architectures. Additionally, this approach significantly reduces the computational resources required compared to using the original 3D representations. However, *inr2vec* did not explore a very recent and promising type of INR, namely Neural Radiance Fields (NeRFs) [6]. They are a class of neural network models used for 3D scene representation, which capture the 3D structure and appearance of scenes by modeling the radiance (color) and occupancy at each point in space. NeRFs can be very efficiently trained using datasets of images and corresponding camera poses, allowing them to render novel views of scenes and synthesize 3D reconstructions, obtaining high-quality results.

At this point, a new research question arises, and it serves as the focal point to which this thesis aims to provide a response: *would it be possible to directly integrate NeRFs within deep learning pipelines to solve downstream tasks?* While this integration has already been demonstrated with other types of implicit representations such as unsigned distance functions, signed distance functions and occupancy fields, NeRFs remain uncharted territory for such applications.

In the pursuit of answering this new research question, two key challenges were addressed. The first challenge lies in the scarcity of suitable datasets for NeRF-based research. In response, a new dataset was developed, building upon ShapeNet [4], a well-known dataset of 3D objects. Renderings of each 3D object were captured from various viewpoints to form the image dataset necessary to train NeRFs. Additionally, data augmentation techniques were applied to enrich the newly established dataset. The second challenge is the inherent parameters redundancy of NeRFs due to their MLP nature, and this

could potentially lead to a notable increase in computational costs when deploying naive solutions. Therefore, a new framework called *nerf2vec* was developed with the primary objective of generating compact embedding representations for each NeRF, while maintaining their original quality. Subsequently, the focus of the thesis shifted towards assessing the quality of the embeddings generated by *nerf2vec*, directly utilizing them as inputs for neural architectures designed to tackle different deep learning tasks.

nerf2vec adopts an encoder-decoder architecture, shown in Figure 1.1. Embeddings are extracted from the encoder's output, while the decoder reconstructs the original discretized representation used to train NeRFs, which are images. These embeddings serve as inputs for neural networks meant to tackle tasks such as classification and shape retrieval. Furthermore, this work demonstrated the ability of *nerf2vec* to create a smooth and well-organized latent space through embedding interpolation. This enabled the generation of novel 3D objects by interpolating the embeddings of two different input NeRFs. Moreover, leveraging *nerf2vec*'s embeddings, adversarial networks were trained, ultimately attaining the ability to generate credible and previously unseen NeRFs.

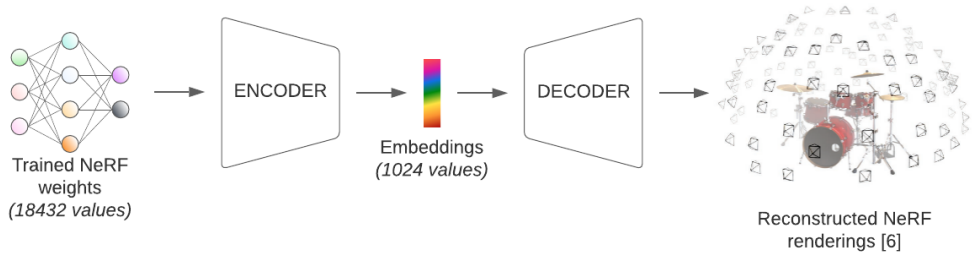

Figure 1.1: *nerf2vec* framework overview.

To help the reader understand how the thesis is structured, here there is a brief description of what each chapter entails:

- **Related work**: explore the advancements made in the research field, offering insights and foundations for the development of this work.

- **Methodology**: outline the steps employed for creating the dataset of NeRFs, and provides details about the creation and training of the *nerf2vec* framework.
- **Experiments and results**: provide a list of all conducted deep learning tasks and their corresponding results.
- **Conclusions and future work**: wrap up, highlighting the positive results and suggesting areas for further improvements.

Chapter 2

Related work

This chapter will delve into the pertinent related work and background knowledge upon which the current study is built. Special emphasis will be placed on classical techniques that currently underpin the representation of 3D scenes, as well as their applications within deep learning frameworks. Subsequently, the chapter will scrutinize the representations of scenes that serve as the foundation for this research, namely Implicit Neural Representations (INRs) and Neural Radiance Fields (NeRFs). These latter representations offer numerous advantages, but also present challenges when used as inputs for neural networks. It's worth noting these challenges has been addressed by the current work, and will be explained in-depth in the following chapter.

2.1 Deep Learning for 3D Scenes

In the realm of computer graphics and computer vision, 3D scene modeling and view synthesis hold immense significance. These techniques serve as foundational components in various applications, encompassing virtual reality, augmented reality, autonomous navigation, and the creation of 3D content. Their overarching objective involves the reconstruction of intricate 3D environments and the generation of fresh perspectives on scenes, facilitating immersive visualization and interaction within virtual spaces.

Unlike 2D images, which already have a standard way of being discretized as 2D grids of pixels, 3D scenes have still many different ways to be represented. Many of these representations have been also used as input of neural networks, allowing a vast variety of downstream tasks, further enhancing their utility and versatility.
The following paragraphs delve into the essential methodologies that have driven the advancement of 3D scene modeling and view synthesis, as well as the deep neural networks adept at effectively harnessing these techniques.

Voxel-based methods employ 3D grids or volumetric representations to model scenes. They discretize space into small cubic elements (voxels) and associate properties such as color, occupancy, or material information with each voxel. These voxels are an intuitive extension of 2D pixels in the 3D space, capable of maintaining an ordered and regular structure. Therefore, neural networks used to process 2D images can be easily adapted to work with this 3D representation, both for discriminative [16, 12, 65] and generative tasks [61, 17, 19, 30, 80]. Nevertheless, in order to work with 3D space, it is necessary to add a dimension to all the learning operators (e.g., convolution, pooling, etc.), which demands a huge increase of the amount of required memory and computational costs. This aspect typically forces to decrease the resolution of the 3D voxel grid, affecting the quality of the generated shapes. There exist an active area of research whose goal is to increase the quality of this representation, while keeping a reasonable amount of computational costs. As an example, FPNN [75] effectively tackles the problem related to the sparsity of the grid, which appears when the resolution is increased and that causes unnecessary computation. Furthermore, to reduce the dimensionality of the input data, other researchers have explored encoding the voxel grid using sparse and adaptive data structures like octrees [45, 14] and their variants [26, 55, 54].

Point clouds have gained significant attention in recent years as an expressive representation of 3D scenes. They consist of a collection of 3D points, each with a position in space. This heightened interest in point clouds is greatly facilitated by the widespread adoption of acquisition devices that directly output point clouds. The processing of point clouds is inherently complex due to their lack of organization, making it far from straightforward. Some studies involved the transformation of the initial point cloud data into intermediary regular grid structures, such as voxels or images [63, 27]. Others, such as PointNet [10] and its successor PointNet++ [11], successfully processed point clouds by coping their unorganized nature with maxpooling layers as symmetry functions. Additionally, many other works with the goal of finding new aggregating operators have been created, and they are mostly based on convolution [8, 76, 43, 66, 50, 28], graph [40] and attention [46].

Mesh-based methods leverage the underlying data structure used to model 3D objects and surfaces by interconnecting vertices, edges, and faces. According to the interconnection type, there are different kinds of representations:

- *Vertex-based*, which relies on defining the shape primarily through its vertices. Each vertex represents a point in 3D space and often contains information about its position and attributes. The connectivity between vertices is used to derive edges and faces. Neural networks using this representation as input capitalize on the presence of a structured domain to capture information regarding the neighborhoods of points, either by utilizing convolution or kernel functions [34, 20, 31, 77, 51, 33, 23, 21].

- *Edge-based*, that focuses on the edges that connect the vertices of a 3D mesh. These edges define the structural elements of the mesh and represent pairwise relationships between vertices. Edge-based representations are used to emphasize the connectivity between vertices. The

connectivity characteristic enables neural networks to handle this kind of shape representation through various methods, including the application of ordering-invariant convolution [59], navigating the shape's structure [25], or generating graphs based on input meshes [1].

- *Face-based*, which defines the mesh primarily by its faces, that are planar polygons covering the surface of the mesh. Each face is composed of vertices and edges, and they collectively describe the surface topology and appearance of the object. Neural networks applied to this representation operate by capturing and encoding information from adjacent faces [15, 78, 74, 2].

One notable limitation of meshes is their lack of topology-invariance, which can pose challenges for neural networks aiming to learn resilient mesh representations. Furthermore, manipulating meshes with large number of vertices and faces can be inherently difficult, due to the substantial amount of time and memory required, making them inefficient for representation purposes.

2.2 Implicit neural representations

As discussed in the preceding section, the absence of a universally accepted standard for discretizing 3D scenes has been a notable challenge in the field. Consequently, researchers have been diligently exploring novel representations with the potential to establish a new standard for representing 3D scenes. Among these emerging representations, Implicit Neural Representations (INRs) have shown great promise. INRs belong to a class of deep learning models specifically designed to represent complex, high-dimensional data, including images, 3D scenes, point clouds, and meshes. What sets INRs apart is their ability to capture such data without relying on explicit geometric or grid-based representations.

Expanding on the versatility of deep learning models, recent advances have demonstrated the remarkable capacity of Multi-Layer Perceptrons (MLPs) to implicitly represent various types of data. Notable examples include objects [38, 37, 29, 42, 44, 3], as well as scenes [73, 13, 67, 60]. Many of these works have focused on representing 3D data using MLPs, relying on the fitting of implicit functions tailored to specific data types. For instance, these functions encompass unsigned distance representations for point clouds and signed distance representations for meshes [42, 29, 3, 73, 13, 67]. For voxel grids, the focus has been on occupancy-based representations [39, 79].

Beyond shape representations, some of these models have been extended to encode object appearance, as evidenced in [64, 73, 49, 47]. Additionally, temporal information has been incorporated into certain models [48].

Among this diverse array of approaches, two notably stand out: Neural Radiance Field (NeRF) [6], and Sinusoidal Representation Network (SIREN) [72]. While SIREN leverages periodic activation functions to capture high-frequency details within input data, NeRF has played a pioneering role in the field of 3D scene representation by introducing the concept of modeling scenes as continuous 3D radiance fields. It has been influential in synthesizing novel views of scenes, enabling high-quality 3D reconstructions, and redefining how scene representation is approached. NeRF serves as an essential component of this study, and therefore more details on it are given in the upcoming section.

2.2.1 NeRFs

NeRFs [6] have depicted a significant advancement in the realm of synthesizing novel views of intricate scenes, characterized by complex geometry and appearance. Its significance primarily lies in its ability to mitigate the computational challenges associated with discretized voxel grids, all while retaining the capability to generate exceptionally high-resolution scenes.

NeRF works by optimizing an underlying volumetric scene function, using a sparse set of input views. This function is represented by a MLP, whose input is a 5D coordinate composed of a spatial location (x, y, z) and a viewing direction (θ, ϕ); and whose output is the volume density and view-dependent emitted radiance at that spatial location. The input is created by marching camera rays through the scene, and the output is used to synthesize novel 2D views through the usage of classic volume rendering techniques. Given the fact that this process is naturally differentiable, the whole system is optimized by leveraging the gradient descent, and minimizing the error between each observed image (i.e., the ground truth) and the corresponding views obtained at the end of the pipeline just described. Figure 2.1 shows the entire process.

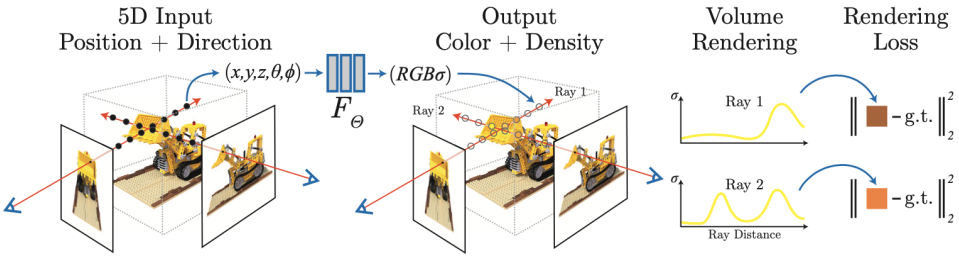

Figure 2.1: NeRF training pipeline [6].

In order to obtain the best results, two important novelties has been introduced: a fixed positional encoding of the input coordinates, which allows the MLP to represent high frequency functions, and a hierarchical sampling procedure, which permits to efficiently query this high-frequency scene representation.

While vanilla NeRF yielded remarkable results, it suffered from lengthy training times. As a result, subsequent research endeavors have led to the development of new methodologies for creating NeRFs that achieve significantly faster training times, while preserving quality. The following paragraphs report notable examples that also contribute to a better understanding of the presented work.

instant-ngp [68] introduces a noteworthy innovation known as *multiresolution hash encoding*. As for vanilla NeRFs, the encoding serves the purpose of mapping neural network inputs to a higher-dimensional space, a key factor in extracting high-quality approximations from compact models like the MLPs employed in training NeRFs. Essentially, it functions by mapping a series of grids to fixed-size arrays of feature vectors. Each array is treated as a hash table and indexed using a spatial hash function. Depending on the resolution, multiple grid points may map to the same array entry, resulting in hash collisions. During the training phase, these collisions lead to the averaging of training gradients, with greater weight assigned to the largest gradients, which ultimately dominate the loss function. This mechanism enables the model to autonomously learn how to prioritize sparse regions that contain the majority of intricate details.
It's worth noting that the adoption of the *multiresolution hash encoding* entails the learning of not only the parameters of the MLPs, but also those within the hash tables. However, despite the increased memory requirements, the additional computational load introduced during the training stage remains minimal. Furthermore, the access time to the hash tables is very efficient, with $O(1)$ time-complexity.
Another significant contribution introduced by *instant-ngp* is the utilization of an *occupancy grid*. This grid serves to provide a coarse mapping of empty and non-empty space, leading to improved efficiency in the volume rendering process.

NeRF Acceleration (*NerfAcc*) [62] successfully follows in the footsteps of *instant-ngp*, with the former designed to address certain challenges posed by the latter. These challenges include the reliance on specialized CUDA implementations, and the need for high customization tailored to bounded static

scenes. In particular, *NerfAcc* focuses on efficient volume rendering of radiance fields, while supporting also dynamic and unbounded scenes. The volume rendering process is optimized by splitting it in two steps:

1. *Ray marching*, which involves the procedure of projecting a ray through the scene, and producing discrete samples along the trajectory of that ray. In order to optimize this step, the goal is to reduce as much as possible the number of these samples. In particular, those samples that occupy empty or occluded regions should be skipped, because they do not contribute to the creation of the final rendering. Following the idea of *instant-ngp*, an *occupancy grid* has been deployed to cache a binary grid which is updated during the training, and whose goal is to map regions that are empty or occluded. For understanding which part of the space are empty or occluded, the density of the samples must be computed along the ray, so as to compute transmittance. However, during this phase, the computation of the gradients is disabled, greatly increasing the efficiency.

2. *Differentiable rendering*, that is responsible of accumulating the color of the samples along the rays, and to convert them into pixel colors. During this phase, the computation of the gradients is enabled again, and the training is supervised with the original RGB values of the pixels coming from the ground truth.

Moreover, *NerfAcc* achieves the support for *unbounded scenes* by applying a non-linear function to the coordinates of the occupancy grid. This function maps the unbounded space into a finite grid. Finally, *dynamic scenes* are facilitated by introducing a new term derived from the ground truth. Specifically, this term comprises the timestamp associated with a particular input sample. Subsequently, it is incorporated as supplementary information provided to the network during the training process.

2.3 Deep Learning on INRs

The previous section emphasized the significance of INRs in accurately and efficiently representing and synthesizing 3D scenes. Consequently, this has given rise to a highly active research domain. Its primary objective is to explore the feasibility of employing INRs, namely neural networks, as direct inputs for other neural networks to address a wide range of downstream deep learning tasks.

In the earliest works that attempted the above mentioned approach, neural networks were regarded simply as algorithms, with a primary focus on predicting certain properties, such as accuracy. Several examples align with this approach. For instance, [70] endeavors to forecast classification accuracy by directly inputting MLP weights into the model. Meanwhile, [36] employs a self-supervised learning approach to predict various characteristics of the input classifier. In a different vein, [9, 24, 32] employs a Graph Neural Network (GNN) to process neural networks represented as computational graphs, which allows it to predict optimal parameters or generate adversarial examples.
However, as previously discussed, the objective is to treat the input INRs as 3D shapes, which have specific features and properties, and only recently have emerged novel approaches for achieving this. The following paragraphs will provide examples of these novel approaches.

INRs that are to be thought of as data (*functa*) [22] represents the first instance in which the processes of dataset creation and its subsequent utilization as input for neural networks in deep learning tasks have been decoupled. An intriguing aspect of this work is the introduction of *modulations*, which are learnable parameters unique to each individual data point. *Modulations* allow the network to learn the variations of each data point, while a shared network learns the overarching structure common to them all. The decoupling

of dataset creation and utilization has a number of benefits. For example, it allows for the creation of datasets that are more efficient to train on, and it makes it easier to transfer knowledge between different datasets. Nevertheless, this work also presents an important downside delineated by the necessity of a shared MLP that must be used to fit new shapes. This aspect significantly undermines the deployability of *functa* in the wild, favoring other works that fit an individual network for each shape, such as [41].

Deep Weight Space Network (*DWSNet*) [5] has been designed with the goal of finding the best architecture able to effectively learn and process neural models that are represented as sequences of weights and biases, such as MLPs. Specifically, *DWSNet* is based on the symmetry property that characterizes the weights of MLPs [57]. This property enables the processing of input MLPs with only three fundamental operations: broadcasting, pooling, or standard linear layers, substantially reducing the number of parameters compared to previous approaches.

inr2vec [41] is a significant precursor to the current work, providing an efficient framework for creating compact representations of INRs. *inr2vec* has demonstrated its versatility by encoding various input shapes, including point clouds, triangle meshes, and voxel grids. These encodings have proven successful in a range of deep learning tasks, such as classification, retrieval, segmentation, and generation. Unlike *functa*, *inr2vec* utilizes individually trained INRs, a strategy that has exhibited superior effectiveness in capturing the underlying signal and offers enhanced practical deployability. It's worth highlighting that *inr2vec* plays a foundational role in shaping this research, emphasizing its pivotal contribution to advancing deep learning applications by enabling compact representations and effective utilization of input INRs.

Chapter 3

Methodology

This chapter will build upon the concepts previously introduced, elucidating the central objectives of the current work. In particular, it will provide an in-depth exploration of the methodology employed to construct the dataset of NeRFs. Furthermore, it will exhibit the innovative architectural framework devised within this research, referred to as *nerf2vec*, which serves as a pivotal component in the encoding and processing of NeRFs.

3.1 Dataset

The initial challenge encountered during the development of this project is a commonplace issue within the realm of artificial intelligence: the absence of an appropriate dataset. Therefore, the first step was indeed to create a suitable dataset of NeRFs. To generate it, a set of 2D images was required for each 3D object that needed to be implicitly represented through NeRFs. To address this requirement, it was leveraged a rendering dataset originally created for *Deep Implicit Surface Network* [56], which will be hereafter referred to as *ShapeNetRenders*. It was generated by extracting 3D shapes from *ShapeNet* [4], a comprehensive and extensive resource tailored for 3D shape understanding and computer vision research. Table 3.1 contains a summary of the items contained in *ShapeNetRenders*, and it reveals two significant observations.

CLASS	FREQUENCY
Airplane	4045
Bench	1813
Cabinet	1571
Cars	3514
Chair	6777
Display	1093
Lamp	2292
Speaker	1597
Rifle	2369
Sofa	3173
Table	8433
Phone	1050
Watercraft	1939

Table 3.1: *ShapeNetRenders* class frequencies.

Firstly, the total count of 3D objects is not particularly large, totaling 33296 elements. Secondly, there is a noticeable imbalance among the classes, which has the potential to pose challenges in various deep learning tasks, such as classification. Nevertheless, only the first problem was addressed in this work, while the second could be considered for future extensions.

In order to expand the dataset of NeRFs and include a greater number of elements, offline data augmentation techniques were applied. Specifically, the size of the dataset was tripled by creating two augmented versions of each object from *ShapeNetRenders*, achieved through direct manipulating the corresponding 3D shapes found in *ShapeNet*. In creating these augmented versions, the shapes were subjected to random deformations while preserving their overall structure. Additionally, a random color was assigned to each component of these augmented shapes, for further increasing diversity between each object. Figure 3.1 depicts some of the created augmented samples.

In the end, a total of 36 renderings were generated for each 3D object, each featuring a distinct camera pose. Specifically, the camera poses were adjusted

Figure 3.1: Data augmentation samples. Column (a) contains the original image taken for *ShapeNetRenders*. Columns (b) and (c) contain the augmented versions.

to complete a full rotation around the object's horizontal axis, while introducing random variations in the vertical axis and camera-to-object distance. The ability to rotate around the object was made possible by the unique characteristic of the *ShapeNet* dataset, where each 3D object has its axes aligned, providing a consistent reference position for rotation. Moreover, it's important to highlight that these renderings only captured the upper portion of the object, aligning with the original *ShapeNetRenders* dataset. Furthermore, it's worth noting that *ShapeNetRenders* initially included 36 training images and 36 test images, but the latter were excluded due to their lower quality and accuracy.

As a final note, it's important to mention that the dataset was divided into three distinct subsets: the training set, the validation set, and the test set. This division aligns with common practices frequently engaged in deep learning research. The training set is employed for updating the model weights, the validation set is used to evaluate model performance on unseen data during training, and the test set is exclusively reserved for assessing the final quality of the model, without any involvement in the training process.

3.2 NerfAcc adaptations

Once the dataset of renderings explained in the previous section was ready, the subsequent stage of this project was to create the actual database of NeRFs, and *NerfAcc* [62] served as the chosen framework for constructing it. However, it is crucial to note that this work seeks to expand upon the previous *inr2vec* research [41], which introduced specific constraints that must be adhered to. These constraints necessitated modifications to the original *NerfAcc* implementation, encompassing some aspects explained in the following paragraphs.

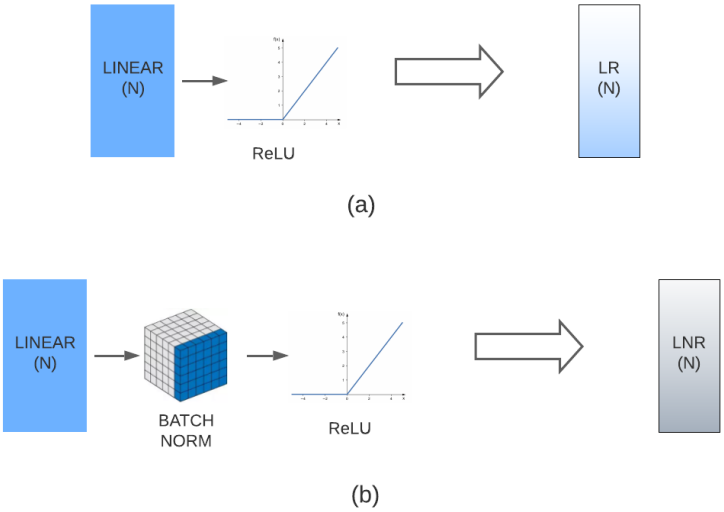

Figure 3.2: Building blocks used to increase readability of neural architecture representations. (a) depicts the *LR block*, consisting of a sequence of two operations: a **L**inear transformation followed by a **R**eLU activation. (b) is an extension of (a), and it includes an additional computation, namely batch **N**ormalization, placed between the linear transformation and ReLU activation. In all these blocks, N represents the number of hidden units contained in the linear layer.

Note that from this point on, specific building blocks will be introduced and utilized to streamline the representation of neural network architectures. Through the utilization of these predefined blocks, the aim is to enhance the clarity and conciseness of neural network diagrams while ensuring consistency throughout the rest of the thesis. Figure 3.2 contains a summary and an explanation of these blocks.

A significant modification was related to the input encoding. It was changed from *multiresolution hash encoding* [68] to *frequency encoding* [6]. This change was necessary because the *inr2vec*'s encoder was originally designed to accept as input only the weights of MLPs, whereas the *multiresolution hash encoding* would require additional parameters to be inputted and learned during training. With the selected encoding method, the dimension of each 3D coordinate provided as input to the network was expanded according to the

following formula:

$$\gamma(p) = \sin(2^0\pi p), \cos(2^0\pi p), ..., \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p) \tag{3.1}$$

where p is the input coordinate, and L decides the dimensionality of the encoding, that was set to 24 for the present study.

Additionally, certain hyperparameters were adjusted to enhance performance and quality. In particular, the axis-aligned bounding box size containing each 3D object was changed from 1.5 to 0.7. Regarding the background color used during the training phase, it was set to random, following the approach detailed in [68].

The next step involved revising the architecture originally used to train NeRFs, with two key changes deployed to respect *inr2vec*'s constraints. These changes also aimed to reduce the complexity of the experiments on NeRFs employed in this thesis, leaving room for potential future enhancements.
The first modification entailed removing one of the two output layers commonly utilized in standard NeRF training architectures [6, 68], while ensuring the preservation of high-quality results.
The second change reduced the dimension of the input used for training NeRFs, which typically is 5D. More specifically, usually the input is composed of spatial locations (x, y, z) and viewing directions (θ, ϕ), and the latter were eventually omitted from those inputs.
Figure 3.3 depicts the final architecture designed to learn NeRFs. It's worth noting that this architecture is based on *FullyFusedMLP*, a highly optimized and exceptionally efficient MLP introduced by NVIDIA [69].

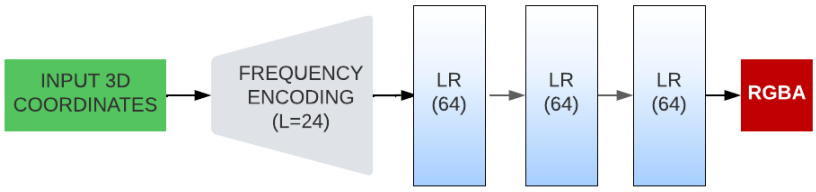

Figure 3.3: Visualization of the fully-connected architecture employed for training NeRFs. All the intermediate layers are *LR blocks*. The frequency encoding parameter L is set to 24, as defined in Equation 3.1. The output is then processed by passing the *RGB* component, namely the color, through a Sigmoid activation function; while the A component, namely the density, is passed through an exponential activation function specifically tailored in [62].

3.3 Learning NeRFs embedded representation

The literature has demonstrated that INRs are a highly promising approach for representing 3D shapes, and it also has explained how to use them within deep learning pipelines. Notably, *inr2vec* [41] stands out as one of the most successful examples of solving several deep learning tasks using directly INRs as inputs. Yet, it's important to note that *inr2vec* did not address NeRFs. Indeed, the goal of current work is to extend the capabilities of *inr2vec*, and to explore the feasibility of directly applying deep learning techniques to NeRFs, which intrinsically encode 3D shapes.

3.3.1 nerf2vec

The framework developed in this work is called *nerf2vec*, and its architecture is based on an encoder-decoder structure, summarized in Figure 1.1.

The encoder, detailed in Figure 3.4, takes the weights of a NeRF as input and produces its embeddings, namely a compressed representation of the original input. How to input the weights into the encoder was a critical first step,

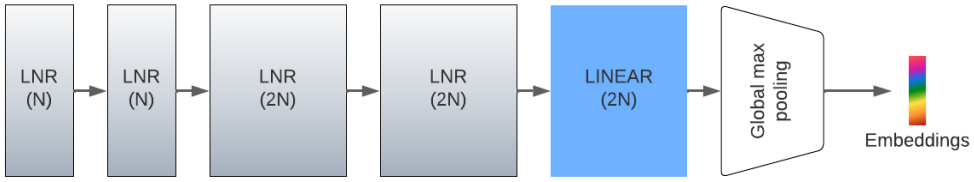

Figure 3.4: Illustration of the whole encoder's architecture, detailing how the embeddings are created.

and it was achieved by following standard practices [72]. The employed techniques involved stacking the weights of each layer of a NeRF to create a bidimensional matrix that can be readily processed by the encoder. Nonetheless, as previously discussed, each NeRF was trained using the *FullyFusedMLP*, which has unique characteristics that need to be taken into account. Consequently, it was essential to consider the following aspects in order to properly prepare the encoder's input:

- every layer within a *FullyFusedMLP* lacks of the bias term;
- the input and all the hidden layers have the same fixed feature dimension H;
- any input/output dimension is automatically padded to the nearest multiple of 16 to optimize both performance and efficiency. For example, the output layer is automatically padded from dimension 4 (i.e., RGBA) to 16. This padding is afterward ignored internally while performing any computation involving it.

This information permitted to manipulate the weights of each layer of a MLP for creating the required bidimensional matrix by applying two subsequent operations. The first was to add 0-padding to all those layers whose dimensions were not a multiple of 16. In the current work, this occurred only for the output layer, which received a padding of dimension $(H - 16) * H$, where H represents the feature dimension. Once the padding was added, it became possible to create W, which is a single tensor with all the layers' weights stacked

together [72]. This preparation allowed for the proper execution of the second operation, which involved reshaping W into a bidimensional matrix. This reshape was carried out by fixing its second dimension at feature dimension H. Figure 3.5 provides a visual summary of this procedure.

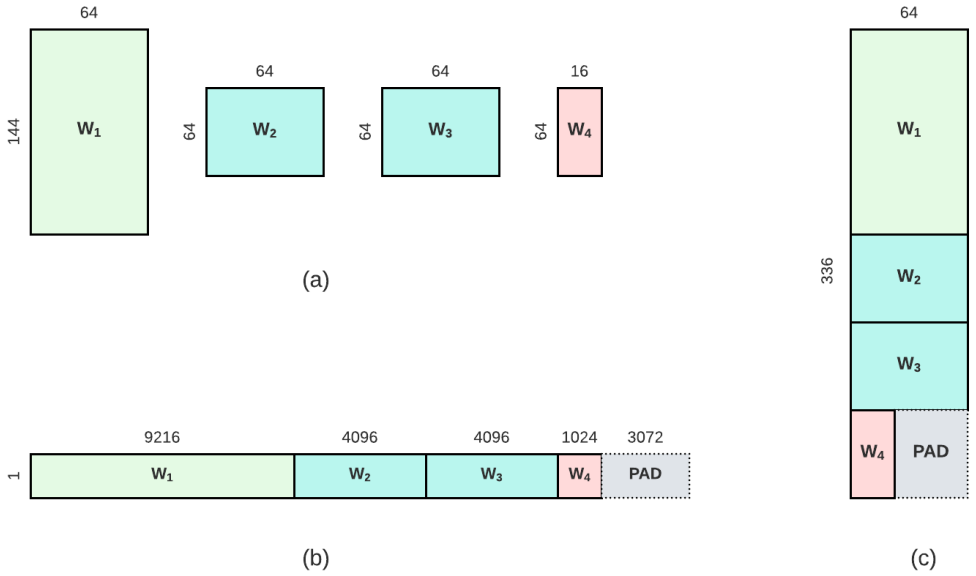

Figure 3.5: Encoder's input vector creation procedure. (a) shows the weight matrices W_x of the layers that compose the MLP used to train a NeRF, as depicted in Figure 3.3, where x identify a specific layer. (b) shows the flattened version of the weights, obtained by concatenating each flattened W_x. Furthermore, padding *PAD* has been incorporated to ensure proper weight distribution during the application of the final reshape operation, which is necessary to obtain the bidimensional matrix that ultimately expresses the encoder's input vector (c). This reshaping process sets the second dimension to a fixed value of 64, representing the number of features.

The encoder's primary goal is to construct a well-structured latent space with strategically positioned embeddings for subsequent deep learning tasks. It's crucial to note that during the training of NeRFs, various random elements are at play, including the initialization of weights and data shuffling. This randomness can result in a completely disordered latent space, scattering weights that should ideally occupy similar positions across different areas.

The resulting disorganization in the latent space presents a significant challenge, rendering it ineffective for applying deep learning tasks. Therefore, a critical step to address this issue was to ensure that all NeRFs were trained with the same randomly generated initialization vector. This vector was used to initialize the weights for every NeRF employed in the dataset creation. As demonstrated in *inr2vec*, this simple yet highly effective practice promotes weight alignment across various NeRFs, facilitating the convergence of the proposed framework.

The second part of the architecture comprises an implicit decoder, which draws inspiration from [29]. It earns the label *implicitly* by deviating from the conventional auto-encoder paradigm, which typically would aim to replicate the encoder's input (i.e., NeRF weights). Instead, this decoder operates under supervision to reconstruct the underlying function approximated by the input NeRF, with its ultimate objective being the prediction of RGBA values for specific 3D coordinates. To accomplish this, the decoder takes as input the embeddings generated by the encoder, along with the 3D coordinates, in order to predict the associated RGBA values. It's important to note that these 3D coordinates are treated in the same manner as explained in Section 3.2, involving an increase in dimensionality through the use of *frequency encoding*. Figure 3.6 details how the decoder's input is computed, while Figure 3.7 shows the architecture of the employed decoder.
A crucial technical aspect pertains to how the embeddings and 3D coordinates are transmitted to the decoder, ultimately leading to the prediction of RGBA values. In the training procedure employed for *NerfAcc*, the number of 3D samples is notably large and it also grows during the training procedure, thanks to the usage of occupancy grids able to filter out those coordinates already recognized as background. However, as detailed in Section 3.3.2, *nerf2vec* did not use occupancy grids in the same way as *NerfAcc* did, and therefore different strategies were required. Additionally, *nerf2vec* was trained to learn

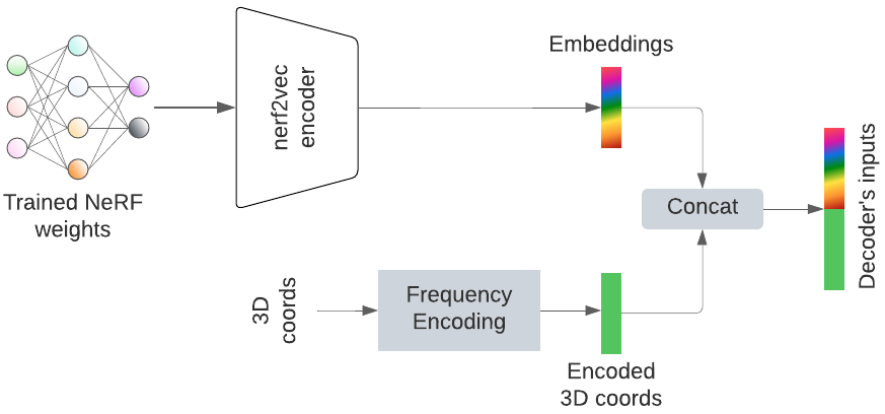

Figure 3.6: A visual representation of the decoder's input computation process. Specifically, the embeddings obtained by the encoder are concatenated with the 3D coordinates, which have been previously encoded by means of the frequency encoding.

multiple NeRFs simultaneously, and therefore it was essential to set a limit on the number of 3D coordinates used as input for the decoder. This limit was determined by considering the capabilities of the machine on which the framework was trained, detailed in Appendix B, and with the aim of minimizing training time. Specifically, the limit for foreground coordinates was set to $25k$, and $10k$ for the background.

Nonetheless, during the ray marching's operation (see Section 2.2.1), two opposite scenarios can occur. In the first, the number of foreground coordinates exceeds the limit of $25k$, which was managed randomly selecting 3D points until the limit is met. In the second, the count falls short of these coordinates, and it was addressed by adding additional background coordinates as padding. Intuitively, this is the most promising approach since NeRFs lacking of foreground coordinates are typically associated with smaller and more compact 3D objects, and adding more background coordinates should facilitate faster learning for these particular objects.

On the other hand, the $10k$ limit imposed for the background coordinates was consistently met with ease. As discussed previously, this is because the vast majority of the scenes consisted of background coordinates.

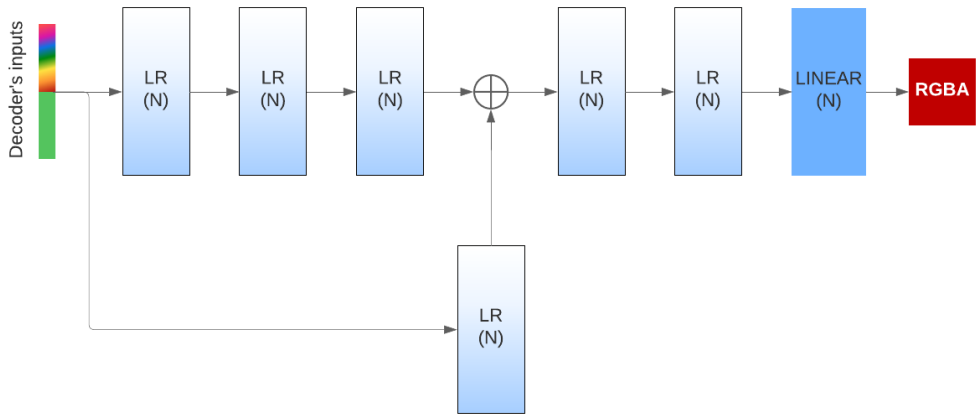

Figure 3.7: Presentation of the whole decoder's architecture, showing how the final RGBA predictions are computed. Refer to Figure 3.6 to see the details about how the decoder's inputs are created.

After training, embeddings obtained by the frozen encoder can be utilized as inputs for any neural architecture to tackle various deep learning tasks, such as classification and retrieval. Meanwhile, the frozen decoder can be employed for reconstructing the discrete representation of embeddings, namely 2D renderings. Therefore, it is well-suited for tasks like assessing the quality of learned NeRFs and generating new 3D object views through the interpolation of embeddings.

3.3.2 Occupancy grids

As referenced earlier, NerfAcc utilizes occupancy grids to significantly reduce the training time required for NeRFs. However, it's worth mentioning that these grids introduce additional parameters that must be learned, which increase the overall complexity of the training process. In contrast, *nerf2vec* aims for a simpler training procedure focused solely on learning the underlying function approximated by NeRFs, without introducing more overhead. As a result, occupancy grids were excluded from the learning pipeline. Yet, obtaining pre-trained occupancy grids for each NeRF belonging to the dataset is a straightforward and trivial operation [62]. Thus, the information they contain was still exploited for the following two reasons:

1. provide an active support to the sampling of 3D points from the ground truth data, which are then used to create the training batches. The goal was to have approximately 40% of the sampled points in the scene's background, and the remaining 60% in the foreground. Given that background coordinates predominate in the scene, this approach promotes a more balanced learning. These proportions were empirically determined for optimal scene representation and reconstruction;

2. increase efficiency of the ray marching algorithm, which is responsible to return only 3D coordinates belonging to the 3D object (i.e., foreground coordinates). This approach, as demonstrated in the *NerfAcc* implementation, ensures smooth operation of the ray marching algorithm without the need for *nerf2vec* to learn additional parameters.

3.3.3 Loss function

Another essential aspect regards the loss function used to train *nerf2vec*. *NerfAcc* natively computes the loss on a batch of RGB values obtained by randomly selecting N pixel values from all the training images, where N is a hyperparameter. The following formula formalizes how the *NerfAcc*'s loss is calculated:

$$loss = L1_{loss}(rgb_{gt}, rgb_{pred}) \tag{3.2}$$

where $L1_{loss}$ is the L1 loss computed between rgb_{gt},namely the RGB values derived from the ground truth, and rgb_{pred}, which are the predicted RGB values inferred by the MLP's output.

The *nerf2vec*'s loss can be computed by adapting the *NerfAcc*'s loss to consider an additional dimension, which represents the number of NeRFs trained simultaneously, namely those contained in each training batch. However, there was a critical concern to address. As already stated, occupancy grids were removed from the learning pipeline, and therefore *nerf2vec* was exposed

to an unacceptable limitation: the incapability to learn anything about the background, and this was fundamentally detrimental to the process of NeRF learning. For these reasons, the background coordinates were forcefully added to the *differentiable rendering* process explained in Section 2.2.1. This addition enabled predictions for background coordinates to be factored into the loss through a newly introduced term. Furthermore, knowing that the number of background coordinates significantly exceeded the number of foreground coordinates, the two terms in the loss were weighted differently to optimize the learning process. Equation 3.3 shows how the new loss was eventually computed.

$$\begin{aligned} fg_{loss} &= L1_{loss}(fg\_rgb_{gt}, fg\_rgb_{pred}) * fg_{weight} \\ bg_{loss} &= L1_{loss}(bg\_rgb_{gt}, bg\_rgb_{pred}) * bg_{weight} \\ total_{loss} &= fg_{loss} + bg_{loss} \end{aligned} \tag{3.3}$$

where fg means *foreground*, while bg stands for background. fg_{weight} and bg_{weight} were set to 0.8 and 0.2, respectively. These values were determined through a *grid search* using a subset of $10k$ elements from the dataset of NeRFs, and were selected based on the best results achieved. $total_{loss}$ was the loss ultimately used for updating the gradients.

3.3.4 Evaluation metrics

The Peak Signal-to-Noise Ratio, often abbreviated as *PSNR*, is a fundamental and widely adopted metric for quantifying the fidelity and quality of reconstructed or compressed signals, such as digital images and videos. *PSNR* measures the similarity between a reference (original) signal and a reconstructed or processed signal by assessing their pixel-wise differences. It is expressed in decibels (dB) and provides a standardized and intuitive way to evaluate the level of signal degradation or distortion, with higher *PSNR* values indicating a higher degree of fidelity and lower signal distortion.

In the specific context of this study, *PSNR* served as a critical evaluation metric for assessing the reconstruction quality of 3D scene renderings produced by the *nerf2vec* framework. These renderings were generated by performing inference on the trained *nerf2vec* model, utilizing known camera poses extracted from the dataset, which also contained ground truth renderings. The comparison of *nerf2vec*-generated renderings to these ground truth images using PSNR offered a quantitative measure of the reconstruction quality, helping to gauge the faithfulness of *nerf2vec*'s ability to model and render 3D scenes. From a formal point of view, PSNR is defined as follows:

$$\begin{aligned} MSE &= \frac{1}{n}\sum_{i=1}^{n} \|RGB_{pred}[i] - RGB_{gt}[i]\|^2 \\ PSNR &= -10\log_e\left(\frac{MSE}{10}\right) \end{aligned} \tag{3.4}$$

where MSE is the mean squared error. RGB_{pred} and RGB_{gt} are respectively the predicted and ground truth RGB images, and n indicates the number of pixels.

3.3.5 Ground truth retrieval

Lastly, in this section, the critical aspect of meticulously executing ground truth data retrieval to ensure the smoothest possible *nerf2vec* training is addressed.

In the initial development stage, a conventional approach was employed. In particular, the ground truth was directly acquired from the 2D renderings initially utilized as the training data for NeRFs. Nevertheless, this approach proved to be inefficient due to the significant number of I/O operations required. Each NeRF was trained on 36 different renderings, and multiple NeRFs were trained simultaneously, resulting in unacceptable I/O overloads. This bottleneck led to a substantial delay in generating the batches necessary to

train *nerf2vec*, also threatening its scalability.

As a result, a new procedure was thoroughly studied and subsequently implemented. This approach involved in obtaining the ground truth directly from the trained NeRFs, rather than retrieving it from original renderings. Utilizing the high optimization and efficiency offered by the *FullyFusedMLP*, as illustrated in Section 3.2, this method considerably improved the speed and smoothness of ground truth retrieval when compared to the previously outlined conventional approach.

Chapter 4

Experiments and results

Upon completing the training of *nerf2vec*, a series of tasks were undertaken to address the central question of this thesis about the possibility of using NeRFs as input and output data for other algorithms, possibly involving other neural networks. These tasks encompassed quality reconstruction assessment, classification, embedding interpolation, and shape retrieval, each of which is exhaustively examined in the subsequent sections.

4.1 Reconstruction quality

One of the key aspects of *nerf2vec* involves learning embeddings for the input NeRFs, which act as a compressed representation of these NeRFs. PSNR, as introduced in section 3.3.4, was employed to assess the reconstruction quality of *nerf2vec* during training. Specifically, PSNR helped to evaluate the differences between the original input signal and the signal reconstructed by the decoder, which utilized only embeddings and 3D coordinates as inputs. Therefore, this metric was used to assess both the framework's quality and the embeddings themselves. Thus, significant effort was devoted to fine-tuning the many hyperparameters integrated into the framework with the aim of achieving the highest possible PSNR values. What follows is a concise summary of the main hyperparameters that were chosen based on their ability to yield the

highest reconstruction quality, as measured by the PSNR:

- Number of epochs: 400
- Learning rate:
  - First 400 epochs: Learning rate 10^{-4}
  - Last 100 epochs: Learning rate 10^{-5}
- Batch size: 16
- Optimizer: Adam
- Weight decay: 10^{-2}
- Encoder linear layer units (see Figure 3.4):
  - First two: 512
  - All others: 1024
- Embeddings size: 1024
- Decoder linear layer dimensions (see Figure 3.7): 1024

While the embeddings are significantly more compact compared to the original NeRF weights, the shape reconstructions achieved with *nerf2vec* closely resemble the original ground truth, preserving a good level of detail. In particular, the PSNR reached approximately 30 for the training set and approximately 25 for the validation set. Figure 4.1 contains examples showing the quality of these reconstructions.

4.2 Classification

As it is often the case, the very first real deep learning task applied to *nerf2vec* involved shape classification. In this context, the embeddings generated by the *nerf2vec*'s encoder, with its weights frozen, were passed through a new

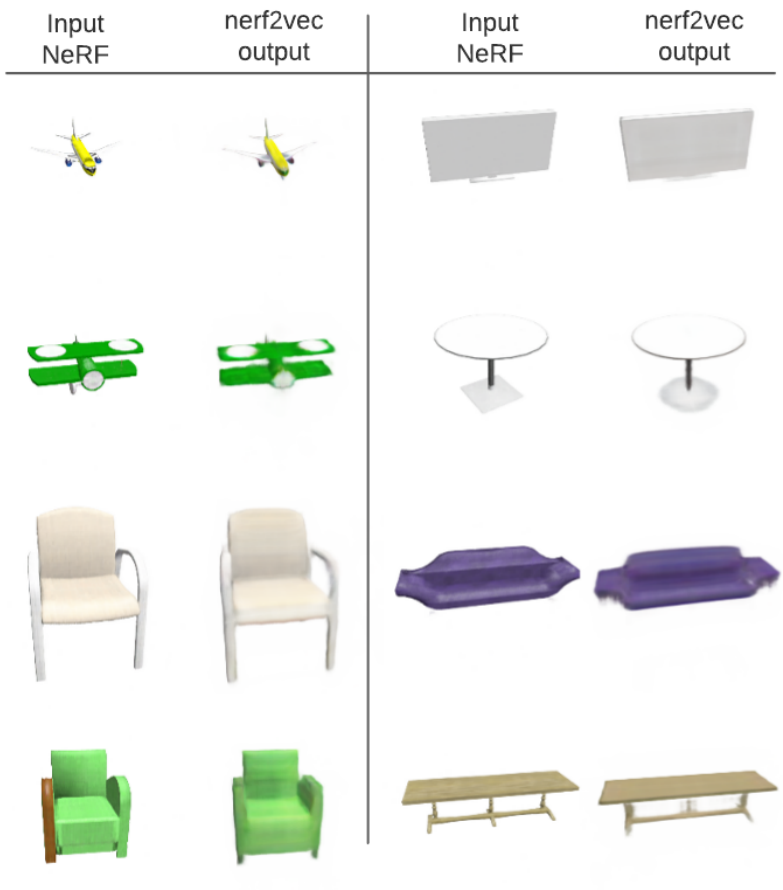

Figure 4.1: *nerf2vec* reconstruction examples.

neural network, namely the classifier itself. The utilized architecture is relatively straightforward, and an illustration of it can be found in Figure 4.2. It underwent training on the complete dataset described in Section 3.1, employing the standard *cross-entropy loss* during the training process. The training was completed after 150 epochs.

To better evaluate this classifier's performance, two additional baseline classifiers were trained for comparison and assessment.

The first baseline classifier leveraged the same architecture shown in Figure 4.2. However, differently from the architecture in the figure, this classifier takes as inputs the same vectors used in training *nerf2vec*. Thus, the goal of this classifier was to directly classify NeRF weights.

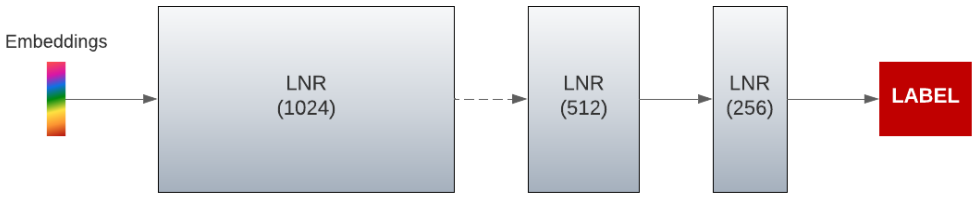

Figure 4.2: Classifier architecture overview. The input is represented by the *nerf2vec*'s encoder output, while the final label is determined through softmax applied to the last LNR block's output logits. The dashed line identify the *dropout* operation [6].

Conversely, the second classifier was based on a different dataset composed of images, as opposed to NeRF weights or embeddings. These images were generated by performing inferences on each NeRF. To be precise, a set of N renderings for each NeRF were created. More specifically, N was set to 1 and 9, representing a single-view and a multi-view classifier, respectively. In the latter case, the renderings were created by encompassing a 360-degree rotation around each 3D object, following the same approach explained in Section 3.1. Furthermore, the ultimate label for a given input is determined through a voting system that takes into account all the N renderings of each object, selecting as the predicted class the one with the highest frequency count. Regardless from the chosen value of N, this classifier utilized the *ResNet50* architecture [35], a widely recognized and standard framework for image classification. The tests were carried out with both training the model from scratch, but also starting from weights obtained by pre-training it on ImageNet [52].

The results in Table 4.1 underscore that the classifier directly leveraging *nerf2vec*'s embeddings achieves comparable scores to the baselines. Additionally, while the pre-trained version of *ResNet50* attains higher accuracy, it incurs significantly longer processing times due to the rendering process, making it unsuitable for real-time applications. Table 4.2 provides further insights into classification time, highlighting the remarkable efficiency of *nerf2vec* in

Classifier	Accuracy	Classification time	N. views
nerf2vec	87.28%	1 ms	-
NeRF weights	84.53%	0.9 ms	-
ResNet50-v1 (P*)	94.12%	11 ms	1
ResNet50-v2	86.88%	11 ms	1
ResNet50-v3	93.28%	100 ms	9

Table 4.1: Accuracy results for each employed classifier. The *nerf2vec* classifier works on the embeddings produced by *nerf2vec*, while the *NeRF weights* classifier directly classifies NeRF parameters. In contrast, the *ResNet50 classifiers* take images as input, with (P*) indicating pre-training on ImageNet. Only for these latter classifiers, it is also reported the number of different views portraying each classified NeRF. Finally, the table also includes details on the average classification time, which comprises encoding and inference times, the two operations necessary for labeling.

Method	Encoding (ms)	Inference (ms)
nerf2vec	0.75 ± 0.0678	0.28 ± 0.0000161
Baseline	4.63 ± 0.0213	6.21 ± 0.0748

Table 4.2: Encoding and inference timings of classifiers. All the values reported in the table are in milliseconds (ms) and represent mean $\pm$ standard deviation.

these operations. In particular, the table provides details about the time requirements for essential classification tasks, namely encoding and inference. This timing data holds pivotal significance in comparing the two previously referenced methods adopted to implement the classifiers: the baseline, which leverages the *ResNet50* backbone, and the *nerf2vec*, based on the framework developed for this work. In the baseline method, *NerfAcc* is engaged to encode input NeRFs into renderings, followed by the labeling process using the *ResNet50* architecture. To facilitate clarity, the baseline's timings reported in the table refer to a single rendering per NeRF. On the other hand, the *nerf2vec* approach relies on the *nerf2vec* framework, specifically designed for generating embeddings from NeRFs. Subsequently, the labeling process in this method incorporates the architecture depicted in Figure 4.2. Once again, these outcomes solidify the credibility of *nerf2vec*'s embeddings, showcasing their effectiveness and efficiency in representing NeRFs.

4.3 Embeddings interpolation

Another intriguing task that aided in evaluating the quality of the embeddings generated by *nerf2vec* is the interpolation between different embeddings. Essentially, the goal of this task is to demonstrate that getting two random embeddings and performing a linear interpolation between them results in a completely new embedding, from which it is possible to create novel views of a plausible model leveraging the decoder's capabilities. The following equation details of how the interpolation was computed:

$$embeddings_{AB} = (1 - \gamma) * embeddings_A + \gamma * embeddings_B \quad (4.1)$$

where $\gamma \in [0.1, 0.9]$, $embeddings_A$ and $embeddings_B$ are the embeddings of two randomly sampled NeRFs. Note that the lower the value of γ is, the more similar the final interpolation will be to $embeddings_A$. Conversely, when γ is very high, the result becomes more similar to $embeddings_B$. Thus, these interpolations were more appreciated when the γ values fell within the middle of the supported range.

The results in Figure 4.3 confirm the meaningfulness of the latent space learned by *nerf2vec*, showing smooth color and shape changes during the interpolation process. Further affirmation of this meaningfulness is evident in Figure 4.4, highlighting the remarkable superiority in interpolating *nerf2vec*'s embeddings over directly interpolating NeRF weights. Thus, this process demonstrates its effectiveness in generating new radiance fields in the form of *nerf2vec*'s embeddings.

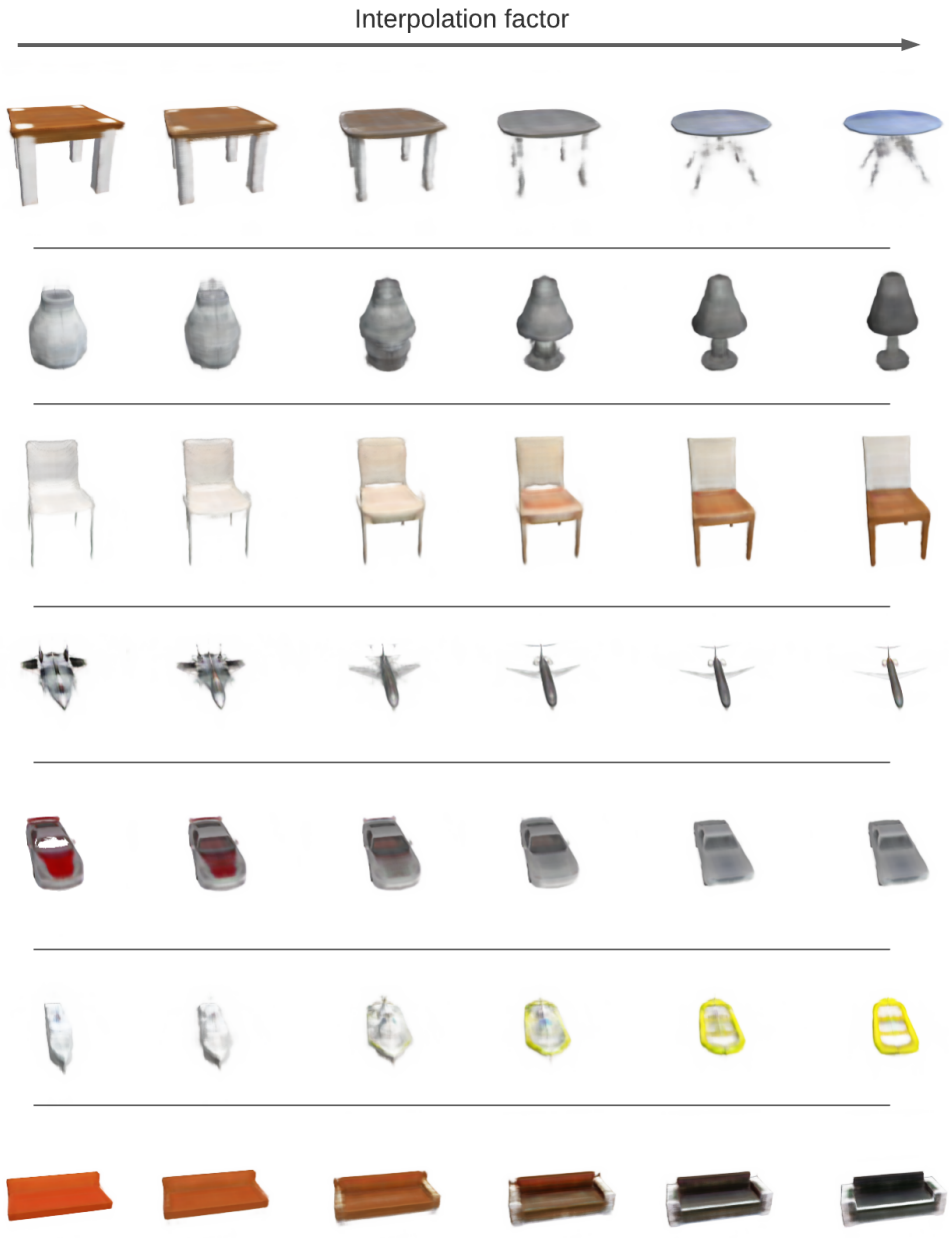

Figure 4.3: Examples of new shapes obtained by gradually interpolating the embeddings of 2 randomly sampled NeRFs.

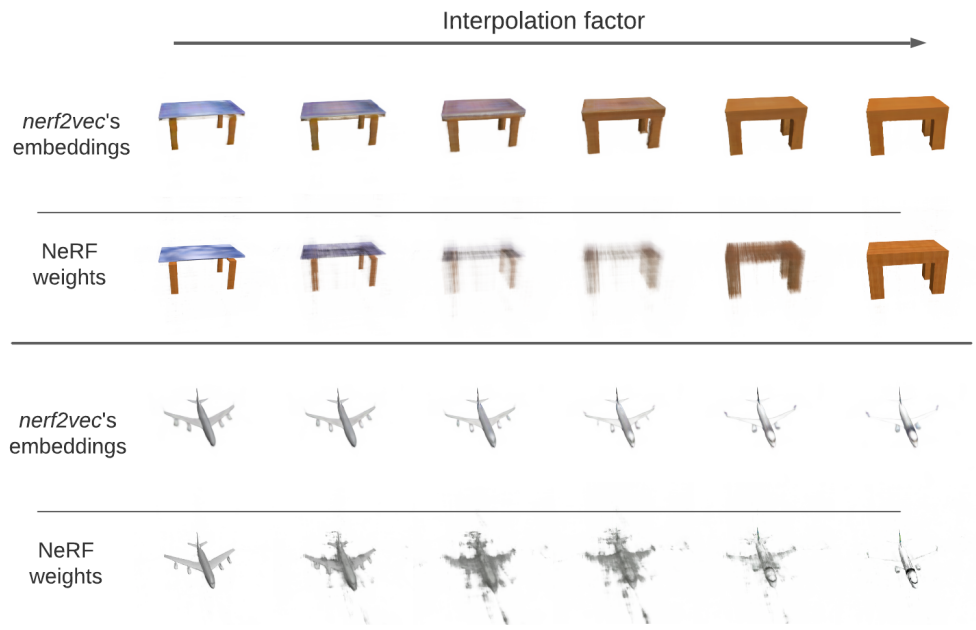

Figure 4.4: Comparison of the results obtained by interpolating NeRF weights and the embeddings created through *nerf2vec*.

4.4 Shape retrieval

Shape retrieval, typically associated with representation learning, involves extracting the nearest neighbors of a given input shape from a provided dataset of shapes. This task was adapted and applied to the current work by considering embeddings rather than shapes, serving as another method to evaluate the quality of the latent space produced by the proposed framework. More specifically, neighbors of the embeddings of a given input NeRF were extracted from the latent space, which contains all the NeRFs' embeddings included in the considered dataset.

The method employed is inspired by [4], and it uses the Euclidean distance to assess the similarity between embeddings of unseen NeRFs from the test set introduced in Section 3.1. For each selected embedding, its *k-nearest neighbors* are identified, then their classes are extracted, and finally they are compared to the initially selected embedding's class. When executing this

Method	mAP@1	mAP@5	mAP@10
nerf2vec	72.38	91.89	95.96
Baseline single-view	74.65	91.52	95.10
Baseline multi-view (9)	82.74	91.66	93.79

Table 4.3: Shape retrieval mAP values are derived from retrieving the K nearest neighbors of a query input NeRF and verifying label equivalence. The value of K used for these experiments is specified after the @ character. Baseline architectures utilized renderings of NeRFs as inputs, and applied the retrieval task on the feature vectors extracted by the *ResNet50* backbone. The number of views for the multi-view baseline is denoted within brackets.

task, the expectation is that the majority of the nearest neighbors share the same class of the originally selected embedding, confirming the optimal organization of the latent space.

To compare the results obtained from the retrieval task using *nerf2vec*'s embeddings, a similar task was executed using two baseline architectures. These architectures utilized N renderings of NeRFs as input, each with varying camera poses. Following this input, feature vectors were extracted using the *ResNet50* backbone. Subsequently, the retrieval task was performed based on these feature vectors. In the experiments, N was fixed to 1 and 9. Table 4.3 demonstrates that the retrieval task can be successfully applied to *nerf2vec*'s embedding, showing high mean Average Precision (mAP) values, comparable to those obtained by the baseline architectures. Moreover, as shown in Figure 4.5, the selected neighbors also exhibit similar structures and colors. This finding served as an additional evidence that *nerf2vec* is proficient at creating embeddings that effectively summarize significant shape information.

4.5 Shape generation

Previous task results have demonstrated the feasibility of utilizing embeddings generated by *nerf2vec* as inputs for diverse deep learning architectures. This

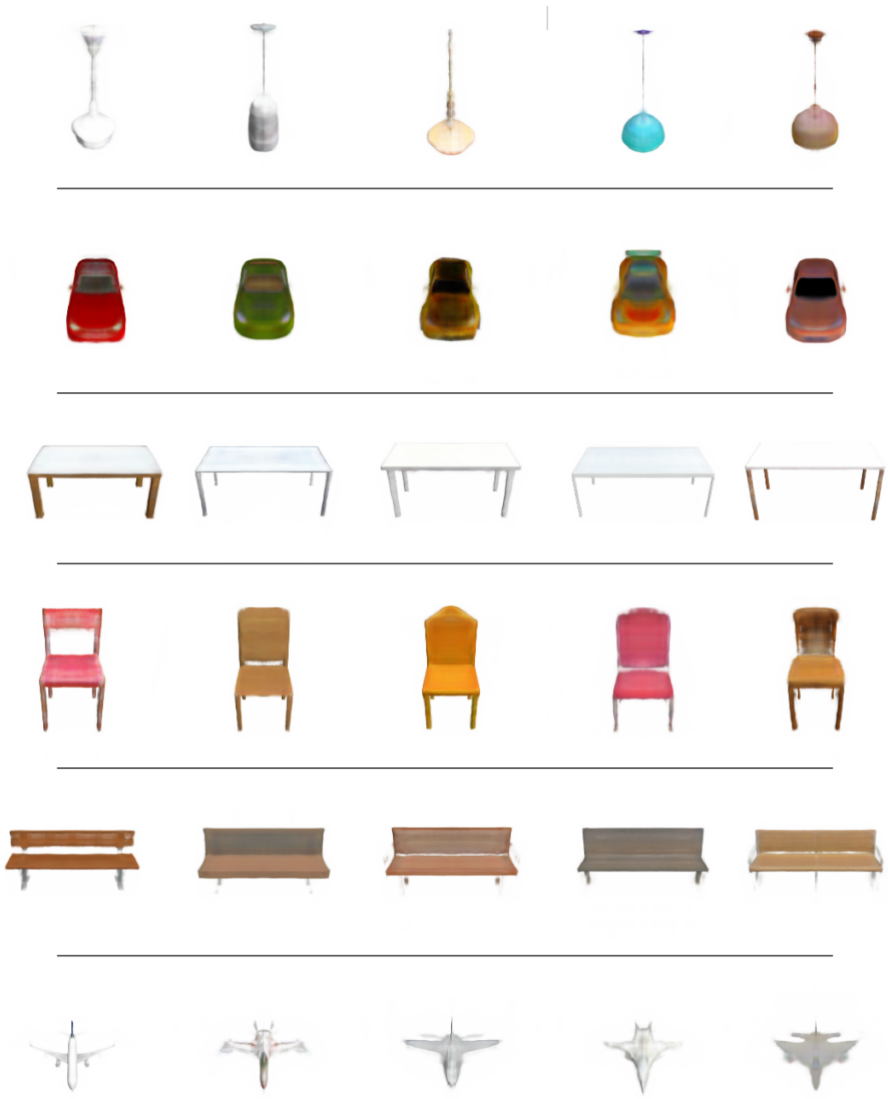

Figure 4.5: Example of the first four nearest neighbors of a query NeRF, which is the leftmost one in each row.

section aims to take a significant step forward by elaborating on another conducted experiment. This experiment involved the generation of new embeddings within an adversarial framework. The ultimate objective was to evaluate the potential for creating credible embeddings for previously unseen NeRFs.

The architecture employed for this task was a *Latent-GAN* [53], outlined in Figure 4.6. This framework aimed to generate embeddings that closely resembled those created by *nerf2vec*, beginning from random noise. Subsequently, these newly generated embeddings could be decoded into discrete representations using the same implicit decoder utilized during the framework's training.

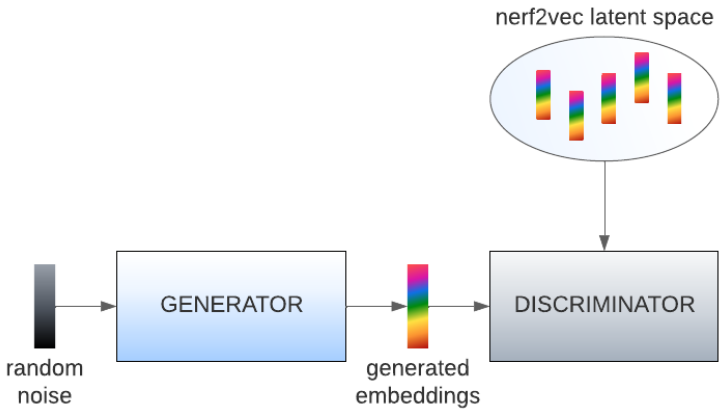

Figure 4.6: *Latent-GAN* architecture overview. Both the discriminator and the generator are composed of two fully connected layers, and adopting standard activation functions. Additional details are provided in the original paper [53].

Figure 4.7: *nerf2vec* generations qualitative results.

Specifically, this task required training multiple adversarial networks, one for each class, utilizing the dataset introduced in Section 3.1. Subsequently, each trained *Latent-GAN* was utilized for inference to generate new embeddings for every class. Examples of these generated embeddings are presented in Figure 4.7, showcasing a notable diversity and intricate detail. Once again, these results highlight the applicability of embeddings generated by *nerf2vec* within deep neural architectures.

Chapter 5

Conclusions and future work

This work was able to eventually provide an answer to the question presented in Chapter 1: *Would it be possible to directly integrate NeRFs within deep learning pipelines to solve downstream tasks?* As extensively shown in the preceding chapters, the answer to this question is *yes*.

The proposed framework *nerf2vec* demonstrated the feasibility of embedding NeRFs into a compact latent vector, creating a latent space for NeRFs serviceable for various deep learning tasks. Specifically, employing these embeddings for classification and shape retrieval yielded highly promising results. Furthermore, the interpolation of these embeddings not only validated the well-organized nature of the learned latent space, but it also enabled the generation of entirely new, previously unseen 3D objects. Similar outcomes were achieved through the training of *Latent-GAN* networks, allowing for the creation of novel NeRFs. In addition, this work added further credit to the *inr2vec* framework [41] by exploring NeRFs, a type of INR that was not treated before.

However, there are two main limitations that must be acknowledged, and these can serve as the foundation for future work to further enhance the performance of *nerf2vec*.

First and foremost, while the reconstruction quality is notably high, it does lose some level of detail compared to the original input NeRFs. Potential strategies to enhance results include increasing the capacity of *nerf2vec*, exploring alternative architectures such as HyperNetworks [18], and increasing the number of renderings used to train each NeRF.
Additionally, a potential constraint on the real-world deployment of this work is the reliance on a common initialization vector for training NeRFs. In this field, significant progress has already been made, as indicated by [5, 58] and other sources. Integrating these advancements into *nerf2vec* could further enhance its capabilities.

Besides the known limitations, there are also several promising avenues for future work and development:

- conducting ablation studies to optimize *nerf2vec*'s training time;
- exploring methods to eliminate the need of occupancy grids;
- adding also the view directions to the decoder's input;
- investigating techniques for online data augmentation;
- expanding the dataset of NeRFs to include unbounded scenes;
- exploring other deep learning tasks, including segmentation and diverse generative methodologies.

In conclusions, this thesis represents an important contribution which demonstrates that NeRFs, and INRs in general, hold great promise as the standard and unified approach for efficient 3D scene representation, and that it is possible to leverage Artificial Intelligence to fully harness their capabilities.

Bibliography

[1] Alon Lahav and Ayellet Tal. MeshWalker: Deep mesh understanding by random walks. 2020. URL: https://arxiv.org/abs/2006.05353.

[2] Amir Hertz, Rana Hanocka, Raja Giryes and Daniel Cohen-Or. Deep geometric texture synthesis. 2020. URL: https://dl.acm.org/doi/10.1145/3386569.3392471.

[3] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon and Yaron Lipman. Implicit geometric regularization for learning shapes. 2020. URL: https://arxiv.org/abs/2002.10099.

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi and Fisher Yu. ShapeNet: An information-rich 3d model repository. 2015. URL: https://arxiv.org/abs/1512.03012.

[5] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik and Haggai Maron. Equivariant architectures for learning in deep weight spaces. 2023. URL: https://arxiv.org/abs/2301.12780.

[6] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. 2020. URL: https://arxiv.org/abs/2003.08934.

[7] L. Biewald. Experiment tracking with weights and biases, 2020. URL: https://www.wandb.com/.

[8] Binh-Son Hua, Minh-Khoi Tran and Sai-Kit Yeung. Pointwise convolutional neural networks. 2018. URL: https://arxiv.org/abs/1712.05245.

[9] Boris Knyazev, Michal Drozdzal, Graham W. Taylor and Adriana Romero. Parameter prediction for unseen deep architectures. 2021. URL: https://arxiv.org/abs/2110.13100.

[10] Charles R Qi, Hao Su, Kaichun Mo and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. 2017. URL: https://arxiv.org/abs/1612.00593.

[11] Charles R Qi, Hao Su, Kaichun Mo and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. 2017. URL: https://arxiv.org/abs/1706.02413.

[12] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. 2016. URL: https://arxiv.org/abs/1604.03265.

[13] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser. Local implicit grid representations for 3d scenes. 2020. URL: https://arxiv.org/abs/2003.08981.

[14] Christian Häne, Shubham Tulsiani and Jitendra Malik. Hierarchical surface prediction for 3D object reconstruction. 2017. URL: https://arxiv.org/abs/1704.00710.

[15] Chunfeng Lian, Li Wang, Tai-Hsien Wu, Mingxia Liu, Francisca Durán, Ching-Chang Ko and Dinggang Shen. MeshSNet: Deep multi-scale mesh feature learning for end-to-end tooth labeling on 3d dental surfaces. 2019. URL: https://link.springer.com/chapter/10.1007/978-3-030-32226-7_93.

[16] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*:922–928, 2015.

[17] Danilo Jimenez Rezende, SM Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg and Nicolas Heess. Unsupervised learning of 3d structure from images. 2018. URL: `https://arxiv.org/abs/1607.00662`.

[18] David Ha andrew Dai and Quoc V. Le. HyperNetworks. 2016. URL: `https://arxiv.org/abs/1609.09106`.

[19] David Stutz and Andreas Geiger. Learning 3d shape completion from laser scan data with weak supervision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*:1955–1964, 2018.

[20] Davide Boscaini, Jonathan Masci, Emanuele Rodolà and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. 2016. URL: `https://arxiv.org/abs/1605.06437`.

[21] Dmitriy Smirnov and Justin Solomon. HodgeNet: learning spectral geometry on triangle meshes. 2021. URL: `https://arxiv.org/abs/2104.12826`.

[22] Emilien Dupont, Hyunjik Kim, SM Ali Eslami, Danilo Jimenez Rezende and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. 2022. URL: `https://arxiv.org/abs/2201.12204`.

[23] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. 2016. URL: `https://arxiv.org/abs/1611.08402`.

[24] Florian Jaeckle and M Pawan Kumar. Generating adversarial examples with graph neural networks. 2021. URL: https://arxiv.org/abs/2105.14644.

[25] Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza and Luca Carlone. Primal-dual mesh convolutional neural networks. 2020. URL: https://arxiv.org/abs/2010.12455.

[26] Gernot Riegler, Ali Osman Ulusoy and Andreas Geiger. OctNet: Learning deep 3D representations at high resolutions. 2017. URL: https://arxiv.org/abs/1611.05009.

[27] Haoxuan You, Yifan Feng, Rongrong Ji and Yue Gao. PVNet: A joint convolutional network of point cloud and multi-view for 3d shape recognition. 2018. URL: https://arxiv.org/abs/1808.07659.

[28] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. 2019. URL: https://arxiv.org/abs/1904.08889.

[29] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. 2019. URL: https://arxiv.org/abs/1901.05103.

[30] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative adversarial modeling. *Advances in neural information processing systems*:29, 2016.

[31] Jingwei Huang, Haotian Zhang, Li Yi, Thomas Funkhouser, Matthias Nießner and Leonidas J Guibas. TextureNet: Consistent local parametrizations for learning from high resolution signals on meshes. 2019. URL: https://arxiv.org/abs/1812.00020.

[32] Jingyue Lu and M. Pawan Kumar. Neural network branching for neural network verification. 2019. URL: https://arxiv.org/abs/1912.01329.

[33] Jonas Schult, Francis Engelmann, Theodora Kontogianni and Bastian Leibe. DualConvMesh-Net: Joint geodesic and euclidean convolutions on 3d meshes. 2020. URL: https://arxiv.org/abs/2004.01002.

[34] Jonathan Masci, Davide Boscaini, Michael Bronstein and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. 2018. URL: https://arxiv.org/abs/1501.06297.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep residual learning for image recognition. 2015. URL: https://arxiv.org/abs/1512.03385.

[36] Konstantin Schürholt, Dimche Kostadinov and Damian Borth. Hyper-Representations: Self-supervised representation learning on neural network weights for model characteristic prediction. 2022. URL: https://arxiv.org/abs/2110.15288.

[37] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna and Thomas Funkhouser. Local deep implicit functions for 3D shape. 2020. URL: https://arxiv.org/abs/1912.06126.

[38] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman and Thomas Funkhouser. Learning shape templates with structured implicit functions. 2019. URL: https://arxiv.org/abs/1904.06447.

[39] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. 2019. URL: https://arxiv.org/abs/1812.03828.

[40] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang and Jie Shan. Graph attention convolution for point cloud semantic segmentation. 2019. URL: https://ieeexplore.ieee.org/document/8954040.

[41] Luca De Luigi, Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti and Luigi Di Stefano. Deep learning on implicit neural representations of shapes. 2023. URL: https://arxiv.org/abs/2302.05438.

[42] Matan Atzmon and Yaron Lipman. SAL: Sign agnostic learning of shapes from raw data. 2020. URL: https://arxiv.org/abs/1911.10414.

[43] Matan Atzmon, Haggai Maron and Yaron Lipman. Point convolutional neural networks by extension operators. 2018. URL: https://arxiv.org/abs/1803.10091.

[44] Mateusz Michalkiewicz, Jhony K Pontes, Dominic Jack, Mahsa Baktashmotlagh and Anders Eriksson. Implicit surface representations as layers in neural networks. 2020. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9010266.

[45] Maxim Tatarchenko, Alexey Dosovitskiy and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs. 2017. URL: https://arxiv.org/abs/1703.09438.

[46] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin and Shi-Min Hu. PCT: Point cloud transformer. 2021. URL: https://arxiv.org/abs/2012.09688.

[47] Michael Niemeyer, Lars Mescheder, Michael Oechsle and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. 2020. URL: https://arxiv.org/abs/1912.07372.

[48] Michael Niemeyer, Lars Mescheder, Michael Oechsle and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. 2019. URL: https://ieeexplore.ieee.org/document/9008276.

[49] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss and Andreas Geiger. Texture fields: Learning texture representations in function space. 2019. URL: https://arxiv.org/abs/1905.07259.

[50] Mutian Xu, Runyu Ding, Hengshuang Zhao and Xiaojuan Qi. PAConv: Position adaptive convolution with dynamic kernel assembling on point clouds. 2021. URL: https://arxiv.org/abs/2103.14635.

[51] Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron and Yaron Lipman. Surface networks via general covers. 2019. URL: https://arxiv.org/abs/1812.10705.

[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet large scale visual recognition challenge. 2015. URL: https://arxiv.org/abs/1409.0575.

[53] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas and Leonidas Guibas. Learning Representations and Generative Models for 3D Point Clouds. 2018. URL: https://arxiv.org/abs/1707.02392.

[54] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu and Xin Tong. Adaptive O-CNN: A patch-based deep representation of 3D shapes. 2018. URL: https://arxiv.org/abs/1809.07917.

[55] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun and Xin Tong. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. 2017. URL: https://arxiv.org/abs/1712.01537.

[56] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech and Ulrich Neumann. DISN: Deep implicit surface network for high-quality single-view 3D reconstruction. 2021. URL: https://arxiv.org/abs/1905.10711.

[57] R. Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. *Advanced Neural Computers*:129–135, 1990.

[58] Rahim Entezari, Hanie Sedghi, Olga Saukh and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. 2022. URL: https://arxiv.org/abs/2110.06296.

[59] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman and Daniel Cohen-Or. MeshCNN: A network with an edge. 2019. URL: https://arxiv.org/abs/1809.05910.

[60] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove and Richard Newcombe. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. 2020. URL: https://arxiv.org/abs/2003.10983.

[61] Rohit Girdhar, David F Fouhey, Mikel Rodriguez and Abhinav Gupta. Learning a predictable and generative vector representation for objects. 2016. URL: https://arxiv.org/abs/1603.08637.

[62] Ruilong Li, Matthew Tancik and Angjoo Kanazawa. NerfAcc: A general NeRF acceleration toolbox. 2023. URL: https://arxiv.org/abs/2210.04847.

[63] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang and Hongsheng Li. PV-RCNN: Point-voxel feature set abstraction for 3d object detection. 2020. URL: https://arxiv.org/abs/1912.13192.

[64] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa and Hao Li. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. 2019. URL: https://arxiv.org/abs/1905.05172.

[65] Shuran Song and Jianxiong Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. 2016. URL: https://arxiv.org/abs/1511.02300.

[66] Siqi Fan, Qiulei Dong, Fenghua Zhu, Yisheng Lv, Peijun Ye and Fei-Yue Wang. SCF-Net: Learning spatial contextual features for large-scale point cloud segmentation. 2021. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9577763.

[67] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys and Andreas Geiger. Convolutional occupancy network. 2020. URL: https://arxiv.org/abs/2003.04618.

[68] Thomas Müller, Alex Evans, Christoph Schied and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. 2022. URL: https://arxiv.org/abs/2201.05989.

[69] Thomas Müller, Fabrice Rousselle, Jan Novák and Alexander Keller. Real-time neural radiance caching for path tracing. 2021. URL: https://tom94.net/data/publications/mueller21realtime/mueller21realtime.pdf.

[70] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet and Ilya O. Tolstikhin. Predicting neural network accuracy from weight. 2021. URL: https://arxiv.org/abs/2002.11448.

[71] Tiny CUDA neural networks. URL: https://github.com/NVlabs/tiny-cuda-nn.

[72] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell and Gordon Wetzstein. Implicit neural representations with periodic activation functions. 2020. URL: https://arxiv.org/abs/2006.09661.

[73] Vincent Sitzmann, Michael Zollhöfer and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. 2020. URL: https://arxiv.org/abs/1906.01618.

[74] Xianzhi Li, Ruihui Li, Lei Zhu, Chi-Wing Fu and Pheng-Ann Heng. DNF-Net: A deep normal filtering network for mesh denoising. 2020. URL: https://arxiv.org/abs/2006.15510.

[75] Yangyan Li, Soeren Pirk, Hao Su, Charles R. Qi and Leonidas J. Guibas. FPNN: Field probing neural networks for 3D data. 2016. URL: https://arxiv.org/abs/1605.06240.

[76] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng and Yu Qiao. SpiderCNN: Deep learning on point sets with parameterized convolutional filters. 2018. URL: https://arxiv.org/abs/1803.11527.

[77] Yuqi Yang, Shilin Liu, Hao Pan, Yang Liu and Xin Tong. PFCNN: Convolutional neural networks on 3d surfaces using parallel frames. 2020. URL: https://arxiv.org/abs/1808.04952.

[78] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao and Yue Gao. MeshNet: Mesh neural network for 3d shape representation. 2018. URL: https://arxiv.org/abs/1811.11424.

[79] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. 2019. URL: https://arxiv.org/abs/1812.02822.

[80] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang and Jianxiong Xiao. 3D ShapeNets: A deep representation

for volumetric shapes. 2015. URL: https://ieeexplore.ieee.org/document/7298801.

Acknowledgements

I would like to express my deepest gratitude to *Professor* Samuele Salti for his continuous support and mentorship throughout my academic pursuit. *Doctor* De Luigi Luca played an indispensable role in shaping the direction of my research, offering invaluable insights, guidance, and consistent assistance during the thesis composition. I am also thankful for the contributions of *Professors* Luigi Di Stefano and Pierluigi Zama Ramirez, alongside *Doctors* Adriano Cardace and Riccardo Spezialetti, whose expertise significantly enriched my academic experience.

My utmost appreciation goes to my wife Serena and my daughter Sofia. Your steadfast support, enduring patience, and constant presence have been my pillars throughout this challenging journey. Through every hardship and triumph, you stood by me, sharing the burdens and celebrating each milestone. Your love and encouragement have been my guiding strength, enabling this achievement. I'm profoundly grateful for your absolute trust in me, which made this journey extraordinary.

I am also profoundly grateful to my parents, Gianni and Franca, for their unwavering sustenance and relentless encouragement. Their consistent motivation and invaluable guidance were instrumental in helping me achieve my goals. Their belief in me has been an unshakable source of strength, shaping me into the person I am today. Additionally, I extend heartfelt thanks to my *older* twin, Simone, whose invaluable suggestions have been a guiding light

throughout my entire learning journey.

Francesco Abbo, your friendship, assistance, and resolute endorsement have been priceless throughout this adventure; your presence has made this academic pursuit infinitely more meaningful. I also wish to express my sincere appreciation to my *mellon* Mattia Carbognani, as well as my friends Mattia Baccaro, and Andrea Righelli for their constant presence whenever a shoulder was needed. Moreover, I cannot overstate my gratitude to Gaetano Signorelli, who not only served as a significant colleague during exams but has also become a good friend. Your encouragement and camaraderie have profoundly impacted my path.

Finally, I extend my heartfelt gratitude to Gianfranco Sinesi, whose inspiration many years ago ignited within me a fascination for the captivating and boundless world of computers.

Appendix A

NeRF training

As detailed in chapter 3, the used MLP to train NeRFs is called *FullyFusedMLP*. However, this is not the only MLP available inside the *Tiny CUDA Neural Networks* framework [71]. More specifically, another MLP was tested in the presented work, and its name is *CutlassMLP*. Additionally, the *Tiny CUDA Neural Networks* framework allows to straightforwardly use different types of input encoding that were experimented so as to keep the training procedure as simple as possible, while trying to preserve high quality results.

The *CutlassMLP* is notably slower compared to the *FullyFusedMLP*, but it offers greater flexibility. For instance, it permits more customization of the number of units in its linear layers and supports various activation functions. Due to these advantages, the initial NeRFs were trained using the *CutlassMLP* with linear layers and activation functions identical to those used by *inr2vec* to train INRs. Additionally, an identity encoding was employed to keep the training procedure in its simplest form. Despite these choices, the results were disappointing, primarily due to the slow training and poor output quality. Figure A.1 illustrates the diverse PSNR scores achieved alongside the training time needed for performing a fixed number of iterations. It accentuates the superiority of frequency encoding, which attains higher PSNR values with significantly fewer units per linear layer and markedly shorter training times.

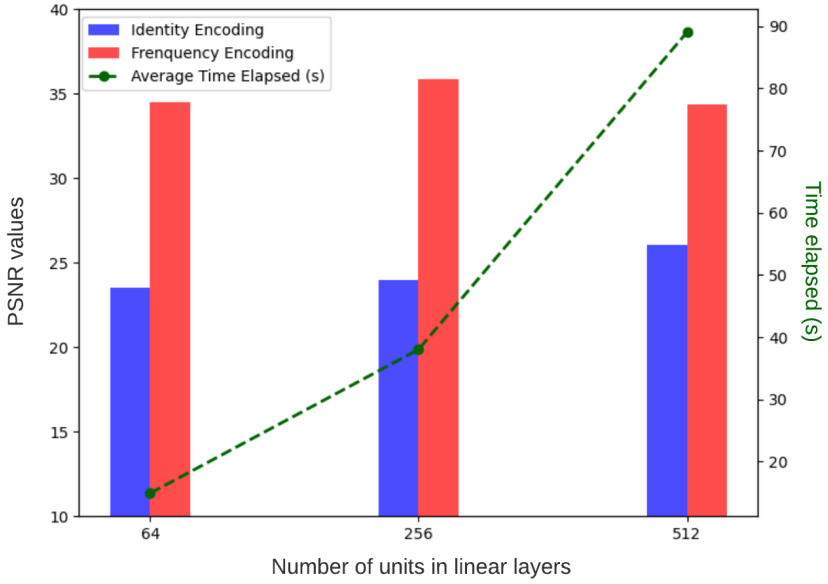

Figure A.1: Comparison of PSNR values and training times for different numbers of units in the MLP's linear layers, using identity and frequency encodings.

Furthermore, Figure A.2 visually compares different unit-encoding combinations, emphasizing the superiority of frequency encoding over identity encoding.

After selecting the main architecture for training NeRFs, a switch was made from *CutlassMLP* to *FullyFusedMLP*. The latter achieved similar PSNR scores with respect to the former but significantly reduced training time. As a result, *FullyFusedMLP* became the final choice for NeRF training. The remaining hyperparameters, including encoding dimension, hidden layer count, and units per layer, were determined based on the highest achieved PSNR. It's worth noting that these hyperparameters were eventually selected to strike the optimal balance between training time and final quality.

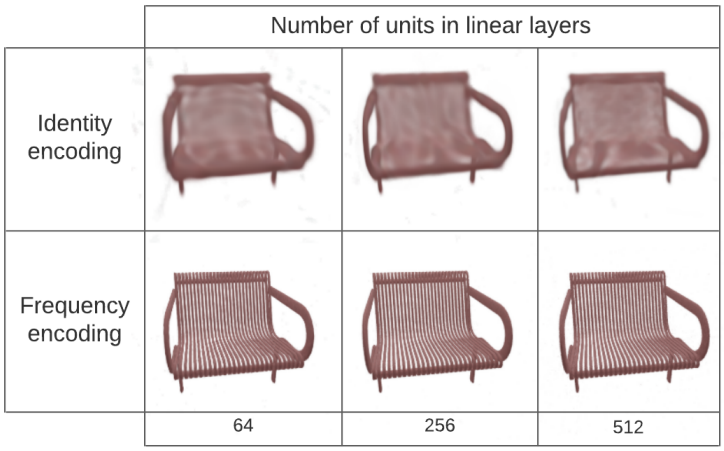

Figure A.2: Depiction of renderings showcasing diverse qualities influenced by the number of units in each MLP layer and encoding methodologies.

Appendix B

Implementation and hardware

B.1 General settings

All experiments were primarily conducted using the PyTorch library on a machine equipped with an Intel Core i7-9700K CPU and a single NVIDIA GeForce RTX 3090 GPU. Visualizations and statistical data were generated using WandB [7], a tool designed to efficiently track experiments, measure model performance, and monitor the training processes.

B.2 Mixed precision

Compared to *inr2vec*, *nerf2vec* demands significantly more resources. Specifically, a substantial amount of VRAM is required to train a single batch due to the simultaneous training of multiple NeRFs. To address this, mixed precision was employed during training, enabling *nerf2vec* to quickly learn each NeRF by processing a larger volume of 3D coordinates at every iteration, thereby expediting the overall training process, and hugely reducing the memory footprint required.

B.3 Timings

To train a single NeRF, which is an operation necessary to create the dataset of NeRFs, the employed *FullyFusedMLP* required an average of 12 seconds to train each NeRF, which is equivalent to roughly $1500 - 2000$ iterations.

To train *nerf2vec* for a single epoch on the full training set, about 58 minutes were required. Note that the training set comprises approximately 90k NeRFs.

Appendix C

NeRF weights removal comparison

Appendix I of the *inr2vec* paper [41] provides a detailed explanation for the exclusion of input and output layers from the vector representation of each INR used to train the framework. This exclusion was essential to reduce the encoder's input dimension and accelerate the training process. However, it's important to note that this technique, which involves removing input and output layer weights of each NeRF, didn't align with the goals of the presented work, as became evident during the initial classification tasks. To reach this conclusion, a comparison was made between the results of two different classifiers: the first one directly classified NeRF weights, while the second classified embeddings obtained by applying the aforementioned technique. As illustrated in Figure C.1, the first classifier significantly outperforms the accuracy of the second one, highlighting a potential weakness in the created embeddings.

The most probable reason for this accuracy gap is that the number of parameters in the MLPs used to train NeRFs in *nerf2vec* was significantly smaller compared to those used in training INRs in *inr2vec*. As a result, the removal of input and output layer weights was causing the loss of a substantial amount of information. These considerations led to the decision of retraining the *nerf2vec* framework from scratch, this time without removing the input and

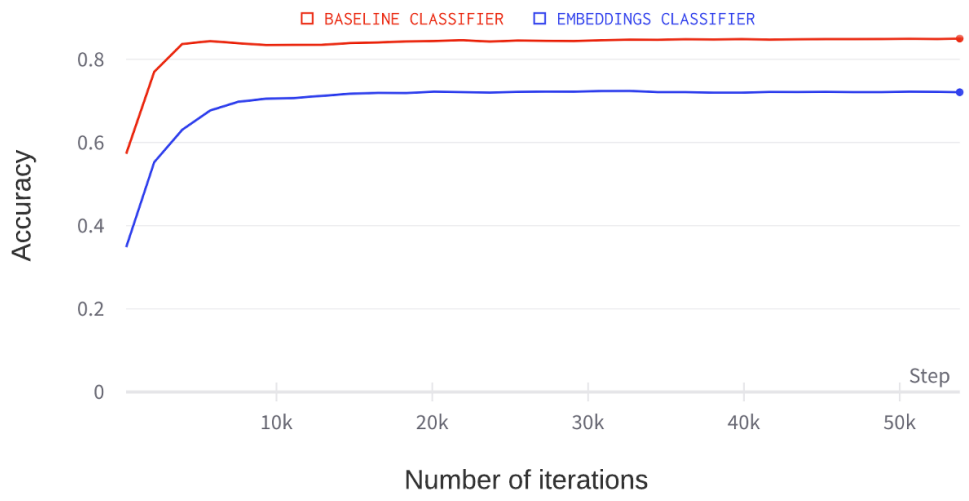

Figure C.1: Accuracies achieved by the embeddings and baseline classifiers on the validation set when the input and output layer weights were omitted from the training of the *nerf2vec* framework.

output layer weights. Ultimately, this choice allowed the embeddings classifier to even outperform the baseline classifier, as demonstrated in Chapter 4.