Alma Mater Studiorum - Università di Bologna

Dottorato Di Ricerca in

Computer Science and Engineering

Ciclo 36

Settore Concorsuale: 09/H1
Settore Scientifico-Disciplinare: ING-INF/05

Learning With Limited Data

Presentata da:
Adriano Cardace

Coordinatore di Dottorato:
Prof.ssa Ilaria Bartolini

Supervisore:
Prof. Luigi Di Stefano

Esame finale anno 2024

UNIVERSITÀ DI BOLOGNA

Abstract

Facoltà di Ingegneria ed Architettura
Dipartimento di Informatica - Scienza e Ingegneria

Dottorato di Ricerca

Learning With Limited Data

by Adriano CARDACE

In recent years, Deep Learning techniques have demonstrated remarkable achievements across various Computer Vision tasks, frequently surpassing human capabilities. Nevertheless, these data-driven methodologies often demand large volumes of annotated data, necessitating laborious and costly manual annotation procedures. The objective of this thesis is to introduce novel methods designed to mitigate this challenge by harnessing knowledge obtained from diverse domains or tasks, even in the presence of limited annotations. This challenge is commonly known as the Transfer Learning problem. Our exploration will delve into the forefront of Transfer Learning, with a predominant emphasis on the advancement of techniques for Domain Adaptation in diverse computer vision tasks. This research journey begins with a comprehensive investigation into 2D Semantic Segmentation, and we demonstrate how other tasks such as Depth Estimation and Edge Detection can enhance the adaptability of models across different visual domains. Subsequently, the exploration extends to the realm of 3D point cloud classification, where the challenges posed by diverse domain shifts are addressed once again exploiting auxiliary tasks such as shape reconstruction or recent Self-Supervised techniques. The proposed works for 2D Semantic Segmentation and 3D point cloud classification lay the foundation for the development of novel frameworks aimed at tackling the challenging task of multi-modal Domain Adaptation for 3D Semantic Segmentation, where multiple sensors such as RGB cameras and LiDARs are available. Finally, we shed some light on a new exciting and emerging topic which is solving common vision tasks on Neural Fields, which are an emerging paradigm used to represent signals such as images or 3D shapes. We will specifically focus on the 3D scenario, and in the context of Transfer Learning, show for the first time how acting directly on Neural Fields allows the possibility to transfer knowledge among different representations such as from 3D point clouds to meshes.

Acknowledgments

I am profoundly grateful for the support and guidance I have received throughout my Ph.D. journey, and I would like to express my deepest appreciation and gratitude to my supervisors, Prof. Luigi Di Stefano and Prof. Samuele Salti. They have been a constant source of inspiration and encouragement in this difficult and at the same time exciting journey. Their commitment to my personal and professional development has played a pivotal role in shaping my career and I will never forget it.

A huge thanks to my colleagues at the CVLab as well, who helped me live this experience in the best possible way. Thank you, guys, for making my Ph.D. much simpler and funnier.

Let me conclude in Italian to spend some final words for my family and friends.

Grazie ai miei genitori e a mio fratello, che mi hanno sempre sostenuto in questo percorso e non mi hanno mai fatto mancare nulla. Senza voi questo traguardo sarebbe stato impossibile.

Un grazie infinite anche ai miei amici, che riescono sempre a rendere tutto più leggero e a farmi sorridere in qualunque momento.

Infine, ci tengo particolarmente a ringrazie Lucy, il mio bellisismo cane che mi ha accompagnato lungo tutto il percorso, passando ore ed ore, notte e giorno, sempre fedelmente al mio fianco.

Contents

Chapter 1

Introduction

1.1 The Transfer Learning Problem

Deep learning has revolutionized Computer Vision in recent years. The success can be largely attributed to the undeniable effectiveness of Convolutional Neural Networks (CNNs) [12, 13, 14] and more recently Transformers [15]. These models have remarkable performance when trained on high-quality annotated training data. Moreover, the availability of numerous pre-trained models allows for the reuse of these networks as feature extractors, enabling the resolution of complex tasks with minimal effort. For example, in a classification problem, one can utilize standard network architectures like ResNet[14] or VGG [13] and train them using their own dataset, potentially yielding excellent results if the data are accurately annotated. These capabilities can be effectively utilized to tackle various complex tasks that demand a comprehensive understanding of images, such as semantic segmentation, depth estimation, and many more. In 2014, noteworthy advancements were made by [16], that achieved state-of-the-art outcomes by simply employing a linear SVM classifier on top of a CNN. However, thus far, most approaches have tackled each task independently, involving the collection of a dataset, training of the model, and subsequent testing. But what if we could transfer the knowledge gained from a previous task to a new one? Or train our network in a specific domain and deploy it in another? After all, humans tend to learn progressively, building upon past knowledge. In a way, we aim to make these algorithms more human-like. Leveraging previously acquired knowledge can be beneficial in several ways. For instance, collecting large datasets for tasks like semantic segmentation is often impractical. To this end, it is essential to develop strategies that exploit synthetic datasets to save both time and money. Transferring knowledge presents a significant challenge for computers, and despite ongoing efforts by researchers to push the boundaries of Deep Learning, we are still far from achieving human-level capabilities. In the realm of Machine Learning, the task of transferring knowledge is commonly referred to as Transfer Learning (TL). This area of research is expansive and dynamic, with roots in Computer Vision before the rise of Deep Learning. Before delving into the specific focus of this thesis, which is Transfer Learning in the context of Domain Adaptation (DA) and Task Transfer (TT), it is essential to briefly clarify some fundamental concepts that will be utilized throughout this dissertation. Depending on the available data, different variations of machine learning problems can arise, and the most common ones are as follows:

- Supervised Learning: under the supervised setting, we are given input-label pairs, and the goal is to learn a mapping function between input and labels. A simple example is image classification, in which the input is an image and the label is the class it belongs to.
- Unsupervised Learning: In this scenario, the absence of labels makes it challenging to assign categories. The main objective is to develop a feature space that effectively represents the distinctive qualities of the input data. This is achieved by optimizing an objective function without relying on any form of annotations. Typical tasks in this context include clustering and anomaly detection.
- Semi-Supervised Learning: These algorithms aim to learn from both labeled and unlabeled samples. The underlying assumption is that both types of samples are drawn from the same or similar distribution, allowing them to be utilized together to enhance performance. In many real-world scenarios, we often encounter situations where only a small portion of the data is labeled, while a vast amount remains unlabeled. Therefore, it becomes crucial to employ techniques that can effectively leverage both labeled and unlabeled data to maximize learning potential.
- Self-Supervised Learning: Self-supervised learning is a relatively recent learning technique where the training data is labeled autonomously. While it can be considered a type of supervised learning, the key distinction lies in the fact that the datasets are not manually annotated by humans. Instead, they are automatically labeled through a surrogate task. By designing a complex task that provides free labels, it becomes possible to learn high-quality features that can be applied to the intended target task. An example of such a task is image rotation, where the input image is rotated, and the model predicts the degree of rotation. Solving these types of tasks requires a deep understanding of high-level semantics. Consequently, the model learns representations that can subsequently be utilized to address downstream problems. Self-Supervised Learning is sometimes referred to as Weak-Supervised Learning.

Transfer Learning can be employed in all the aforementioned settings as these concepts are independent of each other. To provide an intuitive and general example of the Transfer Learning problem, let's consider a scenario where our goal is to develop a model capable of performing semantic segmentation, which involves assigning pixel-wise labels to different objects in an image, such as buildings, trees, cars, pedestrians, and traffic lights. An example of such a dataset in the autonomous driving field is the Cityscapes dataset [17]. However, when we deploy the same model on another dataset, such as Kitti [18], we encounter significant performance degradation. The reason behind this poor performance is that the domain has changed. Despite both datasets representing street scenes, there are variations such as the different type of buildings or cars, and lighting conditions, among other factors, which affect the model's ability to generalize. This is where Domain Adaptation comes into play. Domain Adaptation is a specific sub-discipline of Transfer Learning that addresses these types of scenarios. In case no forms of annotations are available for the target domain, we specifically talk about Unsupervised Domain Adaptation (UDA). Moreover, Transfer learning can also involve the Task Transfer (TT) problem [19], *i.e.*, on exploiting supervised data to tackle multiple tasks in a single domain more effectively by leveraging on the relationships between the learned representations. Thereby, in many of the approaches we introduce in the next chapters, we focus on merging DA and TT by explicitly addressing a cross-domain and cross-task problem where on one source domain (*e.g.*, synthetic data) we have supervision for many tasks, while in another target one (*e.g.*, real data) annotations are available only for a specific task while we wish to solve many.

1.2 Notation and Formal Definition of Transfer Learning

Let's now give a formal definition of Transfer Learning and introduce some common notation that will be used throughout this thesis. For the adopted notation, we closely follow most of the surveys about Transfer Learning [20] [21]. A domain $\mathcal{D}$ consists of a feature space $\mathcal{X}$ and a marginal probability distribution P(X), where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. We may consider X as our training data. If the domain consists of RGB images with size $W \times H \times C$, we have a three-dimensional feature space of size $W \times H \times C$, *i.e.*, the space of the possible three-channels images that can be generated. A dataset can be thought of as a small volume inside this huge feature space. For example, all the natural images, lie in a specific portion of this multi-dimensional feature space, and we are only interested in modeling its marginal probability distribution P(X). For a given domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task $\mathcal{T}$ is defined by two components, $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$, where $\mathcal{Y}$ is the label space and $f(\cdot)$ an objective predictive function that under a probabilistic perspective can be seen as the conditional probability distribution $P(Y|X)$. In the classical supervised setting, $P(Y|X)$ can be learned directly from the labeled data $\{x_i, y_i\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. We can generalize to the situation in which we have two domains, the source domain with sufficient labeled data $\mathcal{D}^s = \{X^s, P(X)^s\}$, and the target one with a small amount of labeled data or no annotated data $\mathcal{D}^t = \{X^t, P(X)^t\}$. $\mathcal{D}^t$ can thereby be decomposed in two sets: the labeled part, $\mathcal{D}^{tl}$, and the unlabeled part, $\mathcal{D}^{tu}$. The entire target domain is $\mathcal{D}^t = \mathcal{D}^{tl} \cup \mathcal{D}^{tu}$. Each domain is coupled with its corresponding task: the former is $\mathcal{T}^s = \{\mathcal{Y}^s, P(Y^s|X^s)\}$, and the latter is $\mathcal{T}^t = \{\mathcal{Y}^t, P(Y^t|X^t)\}$. The previous definition gives a general framework that can be instantiated in different settings to obtain several cases. For example, if we fix $\mathcal{D}^s = \mathcal{D}^t$ and $\mathcal{T}^s = \mathcal{T}^t$, we are in the traditional Supervised Learning case. In general, since both domain and task are composed by two elements, we have in total four possibilities:

1. The two datasets are different because the feature space is different: $\mathcal{X}^s \neq \mathcal{X}^t$. A typical example can be digit recognition. The source domain only contains one channel images, while in the test domain, we have to classify colored digits.

2. The difference between the two datasets is caused by a distribution shift: $P(X^s) \neq P(X^t)$. Fig. 1.1 illustrates such scenario. In a simplistic two-dimensional feature space, we have images belonging to the same class (the digits 5) that are occupying different portions of the feature space. Hence, training a classifier on domain A, and applying it on domain B, would lead to very poor performance.

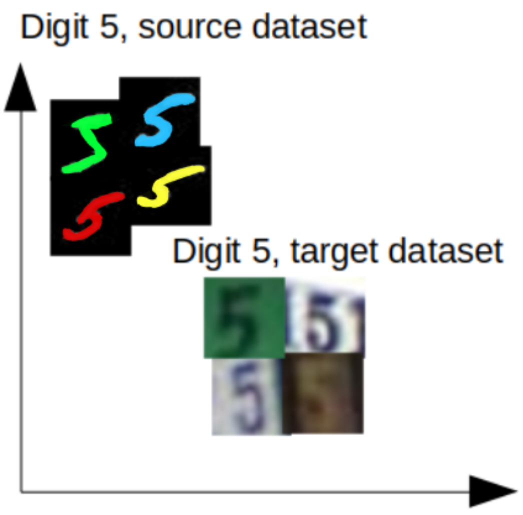


FIGURE 1.1. Distribution shift

3. Tasks divergence is caused by a label space discrepancy: $\mathcal{Y}^s \neq \mathcal{Y}^t$. A typical example of this case is face recognition, because in the source domain, we may have some faces, while in the target domain, we would like to recognize other people as depicted in Fig. 1.2

4. The conditional probability distribution of source and target tasks are different: $P(Y^s|X^s) \neq P(Y^t|X^t)$. This case appears quite often in practice. This happens for example when we train a model for image classification on a balanced dataset, while the test set is strongly unbalanced.

The four previous cases are not exclusive and can also appear together as is often the case in real-world problems. Again, in the autonomous driving scenario, we may have at our

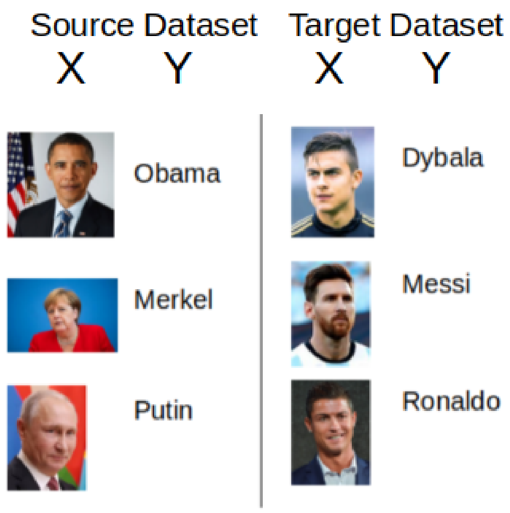


FIGURE 1.2. Different label space

disposal additional sensing information such as depth maps for both the source and the target domain. if we want to solve semantic segmentation in the target domain by exploiting such data, we are in a situation in which both the tasks (label space discrepancy) and the domains (different feature space) differ.

1.3 Structure of the thesis

This dissertation aims to introduce novel Transfer Learning algorithms capable of transferring knowledge across diverse domains and tasks. This research encompasses various applications, such as 2D semantic segmentation, 3D point cloud classification, and multi-modal semantic segmentation. The chosen order of these topics is deliberate and strategically designed. Firstly, we focus on 2D UDA for semantic segmentation, a well-studied field within the UDA literature [22, 23, 24, 25, 26, 27]. Following our contributions in this domain, we analyze and propose novel UDA algorithms for 3D point cloud classification [28, 29, 30]. Next, we investigate 3D semantic segmentation from multi-modal data (i.e., RGB and LiDAR scans). This task can be considered the culmination of the previous two steps and only few approaches delt with this complex scenario [31, 32, 33]. Notably, 3D semantic segmentation using multi-modal data presents substantial challenges and necessitates a comprehensive understanding of the prior topics before tackling its intricacies.

Finally, while Transfer Learning remains the primary focus of this thesis, an additional chapter is dedicated to Neural Fields. This emerging topic has gained significant interest within the computer vision community. In this context, we present two works exploring how to perform standard Deep Learning tasks on this novel representation. Although Neural Fields have not yet been investigated concretely in the context of Transfer learning, they hold immense potential as a unified framework for representing various 3D data forms. To this end, we present in the last chapter some preliminary results on transferring knowledge across modalities *i.e.*, classifying Neural Fields of point clouds with a network trained on neural fields of meshes. By introducing these works, we hope to inspire further research towards leveraging neural Fields in Transfer Learning problems.

To summarize, the structure of the thesis is as follows:

Part 1 - Domain Adaptation for 2D Semantic Segmentation. We propose here two lines of research in this area. The first two works focus on how to exploit different tasks such as depth estimation, to improve the performance of a segmentation model in the Domain Adaptation scenario. In the last work instead, we focus on how to obtain better segmentation masks at object boundaries when the model has to be tested in a different domain.

- **Chapter 3 - Learning Good Features to Transfer Across Tasks and Domains.** In this work [4] we introduce a set of strategies to constrain the learned feature spaces so that the transfer of such features across tasks and domains can be considerably improved.
- **Chapter 4 - Plugging Self-Supervised Monocular Depth into Unsupervised Domain.** In all previous works, the domain shift has always been addressed by focusing on the differences between the two domains. This gap is usually addressed in the input space, feature space, or output space. However, a different perspective can also be introduced, and in this work presented in [2], we investigate whether it is possible to boost UDA performance for 2D semantic segmentation by transferring knowledge learned from *another task*.
- **Chapter 5 - Shallow Features Guide Unsupervised Domain Adaptation for Semantic Segmentation at Class Boundaries.** Previous UDA methods for 2D semantic segmentation can correctly segment out coarse blobs of large elements in a scene such as cars or buildings, failing however in providing pixel-wise accurate segmentation masks. Our approach presented in [34], exploits shallow features of a CNN encoder to refine the coarse segmentation.

Part 2 - Domain Adaptation for 3D Point Cloud Classification. In this part, we specifically address UDA for Point Cloud Classification, and we propose two novel methods that focus mainly on the Self-training step for boosting performance on the target domain.

- **Chapter 7 - RefRec: Pseudo-labels Refinement via Shape Reconstruction for Unsupervised 3D Domain Adaptation.** In this chapter, we present RefRec [1], a novel algorithm that relies on the self-training protocol to learn domain-specific decision boundaries and reduce the negative impact of mislabelled target samples, showcasing the effectiveness of pseudo-labels for this important problem.
- **Chapter 8 - Self-Distillation for Unsupervised 3D Domain Adaptation.** This chapter reports a novel UDA algorithm introduced in [5]. Here, we focus on obtaining a

discriminative feature space for the target domain enforcing consistency between a point cloud and its augmented version. We then propose a novel iterative self-training methodology that exploits Graph Neural Networks (GNNs) in the UDA context to refine pseudo-labels.

Part 3 - Domain Adaptation for multi-modal Semantic Segmentation. Here, we combine the knowledge acquired with the previously proposed methods and investigate UDA for multi-modal data that involves both 2D and 3D sensors.

- **Chapter 10 - Boosting Multi-Modal Unsupervised Domain Adaptation for LiDAR Semantic Segmentation by Self-Supervised Depth Completion**. In this chapter, we propose a novel multi-modal UDA method that leverages depth completion as an auxiliary task to align features extracted from 2D images across domains.

- **Chapter 11 - Exploiting the Complementarity of 2D and 3D Networks to Address Domain-Shift in 3D Semantic Segmentation**. In this chapter, we introduce MM2D2D[6], a two-branches architecture that processes 2D and 3D information independently. We explain why this design choice is effective and then show how this architecture can be improved to make the multi-modal semantic segmentation more robust to domain shift.

Part 4 - Neural Fields. Neural Fields have emerged in the last few years as a powerful tool to encode continuously a variety of different signals like images, videos, audio and 3D shapes. When a single Multi-Layer perception (MLP) is used to parametrize a Neural Field, we usually refer to them as Implicit Neural Representations (INRs) In the last part of this thesis, we introduce the first work that shows how to effectively process INRs to solve common downstream tasks. Then, in the last chapter, we go beyond the concept of INRs and consider also other hybrid parametrizations for Neural Fields to solve downstream tasks more effectively. Moreover, this representation enables the possibility to perform preliminary experiments in the context of Transfer Learning.

- **Chapter 13 - Deep Learning on Implicit Neural Representations of Shapes** Since INR are parametrized as neural networks, it is not clear whether and how it may be possible to feed them into deep learning pipelines aimed at solving a downstream task. Here, we put forward this research problem and propose inr2vec, a framework that can compute a compact latent representation for an input INR in a single inference pass. We verify that inr2vec can embed effectively the 3D shapes represented by the input INRs and show how the produced embeddings can be fed into deep learning pipelines to solve several tasks by processing exclusively INRs.

- **Chapter 14 - Neural Processing of Tri-Plane Hybrid Neural Fields** Using a single neural network to parametrize a neural field to solve downstream tasks still holds many limitations, such as being significantly inferior to those achieved by processing explicit representations, e.g., point clouds or meshes. In the meantime, hybrid representations, in particular based on tri-planes, have emerged as a more effective and efficient alternative to realize neural fields, but their direct processing has not been investigated yet. In this Chapter, we show that the tri-plane discrete data structure encodes rich information, which can be effectively processed by standard deep-learning machinery. We define an extensive benchmark covering a diverse set of fields such as occupancy, signed/unsigned distance, and, for the first time, radiance fields. While processing a field with the same reconstruction quality, we achieve task performance far superior to frameworks that process large MLPs and, for the first time, almost on par with architectures handling explicit representations.

Part I

Transfer Learning for 2D Semantic Segmentation

Chapter 2

Initial Remarks

Semantic segmentation is the task of classifying each pixel of an image. Nowadays, Convolutional Neural Networks can achieve impressive results in this task but require huge quantities of labeled images at training time [35, 36, 37, 38]. A popular trend to address this issue concerns leveraging computer graphics simulations [89] or game engines [39] to obtain automatically synthetic images endowed with per-pixel semantic labels. Yet, a network trained on synthetic data only will perform poorly in real environments due to the so-called *domain-shift* problem. In the last few years, many Unsupervised Domain Adaptation (UDA) techniques aimed at alleviating the domain-shift problem have been proposed in literature. These approaches try to minimize the gap between the labeled source domain (e.g. synthetic images) and the unlabeled target domain (e.g. real images) by either hallucinating input images, manipulating the learned features space or imposing statistical constraints on the predictions [26, 40, 41, 42].

At a more abstract level, UDA may be thought of as the process of transferring more effectively to the target domain the knowledge from a task solved in the source domain. This suggests that it may be possible to improve UDA by transferring also knowledge learned from *another task* to improve performance in the real domain. In fact, the existence of tightly related representations within CNNs trained for different tasks has been highlighted since the early works in the field [43], and it is nowadays standard practice to initialize CNNs deployed for a variety of diverse tasks, such as, e.g., object detection [44], semantic segmentation [45] and monocular depth estimation [46], with weights learned on Imagenet Classification [47]. The notion of *transferability* of representations among CNNs trained to solve different visual tasks has been formalized computationally by the Taskonomy proposed in [19]. Later, [48] has shown that it is possible to train a CNN to hallucinate deep features learned to address one task into features amenable to another task related to the former by means of a *transfer* function. In Chapter 3, we exploit the framework introduced in [48], and we build upon two intuitions. First, the transfer function itself suffers from the domain shift. Second, solving multiple tasks in a multi-task learning scenario can help to enrich the feature representation and hence help to boost performance. Following this line of research, in Chapter 4, we specifically focus on exploiting monocular depth estimation to learn useful features to boost the performance of any existing UDA method. Finally, in Chapter 5, we again take advantage of an auxiliary task such as edge detection to propose a low-level feature adaptation stage that leads to more accurate segmentation masks in the target domain.

2.1 Related works

2.1.1 Transfer Learning and Task Transfer

Collecting training data is often expensive, time-consuming, or even unrealistic in many scenarios. Many works have tackled this problem by exploiting the existence of a relationship between the weights of CNNs trained for different tasks [49]. In particular, [50] showed that this strategy, referred to as transfer learning, can lead to better results than using random initialization even if applied on quite diverse tasks. Transfer learning has become a common practice, for instance, in object detection, where networks are usually initialized with Imagenet [51] classification weights [52, 53, 54, 55]. Additional insights on the transferability of learned representations between different visual tasks were provided in [19], where the authors present Taskonomy, a computational approach to represent a taxonomy of relationships among visual tasks. Along similar lines, [56] proposed to exploit the correlation between known supervised tasks and novel target tasks, in order to predict the parameters of models deployed to solve the target tasks starting from the parameters of networks trained on the known tasks.

2.1.2 Domain Adaptation

Domain adaptation techniques aim at reducing the performance drop of a model deployed on a domain different from the one the model was trained on [57]. Throughout the years, adaptation has been performed at different levels. Early approaches tried to model a shared feature space relying on statistical metrics such as MMD [58, 59]. Later, some works proposed to align domains by adversarial training [60, 61, 62]. Recently [63] noticed that, for classification tasks, aligning feature norms to an arbitrarily large value results in better transferability across domains. Generative adversarial networks [64] have also been employed to perform image-to-image translation between different domains [65, 66, 67], and, in particular, to render cheaply labeled synthetic images similar to real images from a target domain. However, when dealing with dense tasks such as semantic segmentation, feature-based domain adaptation approaches tend to fail. Thus, several approaches to address domain adaptation for dense tasks, such as semantic segmentation [68, 42, 69, 70, 71, 72, 73, 74, 75, 76, 77] or depth estimation [78, 79, 80] have been proposed recently. Akin to UDA methods, we learn from a labeled source domain to perform well on a different target domain. However, unlike the classical UDA setting, we assume the existence of an additional task where supervision is available for both domains.

2.1.3 Multi-task Learning

The goal of multi-task learning is to solve many tasks simultaneously. By pursuing this rather than solving the tasks independently, a neural network may use more information to obtain more robust and reliable predictions. Many works try to tackle several tasks jointly [81, 82, 83, 84]. For example, [83] showed that by learning to correctly weigh each task loss, multi-task learning methods can outperform separate models trained individually. [70, 84] show how learning multiple perception tasks jointly while enforcing geometrical consistency across them can lead to better performances for almost all tasks. Recently, [85] proposes a method to improve the performances of multiple single-task networks by imposing consistency across them during training. Finally, Taskonomy [19] investigates the relationship between the deployed tasks to accomplish multi-task learning effectively.

2.1.4 Task Transfer and Domain Adaptation

Most existing approaches address independently either task transfer or domain adaptation. Yet, a few works have proposed to tackle these two problems jointly. [86] was the first paper to propose a cross-tasks and cross-domains adaptation approach, considering as tasks different image classifications problems. UM-Adapt [87], instead, learns a cross-task distillation framework with full supervision on the source domain and deploys such framework on the target domain in a fully unsupervised manner, while minimizing adversarially the discrepancy between the two domains.

2.1.5 Semantic Segmentation Datasets

As explained in Sec. 1.1, we aim at transferring knowledge from a source domain to a target domain. Since collecting a large amount of labeled data is often time-consuming and expensive for tasks such as semantic segmentation, the source domain is typically obtained leveraging computer graphics. This not only provides the advantage of collecting data in a convenient way, but it also allows the possibility to extract different forms of supervision such as depth maps that can be used at training time to boost UDA algorithms. Examples of such datasets that will be used in the next chapters are GTA5 [88] and SYNTHIA [89]. The former is one of the largest available synthetic datasets for autonomous driving scenarios, and it contains 24,966 annotated images of 1914×1052 resolution extracted from the popular video-game Grand Theft Auto V (GTAV). As regards as SYNTHIA, multiple subsets have been released with different characteristics. In Chapter 4, we adopt the SYNTHIA VIDEO

SEQUENCES (SYNTHIA-SEQ) subset as video sequences are required to train our method, while for the experiments in Chapter 5 we utilize the SYNTHIA-RAND-CITYSCAPES (SYNTHIA) subset, which is a collection of 9,400 synthetic images with resolution 1280 $\times$ 760. For the experiment conducted in Chapter 3, we even collect our own synthetic dataset exploiting the Carla simulator [90], which is composed of 3500, 500, and 1000 images for training, validation, and testing respectively. The real dataset used in this dissertation for addressing UDA for 2D semantic segmentation are instead the Cityscapes dataset [91] and the NTHU [92]. The Cityscapes dataset is a high-quality collection of real images of 2048 $\times$ 1024 resolution acquired in several German cities. It is composed of 2975 and 500 images for the training and validation split, respectively. The NTHU dataset is a collection of images taken from four different cities with 2048 $\times$ 1024 resolution: Rio, Rome, Tokyo, and Taipei. For each city, 3200 unlabeled images are available for the adaptation phase, and 100 labeled images for the evaluation.

Chapter 3

Transfer Features Across Tasks and Domains.

This chapter expands upon the foundations laid by the AT/DT framework [48]. Before delving into our own contributions, we briefly summarize the key concepts of this prior work. At a high level, AT/DT tries to transfer features learned across tasks within a source domain in a supervised fashion, and then apply this mapping to a target domain, where only partial supervision is available. The key point is that the source domain can be synthetic, hence easy and cheap to generate, even with labels for complex tasks. More specifically, the core idea is to lean a mapping function $G_{1\to 2}$ (colored in yellow in Fig. 3.1) in feature space between two tasks in a given domain, so that the same mapping can be applied in another domain. More precisely, the architecture foresees a classical encoder-decoder architecture that is used to solve independently $\mathcal{T}_1$ and $\mathcal{T}_2$. These two networks are referred as to $N_1 = D_1(E_1(x))$ (red network in Fig. 3.1) and $N_2 = D_2(E_2(x))$ (green network in Fig. 3.1) respectively. Since we assume to have complete supervision for $\mathcal{T}_1$, N_1 is trained with images belonging to both domains. N_2 is of course only trained with synthetic images since we do not have labels for $\mathcal{B}$ in $\mathcal{T}_2$. Up to this point, we obtained an encoder $E_1(x)$ capable of extracting depth features $f_1 = E_1(x)$ given both real and synthetic images, and an encoder $E_2(x)$ capable of encoding deep semantic features $f_2 = E_2(x)$. The final step is thereby to train the transfer network $G_{1\to 2}$ to map depth features into semantic segmentation features: $G_{1\to 2} : f_1 \to f_2$. Considering that N_2 is trained on $\mathcal{A}$, and due to the domain shift, $E_2(x)$ can only work reasonably well on the domain it has been trained on. Hence, $G_{1\to 2}$ is optimized on $\mathcal{A}$ as well.

To solve $\mathcal{T}_2$, we can now extract depth features from a natural image, convert them into features for the downstream task and feed them to the corresponding decoder. The whole protocol can be summarized in the following steps:

1. Learn to solve task $\mathcal{T}_1$ on domains $\mathcal{A}$ and $\mathcal{B}$.
2. Learn to solve task $\mathcal{T}_2$ on domain $\mathcal{A}$.
3. Train $G_{1\to 2}$ on domain $\mathcal{A}$.

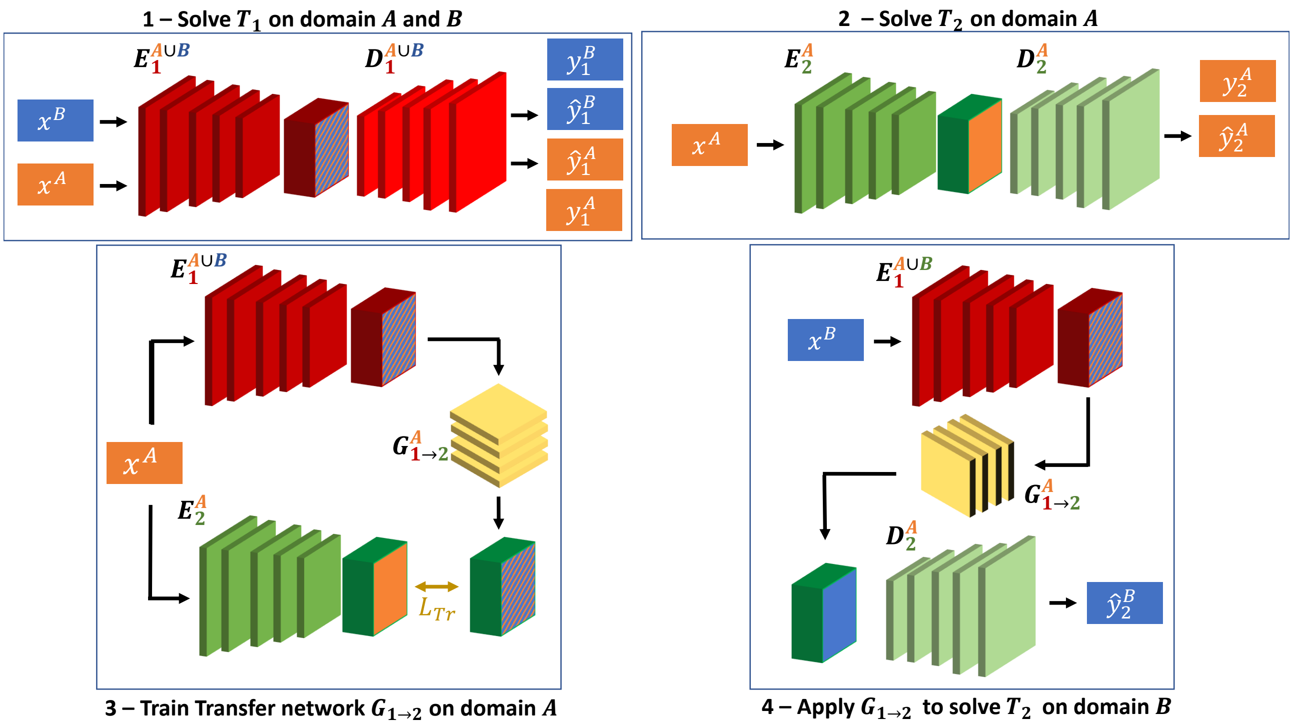


FIGURE 3.1. AT/DT framework [48]

4. Apply $G_{1\to2}$ to solve $\mathcal{T}_2$ on domain $\mathcal{B}$.

We describe in the next sections the additional improvement made to AT/DT to achieve better transferability across tasks and domains. A graphical overview of the additional improvements is depicted in Fig. 3.2.

3.1 Extended AT/DT

3.1.1 Feature Alignment Across Domains

In order to achieve good performances, it is crucial to make $G_{1\to2}$ generalize well in a target unseen domain $\mathcal{B}$ even if trained only source data from $\mathcal{A}$. Domain Adaptation (DA) literature already offers several ways to accomplish this. One may act on the input space [66], on the feature space [62] or on the output space of the network [93]. In our case though, both input and output space of $G_{1\to2}$ are high dimensional latent spaces and, as reported in [93], unsupervised domain adaptation techniques tend to fail when applied to such spaces for dense tasks. However, we can address the domain shift directly on the input space of $G_{1\to2}$ since in our framework it boils down to the feature space of N_1, where partial alignment is already achieved by simultaneous supervised training on $\mathcal{A}$ and $\mathcal{B}$. We can further diminish the domain-shift between the two feature spaces by regularizing them, enforcing that f_1 features extracted from E_1 in $\mathcal{A}$ and $\mathcal{B}$ have similar L_2 norms across channels. We preserve

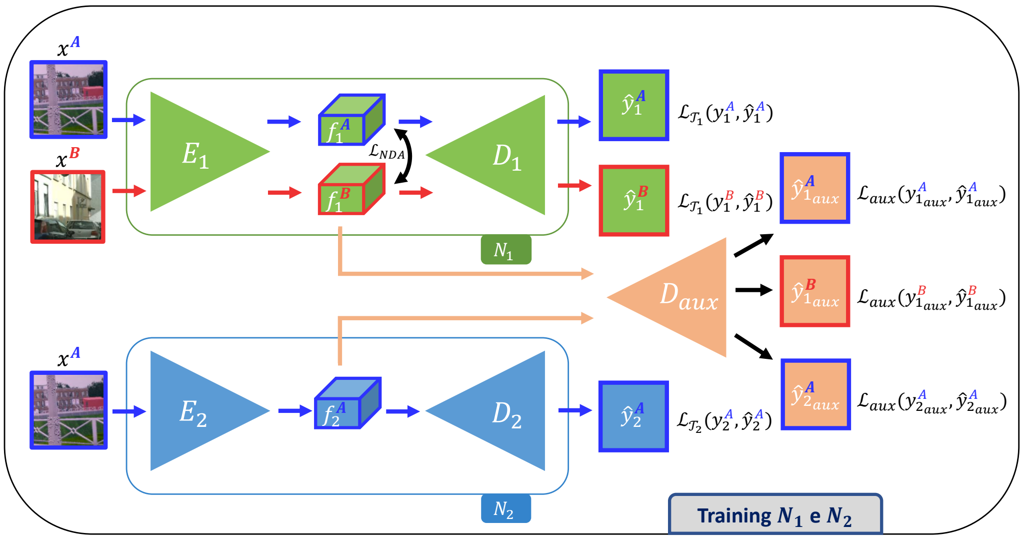

x^A
x^B
x^A
E_1
f_1^A
f_1^B
$\mathcal{L}_{NDA}$
D_1
N_1
$\hat{y}_1^A$
$\hat{y}_1^B$
$\mathcal{L}_{\mathcal{T}_1}(y_1^A, \hat{y}_1^A)$
$\mathcal{L}_{\mathcal{T}_1}(y_1^B, \hat{y}_1^B)$
D_{aux}
$\hat{y}_{1aux}^A$
$\hat{y}_{1aux}^B$
$\hat{y}_{2aux}^A$
$\mathcal{L}_{aux}(y_{1aux}^A, \hat{y}_{1aux}^A)$
$\mathcal{L}_{aux}(y_{1aux}^B, \hat{y}_{1aux}^B)$
$\mathcal{L}_{aux}(y_{2aux}^A, \hat{y}_{2aux}^A)$
E_2
f_2^A
D_2
N_2
$\hat{y}_2^A$
$\mathcal{L}_{\mathcal{T}_2}(y_2^A, \hat{y}_2^A)$
Training N_1 e N_2

FIGURE 3.2. Features alignment strategies across tasks and domains. We train jointly the networks N_1, N_2 and a shared auxiliary decoder D_{aux}. We train N_1 to solve $\mathcal{T}_1$ on images from domains $\mathcal{A}$ and $\mathcal{B}$ using a supervised loss $\mathcal{L}_{\mathcal{T}_1}$ for $\mathcal{T}_1$ alongside a novel feature Norm Discrepancy Alignment loss $\mathcal{L}_{NDA}$ which helps better aligning the features computed by N_1 across the two domains. We train N_2 using a supervised loss $\mathcal{L}_{\mathcal{T}_2}$ for $\mathcal{T}_2$ on images from $\mathcal{B}$. D_{aux} is trained to solve an auxiliary task $\mathcal{T}_{aux}$ using the loss $\mathcal{L}_{aux}$ and based on the features computed by E_1 on images from $\mathcal{A}$ and $\mathcal{B}$ as well as by E_2 on images from $\mathcal{B}$.

spatial information while calculating the norms assuming that the two domains contains scenes with similar structure (as it is the case for autonomous driving applications). To visualize this property we select a synthetic domain $\mathcal{A}$ CARLA [90], and a real domain $\mathcal{B}$ Cityscapes [91]. Then, we count for each pixel location the number of occurrences of each class. We show the result of this experiment in Fig. 3.3, using a *viridis* colormap to display these occurrency maps for each class and for both domains $\mathcal{A}$ and $\mathcal{B}$. We can clearly see that the maps have a structure similar across domains, e.g., building are concentrated in the top image regions.

Thereby, starting from features f_1^A and f_1^B of dimensions $H \times W \times C$, where H, W and C are the height, width and channels of the feature maps respectively, we calculate the L_2 norm along the C axis and we minimize the absolute difference between each spatial location i, j along the H and W dimensions. Formally our NDA Loss is defined as follows:

$$L_{NDA} = \frac{1}{W \times H} \sum_{i=1}^{H} \sum_{j=1}^{W} |||f_{1_{i,j}}^A||_2 - ||f_{1_{i,j}}^B||_2| \tag{3.1}$$

3.1.2 Feature alignment across tasks

We have previously shown a practical solution to improve the generalization across domains of our mapping. However, we want to go a step further and align features also across tasks to ease the learning of a mapping function. We believe that, f_1 should contain as

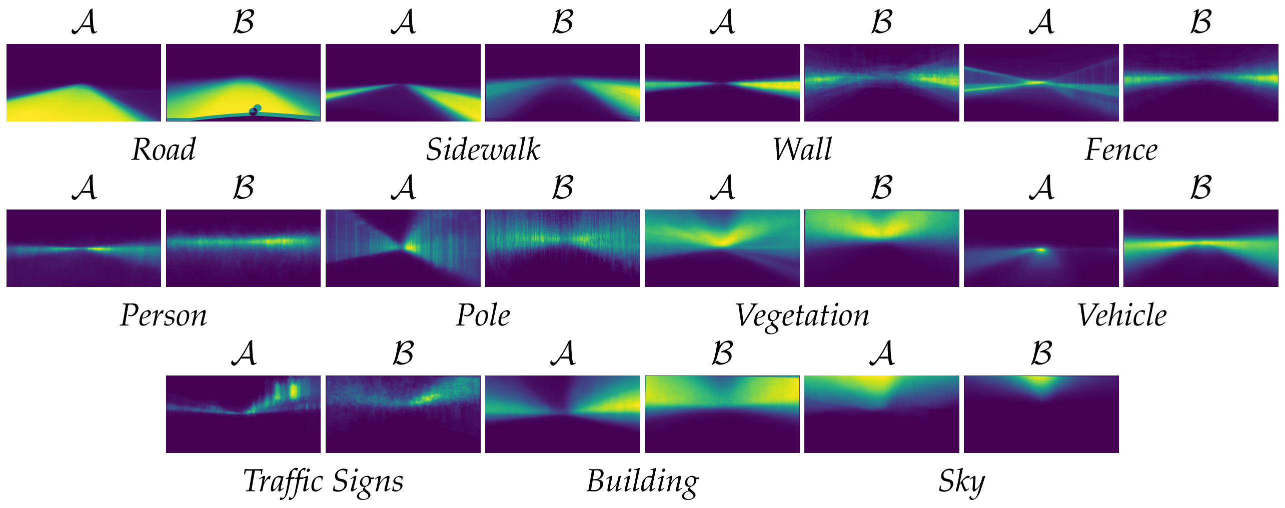


FIGURE 3.3. Spatial Priors Similarities Across Domains. Considered the semantic segmentation task, we compute the number of occurrences of each class at each pixel location for both domains. Domain $\mathcal{A}$ is CARLA, $\mathcal{B}$ is Cityscapes. We visualize the occurrence maps with a *viridis* colormap.

much information as possible, even if they are not strictly needed to solve $\mathcal{T}_1$, because they could be useful for $\mathcal{T}_2$. For this reason, while training networks for $\mathcal{T}_1$, we simultaneously train a decoder to solve an auxiliary task $\mathcal{T}_{aux}$ to enrich representations. However, though multi-task learning of $\mathcal{T}_1$ and $\mathcal{T}_{aux}$ can help to encode more relevant information in the T_1 features f_1, it does not guarantee that the decoder D_2 used at inference time on the transferred features from T_1 to T_2, $f_{1\to2}$, can make proper use of them if it has been trained only to solve $\mathcal{T}_2$ in isolation. $\mathcal{T}_{aux}$ can be used to this end and learn to solve it with the same decoder D_{aux} also from features f_2 computed by E_2.

In detail, given auxiliary task labels Y^A_{aux} and Y^B_{aux} for $\mathcal{A}$ and $\mathcal{B}$, we train N_1 and N_2 simultaneously with an auxiliary decoder D_{aux} using an auxiliary loss $\mathcal{L}_{aux}$. Therefore, we obtain auxiliary predictions in the following way: $y_{aux,k} = D_{aux}(E_k(x)), k \in [1,2]$. Again, we feed images of both domains through E_1, while we pass only images from $\mathcal{A}$ through E_2. We do not pass images belonging to $\mathcal{B}$ through E_2 while training D_{aux} since this would be the only supervision for E_2 on $\mathcal{B}$ and it may skew E_2 output to be more effective on $\mathcal{T}_{aux}$ than on $\mathcal{T}_2$.

3.2 Experimental Settings

Tasks. We fix $\mathcal{T}_1$ and $\mathcal{T}_2$ to be monocular depth estimation or semantic segmentation. These two visual tasks can be addressed using the same base architecture and changing only the final layer. Semantic segmentation is solved by minimizing a cross-entropy loss, monocular depth estimation by minimizing a L_1 loss. We select edge detection as our $\mathcal{T}_{aux}$ since this task has many advantages. First, from an implementation point of view, it can be solved using again the same decoder as $\mathcal{T}_1$ and $\mathcal{T}_2$. Second, since our main goal is to

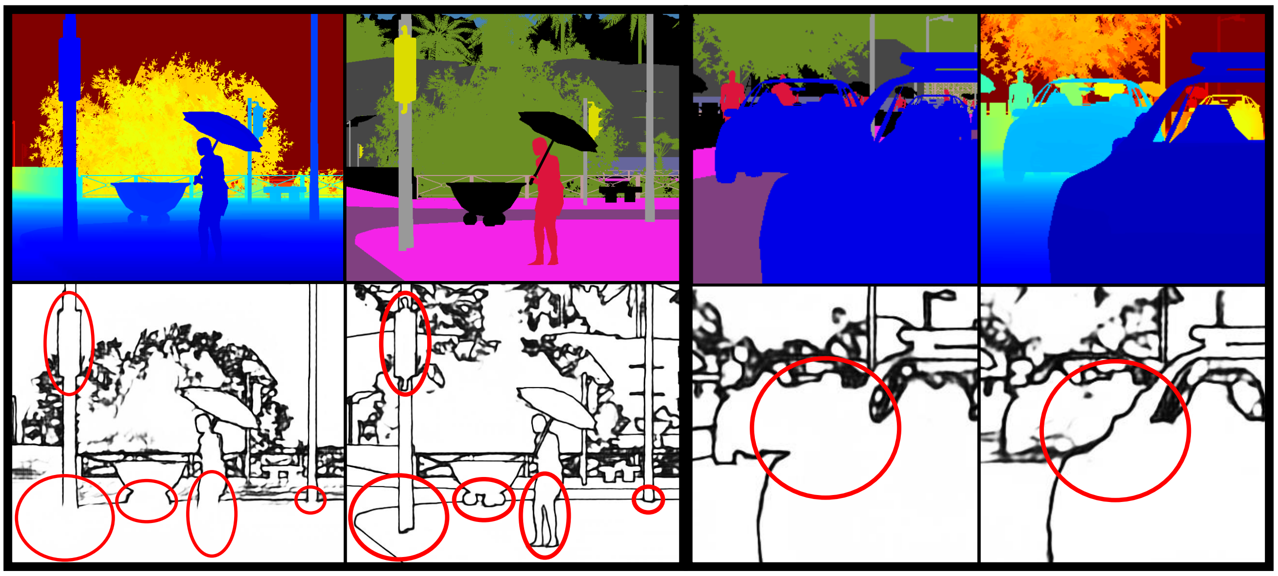


FIGURE 3.4. Two task transfer scenarios: on the left, the depth-to-semantic case; the opposite on the right. Red circles highlight image details that are needed to perform the second task but are not present in the first one. We propose to incorporate relevant details of the scene within deep features by exploiting an auxiliary edge detection task, with the aim of making these representations easier to transfer across tasks and domains.

improve the transferability of features among tasks, we force features for $\mathcal{T}_1$ to contain as many details as possible of the scene, even if they are not strictly needed to solve $\mathcal{T}_1$. To make a concrete example, we can think about the case of $\mathcal{T}_1$ being depth estimation and $\mathcal{T}_2$ semantic segmentation. Features f_1 used to compute depth can ignore boundaries between semantically distinct regions of the image that are not needed to correctly predict depth, as shown in Fig. 3.4 (*e.g.*, legs or tires touching the ground, or between street signs and poles). Therefore, even if fed to a perfect $G_{1\rightarrow 2}$, f_1 may not contain all the information needed to restore the semantic structure of the image. By solving edge detection, instead, we force the network to extract additional information from the image, not normally encoded when training depth features in isolation. We define L_{aux} as a L_2 loss for training the edge decoder.

Datasets. To assess our contributions, we set $\mathcal{A}$ and $\mathcal{B}$ to be the synthetic and real datasets, respectively. We use as $\mathcal{A}$ a collection of images generated with the Carla simulator, while as $\mathcal{B}$ the Cityscapes dataset. We generate a new version of the Carla dataset w.r.t. the one used to evaluate the original AT/DT to reduce the gap between synthetic and real scenes. For each image, we store the associated depth and semantic labels easily provided by the simulator. Since for the Cityscapes dataset only the semantic labels are provided, we use depth maps obtained with the SGM stereo algorithm [94], by filtering the erroneous predictions in the generated disparities with a left-right consistency check. This can be considered as an added value because it shows the ability to transfer knowledge when learning from noisy labels, although a stereo setup is required. Finally, we use a pre-trained state-of-the-art neural network[95] (trained on datasets different from $\mathcal{A}$ and $\mathcal{B}$) as an off-the-shelf black-box edge detector to extract the edges from both $\mathcal{A}$ and $\mathcal{B}$ to be used as proxy labels when learning $\mathcal{T}_{aux}$.

Architecture. We use the same architecture as described in [48] to solve each task. The two encoders are also used to capture good features for edge detection, which is solved using D_{aux}, which shares the same architecture as the decoders used in N_1 and N_2. $G_{1\to2}$ is a simple CNN made out of 6 pairs of convolutional and batch normalization layers with kernel size 3×3 which, differently from the original AT/DT version, do not perform any downsampling or upsampling operation.

Evaluation protocol. During the first step of the training, *i.e.*, training N_1 and N_2, we found to be extremely relevant to monitor the performance of the two networks. Indeed, we have observed that the more effective is N_2 on the downstream task, the higher is the final performance of our method and that the same reasoning may be applied on N_1 when trained on $\mathcal{A}$ and $\mathcal{B}$: better results lead to a superior domain adaptation method.

During the training phase of the transfer network, the model is evaluated on the validation set of Carla. Of course, it is possible that the global optimum for Carla is not the global optimum for Cityscapes. Yet, we cannot use data from the target domain neither for hyper-parameters tuning nor for early stopping, because these data would not be available in a real-case scenario. Therefore, the Cityscapes validation set is only used at test time to measure the final performances of our adaptation method.

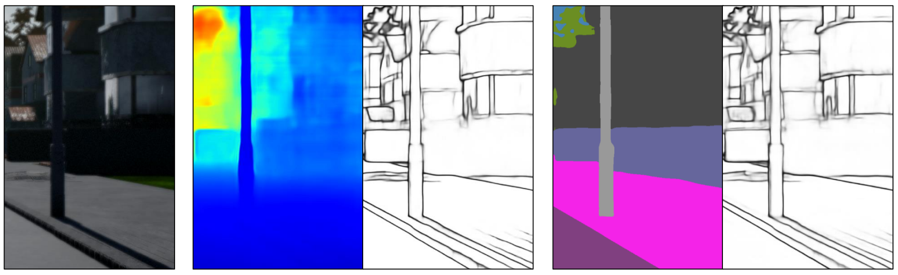


FIGURE 3.5. From left to right: RGB input image of domain $\mathcal{A}$, depth prediction from N_1, edges from f_1, semantic segmentation from N_2 and edges from f_2. Task features f_1 and f_2 encode richer details than strictly needed to solve either task as we can recover all edges from both of them.

3.3 Experimental Results

We provide results for two different settings: transferring features from depth estimation to semantic segmentation (Sec. 3.3.1) as well as from semantic segmentation to depth estimation (Sec. 3.3.2).

In both scenarios, as already mentioned, we used edge detection as an auxiliary task, motivated by the idea that both semantic segmentation and depth estimation can benefit

$\mathcal{A}$	$\mathcal{B}$	Method	Road	Sidewalk	Walls	Fence	Person	Poles	Vegetation	Vehicles	Tr. Signs	Building	Sky	**mIoU**	**Acc**
Carla	CS	Source	78.99	38.81	1.34	5.80	24.02	24.47	71.98	52.23	5.57	65.17	59.10	38.86	78.58
Carla	CS	**AT/DT**	**90.57**	**48.46**	**7.37**	**12.27**	**41.16**	**31.90**	**81.96**	**72.77**	**23.44**	**77.85**	**76.33**	**51.28**	**87.57**
CS	CS	Transfer Oracle	89.69	48.05	11.46	29.58	59.68	35.84	85.83	85.57	34.03	78.17	85.54	58.50	88.84
-	CS	Oracle	96.74	78.28	29.26	40.78	72.39	51.28	90.69	91.94	58.92	86.33	89.23	71.44	93.90

TABLE 3.1. Experimental results of *Dep.* $\rightarrow$ *Sem.* scenario. Source stands for N_2 trained on $\mathcal{A}$ and tested on $\mathcal{B}$, Transfer Oracle represents $G_{1\rightarrow 2}$ trained only on $\mathcal{B}$, Oracle refers to N_2 trained and tested on $\mathcal{B}$. Best results highlighted in bold.

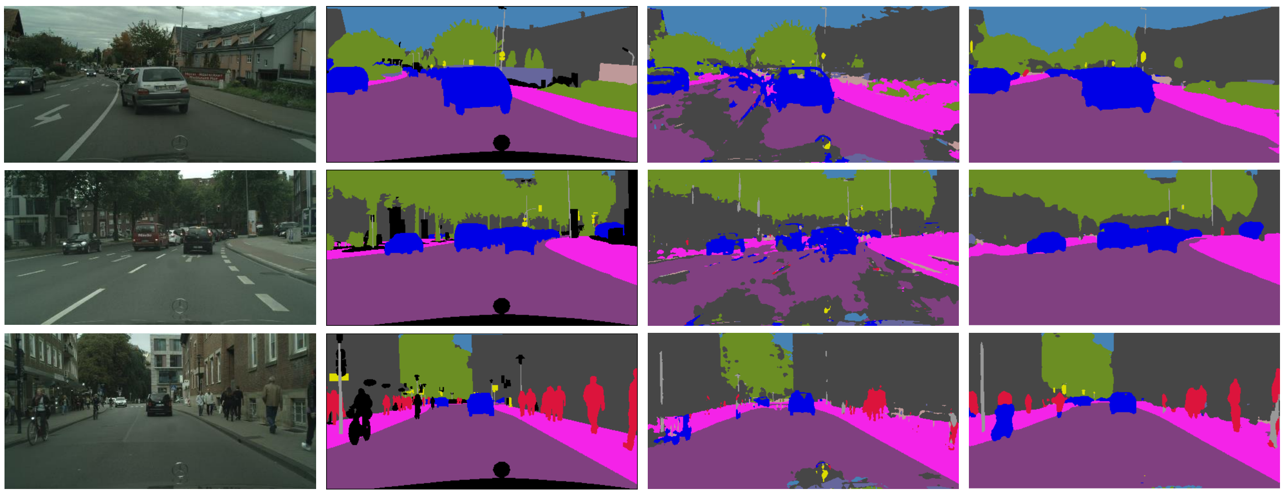


FIGURE 3.6. Qualitative results of the *Dep.* $\rightarrow$ *Sem.* scenario. From left to right: RGB image, ground truth, baseline trained only on domain $\mathcal{A}$, ours.

from edge information. Fig. 3.5 shows that with our multi-task learning protocol, we are able to restore all the details of the scene from both f_1 and f_2, proving that N_1 and N_2 learned to encode richer information than strictly needed to solve $\mathcal{T}_1$ and $\mathcal{T}_2$.

			Lower is better				Higher is better		
$\mathcal{A}$	$\mathcal{B}$	Method	Abs Rel	Sq Rel	RMSE	RMSE log	δ_1	δ_2	δ_3
Carla	CS	Source	0.7398	15.169	14.774	0.641	**0.406**	0.650	0.781
Carla	CS	**AT/DT**	**0.3928**	**4.9094**	**12.363**	**0.444**	0.372	**0.757**	**0.923**
CS	CS	Transfer Oracle	0.2210	2.2962	9.032	0.275	0.669	0.914	0.972
-	CS	Oracle	0.1372	1.6214	8.566	0.244	0.816	0.938	0.976

TABLE 3.2. Experimental results of *Sem.* $\rightarrow$ *Dep.* scenario. Source stands for N_2 trained on $\mathcal{A}$ and tested on $\mathcal{B}$, Transfer Oracle represents $G_{1\rightarrow 2}$ trained only on $\mathcal{B}$, Oracle refers to N_2 trained and tested on $\mathcal{B}$. Best results highlighted in bold.

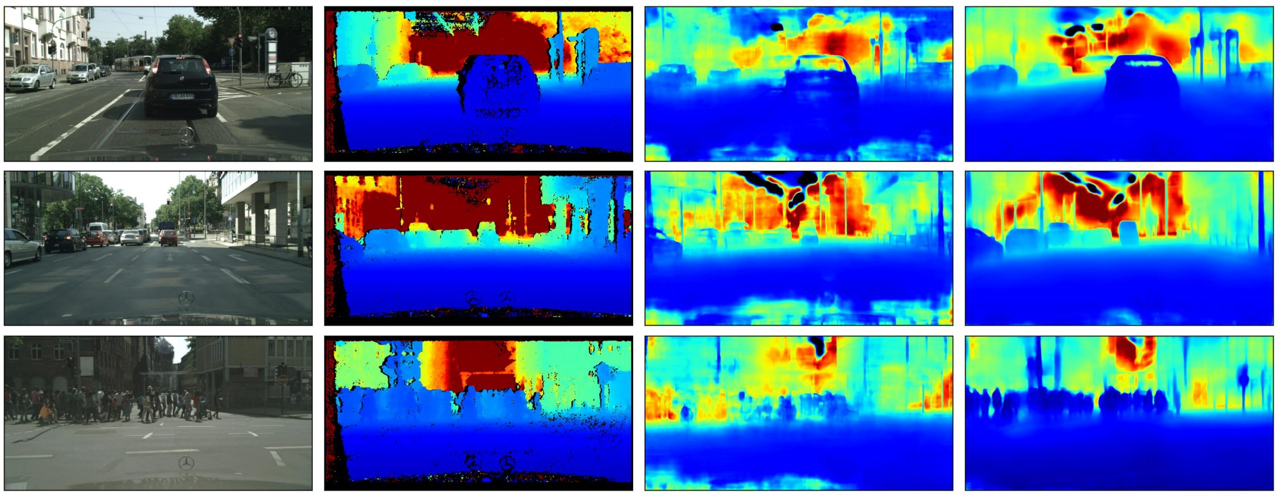


FIGURE 3.7. Qualitative result of the *Sem.* $\rightarrow$ *Dep.* scenario. From left to right: RGB image, ground truth, baseline network trained only on domain $\mathcal{A}$, ours.

3.3.1 Depth to Semantics

In this setup, denoted as *Dep.* $\rightarrow$ *Sem.*, the goal of our framework is to transform depth features into semantic segmentation features. This mapping is learned using Carla as domain $\mathcal{A}$ and Cityscapes as domain $\mathcal{B}$. We report results in Tab. 3.1: the first row shows results obtained with no adaptation (*i.e.*, training N_2 on Carla and testing it directly on Cityscapes), while from the second row we can see that our final framework yields 51.28% mIoU and 87.57% Acc with an improvement of +12.48% and +8.99% respectively in terms of mIoU and Acc wrt to the baseline.

Furthermore, as we are transferring features from another task, it is worth trying to investigate on the upper bound in performance due to the inherent transferability of the features between the two tasks. Thereby, we train $G_{1\rightarrow 2}$ using only Cityscapes to learn a mapping function in a supervised fashion as explained in Sec. 3.1.2 on $\mathcal{B}$ and testing on the validation set of $\mathcal{B}$. These results are shown in the third row of the table (denoted as Transfer Oracle): given a transfer architecture, there seems to be an upper bound in performance due to the nature of the two tasks, which in the considered setting amounts to a 58.5% mIoU. Thus, our proposal shows a gap that is only about -7.2% mIoU. We also report the performance of N_2 trained on $\mathcal{B}$ and tested on $\mathcal{B}$ to show the absolute upper bound (last row of the table, denoted as Oracle).

Some qualitative results dealing with the *Dep.* $\rightarrow$ *Sem.* scenario are depicted in Fig. 3.6. It is possible to appreciate the overall improvement of our method wrt the baseline, either in flat areas (*e.g.*, roads, sidelwalks and walls), in objects shapes (*e.g.*, cars and persons) or in fine-grained details (*e.g.*, poles and traffic signs).

3.3.2 Semantics to Depth

In this setup, which we define as *Sem.* $\rightarrow$ *Dep.*, the goal of our framework is to transform semantic features into depth features. This mapping is learned using Carla as domain $\mathcal{A}$ and Cityscapes as domain $\mathcal{B}$.

Results are reported in Tab. 3.2. Similarly to the *Dep.* $\rightarrow$ *Sem.* scenario, in the first row we show results with no adaptation (denoted as Source), while the second row presents the ones obtained with our framework. In Fig. 3.7, we show some qualitative results of the *Sem.* $\rightarrow$ *Dep.* scenario. While predictions look quite noisy in the background, we can see a good improvement in the foreground area thanks to our method. Shapes are recovered almost perfectly, both for big and small objects, even with difficult subjects like the crowd in the bottom row. Additionally, our method enables a remarkable enhancement of the prediction smoothness.

3.4 Additional Experiments

In the following sections, we study the effectiveness of each implementation choice.

3.4.1 Contribution of $\mathcal{T}_{aux}$ and NDA Loss

We start by studying the effect of introducing in our framework the auxiliary task and the NDA Loss, analyzing their contribution either when used separately or combined together. The second and the third row of Tab. 3.3 report results obtained in the *Dep.* $\rightarrow$ *Sem.* setting respectively when integrating in our method exclusively the auxiliary task (*i.e.*, edge detection) or the NDA loss. We can see that both techniques bring in an improvement of about +2% in

$\mathcal{A}$	$\mathcal{B}$	Edge	NDA	Road	Sidewalk	Walls	Fence	Person	Poles	Vegetation	Vehicles	Tr. Signs	Building	Sky	**mIoU**	**Acc**
Carla	CS			89.95	46.77	5.16	10.21	28.93	28.92	77.50	71.37	19.24	75.29	75.12	48.04	85.90
Carla	CS	✓		90.12	48.90	4.18	11.63	37.40	31.98	**82.34**	71.50	15.11	**78.04**	**80.61**	50.16	87.21
Carla	CS		✓	**91.21**	**50.16**	5.14	**13.78**	36.99	**32.10**	77.72	**73.38**	**23.47**	76.67	72.67	50.30	86.77
Carla	CS	✓	✓	90.57	48.46	**7.37**	12.27	**41.16**	31.90	81.96	72.77	23.44	77.85	76.33	**51.28**	**87.57**

TABLE 3.3. Ablation study in the *Dep.* $\rightarrow$ *Sem.* scenario. Best results highlighted in bold. Edge refers to the framework trained with our details-aware features. NDA refers to the framework trained with our NDA loss.

terms of mIoU wrt to the plain version of AT/DT (first row). Interestingly, though, from the last row of the table we can see that edge detection and the NDA loss result complementary when combined, providing an overall improvement of +3.34% mIoU.

Fig. 3.8 presents some zoomed-in qualitative results: we can see how small details such as poles or car shapes are recovered with our method wrt results obtained without $\mathcal{T}_{aux}$ and NDA loss.

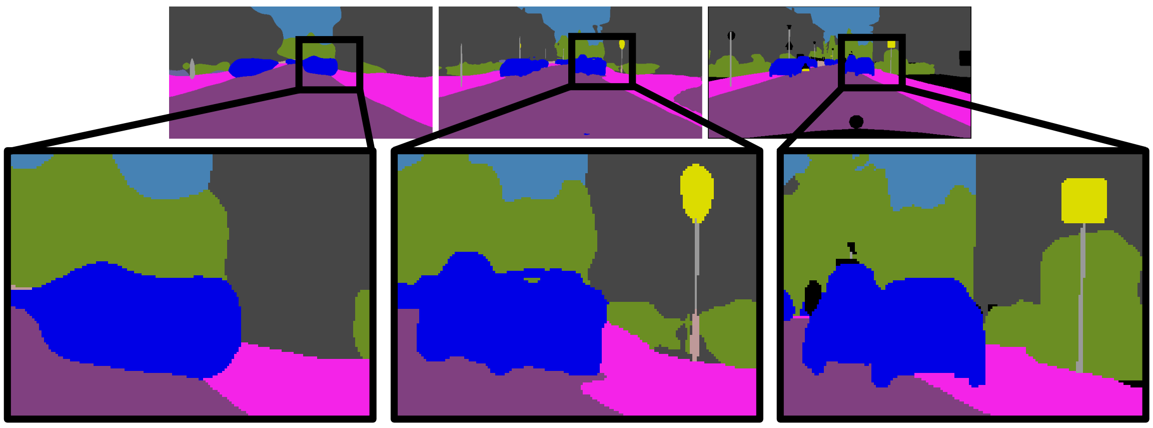


FIGURE 3.8. Zoomed results in a *Dep.* $\rightarrow$ *Sem.* scenario. From left to right: plain AT/DT without edge and NDA, our complete framework, ground truth. We notice how our method is able to recover fine-grained details of the output.

3.4.2 Effectiveness of edge detection as auxiliary task

In this section, we show empirically that the choice of the proper auxiliary task is key to the performance of our framework.

In both the *Dep.* $\rightarrow$ *Sem.* and the *Sem.* $\rightarrow$ *Dep.* scenarios, we proposed to use edge detection as an auxiliary task because it captures information about the shapes of the objects in the input images, and allows for the straightforward computation of proxy labels. To validate this design choice, we tested our framework in the *Dep.* $\rightarrow$ *Sem.* setting, using D_{aux} to simply reconstruct the input images both from f_1 and f_2, *i.e.*, the classical autoencoder

Aux Task	Road	Sidewalk	Walls	Fence	Person	Poles	Vegetation	Vehicles	Tr. Signs	Building	Sky	**mIoU**	**Acc**
None	89.95	46.77	5.16	10.21	28.93	28.92	77.50	71.37	**19.24**	75.29	75.12	48.04	85.90
Autoencoder	**90.68**	**50.12**	**7.45**	9.08	31.40	29.43	78.72	68.51	12.95	74.67	75.68	48.07	86.31
Edge detection	90.12	48.90	4.18	**11.63**	**37.40**	**31.98**	**82.34**	**71.50**	15.11	**78.04**	**80.61**	**50.16**	**87.21**

TABLE 3.4. Comparison between autoencoder and edge detection as auxiliary tasks in the *Dep.* $\rightarrow$ *Sem.* scenario. Best results highlighted in bold.

setting (results in Tab. 3.4). Interestingly, using a reconstruction task as the auxiliary task achieves comparable performances in terms of mIoU over plain AT/DT. We believe that the autoencoder is a trivial task that does not require the extraction of informative features about the image, therefore not providing any additional cues to the downstream task.

3.4.3 Importance of simultaneous training of N_1, N_2 and D_{aux}

In our experiments, we use edge detection as an auxiliary task and train a shared decoder D_{aux} to reconstruct edges of the input image from features extracted by both E_1 and E_2. In fact, we argue that this procedure should force E_1 to encode in the extracted features also edges that are not necessary for $\mathcal{T}_1$ but could be relevant for $\mathcal{T}_2$. Besides, we believe that simultaneous training of N_1, N_2 and D_{aux} is crucial to encourage features coming from E_1 and E_2 to encode edge information in the same way, making it easier to learn $G_{1\rightarrow 2}$.

In Tab. 3.5 we report the ablation study conducted to validate these intuitions. We consider the *Dep.* $\rightarrow$ *Sem.* scenario using the Carla dataset as domain $\mathcal{A}$ and Cityscapes as domain $\mathcal{B}$. The four rows of the table represent the following training schemes:

1. We report the results obtained by *plain* AT/DT (*i.e.*, trained without $\mathcal{T}_{aux}$ and NDA loss) as a baseline.
2. We first train N_1 and D_{aux} on both $\mathcal{A}$ and $\mathcal{B}$. Then, we train N_2 on $\mathcal{A}$. Finally, we train $G_{1\rightarrow 2}$ on features extracted by E_1 and E_2 on domain $\mathcal{A}$.
3. We train N_1 and a first D^1_{aux} on both $\mathcal{A}$ and $\mathcal{B}$. Then, we train N_2 and a second D^2_{aux} on $\mathcal{A}$. Finally, we train $G_{1\rightarrow 2}$ on features extracted by E_1 and E_2 on domain $\mathcal{A}$
4. Our proposed method with a simultaneous training of N_1, N_2 and a shared D_{aux}.

The introduction of edge detection as auxiliary task helps in every scenario. In fact, if we use D_{aux} only during training of N_1 (second row), we already see an increase of 0.6% in the overall mIoU. We believe that this is explained by the presence of edge details (not strictly necessary to solve $\mathcal{T}_1$ but relevant for $\mathcal{T}_2$) in features extracted by E_1. However, $G_{1\rightarrow 2}$ can

method	Road	Sidewalk	Walls	Fendce	Person	Poles	Vegetation	Vehicles	Tr. Signs	Building	Sky	mIoU	Acc
plain AT/DT	89.95	46.77	5.16	10.21	28.93	28.92	77.50	71.37	**19.24**	75.29	75.12	48.04	85.90
Separate (N_1 + edge), N_2	87.24	43.30	3.08	10.17	41.77	29.04	81.81	72.35	16.58	77.10	73.10	48.69	85.89
Separate (N_1 + edge), (N_2 + edge)	88.83	47.31	**7.10**	8.59	**44.53**	30.99	**83.24**	**73.54**	18.05	**78.10**	69.66	49.99	86.72
Simultaneous (N_1 + N_2 + edge)	**90.12**	**48.90**	4.18	**11.63**	37.40	**31.98**	82.34	71.50	15.11	78.04	**80.61**	**50.16**	**87.21**

TABLE 3.5. Ablation study on the use of edge detection as auxiliary task. Best results highlighted in bold. See text for a detailed explanation of the training protocol used in each row.

E_1 Align.	Road	Sidewalk	Walls	Fence	Person	Poles	Vegetation	Vehicles	Tr. Signs	Building	Sky	mIoU	Acc
None	89.95	46.77	**5.16**	10.21	28.93	28.92	77.50	71.37	19.24	75.29	**75.12**	48.04	85.90
Adv.	89.89	46.01	4.22	11.89	**38.20**	30.65	77.00	63.68	12.99	74.35	81.16	48.19	85.42
NDA	**91.21**	**50.16**	5.14	**13.78**	36.99	**32.10**	**77.72**	**73.38**	**23.47**	**76.67**	72.67	**50.30**	**86.77**

TABLE 3.6. Comparison between NDA loss and adversarial training to align E_1 features. Best results highlighted in bold.

have difficulties in adapting f_1 into f_2 when the edge information is not explicitly present in f_2. This is confirmed by the result in the third row of the table, where an additional increase of 1.3% in the overall mIoU is attained by using two different D_{aux} (one during training of N_1 and one during training of N_2). Finally, the best results in terms of mIoU and Acc are achieved by our method, *i.e.*, when simultaneously training N_1, N_2 and a shared D_{aux}. This shows the benefit of encoding in the same way the edge information in f_1 and f_2 to enforce feature alignment across tasks.

3.4.4 Alignment strategies for N_1

An alternative way to align N_1 features between domains to ease the transfer process and favor the generalization of $G_{1\to 2}$, is to apply the widely used adversarial training in feature space. In our settings, this can be done by adding a critic that must discriminate whether the features produced by E_1 come from $\mathcal{A}$ or $\mathcal{B}$. Thus, the encoder E_1 not only has to learn a good feature space for its task, but it is also asked to fool the critic. Afterward, we can proceed to learn a mapping function $G_{1\to 2}$ among tasks as usual. In Tab. 3.6 we compare this standard DA methodology to our NDA loss. Adversarial training (second row) does not introduce significant improvements over not performing DA for $\mathcal{T}_1$ (first row, it even lowers the pixel-wise accuracy), while constraining the features extracted by E_1 in a norm aligned space (third row) significantly increases both metrics with respect to the baseline. Our intuition is that although adversarial training can be useful for domain alignment, it alters the learned feature space with the goal of fooling the critic, and due to the instability of adversarial learning, this training objective can lead to worse performances on the current task. Our NDA loss on the other hands, acts as a simple regularizer that tries to favor the generalization of the network, thereby it directly helps the network to perform better in the specific tasks.

3.4.5 Aligning N_2 features

E_2 Align.	Road	Sidewalk	Walls	Fence	Person	Poles	Vegetation	Vehicles	Tr. Signs	Building	Sky	**mIoU**	**Acc**
plain AT/DT	**89.95**	**46.77**	5.16	**10.21**	28.93	**28.92**	**77.50**	71.37	**19.24**	**75.29**	75.12	**48.04**	**85.90**
Adv.	89.36	46.03	**5.59**	8.22	**36.45**	25.44	75.15	**72.29**	12.69	74.12	**75.79**	47.38	85.31
NDA	44.94	23.82	3.81	2.09	30.74	24.21	42.08	68.84	11.69	35.67	11.10	27.18	56.17

TABLE 3.7. Results of aligning output space of E_2 in a *Dep.* $\rightarrow$ *Sem.* scenario. Best results highlighted in bold.

We tried to perform feature alignment across domains also on the features f_2 extracted by E_2, by either deploying adversarial training or imposing our NDA loss. The idea is to favor the generalization of $G_{1\rightarrow 2}$ by making not only alignment in its input space (*i.e.*, the features produced by E_1, aligned with our NDA loss) more homogeneous, but also its output space, *i.e.*, the features produced by E_2. However, the setting is not completely symmetric: when learning E_2, we do not have supervision available for $\mathcal{B}$, and the only loss shaping the feature space for its images is the alignment loss. We believe this to be the reason why aligning N_2 features turned out always detrimental to performance, as shown in Tab. 3.7 and discussed below.

In the first row, we report the results provided by *plain* AT/DT (no NDA, no aux), in the second those obtained by adversarial training on the features f_2 using the same procedure as described in the previous section for f_1. We can observe that not only adversarial training does not improve (like adversarial training applied to E_1) but it even decreases the overall mIoU of 0.7% compared to the baseline. Finally, in the third row, we report the results obtained by our NDA loss on f_2, like we did successfully on f_1: the NDA loss destroys the feature space of $\mathcal{T}_2$ when applied in this context, as vouched by the drop of 20% in the overall mIoU wrt to plain AT/DT.

We formulate the following hypothesis to explain these results: both adversarial training and NDA perform a comparison between $f_2^{\mathcal{A}}$ and $f_2^{\mathcal{B}}$. While $f_2^{\mathcal{A}}$ are shaped also by the supervision of $\mathcal{T}_2$, $f_2^{\mathcal{B}}$ are evolved only according to the additional loss we impose, as we do not have supervision for $\mathcal{T}_2$ on $\mathcal{B}$. However, E_2 is shared across domains, and therefore may be pushed to produce worse representations for both domains while it tries to accomplish the adversarial objective or the NDA loss minimization for $\mathcal{B}$. If this happens, mappings

Input Align.	Output Align.	Road	Sidewalk	Walls	Fence	Person	Poles	Vegetation	Vehicles	Tr. Signs	Building	Sky	**mIoU**	**Acc**
-	NDA	42.97	19.60	2.31	1.36	4.21	15.74	18.42	11.77	7.19	36.72	38.99	18.12	43.63
-	Adv	90.80	48.91	**6.16**	11.84	35.32	30.29	**78.78**	71.17	18.51	75.66	75.03	49.32	86.43
-	NDA + Adv	91.03	48.93	6.14	12.24	35.91	31.05	77.93	70.28	16.65	75.50	74.47	49.10	86.28
NDA	Adv	90.67	49.49	5.54	12.29	36.73	28.49	78.28	70.19	22.05	76.47	**76.35**	49.69	86.73
NDA	-	**91.21**	**50.16**	5.14	**13.78**	**36.99**	**32.10**	77.72	**73.38**	**23.47**	**76.67**	72.67	**50.30**	**86.77**

TABLE 3.8. Results of aligning input and output space of $G_{1\to2}$ in a *Dep.* $\to$ *Sem.* scenario. Best results highlighted in bold.

learned by $G_{1\to2}$ from $f_1^{\mathcal{A}}$ to $f_2^{\mathcal{A}}$ will hallucinate worse features for $\mathcal{T}_2$ on $\mathcal{B}$. To understand why adversarial training leads to a small decrease in performances compared to the use of NDA loss, we ought to consider that adversarial training implies a discriminator that cannot be easily fooled by totally degenerated features, while, without any additional constrain from task supervision, the NDA loss can yield totally collapsed representations.

3.4.6 Aligning $G_{1\to2}$ features

Although feature alignment did not turn out beneficial when training N_2, one may still expect to obtain better hallucinated features if the representations obtained when transferring $f_1^{\mathcal{A}}$ and $f_1^{\mathcal{B}}$ are aligned. We empirically found out that even though output space aligning strategies deployed when training $G_{1\to2}$ can lead to improvements in performance, input space alignment using our NDA loss deployed when training N_1 is more effective than them. Moreover, combining input and output space alignment techniques does not lead to further improvements. We performed this ablation study in the *Dep.* $\to$ *Sem.* scenario using Carla as $\mathcal{A}$ and Cityscapes as $\mathcal{B}$. Results of these experiments are reported in Tab. 3.8.

First, we applied our NDA loss to the output space of $G_{1\to2}$. Similarly to the previous section, we notice that, without having supervision on $\mathcal{B}$, the representations extracted from $G_{1\to2}$ for its images collapsed into a single value, yielding a drastic drop in transfer performance (row 1). We also tried to align the output-space features by training $G_{1\to2}$ alongside a discriminator in an adversarial fashion. We wanted to fool the discriminator in order to generate indistinguishable features from $\mathcal{A}$ or $\mathcal{B}$. We noticed that this strategy allow us to reach good overall performances with a 49.32 mIoU on Cityscapes (second row). Moreover, we thought that as adversarial training provides supervision on $\mathcal{B}$, using the NDA loss in combination with adversarial loss could avoid feature collapse for $\mathcal{B}$ while reaching a better overall alignment between $\mathcal{A}$ and $\mathcal{B}$. However, we notice that the combination of the two losses lead us to slightly worse results than adversarial training only (rows 2 vs 3). Finally, since we noticed that using adversarial loss on the output space lead us to good overall performances, we tested the combination of input space alignment, through NDA loss applied when training N_1, and output space alignment, through adversarial training for

$G_{1\to2}$. However, the combination of these two methods achieves worse performance than using only NDA alignment loss on input space (rows 5 vs 6).

3.5 Conclusions

We have introduced a framework able to effectively transfer knowledge between different tasks by learning an explicit mapping function between deep features. This mapping function can be parametrized by a neural network and shows interesting generalization capabilities across domains. To further ameliorate performance we have proposed two novel feature alignment strategies. At a domain level, we showed that the transfer function presented in our framework can be boosted by making its input space more homogeneous across domains with our simple yet effective NDA loss. At a task level, instead, we reported how deep features extracted for different tasks can be enriched and aligned with the introduction of a shared auxiliary task, which we implemented as edge detection in our experiments. We reported good results in the challenging synthetic to real scenario while transferring knowledge between the semantic segmentation and monocular depth estimation tasks. Our proposal is complementary to the whole domain adaptation literature and might be integrated with it.

Chapter 4

Plugging Monocular Depth into Unsupervised Domain Adaptation

Inspired by previous findings, we argue that monocular depth estimation could be an excellent task to gather additional knowledge useful to address semantic segmentation in the Unsupervised Domain Adaptation (UDA) settings. First of all, a monocular depth estimation network makes predictions based on 3D cues dealing with the appearance, shape, relative sizes, and spatial relationships of the stuff and things observed in the training images. This suggests that the network has to predict geometry by implicitly learning to understand the scene semantics. Indeed, other works [97, 98, 99] show that a monocular depth estimation network obtains better performances if forced to learn a semantic segmentation task jointly. Moreover, previous experiments in [48] show that depth can help semantics alike. Indeed, it is possible to learn a mapping in both directions between features learned to predict depth and per-pixel semantic labels. It is also worth observing how depth prediction networks tend to extract accurate information for regions characterized by repeatable and simple geometries, such as roads and buildings, which feature strong spatial and geometric priors (e.g. the road is typically a plane in the bottom part of the image) [100, 101, 102, 103]. Therefore, on the one hand, predicting accurately the semantics of such regions from depth information alone should be possible. On the other, a semantic network capable of reasoning on the scene geometry should be less prone to mistakes caused by appearance variations between synthetic and real images, the key issue in UDA for semantic segmentation.

Despite the above observations, injection of geometric cues into UDA frameworks for semantic segmentation has been largely unexplored in literature, except for a few proposals, which either assume the availability of depth labels in the real domain [104], a very restrictive assumption, or can leverage on depth information only in the synthetic domain due to availability of cheap labels [105, 106, 107]. In this respect, we set forth an additional consideration: nowadays, effective self-supervised procedures allow for training a monocular depth estimation network without the need for ground-truth labels [101, 108, 109].

Based on the above intuitions and considerations, in this chapter, we show that thanks to self-supervision, we can deploy depth information from both synthetic and *unlabelled real*

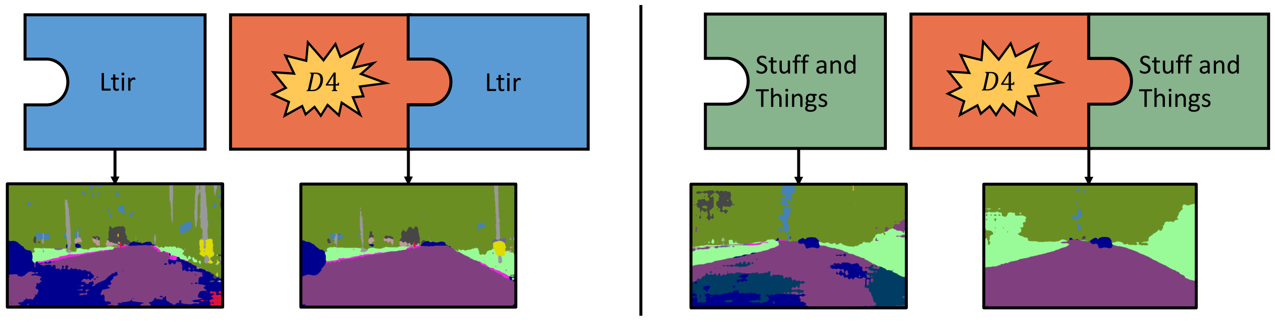


FIGURE 4.1. D4 can be plugged seamlessly into any existing method to improve UDA for Semantic Segmentation. Here we show how the introduction of D4 can ameliorate the performance of two recent methods like Ltir [27] and Stuff and Things [96].

images in order to inject geometric cues in UDA for semantic segmentation. Purposely, we adapt the knowledge learned to pursue depth estimation into a representation amenable to semantic segmentation with [48]. As the geometric cues learned from monocular images yield semantic predictions that are often complementary to those attainable by current UDA methods, as illustrated in Fig. 4.1 we realize our proposal as a depth-based add-on, dubbed D4 (Depth For), which can be plugged seamlessly into any UDA method to boost its performance.

Finally, we also follow a recent trend in UDA for semantic segmentation, the Self-Training (ST), which consists of further fine-tuning the trained network by its predictions [110, 111, 112, 113, 114, 115]. We propose a novel Depth-Based Self-Training (DBST) approach, which deploys once more the availability of depth information for real images to build a large and varied dataset of plausible samples to be deployed in the Self-Training (ST) procedure.

We will show that our framework can improve many state-of-the-art methods by a large margin in two UDA for semantic segmentation benchmarks, where networks are trained either on GTA5 [88] either or SYNTHIA VIDEO SEQUENCES [89] and tested on Cityscapes [91]. Moreover, we show that our DBST procedure enables us to distill the whole framework into a single ResNet101 [116] and achieve state-of-the-art performance.

4.1 Method

In Unsupervised Domain Adaptation (UDA) for semantic segmentation one wishes to solve semantic segmentation in a target domain, $\mathcal{B}$, though labels are available only in another domain, referred to as source domain $\mathcal{A}$. In the following, we describe the two ingredients to better tackle this problem. In Sec. 4.1.1 we show how to use AT/DT to transfer information from self-supervised monocular depth to semantic segmentation and merge this knowledge with any UDA method (D4-UDA, Depth For UDA). Then, in Sec. 4.1.2 we introduce a Depth-Based Self-Training strategy (DBST) to further improve semantic predictions while distilling the whole framework into a single CNN.

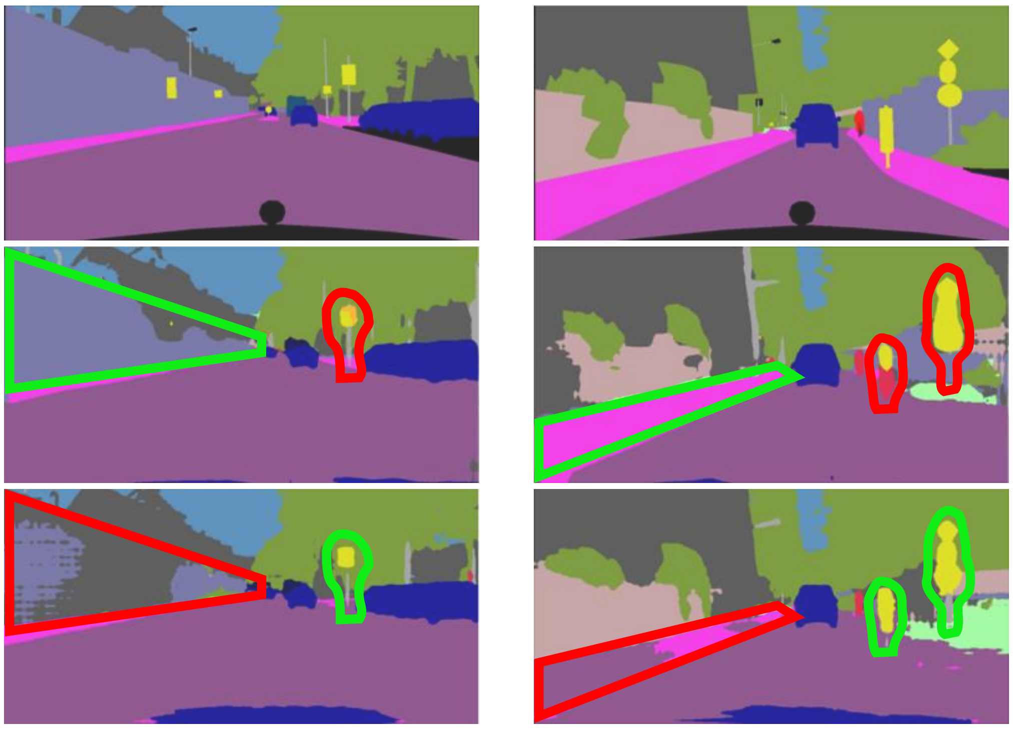


FIGURE 4.2. From top to bottom: ground truth, semantics from depth, semantics by LTIR [27]. The semantic labels predicted from depth are more accurate than those yielded by UDA methods in regularly-shaped objects (such as the *wall* in the left image and the *sidewalk* in the right one), whilst UDA approaches tend top perform better on small objects (see the *traffic signs* in both images).

4.1.1 D4 (Depth For UDA)

Semantics from Depth. As explained above we want to use AT/DT to transfer knowledge from depth to semantic. However, AT/DT assumes the availability of ground-truth labels for the first task (depth estimation in this setting) also in $\mathcal{B}$ (real images), or at least a stereo setup. As pointed out previously, this assumption does not comply with the standard UDA for semantic segmentation problem formulation, which pertains availability of semantic labels for source images ($\mathcal{A}$) alongside with unlabelled target images ($\mathcal{B}$). To address this issue we propose here to rely on *depth proxy-labels* attainable from images belonging to both $\mathcal{A}$ and $\mathcal{B}$ without the need for any ground-truth information. In particular, we propose to deploy one of the recently proposed deep neural networks, such as [101], that can be trained to perform monocular depth estimation based on a self-supervised loss that requires the availability of raw image sequences only, i.e. without ground-truth depth labels. Thus, we will follow the following protocol. First, we train a self-supervised monocular depth estimation network on both $\mathcal{A}$ and $\mathcal{B}$. Then, we use this network to generate *depth proxy-labels* for both domains. Finally, we train AT/DT following the protocol exposed in AT/DT using the previously computed *depth proxy-labels* to train N_1. In the following, we will refer to such predictions as *semantics from depth* because they concern semantic information extracted from features amenable to perform monocular depth estimation.

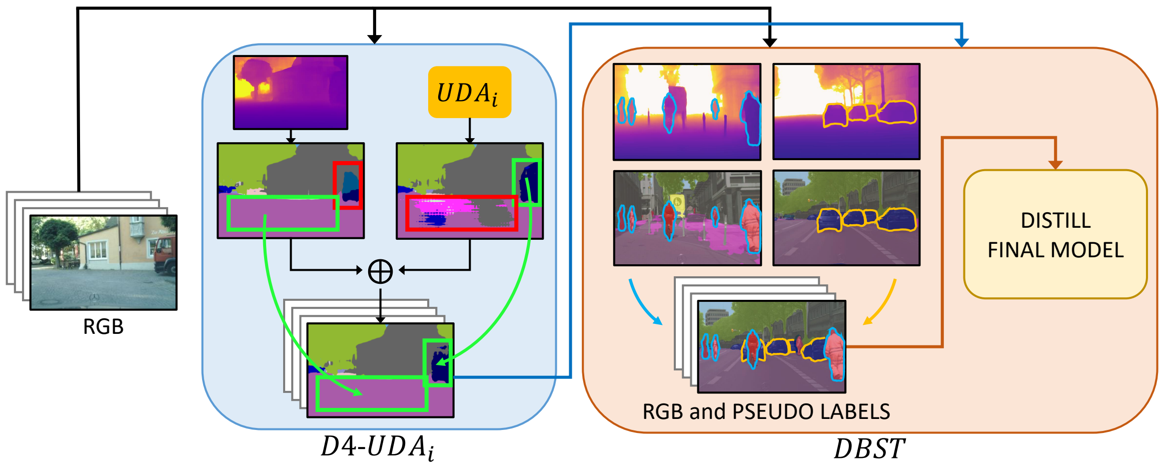


FIGURE 4.3. Overview of D4. RGB images are first processed by two different semantic segmentation engines to produce complementary predictions that are then combined by a weighted sum that accounts for the relative strengths and weaknesses of the two engines (Eq. (4.3)). During the next step, referred to as DBST, predictions from D4-UDA$_i$ are used to synthesize augmented samples by mixing portions of different images according to depth and semantics. The augmented samples are exploited to train a final model, so as to distill the whole pipeline into a single network.

Combine with UDA. Fig. 4.2 compares semantic predictions obtained from depth by the protocol described in the previous sub-section and from a recent UDA method. The reader may observe a clear pattern: predictions from depth tend to be smoother and more accurate on objects with large and regular shapes, like *road*, *sidewalk*, *wall* and *building*. However, they turn out often imprecise in regions where depth predictions are less informative, like thin things partially overlapping with other objects or fine-grained structures in the background. As UDA methods tend to perform better on such classes (see Fig. 4.2), our D4 approach is designed to *combine* the semantic knowledge extracted from depth with that provided by any chosen UDA method in order to achieve more accurate semantic predictions.

Depth information helps on large objects with regular shapes, which usually account for the majority of pixels in an image, and a whole dataset alike. On the contrary, UDA methods perform well in predicting semantic labels for categories that typically concern much smaller fractions of the total number of pixels in an image and dataset, like e.g. the *traffic signs* in Fig. 4.2. This orthogonality suggests that a simple yet effective way to combine the semantic knowledge drawn from depth with that provided by UDA methods consists of a weighted sum of predictions, with weights computed according to the frequency of classes in $\mathcal{A}$ (the domain where semantic labels are available). As weights given to UDA predictions ($\mathbf{w}_{uda}$) should be larger for rarer classes, they can be computed as:

$$\mathbf{w}_{uda} = [w^1_{uda}, \ldots, w^C_{uda}] \quad \text{where} \quad w^i_{uda} = \frac{1}{ln(\delta + f^i)} \tag{4.1}$$

method	Road	Sidewalk	Building	Walls	Fence	Pole	T-light	T-sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorbike	Bicycle	**mIoU**	**Acc**
AdaptSegNet [93]	86.5	36.0	79.9	23.4	23.3	23.9	35.2	14.8	83.4	33.3	75.6	58.5	27.6	73.6	32.5	35.4	3.9	30.1	28.1	42.4	85.6
D4-AdaptSegNet + DBST	93.1	53.0	85.1	42.8	27.3	35.8	43.9	18.5	85.9	39.0	89.9	63.0	31.6	86.6	39.8	36.7	0	42.4	35.0	**50.0**	**90.3**
MaxSquare [24]	88.1	27.7	80.8	28.7	19.8	24.9	34.0	17.8	83.6	34.7	76.0	58.6	28.6	84.1	37.8	43.1	7.2	32.2	34.5	44.3	86.9
D4-MaxSquare + DBST	92.9	51.2	84.7	43.5	22.2	35.7	42.5	20.0	86.2	42.0	90.0	63.7	33.0	86.9	45.5	50.9	0	42.2	41.4	**51.3**	**90.3**
BDL [22]	88.2	44.7	84.2	34.6	27.6	30.2	36.0	36.0	85.0	43.6	83.0	58.6	31.6	83.3	35.3	49.7	3.3	28.8	35.6	48.5	89.2
D4-BDL + DBST	93.2	52.6	86.4	44.1	31.2	36.5	42.4	36.1	86.3	41.0	89.8	63.3	37.4	86.3	42.8	57.8	0	40.3	37.9	**52.9**	**90.7**
MRNET [118]	90.5	35.0	84.6	34.3	24.0	36.8	44.1	42.7	84.5	33.6	82.5	63.1	34.4	85.8	32.9	38.2	2.0	27.1	41.8	48.3	88.3
D4-MRNET + DBST	93.2	51.6	86.1	45.9	24.5	37.9	47.4	40.4	85.3	37.5	89.6	64.7	39.8	85.8	41.1	53.2	8.9	17.1	33.4	**51.7**	**90.0**
Stuff and things* [96]	90.2	43.5	84.6	37.0	32.0	34.0	39.3	37.2	84.0	43.1	86.1	61.1	29.9	81.6	32.3	38.3	3.2	30.2	31.9	48.3	88.8
D4-Stuff and things + DBST	93.3	54.0	86.5	46.4	32.3	37.7	45.2	39.5	85.5	39.4	90.0	63.7	32.8	85.5	32.0	39.5	0	37.7	35.5	**51.4**	**90.5**
FADA [119]	92.5	47.5	85.1	37.6	32.8	33.4	33.8	18.4	85.3	37.7	83.5	63.2	39.7	87.5	32.9	47.8	1.6	34.9	39.5	49.2	88.9
D4-FADA + DBST	93.9	58.2	86.4	45.9	29.6	36.9	44.6	27.0	86.3	39.4	90.0	64.9	41.0	85.8	34.6	51.2	9.9	24.2	37.3	**52.0**	**90.7**
LTIR [27]	92.9	55.0	85.3	34.2	31.1	34.4	40.8	34.0	85.2	40.1	87.1	61.1	31.1	82.5	32.3	42.9	3	36.4	46.1	50.2	90.0
D4-LTIR + DBST	94.2	59.6	86.9	43.9	35.3	36.9	45.7	36.1	86.2	40.6	90.0	65.9	38.2	84.4	33.3	52.4	13.7	46.2	51.7	**54.1**	**91.0**

TABLE 4.1. Results on GTA5→Cityscapes. When available, checkpoints provided by authors are used. * denotes method retrained by us.

where C denotes the number of classes and $f^i = \frac{n^i}{tot}$ denotes their frequencies at the pixel level, *i.e.*,the ratio between the number n^i of pixels labelled with class i in $\mathcal{A}$ and the total number tot of labelled pixels in $\mathcal{A}$. Akin to common practice, we set the constant δ to 1.02 in our experiments. Eq. (4.1) is the standard formulation introduced in [117] to compute bounded weights inversely proportional to the frequency of classes. Accordingly, weights applied to semantic predictions drawn from depth ($\mathbf{w}_{dep}$) are given by:

$$\mathbf{w}_{dep} = [w^1_{dep}, ..., w^C_{dep}] \quad \text{where} \quad w^i_{dep} = 1 - w^i_{uda}. \tag{4.2}$$

Thus, at each pixel of a given image, we propose to combine semantics from depth and predictions yielded by any chosen UDA method as follows:

$$\widehat{\mathbf{y}}_f = \mathbf{w}_{dep} \cdot \phi_T(\widehat{\mathbf{y}}_{dep}) + \mathbf{w}_{uda} \cdot \phi_T(\widehat{\mathbf{y}}_{uda}), \tag{4.3}$$

where $\widehat{\mathbf{y}}_f$ is the final prediction, $\widehat{\mathbf{y}}_{dep}$ and $\widehat{\mathbf{y}}_{uda}$ are the logits associated with semantics from depth and the selected UDA method, respectively, ϕ_T denotes the *softmax* function with a temperature term T that we set to 6 in our experiments.

As illustrated in Fig. 4.3, the formulation presented in Eq. (4.3) and symbolized as $\bigoplus$ can be used seamlessly to plug semantic information extracted from self-supervised monocular depth into any existing UDA method. We will refer to the combination of a given UDA method with our D4 with the expression D4-UDA. Experimental results reported in Sec. 4.2.2 show that, indeed, all recent s.o.t.a. UDA methods do benefit significantly from the complementary geometric cues brought in by D4.

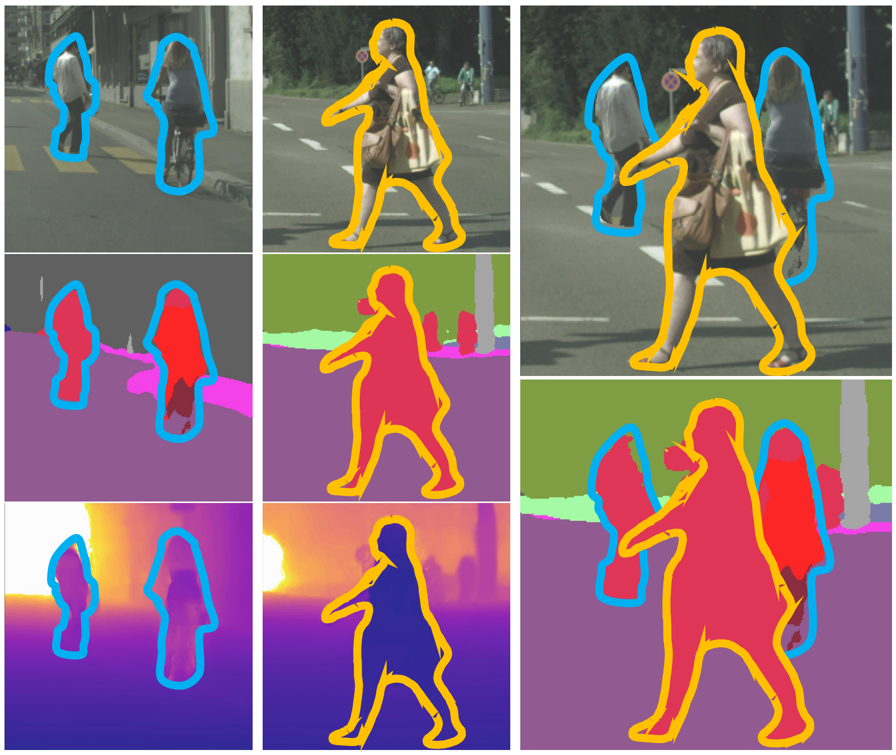


FIGURE 4.4. The rightmost column is a training sample synthesized by copying pixels from left column into the central one. Pixels are chosen according to their semantic class and stacked according to their depths (third row). For example, the two small persons in the left pair are copied behind the one in the middle column. The white pixels in the depth maps represent areas that cannot be copied into other samples due to their depth being too large.

4.1.2 DBST (Depth-Based Self-Training)

We describe now our proposal to further improve semantic predictions and distill the knowledge of the entire system into a single network easily deployable at inference time. First, we predict semantic labels for every image in $\mathcal{B}$ by our whole framework (i.e. D4 alongside a selected UDA method, referred to as D4-UDA); then, we use these labels to train a new model on $\mathcal{B}$. This procedure, also known as Self-Training [120], has become popular in recent UDA for semantic segmentation literature [110, 111, 112, 113, 114, 115] and consists of training a model by its own predictions, referred to as *pseudo-labels*, sometimes through multiple iterations. The novelty of our approach concerns the peculiar ability to leverage on the depth information available for the images in $\mathcal{B}$ to generate plausible new samples.

Running D4-UDA on $\mathcal{B}$ yields semantic pseudo-labels for every image in $\mathcal{B}$. Yet, as described in Sec. 4.1.1 (*Semantics from Depth*), each image in $\mathcal{B}$ is also endowed with a depth prediction, provided by a self-supervised monocular depth estimation network.

We can take advantage of this information to formulate a novel depth-aware data augmentation strategy whereby a portion of images and corresponding pseudo-labels are *copied* onto others so as to synthesize samples for the Self-Training procedure. The crucial difference between similar approaches presented in [121, 122] and ours consists of the deployment of depth information to steer the data augmentation procedure toward generating plausible samples. Indeed, a first intuition behind our method deals with semantic predictions being less accurate for objects distant from the camera: as such predictions play the role of labels in Self-Training, we are led to prefer picking closer rather than distant regions in order to generate training samples. Moreover, we reckon certain kinds of objects, like e.g. persons, vehicles, poles, traffic signs, to be more plausibly transferable across different images as they tend to be small and less bound to specific spatial locations. For example, a piece of road or building from another image would more unlikely merge seamlessly into a given one with respect to a pedestrian or vehicle.

Given N randomly selected images x^n from $\mathcal{B}$, with $n \in \{1, \ldots, N\}$, paired with semantic pseudo-labels s^n and depth predictions d^n, we augment x^1, by copying on it pixels from the set $\mathcal{X}^{src} = \{x^2, \cdots, x^N\}$. For each pixel of the augmented image we have N possible candidates, one from x^1 itself and $N-1$ from the images in $\mathcal{X}^{src}$. We filter such candidates according to two criteria: the predicted depth should be lower than a threshold t and the semantic prediction should belong to a predefined set of classes, C^*. Hence, we define the set of depths of the filtered candidates at each spatial location p as:

$$\mathcal{D}_p = \{d_p^n \mid d_p^n < t \wedge s_p^n \in C^*\} \quad n \in \{1, \ldots, N\}. \tag{4.4}$$

In our experiments, for each image the depth threshold t is set to the 80^{th} percentile of the depth distribution, so as to avoid selecting pixels from the farthest objects in the scene, while C^* contains classes: *pole*, *traffic light*, *traffic sign*, *person*, *car*, *rider*, *truck*, *bus*, *train*, *motorbike*, *bicycle*, *wall* and *fence*, which we found more amenable to synthesize plausible training samples. Then, we synthesize a new image x^z and corresponding pseudo-labels s^z, by assigning at each spatial location p the candidate with the lowest depth, so that objects belonging to different images do overlap plausibly into the synthesized one:

$$x_p^z = x_p^k \quad s_p^z = s_p^k \tag{4.5}$$

$$\begin{cases} 1, & \mathcal{D}_p = \emptyset \\ n \text{ s.t. } d_p^n = \min \mathcal{D}_p, & \mathcal{D}_p \neq \emptyset \end{cases} \tag{4.6}$$

In Fig. 4.4 we depict our depth-based procedure to synthesize new training samples, considering, for the sake of simplicity, the case where N is 2.

Hence, with the procedure detailed above, we synthesize an augmented version of $\mathcal{B}$, used to distill the whole D4-UDA framework into a single model by a Self-Training process. This dataset is much larger and exhibits more variability than the original $\mathcal{B}$. Due to its reliance on depth information, we dub our novel technique as DBST (Depth-Based Self-Training). The

Method	Sky	Building	Road	Sidewalk	Fence	Vegetation	Pole	Car	T-Sign	Person	Bicycle	T-Light	**mIoU**	**Acc**
AdaptSegNet* [93]	75.6	78.0	89.7	28.5	3.4	76.0	28.5	85.1	27.2	55.3	46.6	0	49.5	86.9
D4-AdaptSegNet + DBST	88.0	80.2	95.1	66 .8	5.7	80.4	33.2	87.3	33.2	60.9	52.4	0	**56.9**	**90.7**
MaxSquare* [24]	72.4	79.2	89.2	36.0	4.6	75.7	31.5	84.9	30.7	55.8	45.8	8.6	51.2	87.3
D4-MaxSquare + DBST	88.1	80.1	95.0	66.6	6.0	79.4	34.4	86.7	36.3	60.8	47.2	8.4	**57.4**	**90.6**
MRNET* [118]	84.6	79.7	93.9	56.3	0	80.5	35.4	88.9	27.2	59.4	56.3	0	54.5	90.0
D4-MRNET + DBST	88.3	79.0	95.0	67.0	5.9	78.6	36.2	86.7	31.0	60.6	47.5	0	**56.3**	**90.2**

TABLE 4.2. Results on the SYNTHIA-SEQ→Cityscapes benchmark. * denotes method retrained by us.

results reported in Sec. 4.2.2 prove its remarkable effectiveness, both when used as the final stage following D4 as well as when deployed as a standalone Self-Training procedure applied to any other UDA for semantic segmentation method.

4.2 Experiments

4.2.1 Implementation Details

Network Architectures. We use Monodepth2 [101] to generate depth proxy-labels for the procedure described in Sec. 4.1.1. We change the AT/DT framework with respect to the one used in [48] to this new setting by deploying the popular Deeplab-v2 [123] for depth estimation and semantic segmentation networks. Both networks consist of a backbone and an ASPP module [123], which substitute, respectively, the encoder and decoder used in [48]. The backbone is implemented as a dilated ResNet50 [124]. We also use the transfer function without the downsampling and upsampling operations. More precisely, the transfer function is realized as a simple 6-layer CNN with kernel size 3×3 and Batch Norm [125]. Following the recent trend in UDA for semantic segmentation [93, 24, 22, 118, 96, 119, 27], during DBST we train a single Deeplab-v2 [123] model, with a dilated ResNet101 pre-trained on Imagenet [126] as backbone.

Training Details. Our pipeline is trained on one NVIDIA Tesla V100 GPU with 16GB of memory. In every training and test phase, we resize input images to 1024×512, with the exception of DBST, when we first perform random scaling and then random crop with size 1024×512. During DBST we use also color jitter to avoid overfitting on the pseudo-labels. The depth and the transfer network are optimized by Adam [127] with batch size 2 for 70 and 40 epochs, respectively, while the semantic segmentation network is trained by SGD with batch size 2 for 70 epochs.The final model obtained by DBST is trained again with SGD, batch size 3 and for 30 epochs. We adopt the One Cycle learning rate policy [128] in every training,

Method	UDA	D4-UDA	UDA + DBST	D4-UDA + DBST
AdaptSegNet [93]	42.4	46.7	46.0	**50.0**
MaxSquare [24]	44.3	48.0	48.1	**51.3**
BDL [22]	48.5	49.6	51.7	**52.9**
MRNET [118]	48.3	49.6	50.0	**51.7**
Stuff and Things* [96]	48.3	49.1	50.4	**51.4**
FADA [96]	49.3	49.9	51.4	**52.0**
LTIR [27]	50.2	51.1	53.1	**54.1**

TABLE 4.3. Impact on performance of the two components of our proposal (D4, DBST) when applied separately or jointly to selected UDA methods on GTA5→Cityscapes. * indicates that the method was retrained by us. Results are reported in mIoU.

setting the maximum learning rate to 10^{-4} but in DBST, where we use 10^{-3}. To conduct our experiments, e use two synthetic datasets: GTA5 and SYNTHIA. Since our method requires video sequences to train Monodepth2 [101], we use the split SYNTHIA VIDEO SEQUENCES (SYNTHIA-SEQ) in the experiments involving the SYNTHIA dataset. As for real images, we leverage on the Cityscapes dataset [91].

4.2.2 Results

We report here experimental results obtained in two domain adaptation benchmarks, i.e. GTA5→Cityscapes and SYNTHIA-SEQ→Cityscapes, which show how the combination with our D4 method allows to boost performance of recent UDA for semantic segmentation approaches.

GTA5→Cityscapes. Tab. 4.1 reports results on the most popular UDA benchmark for semantic segmentation, i.e. GTA5→Cityscapes, where methods are trained on GTA5 and tested on Cityscapes. We selected the most relevant UDA approaches proposed in the last years [93, 24, 22, 118, 96, 119, 27], using training checkpoints provided by authors whenever available. We report per-class and overall results in terms of mean intersection over union (mIoU) and pixel accuracy (Acc), when each method is either used stand-alone or deployed within our proposal (i.e. D4 + DBST). The reader may notice how every recent UDA method does improve considerably if combined with our proposal, despite the variability of their stand-alone performances. Indeed, Adaptsegnet [93], which yields about 42% in terms of mIoU, reaches 50% when embedded into our framework. Likewise, LTIR [27], currently considered one of the s.o.t.a. UDA methods, improves in mIoU from 50.2% to 54.1%. Analyzing Tab. 4.1 more in detail, we can observe that our method produces a general improvement for all classes, although we experience a certain performance variability for some of them (such as *train*, *motorbike* and *bicycle*), probably due to noisy pseudo-labels used during DBST. Conversely, our method yields consistently a significant gain on classes

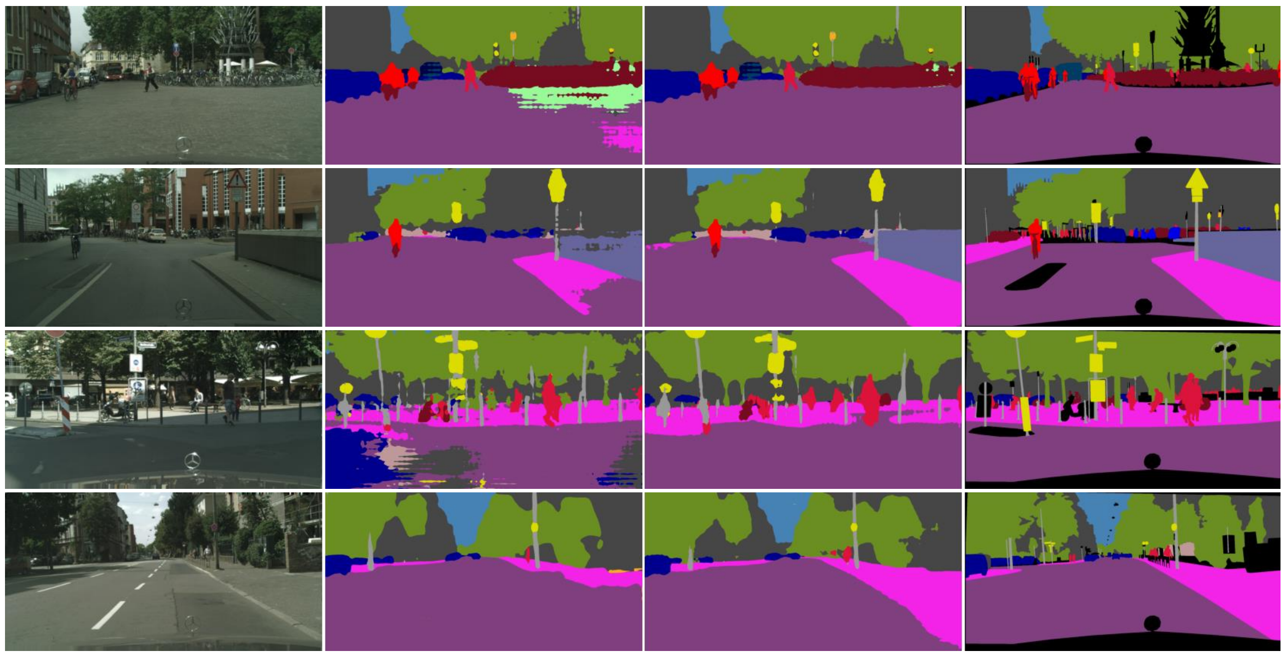


FIGURE 4.5. From left to right: RGB image, prediction from UDA method, prediction from D4-UDA + DBST, GT. The top two rows deal with GTA5→Cityscapes, the other two with SYNTHIA-SEQ→Cityscapes. Selected methods are, from top to bottom: LTIR [27], BDL [22], MaxSquare [24] and MRNET [118]. In all these examples our proposal can ameliorate dramatically the output of the given stand-alone method, especially on classes featuring large and regular shapes, like *road* in rows 1-3, *sidewalk* in rows 2-4 and *wall* in row 2.

characterized by large and regular shapes, namely *road*, *sidewalk*, *building*, *wall* and *sky*, which validates our intuition on the effectiveness of a) the geometric cues derivable from depth to predict the semantics of these kind of objects and b) the methodology we propose to leverage on these additional cues in UDA settings. This behavior is also clearly observable from a qualitative perspective in Fig. Fig. 4.5. Finally, we point out that, to the best of our knowledge, the performance figure obtained by D4-LTIR + DBST, i.e. 54.1% mIoU (last row of Tab. 4.1) establishes the new state-of-the-art for the GTA5→Cityscapes benchmark.

SYNTHIA-SEQ→Cityscapes. Akin to common practice in literature we present results also on the popular SYNTHIA dataset. Due to our pipeline requiring video sequences to

Method	D4-UDA	Self-Training	DBST
D4-BDL [22]	49.6	50.1	**52.9**
D4-MRNET [118]	49.6	50.3	**51.7**
D4-Stuff and Things [96]	49.1	49.4	**51.4**
D4-FADA [119]	49.9	50.0	**52.0**
D4-LTIR [27]	51.1	51.5	**54.1**

TABLE 4.4. Comparison between DBST and baseline Self Training. Results are reported in terms of mIoU on GTA5→Cityscapes.

train the self-supervised monocular depth estimation network, we select the SYNTHIA VIDEO SEQUENCES split for training and the Cityscapes dataset for testing. We will refer to this setting as to SYNTHIA-SEQ→Cityscapes. To address SYNTHIA-SEQ→Cityscapes we re-trained the UDA methods for which the code is available and the training procedure is more affordable in terms of memory and run-time requirements, namely AdaptSegNet [93], MaxSquare [24] and MRNET [118]. The results in Tab. 4.2 show that all the selected UDA approaches exhibit a substantial performance gain when coupled with our proposal, with a general improvement in all classes. In particular, similarly to the results obtained in GTA5→Cityscapes, we observe a consistent improvement for classes related to objects with large and regular shapes (as depicted also in Fig. 4.5), with the only exception of a slight performance drop for the class *building* when using MRNET [118] (last row of Tab. 4.2). We argue that our approach is relatively less effective with MRNET [118] as, unlike AdaptSegNet [93] and MaxSquare [24], it yields already satisfactory results in those classes which are usually improved by the geometric clues injected by D4.

4.2.3 Ablation study

Impact of the individual contributions. In Tab. 4.3, we analyze the impact on the performance yielded by the two main contributions of this approach, i.e. injection of geometric cues into UDA methods by D4 and DBST. Purposely, we select the GTA5→Cityscapes benchmark and, for each of the UDA methods considered in Tab. 4.1, we report the mIoU figures obtained by a) using the stand-alone UDA method, b) combining it with D4, c) applying DBST directly on the stand-alone method and d) embedding the method into our full pipeline. Thus, we can observe that each of our novel contributions allows for improving the performance of the most recent UDA methods by a large margin. Moreover, as shown in the last column of Tab. 4.3, when deployed jointly so as to realize our whole proposal, D4 and DBST further enhance the performances of any selected method suggesting that they are complementary.

Effectiveness of DBST. In Tab. 4.4 we compare our DBST against an alternative Self-Training procedure. We select five UDA methods with appealing performances on GTA5→Cityscapes and combine each of them with our D4, reporting the obtained mIoU figures in the second column. Seeking a viable Self-Training algorithm, we considered those originally proposed together with each method. Yet, we found experimentally that none of the original Self-Training procedure could yield a further performance improvement. This can be explained by the fact that these Self-Training algorithms rely on modeling the uncertainty of the original method for which they are designed and this tends to fail when UDA methods get merged with D4. This leads us to consider a baseline Self-Training procedure, which consists of simply fine-tuning the model by its own predictions on the images of the target domain. Thus, in Tab. 4.4 we compare this baseline Self-Training to DBST (third and fourth column respectively). The figures in Tab. 4.4 highlight how our DBST procedure consistently provides a much larger performance improvement than the considered baseline Self-Training. As the only difference between the two procedures concerns the dataset employed in the fine-tuning process, the results in Tab. 4.4 prove the effectiveness of DBST in generating a large and varied set of plausible training samples more amenable to Self-Training than the original images belonging to the target domain.

4.3 Conclusions

We have shown how to exploit self-supervised monocular depth estimation in UDA problems to obtain accurate semantic predictions for objects with strong geometric priors (like roads and buildings). As all recent UDA approaches lack such geometric knowledge, we designed our method as a depth-based add-on, pluggable into any UDA method to boost performances. Finally, we employed self-supervised depth estimation to realize an effective data augmentation strategy for self-training. Our work highlights the possibility of exploiting auxiliary tasks learned by self-supervision to better tackle UDA for semantic segmentation, paving the way for novel research directions.

Chapter 5

Exploiting Shallow Features for Sharp Segmentation Mask

In the last few years, several UDA techniques have been proposed for the task of semantic segmentation [42, 23, 70, 129, 26, 130]. However, all these methods ignore the main goal of semantic segmentation, which is to obtain sharp prediction masks and only focus on the feature adaptation part. For this reason, previous works can correctly segment out coarse blobs of large elements in a scene such as cars or buildings, while they provide inaccurate segmentation masks along class boundaries as shown in Fig. 5.1. On the other hand, in the supervised semantic segmentation setting, a large number of works focus on obtaining sharp predictions [131, 132, 133, 134, 135]. This is commonly done by better integrating low-level features into high-level features since modern segmentation architectures discard spatial information with down-sampling operations such as max-pooling or strided convolution due to memory and time constraints. Following the supervised setting, we argue that this line of research should also be pursued for the UDA case to obtain sharp predictions across domains even though target labels are not available. Our approach also leverages on low-level features to reach this goal, and we introduce a novel low-level adaptation strategy specifically for the UDA scenario. More precisely, we enforce the alignment of low-level features by exploiting an auxiliary task that can be solved for both domains in a self-supervised fashion, intending to make them more transferable. By doing this, we enable the possibility to exploit shallow features to refine the coarse segmentation masks for both the source and target domains. To achieve this, we estimate a 2D displacement field from the aligned shallow features that, for each spatial location of the predicted coarse feature map, specifies the direction where the representation for that patch is less ambiguous (i.e. at center of the semantic object). Our intuition is that when the coarse feature map is bi-linearly up-sampled to regain the target resolution, the feature representation of those patches corresponding to semantic boundaries in the input image is mixed up, as it contains semantic information belonging to different classes. Thanks to the estimated 2D displacement field, however, we refine each patch representation according to the features coming from the center of the object, which are less prone to be influenced by other classes as they lay spatially far from boundaries. This

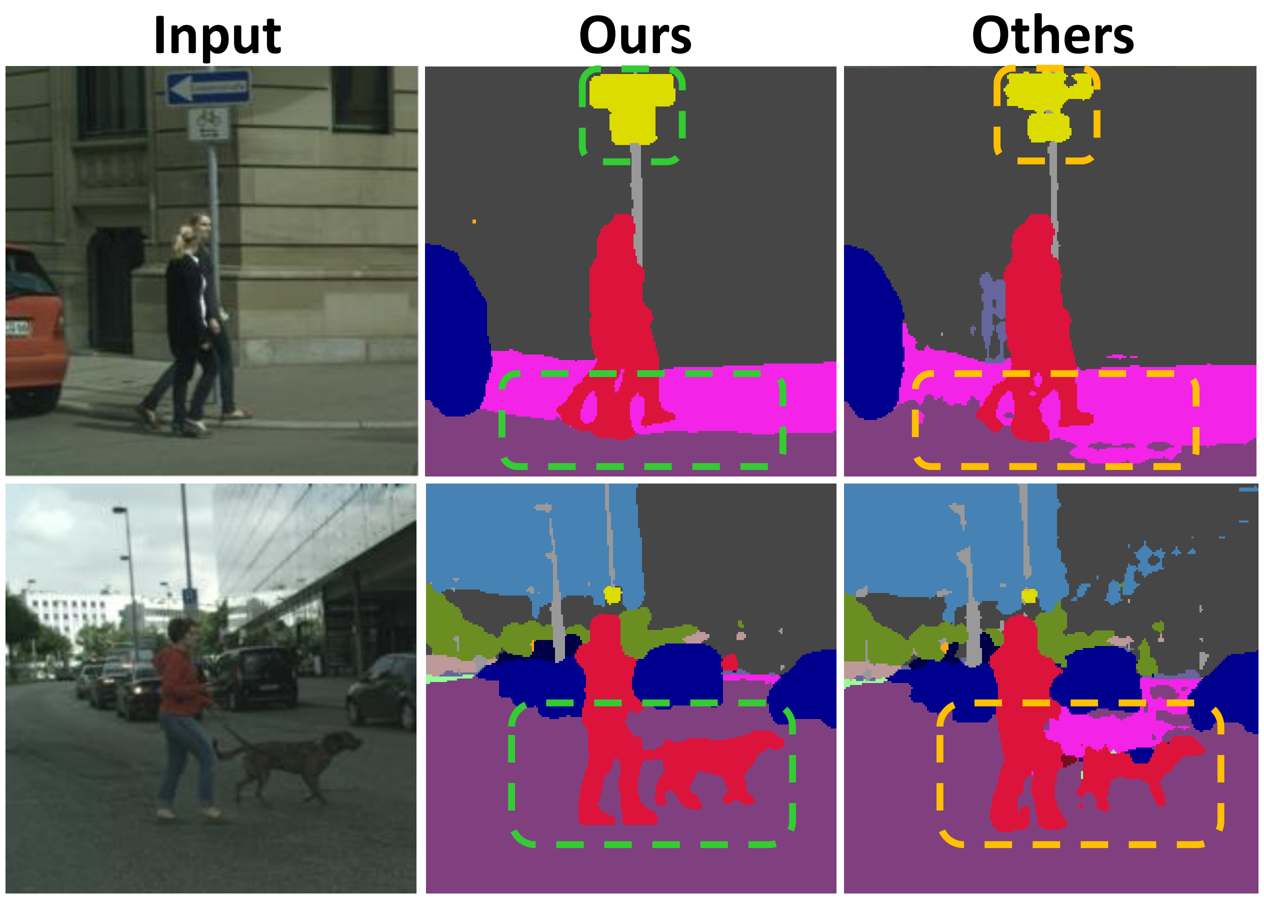


FIGURE 5.1. Given in input an RGB image (left-most column), our model produces sharp predictions along class boundaries (central column), while a model trained on translated images (right-most column) exhibits severe noise.

process will be referred later as the feature *warping* process.

Finally, as done in the previous chapter, we exploit once again self-training. However in this chapter, we are specifically interested in preserving the information at the boundaries rather than naively masking low-confident pixels, and we propose a novel data augmentation that allows us to preserve useful information such as the boundaries of an object. In fact, due to the low confidence of the network in the target domain, pseudo-labels along edges are usually masked by previous methods, resulting in further performance degradation along class boundaries due to the lack of supervision during the self-training process. Thus, we employ a class-wise erosion filtering algorithm that allows us to synthesize new training samples in which only the inner body of the target objects is preserved and copied into other images. By doing this, all pixels have supervision, and the network is trained to classify correctly edges also in the target domain.

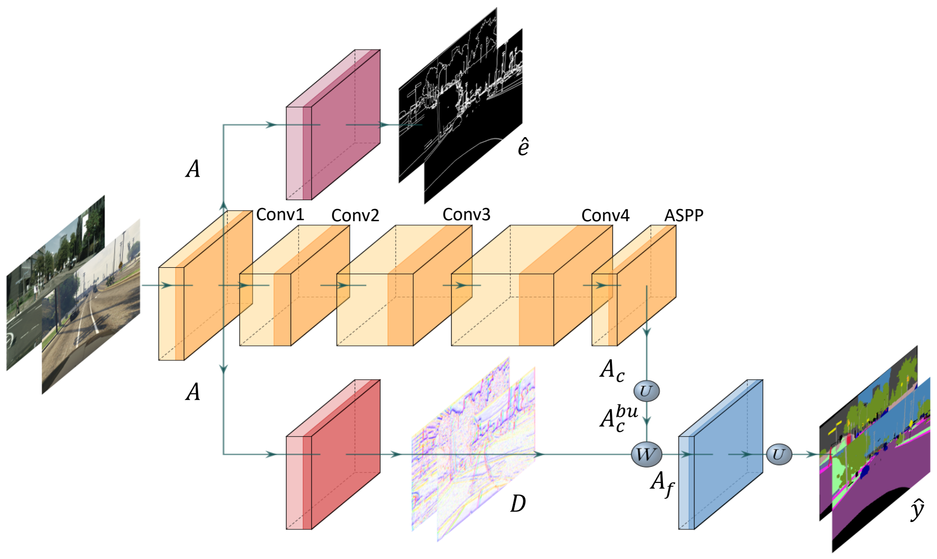


FIGURE 5.2. Illustration of our architecture in the adaptation step. Given an RGB input image, the network learns to extract semantic edges from shallow features. From the same feature map, a 2D displacement map is estimated in order to guide the warping of down-sampled deep features, which lacks of fine-grained details.

5.1 Method

Our proposed framework comprises several components, as depicted in Fig. 5.2. A standard backbone (yellow branch) produces a coarse feature map A_c from an image. A semantic edge extractor (top purple branch) estimates semantic edges $\hat{e}$, given the activation map A produced by the first convolutional block of the backbone. The same shallow features are processed by another convolutional block (bottom red branch) to obtain a 2D displacement map, D. Then, A_c is up-sampled to the same size as D and it is refined according to D to produce a fine-grained feature map A_f. Finally, one last convolutional block that acts as a classifier is applied to produce a C-dimensional vector for each pixel, with C being the number of classes, and a final bi-linear up-sampling yields a prediction map of the same size as the input. We detail each component in the following subsections.

5.1.1 Low-level adaptation

Learning transferable shallow features. We introduce an auxiliary task to push the network to learn domain-invariant features that include details on object boundaries already from early layers. Given the feature map A, a convolutional block γ is applied to predict an edge map $\hat{e}$. Ground truths e are obtained by the Canny edge detector [136] applied directly on semantic annotations for the source domain and on pseudo-labels for the target domain, so that only semantic boundaries are considered. A binary-cross entropy loss is minimized for batches including images from both domains:

$$\begin{aligned}\hat{e} &= \gamma(A), \\ \mathcal{L}_{edge} &= \sum_{h}^{H}\sum_{w}^{W} e^{(h,w)} \log \hat{e}^{(h,w)} \\ &+ (1 - e^{(h,w)}) \log\left(1 - \hat{e}^{(h,w)}\right)\end{aligned} \tag{5.1}$$

Hence, we enforce the auxiliary semantic edge detection task for the very first layers of the network only, rather than, as in typical multi-task learning settings such as [137, 138, 139], at a deeper level, where features are more task-dependent. We believe this design choice to be key for a good generalization for three reasons. First, trying to solve this task from shallow layers guides the network to explicitly reason about object shapes from the beginning, rather than solely texture and colors as typically done by CNNs [140]. Second, solving an auxiliary task for both domains forces the network to learn a shared feature representation, which naturally leads to aligned distributions. Consequently, the displacement field generated from the shallow features is effective also in the target domain, and it can be directly exploited at a deeper level to recover fine-grained details. Finally, the peculiar choice of semantic edge detection is directly beneficial to estimate a displacement field that mainly focuses on edges, making the following warping process more effective where the network is uncertain.

Feature warping. One of the contributions of our method is to refine the bi-linearly up-sampled coarse feature map A_c, hereafter A_c^{bu}, to obtain a fine-grained feature map A_f that better captures the correct class for pixels laying in the boundary regions. The refinement is guided by a 2D displacement field D obtained from the domain-invariant shallow features computed by the first convolutional block of the backbone. The displacement field indicates for each location of A_c^{bu} where the network should look to recover the correct class information, namely the direction that better characterizes that patch. We estimate the 2D displacement map D by applying a convolutional block to the aligned shallow features A that are aligned as described above.

Our intuition is that, due to the unavoidable side-effect of the down-sample operations in the forward pass, the representation of those elements in A_c whose receptive field includes regions at class boundaries in the original image, contains ambiguous semantic information. Indeed, when A_c is bi-linearly up-sampled, patches that receive contributions from ambiguous coarse patches inherit such ambiguity. However, in the higher resolution feature map A_c^{bu} it may be possible to compute a better, unambiguous representation for some of the patches, *i.e.*,those now laying entirely in a region belonging to one class. The correct semantic information may be available in the nearby high-resolution patches closer to the semantic blob centers. Thus, each feature vector at position p on a standard 2D spatial grid of A_c^{bu}, is mapped to a new position $\hat{p} = p + D(p)$, and we use a differentiable sampling mechanism

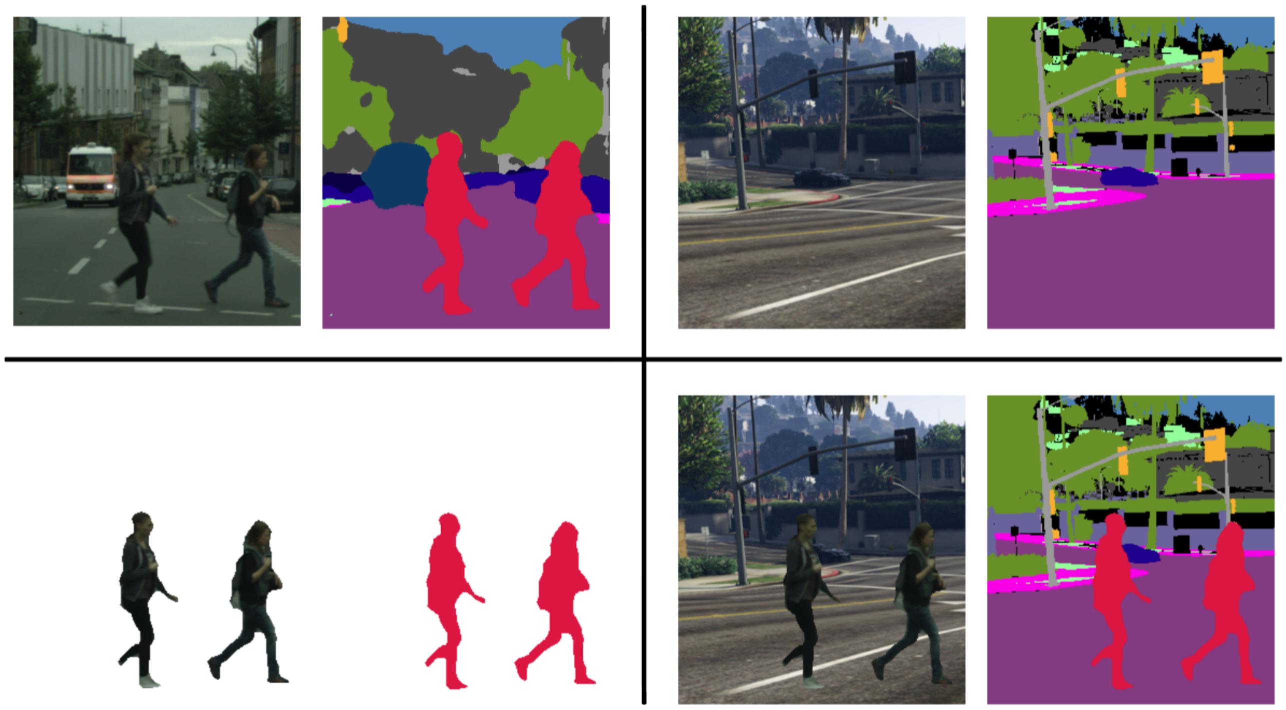


FIGURE 5.3. Given a target image prediction pair (top-left) and a source training pair (top-right), we select classes such as *person* (bottom-left) and apply our class-wise data augmentation pipeline to synthesise a new training pair (bottom-right). The selected shapes are eroded before being pasted.

[141] to approximate the new feature vector representation for that patch:

$$A_f(p) = \sum_{p_l \in \mathcal{N}(\hat{p})} w_{p_l} A_c^{bu}(p_l) \tag{5.2}$$

where w_{p_l}, are the bi-linear kernel weights obtained from D and $\mathcal{N}$ the set of neighboring pixels. Hence, Eq. (5.2) defines a backward warping operation in feature space, where A_f is obtained by warping A_c^{bu} according to D. Finally, the fine-grained feature map A_f is fed to the classifier to obtain the final prediction that is up-sampled by a factor of 2 to regain the input image resolution. We minimize the cross entropy loss using annotations for the source domain and pseudo-labels for the target domain:

$$\mathcal{L}_{sem} = \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{c=1}^{C} y^{(h,w,c)} \log \hat{y}^{(h,w,c)} \tag{5.3}$$

5.1.2 Data Augmentation for Self-Training

Inspired by [142, 143, 144, 122], we use a pre-trained model to select objects based on predictions on target images and paste them over source images (see Fig. 5.3). Peculiarly, our self-training approach relies on a data augmentation process that selects objects from the target scenes rather than the source ones as done [122]. Although selecting source objects may be useful to reduce the unbalanced distributions of classes, it is a sub-optimal choice since the network would be still trained to identify shapes and details peculiar to the source domain, which are different to those found at inference time for the target images. We instead use pseudo-labels to cut objects from the target scenes and paste them into source or target images, forcing the network to look for these patterns on both domains. However, due to the inherent noise of pseudo-labels we need to filter out noisy predictions. In particular, we aim at removing object boundaries as they typically exhibit classification errors and tend to be localized rather inaccurately. Given a target image x_t and its associated predictions $\hat{y}_t$, we compute a binary mask B_c for each class $c \in C^*$, where C^* denotes a random subset of the considered classes. We exclude classes such as *"road"* and *"building"* to avoid occlusion of the whole scene and to counteract the unbalanced distributions of classes, and only use object instances such as *"car"* and *"poles"*. This categorization is similar to the one used in [96], and can be easily adapted to different datasets. For each spatial location p, B_c has value 1 if p is assigned to class c, 0 otherwise. Then, we apply an erosion operation, $\ominus$, with a 5×5 structuring element k to each class mask B_c. To obtain the set of pixels to be copied from the target image to a randomly selected source image we apply the union set operator to all masks:

$$B = \bigcup_{c \in C^*} B_c \ominus k, \tag{5.4}$$

$$x^p = \begin{cases} x_t^p, & B^p = 1 \\ x_s^p, & B^p = 0 \end{cases}, y^p = \begin{cases} \hat{y}_t^p, & B^p = 1 \\ y_s^p, & B^p = 0 \end{cases} \tag{5.5}$$

The new synthesised training pairs are very often enriched with fine-grained details from the target domain. Indeed, as shown in Fig. 5.3, thanks to our data augmentation pipeline, only the inner part of an object is preserved while edges are discarded, producing sharp pseudo-labels even at class boundaries. The whole data augmentation process is applied offline before training, therefore it does not have any impact on the training time.

5.1.3 Training Procedure

The whole pipeline can be summarised in 3 simple steps. We start with the *initialization* step to train our baseline model (i.e. the yellow backbone of Fig. 5.2) on the source domain only. We follow standard practices [22, 96, 145, 146, 147] and, for synthetic-to-real adaptation, we utilize domain-translated source images provided by [22]. We deploy this baseline to produce pseudo-labels for the target domain and obtain an augmented mixed dataset as detailed in Sec. 5.1.2.

Then, we perform the *adaptation* step: we train the model illustrated in Fig. 5.2 that empowers our additional modules for low-level alignment as explained in Sec. 5.1.1. It is important to highlight that the proposed data augmentation extracts objects from only target images and pastes them on images on both domains. Hence, at this stage, the training is done simultaneously in both domains. The training loss is as follows:

$$\mathcal{L} = \mathcal{L}_{sem} + \lambda \mathcal{L}_{edge} \tag{5.6}$$

with λ set to 0.1 in all experiments.

Finally, we use the predictions from the model trained in the previous step to synthesise new training pairs by following again the procedure detailed in Sec. 5.1.2. This allows us to distill the knowledge and the good precision along the class boundaries of the previously enhanced model into a lighter segmentation architecture as the one used in the first step. We do this to avoid the introduction of additional modules at inference time. Differently from the adaptation step, however, we apply our data algorithm using solely images from the target domain. Indeed, as we are now at the third and final stage, we expect pseudo-labels to be less noisy compared to the previous step, and training only on the target domain allows to capture domain-specific characteristics. We denote this third step as the *distillation* step.

method	IT	ST	Road	Sidewalk	Building	Walls	Fence	Pole	T-light	T-sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorbike	Bicycle	**mIoU**
AdaptSegNet [23]			86.5	36.0	79.9	23.4	23.3	23.9	35.2	14.8	83.4	33.3	75.6	58.5	27.6	73.6	32.5	35.4	3.9	30.1	28.1	42.4
MaxSquare [24]			88.1	27.7	80.8	28.7	19.8	24.9	34.0	17.8	83.6	34.7	76.0	58.6	28.6	84.1	37.8	43.1	7.2	32.2	34.5	44.3
BDL [22]	✓	✓	88.2	44.7	84.2	34.6	27.6	30.2	36.0	36.0	85.0	43.6	83.0	58.6	31.6	83.3	35.3	49.7	3.3	28.8	35.6	48.5
MRNET [118]	✓	✓	90.5	35.0	84.6	34.3	24.0	36.8	44.1	42.7	84.5	33.6	82.5	63.1	34.4	85.8	32.9	38.2	2.0	27.1	41.8	48.3
Stuff and things [96]	✓	✓	90.6	44.7	84.8	34.3	28.7	31.6	35.0	37.6	84.7	43.3	85.3	57.0	31.5	83.8	42.6	48.5	1.9	30.4	39.0	49.2
FADA [119]		✓	92.5	47.5	85.1	37.6	32.8	33.4	33.8	18.4	85.3	37.7	83.5	63.2	**39.7**	**87.5**	32.9	47.8	1.6	34.9	39.5	49.2
LTIR [27]	✓	✓	92.9	55.0	85.3	34.2	31.1	34.4	40.8	34.0	85.2	40.1	87.1	61.1	31.1	82.5	32.3	42.9	3	36.4	46.1	50.2
Yang *et al.* [147]	✓	✓	91.3	46.0	84.5	34.4	29.7	32.6	35.8	36.4	84.5	43.2	83.0	60.0	32.2	83.2	35.0	46.7	0.0	33.7	42.2	49.2
IAST [148]		✓	**93.8**	**57.8**	85.1	**39.5**	26.7	26.2	43.1	34.7	84.9	32.9	88.0	62.6	29.0	87.3	39.2	49.6	**23.2**	34.7	39.6	51.5
DACS† [122]		✓	89.9	39.7	**87.9**	30.7	**39.5**	**38.5**	**46.4**	**52.8**	**88.0**	**44.0**	**88.8**	**67.2**	35.8	84.5	**45.7**	**50.2**	0.0	27.3	34.0	52.1
Ours	✓	✓	91.9	48.9	86.0	38.6	28.6	34.8	45.6	43.0	86.2	42.4	87.6	65.6	38.6	86.8	38.4	48.2	0.0	**46.5**	**59.2**	**53.5**

TABLE 5.1. Results on GTA5→Cityscapes. † denotes models pre-trained on MSCOCO [149] and ImageNet [47]. IT: Image Translation; ST: Self-Training.

5.2 Implementation

5.2.1 Architecture

To assess fairly our contributions, we use a similar setup to the one of the previous chapters which is also adopted by previous works [23, 24, 22, 118, 96, 119, 27]. Indeed, we employ the Deeplab-v2 [36] architecture, with a dilated ResNet101 pre-trained on ImageNet [47] for both the initialization step and the distillation step.

method	IT	ST	Road	Sidewalk	Building	Walls*	Fence*	Pole*	T-light	T-sign	Vegetation	Sky	Person	Rider	Car	Bus	Motorbike	Bicycle	**mIoU**	**mIoU***
AdaptSegNet [23]			84.3	42.7	77.5	-	-	-	4.7	7.0	77.9	82.5	54.3	21.0	72.3	32.2	18.9	32.3	-	46.7
MaxSquare [24]			77.4	34.0	78.7	5.6	0.2	27.7	5.8	9.8	80.7	83.2	58.5	20.5	74.1	32.1	11.0	29.9	39.3	45.8
BDL [22]	✓	✓	86.0	46.7	80.3	-	-	-	14.1	11.6	79.2	81.3	54.1	27.9	73.7	42.2	25.7	45.3	-	51.4
MRNET [118]	✓	✓	83.1	38.2	81.7	9.3	1.0	35.1	30.3	19.9	82.0	80.1	62.8	21.1	84.4	37.8	24.5	**53.3**	46.5	53.8
Stuff and things [96]	✓	✓	83.0	44.0	80.3	-	-	-	17.1	15.8	80.5	81.8	59.9	33.1	70.2	37.3	28.5	45.8	-	52.1
FADA [119]		✓	84.5	40.1	83.1	4.8	0.0	34.3	20.1	27.2	84.8	84.0	53.5	22.6	85.4	43.7	26.8	27.8	45.2	52.5
LTIR [27]	✓	✓	**92.6**	53.2	79.2	-	-	-	1.6	7.5	78.6	84.4	52.6	20.0	82.1	34.8	14.6	39.4	-	49.3
Yang *et al*. [147]	✓	✓	82.5	42.2	81.3	-	-	-	18.3	15.9	80.6	83.5	61.4	33.2	72.9	39.3	26.6	43.9	-	52.4
IAST [148]		✓	81.9	41.5	83.3	17.7	**4.6**	32.3	**30.9**	28.8	83.4	85.0	65.5	30.8	**86.5**	38.2	**33.1**	52.7	**49.8**	**57.0**
DACS† [122]		✓	80.6	25.1	81.9	**21.5**	2.6	**37.2**	22.7	24.0	83.7	**90.8**	**67.6**	**38.3**	82.9	38.9	28.5	47.6	48.3	54.8
Ours	✓	✓	90.4	**51.1**	**83.4**	3.0	0.0	32.3	25.3	**31.0**	**84.8**	85.5	59.3	30.1	82.6	**53.2**	17.5	45.6	48.4	56.9

TABLE 5.2. Results on SYNTHIA→Cityscapes. † denotes models pre-trained with MSCOCO [149] and ImageNet [47]. IT: Image Translation; ST: Self-Training. The 13 classes with * are used to compute mIoU*.

5.2.2 Training Details

Our pipeline is implemented in PyTorch [150] and trained on a single NVIDIA 2080Ti GPU with 12GB of memory. We train for 20 epochs in the first two steps, while we set the number of epochs to 25 for the final distillation with batch size 4 in all cases. We use random scaling, random cropping at 1024×892, and color jittering in our data augmentation pipeline. Akin to previous works, we freeze Batch-Normalization layers [151] while performing the initialization and adaptation step. For the last step, instead, we activate these layers. We adopt the One Cycle learning rate policy [128] for each training, with a maximum learning rate 10^{-3} and SGD as optimizer.

5.3 Experiments

5.3.1 Datasets

We test our method on both synthetic-to-real and real-to-real adaptation. We set GTA or SYNTHIA as source datasets and Cityscapes as target for the former setting, while we use Cityscapes as source and the NTHU as target for the latter.

5.3.2 Synthetic-to-real adaptation

To test our framework, we follow standard practice [23, 110, 22, 118, 152, 24] and report the results for the synthetic-to-real adaptation in the GTA5→Cityscapes and SYNTHIA→Cityscapes benchmarks in Tab. 5.1 and Tab. 5.2 respectively. We obtain state-of-the-art performance in the former setting, surpassing also recent methods such as [148] that perform many iterations of self-training. We also improve over [122] for GTA5→Cityscapes, which, differently from

City	Method	ST	road	sidewalk	building	light	sign	veg.	sky	person	rider	car	bus	motor	bike	mIoU (%)
Rome	Source only		85.9	40.0	86.0	9.0	25.4	82.4	90.5	38.8	25.9	81.6	52.0	48.7	6.7	51.9
	CBST [110]	✓	87.1	43.9	**89.7**	14.8	47.7	85.4	90.3	45.4	26.6	**85.4**	20.5	49.8	10.3	53.6
	AdaptSegNet [23]		83.9	34.2	88.3	18.8	40.2	**86.2**	**93.1**	47.8	21.7	80.9	47.8	48.3	8.6	53.8
	MaxSquare [24]		80.0	27.6	87.0	**20.8**	**42.5**	85.1	92.4	46.7	22.9	82.1	53.5	50.8	8.8	53.9
	FADA [119]	✓	84.9	35.8	88.3	20.5	40.1	85.9	92.8	**56.2**	23.2	83.6	31.8	53.2	**14.6**	54.7
	Ours	✓	**89.4**	**48.2**	87.5	**26.3**	37.2	83.1	90.7	55.2	**42.1**	84.8	**66.6**	**59.2**	11.1	**60.1**
Rio	Source only		80.4	53.8	80.7	4.0	10.9	74.4	87.8	48.5	25.0	72.1	36.1	30.2	12.5	47.4
	CBST [110]	✓	84.3	55.2	85.4	19.6	**30.1**	80.5	77.9	55.2	28.6	**79.7**	33.2	37.6	11.5	52.2
	AdaptSegNet [23]		76.2	44.7	84.6	9.3	25.5	**81.8**	87.3	55.3	32.7	74.3	28.9	43.0	27.6	51.6
	MaxSquare [24]		70.9	39.2	**85.6**	**14.5**	19.7	**81.8**	88.1	55.2	31.5	77.2	39.3	43.1	30.1	52.0
	FADA [119]	✓	80.6	53.4	84.2	5.8	23.0	78.4	87.7	**60.2**	26.4	77.1	37.6	**53.7**	**42.3**	54.7
	Ours	✓	**86.6**	**63.3**	82.3	10.3	19.8	73.9	**88.4**	57.5	**41.3**	78.1	**51.5**	40.0	19.4	**54.8**
Tokyo	Source only		86.0	38.8	76.6	11.7	12.3	80.0	89.5	44.9	28.0	71.5	4.7	27.1	42.2	47.2
	CBST [110]	✓	85.2	33.6	**80.4**	8.3	**31.1**	83.9	78.2	53.2	28.9	72.7	4.4	27.0	47.0	48.8
	AdaptSegNet [23]		81.5	26.0	77.8	17.8	26.8	82.7	90.9	55.8	**38.0**	72.1	4.2	24.5	50.8	49.9
	MaxSquare [24]		79.3	28.5	78.3	14.5	27.9	82.8	89.6	57.3	31.9	71.9	6.0	29.1	49.2	49.7
	FADA [119]	✓	85.8	39.5	76.0	14.7	24.9	**84.6**	**91.7**	62.2	27.7	71.4	3.0	29.3	**56.3**	51.3
	Ours	✓	**87.8**	**41.0**	79.6	**20.3**	24.2	80.2	90.0	**62.3**	30.8	**74.0**	**6.4**	**32.7**	50.0	**52.4**
Taipei	Source only		85.0	38.1	82.2	17.8	8.9	75.2	91.4	23.9	19.6	69.2	45.9	49.4	16.0	47.9
	CBST [110]	✓	86.1	35.2	84.2	15.0	22.2	75.6	74.9	22.7	33.1	78.0	37.6	58.0	30.9	50.3
	AdaptSegNet [23]		81.7	29.5	85.2	26.4	15.6	76.7	91.7	31.0	12.5	71.5	41.1	47.3	27.7	49.1
	MaxSquare [24]		81.2	32.8	85.4	31.9	14.7	78.3	92.7	28.3	8.6	68.2	42.2	51.3	32.4	49.8
	FADA [119]	✓	86.0	42.3	86.1	6.2	20.5	78.3	92.7	47.2	17.7	72.2	37.2	54.3	44.0	52.7
	Ours	✓	**95.6**	**78.9**	**94.3**	**45.9**	**70.3**	**93.0**	**96.2**	**63.3**	**51.3**	**90.5**	**83.6**	**84.8**	**56.5**	**55.7**

TABLE 5.3. Results for the Cross-City experiments. ST: Self-Training.

						GTA	Synthia
Step	IT	ST	A	W	D	mIoU	mIoU
Initialization	✓				$\mathcal{S}$	47.3	41.6
Adaptation	✓	✓			$\mathcal{S},\mathcal{T}$	49.8	43.5
	✓	✓	✓		$\mathcal{S},\mathcal{T}$	52.0	46.4
	✓	✓	✓	✓	$\mathcal{S},\mathcal{T}$	52.6	46.9
Distillation	✓	✓	✓		$\mathcal{T}$	53.5	48.4
Oracle					$\mathcal{T}$	63.8	65.1

TABLE 5.4. Ablation studies on GTA5→Cityscapes (second-to-last) and SYNTHIA→Cityscapes (last) columns. IT: image translation; ST: Self-Training; W: low-level adaptation; A: Data Augmentation; D: Training domain.

all other methods, pre-trains the baseline network not only on ImageNet[47] but also on MSCOCO[149]. We argue that pre-training on more tasks and real annotated data notably improves the baseline performance of the synthetic-to-real benchmark. For GTA5→Cityscapes, we note that, thanks to our low-level adaptation, we can boost performances for fine-detailed classes such as *Bicycle* and *Motorcycle*. Regarding SYNTHIA→Cityscapes, we obtain competitive performance, showing that our method can work also in this challenging scenario in which the source synthetic domain exhibits many bird's-eye views that are very different from the one in Cityscapes. Indeed our method is only slightly inferior to IAST[148] and again superior to [122] that performs a similar data augmentation.

5.3.3 Cross-city adaptation

We report in Tab. 5.3 our performance for the real-to-real setting. Our proposal shows great results, confirming the generalization properties of our contributions in diverse settings. We improve performance with respect to previous works for all the cities. Our model achieves 60% in mIoU in Rome, which is likely the most similar to the German cities used in the Cityscapes dataset. Nonetheless, we achieve strong results even for more distant domains, e.g. as in the case of Taipei, improving by 7.8% with respect to the model trained only on the source domain. For the Cross-city adaptation setting, differently from the other settings, we use images of both domains in our *distillation* step to exploit the perfect annotations available in the similar source domain.

5.3.4 Ablation Studies

In this section, we analyze the contribution provided by each component of our framework and motivate our design choices. In Tab. 5.4 we detail the results for both GTA5$\rightarrow$Cityscapes and SYNTHIA$\rightarrow$Cityscapes. The first row reports the performance obtained using only translated source domain images. This is nowadays a common building block of many UDA frameworks, and we also consider it our baseline on which we build our pipeline. In the adaptation section instead, we isolate both our contributions and use the model trained in the initialization step to extract pseudo-labels for the target domain as explained in Sec. 5.1.2 and train on both domains simultaneously. When applying a naive self-training strategy (i.e. training directly on pseudo-labels) we already obtain a significant boost (+2.5% and +1.9%) respectively. However, when deploying the proposed data augmentation (row 3), we observe an even greater boost: +4.8% for both settings. This clearly demonstrates the effectiveness of our data augmentation and its applicability to diverse scenarios. Then, applying the proposed low-level adaptation (row 4) also yields an additional contribution overall: about +0.6% on top of the data augmentation version. We argue that is noticeable, especially when performances are already high, as in our case, and the strongest competitors are all within a narrow window. Finally, in row 5, we distill our full model (i.e. row 4) into a simple Deeplab-v2 for efficient inference time and apply once again the proposed data augmentation. Remarkably, this further improves performance with respect to the distilled model and avoids the typical pseduo-labels overfitting behavior when employing many steps of self-training.

Moreover, to motivate our intuition that shallow features are amenable to guide the warping process, we compare the results obtained by applying our adaptation step in the GTA5$\rightarrow$Cityscapes setting at the three different levels of the backbone before the last module achieving 52.6%, 51.6%, and 51.8% mIoU for layers *Conv1*, *Conv2*, and *Conv3* respectively. Thus, the best result is achieved by using the first convolutional block of the architecture, while on *Conv2* and *Conv3* results are comparable (see Fig. 5.2 for layer names).

5.3.5 Performance Along Class Boundaries

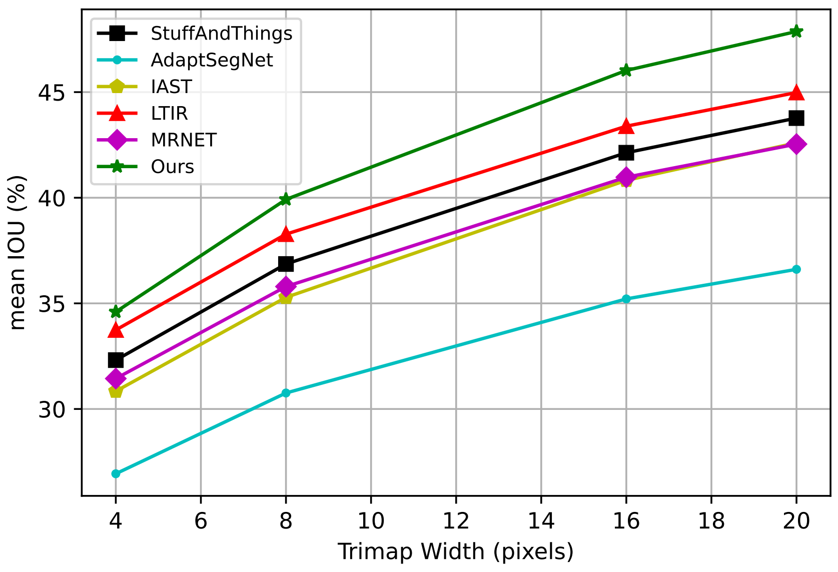


FIGURE 5.4. mIOU on GTA5→Cityscapes as a function of trimap band width around class boundaries.

In this section, we test the segmentation accuracy with the trimap experiment [153, 154, 155, 156] to quantify the accuracy of the proposed method along semantic edges. Specifically, we evaluate in terms of mIoU pixels within four bandwidths (4, 8, 16, 20 pixels) around class boundaries (trimaps). We first compare our final model against other frameworks in Fig. 5.4. We observe that our method is more accurate w.r.t. all other competitors in all the tested bandwidths, validating our main goal which is improving precision along class boundaries. We also highlight that although the green line is obtained from a distilled model (row 5 of Tab. 5.4), that does not include the additional modules presented in Sec. 5.1.1, it is still able to maintain strong performances at semantic boundaries thanks to the precise pseudo-labels extracted from the adaptation step. Then, we assess in Fig. 5.5 how our contributions affect performances on semantic boundaries. To this end, we repeat the same trimap experiment using the intermediate steps of our pipeline i.e. rows 2, 3, and 4 of Tab. 5.4. When applying all our contributions (purple line), we are able to improve by a large margin over the self-training strategy (black line) confirming that the additional modules account for an improvement along semantic edges. Furthermore, activating the low-level adaptation strategy maintains its improvements along semantic edges over the data augmentation only version (cyan line), leading to better pseudo-labels for the distillation step.

5.3.6 Comparison with other data augmentations

We compare our data augmentation, one of our main contributions, with the one introduced in [122]. More specifically, we apply this data augmentation in the adaptation step

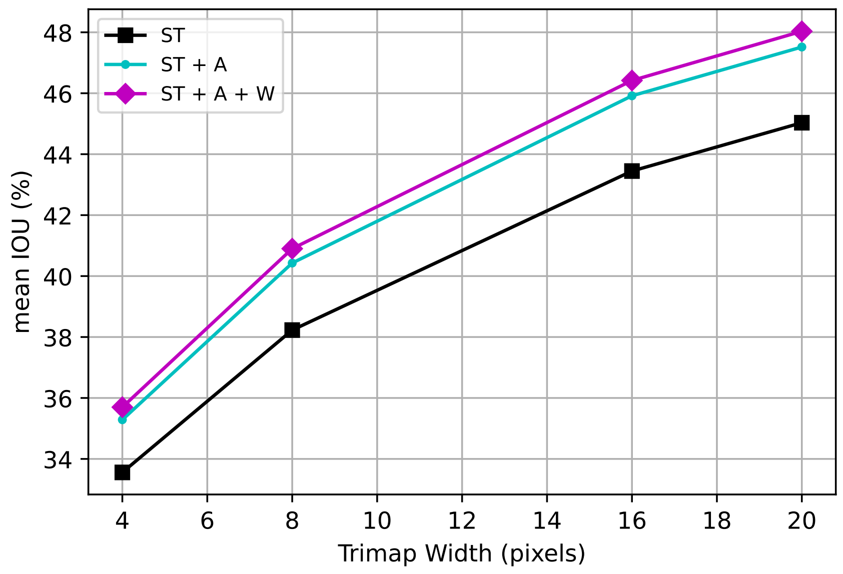


FIGURE 5.5. mIOU on GTA5→Cityscapes as a function of trimap band width around class boundaries. We report results for the three versions of the *adaptation* step of Tab. 5.4.

as in row 3 of Tab. 5.4, i.e. without the low-level adaptation modules to isolate the data-augmentation effect. We augment target images by randomly pasting objects from the source domain, using the open source implementation of [122]. With this strategy, we only obtain 51.0% in terms of mIoU, while with our technique the mIoU raises to 52.2%, confirming our intuition that looking for target instances is more effective than forcing the network to identify source objects as done [122] during the self-training step.

5.3.7 Displacement map visualization

In this section, we analyze the displacement map learned by the model. As Fig. 5.6 shows, the 2D map that guides the warping process is consistent with our intuition that the displacement is more pronounced at the boundaries, while areas within regions such as the body of a person are characterized by a low displacement (i.e. white color). Moreover, we can appreciate that when the warping is applied according to the estimated displacement field (top-right), the contours of small objects such as poles, traffic signs, and persons are better delineated (bottom-right). On the other hand, in the bottom-left mask, these objects are coarsely segmented when using a segmentation model train with translated images only. We also highlight that the displacement field is agnostic to semantic class (it only considers boundaries), and even though it captures other kinds of edges (i.e. not only semantic ones), it leads to computing an average of patches belonging to the same class.

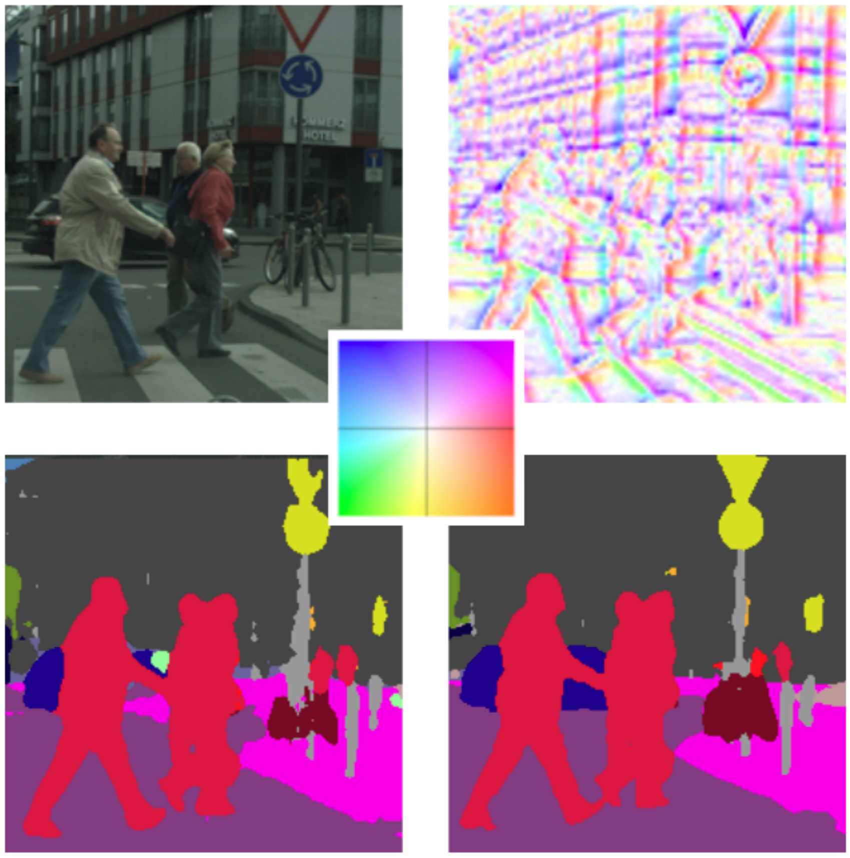


FIGURE 5.6. Top left: input target image. Top right: estimated 2D displacement. Bottom left: semantic map from a model trained on translated images. Bottom Right: Our results, improved on class boundaries by using the warping module. Colors and lightness in the middle indicate the warping direction with the corresponding intensity.

5.4 Conclusions

We proposed a novel framework for UDA for semantic segmentation that explicitly focuses on improving accuracy along class boundaries. We have shown that we can exploit domain-invariant shallow features to estimate a displacement map used to achieve sharp predictions along semantic edges. Jointly with a novel data augmentation technique that preserves fine edge information during self-training, our approach achieves better accuracy along class boundaries w.r.t. previous methods.

Part II

Domain Adaptation for 3D Data

Chapter 6

Initial Remarks

Properly reasoning on 3D geometric data such as point clouds or meshes is crucial for many 3D computer vision tasks, which are key to enabling emerging applications like autonomous driving, robotic perception, and augmented reality. In particular, assigning the right semantic category to a set of points that represent the surface of an object is a required skill for an intelligent system in order to understand the 3D scene around it. Such a problem referred to as *shape classification*, was initially addressed by *hand-crafted* features [157, 158, 159], while, with the advances in deep learning, recent proposals learn features directly from 3D point coordinates by means of deep neural networks [160, 161, 162, 163, 164, 165, 166, 167, 168, 169]. Although data-driven approaches can achieve impressive results, they require massive amounts of labeled data to be trained, which are cumbersome and time-consuming to collect. Typically, 3D deep learning methods use synthetic datasets of CAD models, *e.g.*,ModelNet40 [170] or ShapeNet [171], to harvest a large number of 3D examples. While synthetic datasets enable 3D deep learning, they create a conundrum. On the one hand, shape classifiers trained on ModelNet are very effective on synthetic data, as witnessed by performance saturation on standard benchmarks [172, 173]. On the other hand, though, they are not able to transfer their performance to real-world scenarios [172], where point cloud data are usually captured by RGB-D or LiDAR sensors [174, 175]. This limitation severely restricts the deployment of 3D deep learning methods in real-world applications. In light of the aforementioned rationale, this dissertation introduces two innovative domain adaptation methodologies designed to enhance the transfer of knowledge to an unlabeled target domain. In line with the previous chapters on 2D data, we examine again the usage of auxiliary tasks. More specifically, in Chapter 7, we leverage 3D point cloud reconstruction as a supplementary task. Conversely, in Chapter 8, we advocate the utilization of a self-supervised task to boost performance within the target domain.

6.1 Related Works

6.1.1 Unsupervised 3D Domain Adaptation

Only few papers discuss Unsupervised Domain Adaptation for point cloud classification. Among these, PointDAN [28] is a seminal work that proposes to adapt a classical 2D domain adaptation approach to the 3D world. Specifically, they focus on the alignment of both local and global features, building their framework upon the popular Maximum Classifier Discrepancy (MCD) [176] for global feature alignment. Differently, [29, 30] leverage on Self-supervised learning (SSL) to learn simultaneously distinctive features for both the source and target domains. Similarly, [29] also relies on SSL, and introduces a novel pretext task to learn strong features for the target domain as well.

6.1.2 Deep Learning for Point Clouds Reconstruction

With the recent advances in deep learning several methods for point cloud reconstruction have been suggested. A seminal work in this area [177] proposes a new auto-encoder architecture for point clouds using the permutation invariant operator introduced in [160]. AtlasNet [178] and FoldingNet [179] propose a *plane-folding* decoder to learn to deform points sampled from a plane in order to reconstruct the input surface. TearingNet [180] takes inspiration from [179, 178] and presents a *tearing* module to cut regular 2D patch with holes, or into several parts, to preserve the point cloud topology. In [181], the authors reconstruct the point clouds by training a generative adversarial network (GAN) on the latent space of unpaired clean synthetic and occluded real point clouds. We take inspiration from the finding of [177] and leverage the expressive power of point cloud auto-encoders to pre-train our shape classifier and learn a global shape descriptor deployed for pseudo-labels refinement.

6.1.3 UDA Datasets for Point Cloud Classification and Segmentation

The standard dataset used for UDA for point cloud classification is PointDA-10 [28], which consists of three subsets that share the same ten classes of three popular point clouds classification datasets: ShapeNet [171], ModelNet40 [182] and ScanNet [174]. This allows to define six different scenarios that involve synthetic-to-synthetic, synthetic-to-real and real-to-synthetic adaptation. ModelNet-10 consists of 4,183 training and 856 testing point clouds, that are extracted from synthetic 3D CAD models. Similarly, ShapeNet-10 features synthetic data only. It is the largest and most varied among the three datasets, and it comprises 17,378 training and 2,492 testing samples. ScanNet-10 is the only real datasets among these three, and it consists of 6,110 and 1,769 training and testing point clouds, respectively. It has been obtained from multiple real RGB-D scans. For this reason, it exhibits several forms of noise such as errors in the registration process and occlusions. A second additional real dataset that will be used for our experiments is ScanObjectNN [172] which is composed of 2902 3D scans from 15 categories. Similarly to ScanNet-10, it represents a challenging scenario due to the high diversity with respect to synthetic datasets and the presence of artifacts such as non-uniform point density, missing parts and occlusions. Several variants of the ScanObjectNN dataset are provided. As in [29], we select the OBJ_ONLY split which contains only foreground vertices. As regards the problem of part segmentation, there is no established setting in the literature and we refer to [29] as a reference since it is the only work performing synthetic-to-real adaptation from ShapeNetPart [183] to ScanOBJ-BG, wich is another split from ScanObjectNN. The task is solved only for the *chair* class, which comprises 4 components to segment: *Seat*, *Back*, *Base*, *Arm*.

Chapter 7

RefRec: Pseudo-labels Refinement via Shape Reconstruction

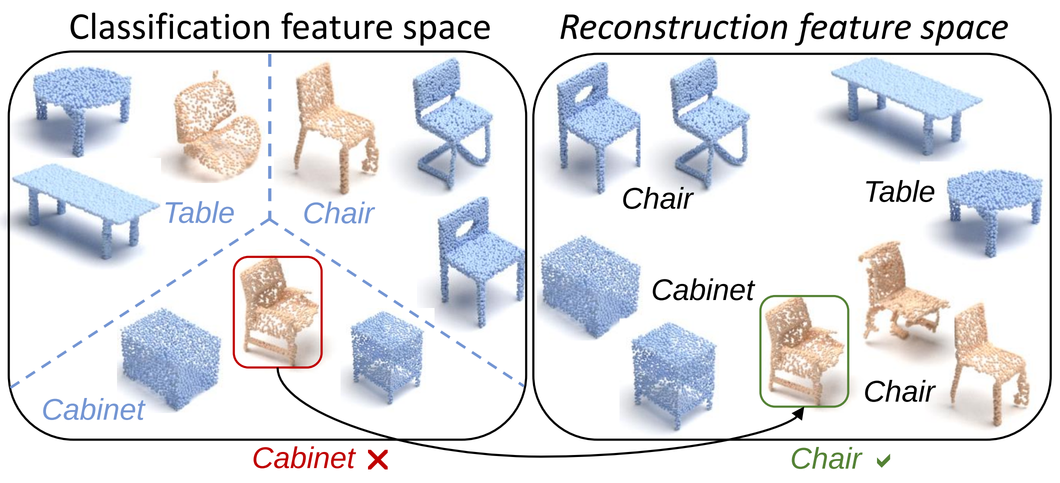


FIGURE 7.1. Feature space of a classifier (left) and a reconstruction auto-encoder (right). A classifier trained on source shapes only (blue) may not be effective on target shapes (orange) and assign wrong pseudo-labels. An auto-encoder, instead, tends to cluster similar shapes together in the learned embedding, such that its features can be used as global shape descriptors to correct wrong pseudo-labels.

UDA has its roots in 2D computer vision, where a multitude of methods have been proposed [184]. Among them, the most widespread approach pertains globally aligning the feature distributions between the source and target domain. This is the paradigm also leveraged by methods tackling UDA for 3D data, either explicitly, by designing losses and models to align features [28], or implicitly, by solving a self-supervised task on both the source and target domains with a shared encoder [29, 30]. We argue that, when moving from a synthetic CAD dataset to a real-world one, feature alignment can only lead to sub-optimal solutions due to the large differences between the two domains. Indeed, acquiring objects in cluttered scenes results oftentimes in partial scans with missing parts due to *occlusions*. Moreover, registration errors arising when fusing multiple 2.5D scans [185] to obtain a full 3D shape, alongside noise from the sensor, resulting in less clean and geometrically coherent shapes than CAD models. Therefore, the shape classifier may need to ground its decision on new cues, different from those learned on the source domain, where the clean full shape is available, to correctly classify target samples.

To let the classifier learn such new cues, we propose RefRec (Refinement via Reconstruction), a novel framework to perform unsupervised domain adaptation for point cloud classification. Key to our approach is reliance on *pseudo-labels* [186] and self-training, wich we introduced and exploited in the previuos chapters as well. As usual however, the pseudo-labels obtained from a model trained on the source domain may be wrong due to the domain shift (as shown in Fig. 7.1-left) and a target domain classifier naively trained on them would underperform. Therefore, our key contribution concerns effective approaches to *refine* pseudo-labels. We propose both an offline and an online refinement, *i.e.*, before training and while training on pseudo-labels. Both refinements are based on the idea that similar shapes should share similar labels. To find similar shapes, we match *global shape descriptors*, *i.e.*, the embedding computed by an encoder given the input shape. Here we make another key observation, illustrated in Fig. 7.1: the space of features learned by a classifier is organized to create linear boundaries among different classes, but it is not guaranteed -nor meant- to possess a structure where similar shapes lay close one to another, especially for target samples which are not seen at training time. Hence, such features are not particularly effective if used as global shape descriptors. In contrast, teaching a *point cloud auto-encoder* to reconstruct 3D shapes is an effective technique to obtain a compact and distinctive representation of the input geometric structures, as proven by recent proposals for local and global shape description [179, 178, 187, 188], which, in our setting, can be trained also on the target domain since it is learned in an unsupervised way. By leveraging on such properties of the reconstruction latent space, in the offline step we focus on reassigning the pseudo-labels of target samples where the source domain classifier exhibits low confidence, while in the online step, we compute *prototypes* [189], *i.e.*, the mean global descriptors on the target domain for each class, and we weight target pseudo-labels according to the similarity of the input shape to its prototype. Peculiarly, by using reconstruction embeddings trained also on the target domain to compute prototypes, we avoid the domain shift incurred when using the classifier trained on the source domain as done by previous 2D methods [190]. In the online refinement step, we also leverage the standard training protocol of 2D UDA methods based on mean teacher [191] to improve the quality of pseudo-labels as training progresses.

7.1 Method

As in the classical UDA setting, we assume the availability of a labeled source domain $\mathcal{S} = \{(\mathbf{x}_s^i \in \mathcal{X}_s, y_s^i \in \mathcal{Y}_s)\}_{i=1}^{n_s}$, and a target domain $\mathcal{T} = \{\mathbf{x}_t^j \in \mathcal{X}_t\}_{j=1}^{n_t}$, whose labels $\{y_t^j \in \mathcal{Y}_t\}_{j=1}^{n_t}$ are, however, not available. As in standard UDA settings [57], we assume to have the same one-hot encoded label space $\mathcal{Y}_s = \mathcal{Y}_t = \mathcal{Y} = \{0, 1\}^k$ and the same input space $\mathcal{X}_s = \mathcal{X}_t$ (*i.e.*, point clouds with $\{x, y, z\}$ coordinates) but with different distributions $P_s(\mathbf{x}) \neq P_t(\mathbf{x})$, *e.g.*, due to source data being synthetic while target data being real or due to the use of different sensors. The final classifier for a point cloud $\mathbf{x}$ can be obtained as a composite function $\Omega = \Phi \circ \Psi$, with $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$ representing the feature extractor and $\Psi : \mathbb{R}^d \rightarrow [0, 1]^k$ the classification head, which outputs softmax scores $\hat{\mathbf{p}} \in [0, 1]^k$. When one-hot labels are needed, we further process softmax scores with $\Lambda : \mathbb{R}^k \rightarrow \mathcal{Y}$ to obtain the label corresponding to the largest softmax score, with such value providing also the confidence associated with the label prediction. Although the largest softmax is a rather naive confidence measure, we found it to work satisfactorily in our experiments. Our overall goal is to learn a strong classifier Ω_t for the target domain even though annotations are not available therein.

An overview of our method is depicted in Fig. 7.2. It encompasses three major steps: warm-up, pseudo-labels refinement, and self-training. The purpose of the first step, described in Sec. 7.1.1, is to train a model effective on target data by using labeled source data and unlabelled target data. Once trained, this model provides the initial pseudo-labels. These are refined offline in the second step, described in Sec. 7.1.2, in order to partially reduce the errors in pseudo-labels due to the limited generalization of the source classifier to the target domain. Finally, in the last step, detailed in Sec. 7.1.3, we introduce an effective way of exploiting pseudo-labels during self-training by combining a domain-specific classifier with an online pseudo-labels weighting strategy that exploits prototypes computed in the target domain.

7.1.1 Pseudo-labels Warm-up

The first step of our pipeline seeks to produce good initial pseudo-labels, which, after refinement, can be used to train the final classifier on the source domain and the target domain augmented with pseudo-labels. The warm-up step is very important, as the effectiveness of self-training is directly related to the quality of pseudo-labels. To obtain good initial pseudo-labels, we focus on pretraining and data augmentation, with the aim of reducing overfitting on source data. Pre-training is largely adopted also in UDA for image classification, where ImageNet pre-training is a standard procedure [190, 192, 193] that learns powerful features able to generalize to multiple domains and alleviates the risk of overfitting when training solely on data coming from the source distribution. Differently from UDA in the 2D world, however, here we focus on *unsupervised* pre-training. This is particularly attractive for the UDA context, where no supervision is available for the target domain. In fact, inspired by recent advances on representation learning for 3D point clouds, which have demonstrated the effectiveness of unsupervised techniques for learning discriminative features [179, 178, 187, 188], we propose to use point cloud reconstruction as unsupervised pre-training for our backbone. The key advantage of such pre-training is the possibility to capture discriminative features also for the target domain since unsupervised pre-training can be conducted on both domains simultaneously. Moreover, it learns a feature extractor Φ_{rec} which can be deployed also to refine labels effectively, as we do in the following steps of our pipeline.

We follow the same strategy proposed in [177], and use a standard PointNet [160] backbone as Φ_{rec} to produce a global d-dimensional descriptor of the input point cloud. This latent representation is then passed to a simple decoder made out of 3 fully connected layers that tries to reconstruct the original shape. During training, we minimize both the Chamfer Discrepancy (CD) $\mathcal{L}_{CD}$ and Earth Mover's distance (EMD) $\mathcal{L}_{EMD}$ [194] as loss functions [177]. Additionally, as mentioned above, data augmentation is a key ingredient to improve generalization, especially for the synthetic-to-real adaptation case, since 3D real scans always exhibit occlusions and non-uniform point density. For this reason, when performing synthetic-to-real adaptation, we apply a data augmentation procedure similar to that proposed in [195] in order to simulate occlusions.

To conclude warm-up, we train a new classifier $\Omega^w = \Phi^w_{cls} \circ \Psi^w$ on the source dataset with a classical cross-entropy loss. Importantly, Φ^w_{cls} and Φ_{rec} have the same architecture and the weights θ of the backbone Φ^w_{cls} are initialized with those learned for Φ_{rec}. We then use Ω^w to obtain pseudo-labels $\{\hat{y}^j_t = \Lambda(\Omega^w(\mathbf{x}^j_t)\}^{n_t}_{j=1})$ alongside their confidence scores.

We may, in principle, exploit these pseudo-labels to perform self-training in the target domain. However, even if we rely on unsupervised pre-training and data augmentation to boost performance on the target domain, they are still noisy. Indeed, due to the domain gap, only a small portion of the pseudo-labels can be considered reliable, while the majority of the samples are assigned wrong labels that could lead to poor performance when applying self-training. Hence, in the next step, we refine the initial pseudo-labels obtained in the warm-up step by leveraging on Φ^w_{rec}.

7.1.2 Pseudo-labels Refinement

To refine pseudo-labels, we exploit the confidence computed by the classifier Ω^w and split pseudo-labels in the two disjoint sets of highly confident predictions, denoted as $\mathcal{E}$ (*i.e.*, *easy split*), and uncertain ones, denoted as $\mathcal{H}$ (*i.e.*, *hard split*). We first build $\mathcal{E}$ by selecting the g=*10%* most confident predictions on the target samples for each class. We perform this operation class-wisely to obtain a sufficient number of examples for each class and to reproduce the class frequencies in $\mathcal{E}$. $\mathcal{H}$ is composed by all the remaining target samples. One of the key ideas behind RefRec is to utilize the embedding of the reconstruction backbone Φ_{rec}, instead of Φ^w_{cls}, to improve the labels of the samples in $\mathcal{H}$. We conjecture that since Φ^w_{cls} has been trained only on the source domain, its embeddings are not discriminative for the target domain, and more importantly there are no guarantees that objects belonging to the same class, yet coming from different domains, would lay close in the feature space. Hence,

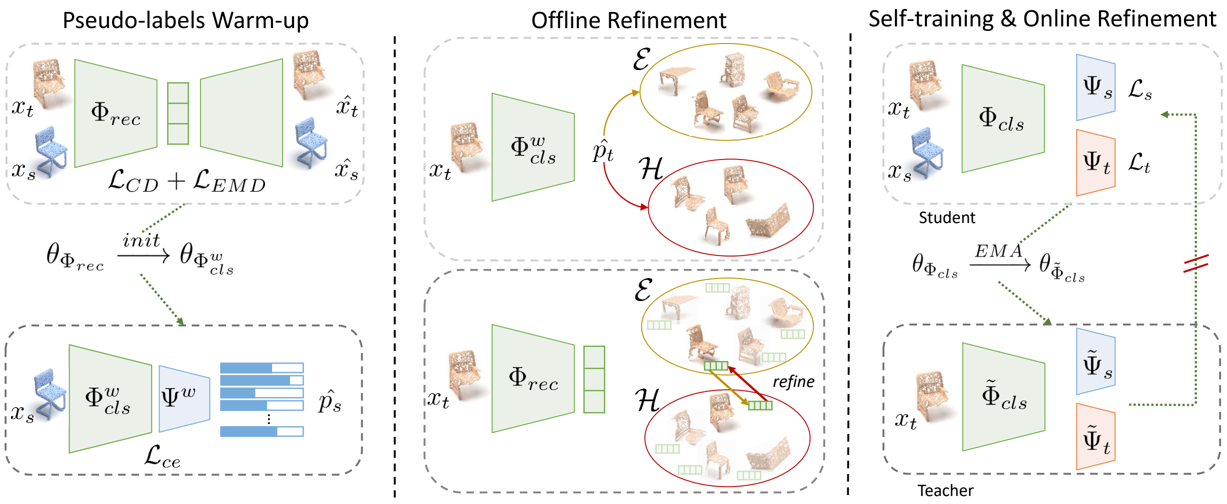


FIGURE 7.2. RefRec comprises three steps. First, in the *pseudo-labels warm up* step, we train a reconstruction network Φ_{rec} on both source and target domains. The weights of the encoder are used to initialize the backbone Φ^w_{cls} of a classifier, that is then trained on the source domain. In the *refinement* step, we use the classifier to split target samples into the easy ($\mathcal{E}$) and hard ($\mathcal{H}$) sets according to their confidence and refine them by performing nearest neighbor queries in the auto-encoder feature space. Finally, in the *self-training* step, we train a target-specific classifier Ψ_t by refined pseudo-labels and online pseudo-labels obtained with the mean teacher architecture [191].

we assign new pseudo-labels to target samples according to similarities in the feature space of Φ_{rec}.

We first seek to expand the set of easy examples $\mathcal{E}$ by finding in the feature space of Φ_{rec} the nearest neighbor in $\mathcal{H}$ for each sample of $\mathcal{E}$ and vice versa. To refine pseudo-labels for samples in $\mathcal{H}$, we adopt a well-known technique employed for surface registration [187, 196, 188] and accept only reciprocal nearest neighbor matches, *i.e.*, pairs of samples that are mutually the closest one in the feature space: if one sample $h \in \mathcal{H}$ is the nearest neighbour in $\mathcal{H}$ of a sample $e \in \mathcal{E}$ and e is, in turn, the nearest neighbor of h in $\mathcal{E}$, we move h to the easy split and label it according to the pseudo-label of e. At the end of this procedure, we obtain a refined set of easy examples $\mathcal{E}^r$.

We then try to refine the pseudo-labels for the samples left in $\mathcal{H}$ exploiting $\mathcal{E}^r$. In particular, we select K-nearest neighbors ($K = 3$ in our experiments) in the refined set $\mathcal{E}^r$ for each remaining sample $h \in \mathcal{H}$, and assign the new pseudo-label to h by majority voting. When there is no consensus among the K neighbors, we assign the pseudo-label of the closest one. This produces the refined set of hard examples $\mathcal{H}^r$. It is important to note that the entire process is applied offline before the self-training step, as illustrated in Fig. 7.2, and the absence of hard thresholds in all the refinement steps facilitates the applicability of the proposed method across datasets.

7.1.3 Self-training

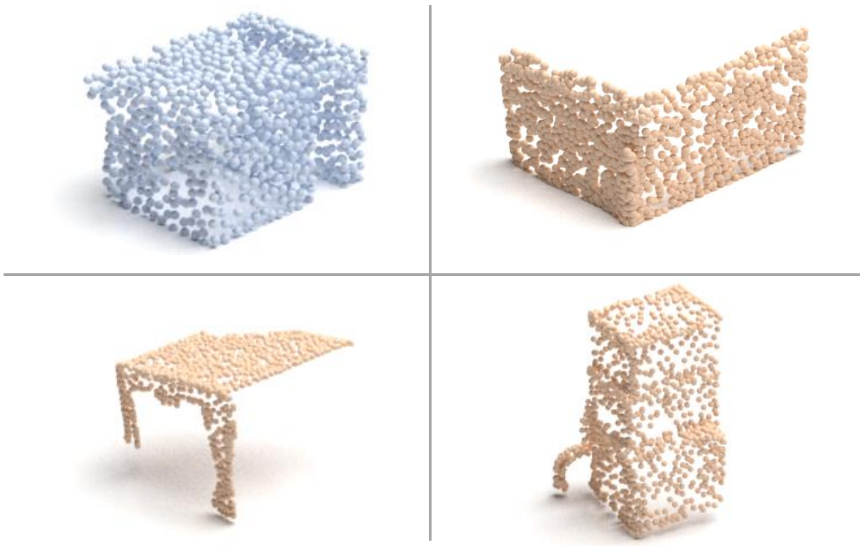


FIGURE 7.3. Samples from *cabinet*. First row: intra-class variability between domains (ModelNet and ScanNet). Second row: intra-class variability in ScanNet.

When performing synthetic-to-real adaptation and vice-versa, the gap between the two distributions could be large and difficult to reduce even in case of perfect supervision. As a matter of example, Fig. 7.3 shows how shapes, such as *cabinets*, may look very different across domains (first row) as well as within a domain (second row). A high intra-class variability can be somehow dealt with in a supervised setting, but it is harder to handle when noisy supervision in the form of pseudo-labels must be used. Hence, in this setting, it is difficult for a neural network to find common features for shapes belonging to the same class across domains. We address this issue by adopting domain-specific classification heads together with online pseudo-labels refinement while performing the self-training step that concludes our pipeline.

Domain-specific classification heads. To tackle intra-class variability across domains, we deploy a shared encoder Φ_{cls}, initialized using the weights from Φ_{rec}, and attach two domain-specific classification heads, Ψ_s and Ψ_t, for the source and target domain, respectively. The benefit of having a target-specific head (right) versus a single head trained on both domains (left) is highlighted in Fig. 7.4.

When using a single classification head, the model tries to separate classes for both domains simultaneously, which may lead to a non-optimal decision boundary. Indeed, due to the high intra-class variability across the domains it is not possible to find a single decision boundary to correctly separate all samples for both, leading to some wrongly classified samples. When employing two domain-specific heads, instead, the model can learn two more effective boundaries.

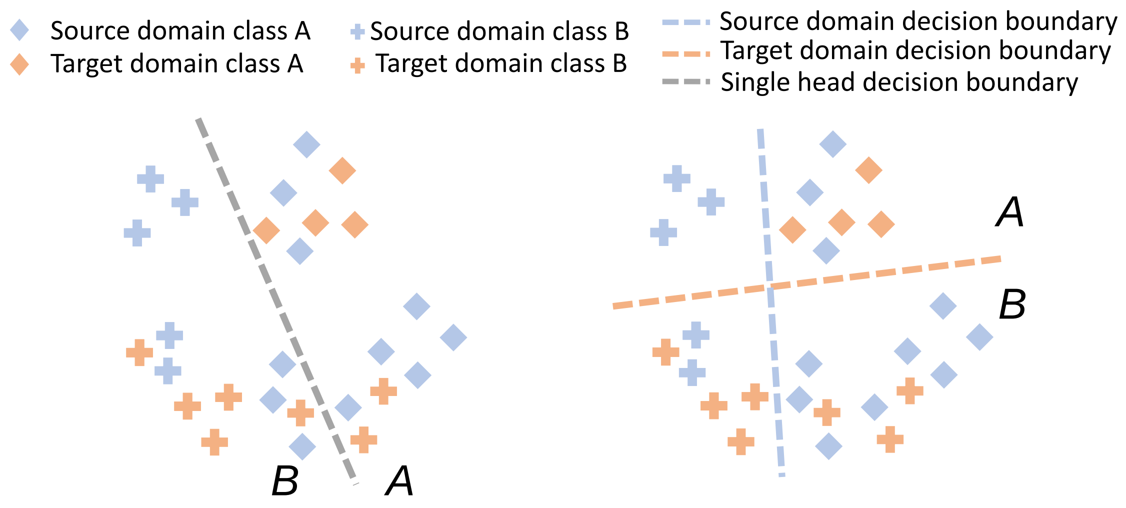


FIGURE 7.4. Single head vs domain-specific classification heads. When a single head is deployed (left side), it may not be possible to find a linear decision boundary that correctly classifies both classes for the source and target domain. When domain-specific classification heads are deployed, the model can focus on each domain separately and learn more effective decision boundaries (right side).

Although domain-specific classification heads have been already explored in UDA for image classification [197], in RefRec we can apply them in a unique way, which makes them more effective, as it will been shown in our ablation studies. In particular, we train the first head Ψ_s on the source domain mixed with $\mathcal{E}^r$, while we supervise the second head using target data only, *i.e.*, both $\mathcal{E}^r$ and $\mathcal{H}^r$. By doing this, we force both heads to correctly classify the easy split, which contains the most confident predictions, *i.e.*, the set of target samples already more aligned with the source domain. Thereby, training with this strategy does not reduce performance on the source domain while it enforces a partial feature alignment across domains. At the same time, by only feeding target data to Ψ_t, we let the model define target-specific boundaries, alleviating the negative impact of the intra-class variability across domains.

Online pseudo-labels refinement. Even when using domain-specific classification heads, the intra-class variability on the target domain can still affect the model performance. To deal with this issue we adopt an online pseudo-labels refinement and weighting strategy. The key intuition behind online refinement is that, as training progresses, our classifier better learns how to classify the target domain thanks to the pseudo-labels and thus we can progressively improve the pseudo-labels by exploiting such freshly learned knowledge. Purposely, we exploit the mean teacher [191] of our model in order to obtain online pseudo-labels $\tilde{y}_t$:

$$\tilde{y}_t = \Lambda(\tilde{\Psi}_s(\tilde{\Phi}_{cls}(\mathbf{x}_t)) + \tilde{\Psi}_t(\tilde{\Phi}_{cls}(\mathbf{x}_t))) \tag{7.1}$$

It is important to note that $\tilde{\Phi}_{cls}$, $\tilde{\Psi}_s$, and $\tilde{\Psi}_t$ are never updated trough gradient back-propagation, as they consist of simple temporal exponential moving averages (EMA)of their student counterparts [191]. At each training step, we feed one batch of samples from $\{\mathcal{S}, \mathcal{E}^r\}$ and one batch from $\{\mathcal{E}^r, \mathcal{H}^r\}$ to train $\Phi_{cls} \circ \Psi_s$ and $\Phi_{cls} \circ \Psi_t$, respectively. As for the source classifier, we train it by the standard cross-entropy loss with labels for the source domain and the pseudo-labels $\hat{y}_t$ obtained from the refinement step for $\mathcal{E}^r$:

$$\mathcal{L}_s = \mathcal{L}_{ce}(\mathbf{p}_s, \mathbf{y}_s) + \mathcal{L}_{ce}(\mathbf{p}_t, \hat{\mathbf{y}}_t) \tag{7.2}$$

We instead exploit both the refined ($\hat{\mathbf{y}}_t$) and the on-line ($\tilde{\mathbf{y}}_t$) pseudo-labels when training the target classifier:

$$\mathcal{L}_t = (1 - \alpha_{it}) z_t \mathcal{L}_{ce}(\mathbf{p}_t, \hat{\mathbf{y}}_t) + \alpha_{it} z_t \mathcal{L}_{ce}(\mathbf{p}_t, \tilde{\mathbf{y}}_t) \tag{7.3}$$

where α_{it} is a weighting factor that starts from 0 (use only refined pseudo-labels) and increases at every iteration up to 1 (use only online pseudo-labels). Intuitively, when self-training starts, we trust the previously refined pseudo-labels and thus give more weight to the first term of Eq. (7.3) as the mean teacher is not reliable yet. As training goes on, we progressively trust more the output of the mean teacher, *i.e.*, $\tilde{y}_t$, and so give more weight to the second term. z_t is instead a weighting factor that accounts for the plausibility of the pseudo-label. z_t is computed for each target sample exploiting once again the embedding of Φ_{rec}. In particular, before the self-training step, we compute the class-wise prototypes $\eta^{(k)} \in \mathbb{R}^d$ as the class-wise average of the target features in the easy split:

$$\eta^{(k)} = \frac{\sum_{x_t \in \mathcal{E}^r} \Phi_{rec}(x_t) * \mathbb{1}(\hat{y}_{t,k} == 1)}{\sum_{x_t \in \mathcal{E}^r} \mathbb{1}(\hat{y}_{t,k} == 1)} \tag{7.4}$$

where $\hat{y}_{t,k}$ is the k-entry in $\hat{y}_t$. We only consider $\mathcal{E}^r$ as it contains the most reliable pseudo-labels for the target domain. We then obtain the confidence score for each sample by simply computing the softmax of the opposite of the distance between the current embedding of x_t and the prototype of the class k assigned to it in its online pseudo-label, $k = \arg\max_{k'} \tilde{y}_{t,k'}$:

$$z_t = \frac{\exp\left(-\|\tilde{\Phi}_{cls}(\mathbf{x}_t) - \eta^{(k)}\|_2\right)}{\sum_{k'} \exp\left(-\|\tilde{\Phi}_{cls}(\mathbf{x}_t) - \eta^{(k')}\|_2\right)}. \tag{7.5}$$

Hence, z_t forces the loss to ignore samples that are far from the expected prototype. In fact, when a sample has a very different representation from the expected class prototype, either the pseduo-label is wrong or the input point cloud is an outlier in the target distribution, and dynamically weighting it less in the self-training process allows for learning a better classifier.

7.2 Experiments

We evaluate our method on two standard datasets for point cloud classification: PointDA-10 [28] and ScanObjectNN [172].

7.2.1 Implementation details

As done in all previous 3D DA methods [28, 29, 30], we use the well-known PointNet [160] architecture. In particular, we use the standard PointNet proposed for point cloud classification for all our backbones Φ^{w}_{cls}, Φ_{cls}, and Φ_{rec}. It produces a 1024-dimensional global feature representation for each input point cloud. We train the reconstruction network for 1000 epochs in the unsupervised pre-training step [177], while we train only for 25 epochs when training classification networks in each step of our pipeline. We set to 0.0001 both learning rate and weight decay. We train with batch size 16 using AdamW [198] with cosine annealing [199] as optimizer. The framework is implemented in PyTorch [150], and is available at `https://github.com/CVLAB-Unibo/RefRec`. At test time, we use the target classifier $\Phi_{cls} \circ \Psi_t \circ \Lambda$ in the target domain.

Method	ModelNet to ShapeNet	ModelNet to ScanNet	ShapeNet to ModelNet	ShapeNet to ScanNet	ScanNet to ModelNet	ScanNet to ShapeNet	Avg
No Adaptation	80.2	43.1	75.8	40.7	63.2	67.2	61.7
PointDAN [28]	80.2	45.3	71.2	46.9	59.8	66.2	61.6
DefRec [30]	80.0	46.0	68.5	41.7	63.0	68.2	61.2
DefRec+PCM [30]	81.1	50.3	54.3	52.8	54.0	69.0	60.3
3D Puzzle [29]	**81.6**	49.7	73.6	41.9	65.9	68.1	63.5
RefRec (Ours)	81.4	**56.5**	**85.4**	**53.3**	**73.0**	**73.1**	**70.5**
Oracle	93.2	64.2	95	64.2	95.0	93.2	

TABLE 7.1. Shape classification accuracy (%) on the PointDA-10 dataset. For each method, we report the average results on three runs. Best result on each column is in bold.

Method	ModelNet to ScanObjectNN
No Adaptation	49.6
PointDAN [28]	56.4
3D Puzzle [29]	58.5
RefRec (Ours)	**61.3**

TABLE 7.2. Shape classification accuracy (%) on the ScanObjectNN dataset. For each method, we report the average results on three runs. Best result is in bold.

7.2.2 Results

We report and discuss here the results of RefRec, and compare its performance against previous work as well as the baseline method trained on the source domain and tested on the target domain (referred to as No Adaptation). For each experiment, we provide the mean accuracy obtained on three different seeds. Since in UDA target annotations are not available, we never use target labels to perform model selection and we always select the model that gives the best result on the validation set of the source dataset.

PointDA-10. We summarize results for each benchmark in Tab. 7.1. Overall, our proposal improves by a large margin the previous state-of-the-art methods. Indeed, on average we obtain 70.5% against the 63.5% obtained by 3D puzzle [29], which is an improvement of 7% in terms of accuracy. From Tab. 7.1, it is also possible to observe how our method is consistently better than previous works in the synthetic-to-real adaptation scenario, which we consider the most important for practical applications. Compared to DefRec+PCM [30], which obtains 50.3 in the ModelNet→ScanNet setting, we improve by 6.2%. As regards ShapeNet→ScanNet, we obtain 53.3, surpassing by 0.5% DefRec+PCM again. Moreover, we highlight how RefRec seems to be the only framework able to generalize well to all adaptation scenarios. In fact, when comparing our proposal to DefRec+PCM which was the strongest method for the synthetic-to-real case, we also improve by a large margin in cases such as ShapeNet→ModelNet and ScanNet→ModelNet, where DefRec+PCM seems to fail. Finally, the ability of RefRec to handle large distribution gaps is also confirmed by the large improvements in the real-to-synthetic cases. Indeed, we observe a +7.1% improvement for ScanNet→ModelNet and +4.1% for ScanNet→ShapeNet.

Experiment		**ModelNet to**	**ModelNet to**	**ShapeNet to**	**ShapeNet to**	**ScanNet to**	**ScanNet to**	
Training data	Offline ref.	**ShapeNet**	**ScanNet**	**ModelNet**	**ScanNet**	**ModelNet**	**ShapeNet**	**Avg**
$\mathcal{S},\mathcal{E},\mathcal{H}$		**82.2**	51.7	80.4	43.4	64.7	**70.0**	65.4
$\mathcal{S},\mathcal{E}$		82.8	49.6	79.0	43.8	64.1	69.8	64.9
$\mathcal{S},\mathcal{E}^r,\mathcal{H}^r$	✓	79.1	**57.3**	**85.6**	**50.7**	**70.1**	69.1	**68.7**

TABLE 7.3. Ablation study on the effect of offline refinement. We report the average shape classification accuracy (%) on three runs.

Experiment				**ModelNet to**	**ModelNet to**	**ShapeNet to**	**ShapeNet to**	**ScanNet to**	**ScanNet to**	
Ψ_s	Ψ_t	EMA	Online ref.	**ShapeNet**	**ScanNet**	**ModelNet**	**ScanNet**	**ModelNet**	**ShapeNet**	**Avg**
$\mathcal{S}$	$\mathcal{E}^r,\mathcal{H}^r$			79.5	55.3	84.7	49.3	72.0	68.5	68.2
$\mathcal{S},\mathcal{E}^r$	$\mathcal{E}^r,\mathcal{H}^r$			79.3	**56.9**	84.7	51.6	71.7	69.0	68.9
$\mathcal{S},\mathcal{E}^r$	$\mathcal{E}^r,\mathcal{H}^r$	✓		80.3	54.2	83.2	52.7	72.8	71.6	69.1
$\mathcal{S},\mathcal{E}^r$	$\mathcal{E}^r,\mathcal{H}^r$	✓	✓	**81.4**	56.5	**85.4**	**53.3**	**73.0**	**73.1**	**70.5**

TABLE 7.4. Ablation study on the effect of the self-training strategy and online refinement. We report the average shape classification accuracy (%) on three runs.

ScanObjectNN. In Tab. 7.2 We report the results for the challenging ModelNet→ScanObjectNN adaptation task. On this challenging benchmark, we achieve 61.3%, which is 2.8% better than the previous state-of-the-art result.

7.2.3 Ablation studies

To validate the importance of our design choices, we conduct some ablation studies on both the pseudo-labels refinement process and the self-training strategy.

Pseudo labels refinement. In Tab. 7.3, we show the effect of our refinement process. When performing self-training with the initial, unrefined pseudo-labels produced by the classifier Ψ_s after the warm-up step (first row), we obtain an overall accuracy of 65.4%. Conversely, when we apply our descriptor matching approach aimed at pseudo-labels refinement (third row), the accuracy increases to 68.7%. This confirms our intuition that using the reconstruction network allows to capture similarities among shapes in feature space and consequently to improve the pseudo-labels. Moreover, we compare self-training using the most-confident pseudo-labels only (second row) against self-training with the refined pseudo-labels (third row). The improvement given by the refinement approach (+3.8%) suggests that only using the most confident pseudo-labels is not enough to reach good performance.

Self-training strategy. In Tab. 7.4, we show the effectiveness of our strategy to perform self-training and ablate our design choices compared with other reasonable alternatives. When deploying the domain-specific classification heads, and training Ψ_s solely with source data (first row), results are worse than when we train Ψ_s with both source and $\mathcal{E}^r$ (second row). This is more evident for the synthetic-to-real adaptation and vice versa, where partial alignment in feature space is more difficult to attain. Indeed in all four cases, forcing Ψ_s to correctly classify the target easy split is beneficial. On the other hand, for the synthetic-to-synthetic case, performances remain stable. This is an expected behaviour since the decision boundaries in these easy adaptation scenarios should not vary significantly across domains. Finally, in the last two rows, we ablate the effect of the mean teacher and the online refinement, respectively. The mean teacher only gives a marginal contribution (+0.2%), while its combination with our online refinement accounts for a 1.4% improvement.

Method	ModelNet to ShapeNet	ModelNet to ScanNet	ShapeNet to ModelNet	ShapeNet to ScanNet	ScanNet to ModelNet	ScanNet to ShapeNet	Avg
No Adaptation	80.2	43.1	75.8	40.7	63.2	67.2	61.7
Warm-up	81.3	51.4	78.9	43.8	59.7	67.5	63.7
Multi-task	80.6	45.4	78.9	46.0	63.9	67.4	63.7
Self-train multi-task	81.2	46.9	76.3	47.7	66.0	66.5	64.1
RefRec (Ours)	**81.4**	**56.5**	**85.4**	**53.3**	**73.0**	**73.1**	**70.5**

TABLE 7.5. Ablation study on the effect of pre-training. We report the average shape classification accuracy (%) on three runs.

Warm-up vs SSL. Finally, we aim to shed some light on the importance of unsupervised pre-training, *i.e.*, warm-up, compared to SSL, which is so far the most studied approach to UDA for point cloud classification. In Tab. 7.5, we compare our warm-up step (second row), which exploits unsupervised pre-training, with a multi-task approach as done in [29, 30] (multi-task, third row), where the SSL task of shape reconstruction is solved by an auxiliary head. For a fair comparison, we adopt in both cases our data augmentation in the synthetic-to-real setting, and train the multi-task architecture for 150 epochs, as done in [29] and [30], since no pre-training is applied. Although a simple comparison between such baselines does not establish a clear winner (63.7 on average in both cases), we observe a remarkable difference after the self-training stage. Indeed, when comparing the classifier self-trained with pseudo-labels obtained with the multi-task approach (fourth row) against the classifier self-trained with refined pseudo-labels (last row), and applying the mean teacher in both cases, we observe a remarkable gap (+7%).

7.3 Conclusions

We improved the state of the art in UDA for point cloud classifications. We showed how solving 3D UDA by means of self-training with supervision from robust pseudo-labels is a superior paradigm with respect to the established way of tackling it by multi-task learning. Key contributions we make are effective ways to refine pseudo-labels, offline and online, by leveraging shape descriptors learned to solve shape reconstruction on both domains, as well as a carefully designed self-training protocol based on domain-specific classification heads and improved supervision by an evolving mean teacher. We hope our results will call for more explorations around the use of pseudo-labels and self-training in this emerging area of research.

Chapter 8

Self-Distillation for Unsupervised 3D Domain Adaptation

The main line of research in 3D UDA for point cloud classification focuses on learning an effective feature space for the target domain by means of auxiliary tasks such as point cloud reconstruction [30, 200], 3D puzzle sorting [29] and rotation prediction [201]. These tasks are referred as auxiliary since they do not directly solve the main task, but at the same time, they are useful to learn features for the target domain without the need of annotations. Although such techniques considerably improve over the baseline (i.e., training only on source data), the design of such tasks is not trivial and typically lead to sub-optimal solutions. It requires identifying one that can drive the network to learn representations *discriminative* enough to perform classification in the target domain effectively.

Despite the fact they force some degree of alignment between the features computed on objects from the two domains, such auxiliary tasks do not explicitly steer the network to learn discriminative representations amenable to classification in the target domain. For instance, if we train a network to reconstruct shapes, we will get similar point cloud embeddings for similar 3D shapes. However, two point clouds could represent objects that, though similar in shape, do belong to different categories, *e.g.*, a cabinet and a bookshelf. As a consequence, relying on reconstruction to perform domain adaptation can align features between the two domains, with similar shapes embedded close one to each other regardless of their domain, but the decision boundaries learnable from the labeled source samples may not discriminate effectively between target samples belonging to different classes. This is also shown in [30], where a simple denoising autoencoder for point clouds only slightly improves performance over the baseline. We reckon that similar considerations apply to the other auxiliary tasks proposed in the literature as they pursue cross-domain feature alignment based on a learning objective that does not ensure cross-domain class discriminability. We support this claim by comparing our proposal with previous works in the experimental section.

In this paper instead, we take inspiration from a recent self-supervised approach, DINO [202], to learn more discriminative representations for the target domain by constraining a sample and a strongly augmented version of itself to be classified similarly. This is typically

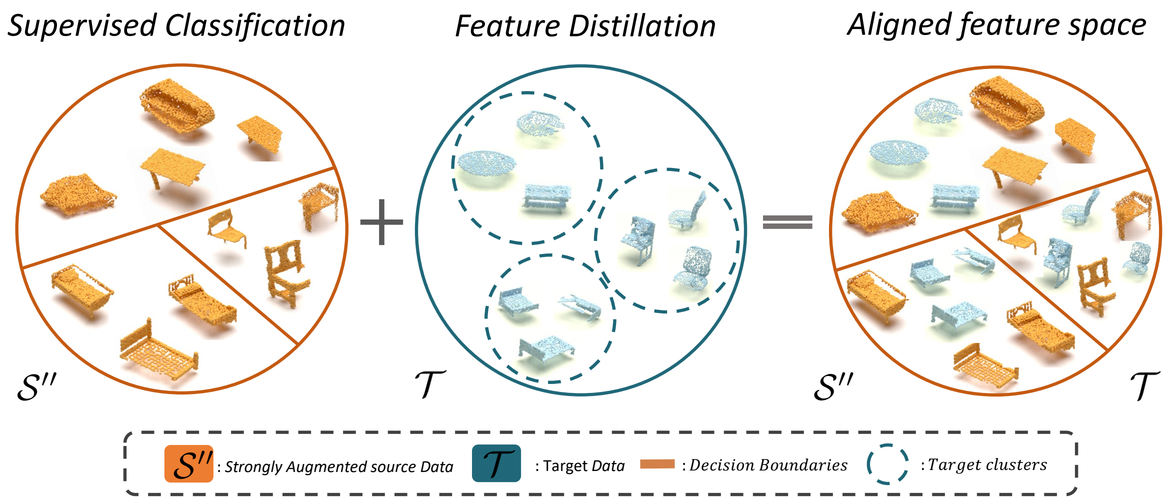


FIGURE 8.1. Proposed UDA method. We combine a supervised training of strongly augmented source data with a self-distillation approach that aims at clustering target shapes unsupervisedly. The combination of these two approaches leads to an alignment in feature space across domains.

achieved through self-distillation, a methodology where the output of a neural network is compared with the output obtained from a mean teacher *i.e.*, a temporal exponential moving average of the weights of the network itself (EMA) [191]. As shown in [202] this training methodology allows for clustering together samples of the same class. However, differently from DINO, we apply self-distillation for the first time in the 3D UDA context, where samples are point clouds, and the main goal is to reduce the gap between representations of two different domains rather than only learning a well-clustered feature space for a single domain. We believe that self-distillation is particularly suited for point cloud domain adaptation due to peculiar 3D data augmentations such as translation, occlusion and point-wise noise that can easily bridge the gap between source and target domain. By exploiting such augmentations to strongly augment source data and by enforcing inter-class discriminability for the target domain via self-distillation, we are able to obtain a shared aligned feature space across domains. The overall idea is illustrated in Fig. 8.1. Moreover, one major limitation of DINO that hinders its wide adoption is mode collapse [202], and previous works usually adopt multiple tricks and hyper-parameters such as clustering constraints [203], predictor [204] and contrastive losses [205] that are difficult to apply and tune in other contexts. In this work, we show how this paradigm can be applied without such tricks to UDA for point cloud classification, where mode collapse is prevented by simultaneously training a classifier on labeled source data that inherently separates the feature space according to semantic categories.

In the second step of our proposal, following recently published works in the field

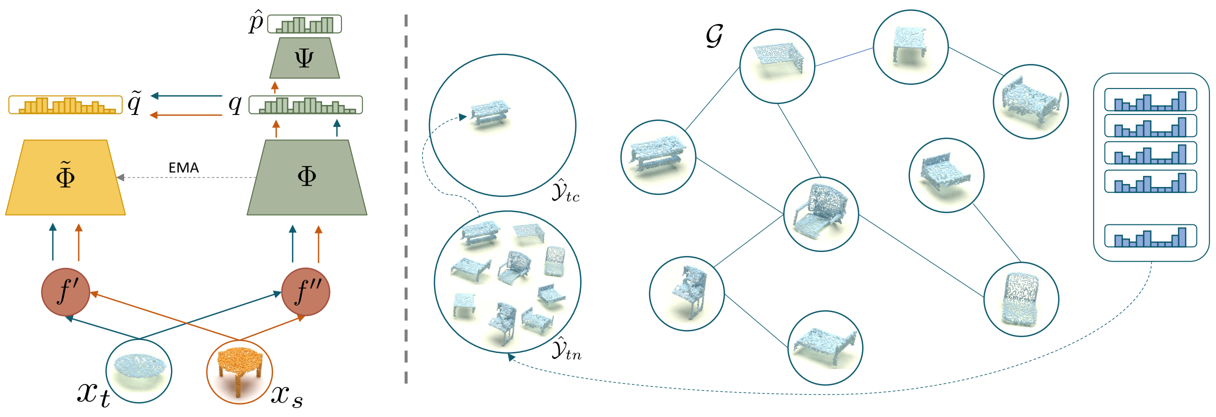


FIGURE 8.2. Illustration of our framework. **Left**: weakly and strongly augmented point clouds are generated with two transformation functions f' and f'' for both domains. The weakly augmented shapes are fed to an exponential moving average (EMA) encoder, the teacher $\tilde{\Phi}$, while the strongly augmented are processed by the student Φ. A consistency loss is applied between the corresponding embeddings. **Right**: the whole target dataset is processed by a GCN $\mathcal{G}$ online during self-training to iteratively refine and update pseudo-labels

[206, 201, 200, 207], we make use of *self-training*, an iterative methodology that exploits the predictions of a pre-trained model (*pseudo-labels*) to provide partial supervision on the target domain as well. However, pseudo-labels are noisy and their naive use typically leads to overfitting of the dominant classes of the source domain as shown in [208, 209]. The strategy proposed by [206] to refine them requires offline training of an additional network for this purpose and the definition of hand-crafted rules based on k-NN queries, limiting its general applicability, while [201] adopted a standard procedure borrowed from the 2D world [210]. As a further contribution of our work, we take a different path and propose to use Graph Neural Networks (GNNs) [211] to refine pseudo-labels online during self-training. Our main intuition is that by using a GNN, pseudo-labels are obtained by considering relationships between all target samples in the dataset rather than on single samples in isolation. This allows for reasoning at the dataset-level and enables to correct misclassified samples and thus refine pseudo-labels. Moreover, the target feature space is clustered thanks to the self-distillation, thus each node of the graph is likely to be connected to samples of the same category. Hence, the GNN can improve the pseudo-labels by reasoning on a neighborhood of samples sharing the same class. This procedure can be done online during training with the graph structure evolving over time, thus avoiding pseudo-labels overfitting.

8.1 Method

Our framework is divided into two main steps: self-distillation (Sec. 8.1.2) and *self-training with pseudo-labels refinement* (Sec. 8.1.3 and Sec. 8.1.4). The overall pipeline is depicted in

Fig. 8.2. We start introducing the notation and a brief review of the basic concepts about GNNs.

8.1.1 Preliminaries

Notation. As in the previous case, we consider the problem of UDA for point cloud classification, *i.e.*, given a point cloud with N elements $x \in \mathbb{R}^{N\times 3}$ we aim at learning a neural network $\Omega : x \rightarrow [0,1]^K$ that takes an input example x and produces a K-dimensional vector representing the confidence scores for K classes. Such a point cloud classifier consists of two components: $\Omega = \Phi \circ \Psi$. The first is a feature extractor network, $\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^D$, producing $g \in \mathbb{R}^D$, *i.e.*, a D-dimensional global feature descriptor for the shape, the second is small MLP $\Psi : \mathbb{R}^D \rightarrow \mathbb{R}^K$ followed by a softmax operator which maps g to a vector of confidence scores $\hat{p} \in [0,1]^K$. Finally, the class predictions can be obtained by the argmax operator $\Lambda : \mathbb{R}^K \rightarrow \mathcal{Y}$. As it is peculiar in UDA settings, we have at our disposal a source domain with labels $\mathcal{S} = \{(x_s^i \in \mathcal{X}_s, y_s^i \in \mathcal{Y}_s)\}_{i=1}^{n_s}$, and a target domain $\mathcal{T} = \{x_t^j \in \mathcal{X}_t\}_{j=1}^{n_t}$, where the point clouds are unlabeled. Our objective is to obtain a classifier able to make correct predictions on $\mathcal{T}$.

Background on GNNs. Graph Neural Networks (GNNs) are models designed to process graphs, *i.e.*, sets of nodes that are optionally joined one to another by edges representing relationships. GNNs are a powerful tool to process unstructured data thanks to their ability to update the representations of each node by aggregation of information from the neighbouring nodes. An undirected graph $\mathcal{G}$ is represented as a tuple $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of N vertices $v_i \in \mathcal{V}$, and $\mathcal{E}$ is the set of edges. The graph topology is determined by the adjacency matrix $A \in \mathbb{R}^{N\times N}$, with $A_{i,j} = 1$ if two nodes i and j are connected. Among the many architectures of GNNs [211], in this work, we adopt the Graph Convolutional Networks (GCNs) [212], which propagate the information between each layer according to the following propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{8.1}$$

where $\tilde{A} = A + I$ represents the adjacency matrix with self-connections, I is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ acts as a scaling factor and $W^{(l)}$ is a layer-specific trainable weight matrix. The aggregation rule is followed by a non-linear activation function $\sigma(\cdot)$ such as ReLU. Note that matrix $H^{(l)}$ deals with the l-th layer of the network, each row i representing the feature vector of a node $v_i \in \mathcal{V}$ in that layer. We refer the reader to [211] for a more detailed discussion.

8.1.2 Self-distillation

In this section, we present the self-distillation module that we use in both steps of our pipeline. The purpose of this component is to distill good features for the target domain unsupervisedly, so that a discriminative feature space directly useful for classification can be learned even though no direct supervision is available in $\mathcal{T}$. Our main intuition is that learning a clustered feature space that minimizes the distance among variations of the same cloud from the target domain, while simultaneously learning decision boundaries amenable to classification thanks to a carefully augmented source domain, is key to obtaining good pseudo-labels to be deployed in the self-training process. Indeed, without enforcing compactness in the feature space, it is more likely that, due to the domain gap, target samples are spread across the different categories defined by the decision boundaries of the classifier. This is undesirable since it would lead to excessive noise in pseudo-labels.

To achieve our goal, we use two data augmentation functions $f', f'' : \mathbb{R}^{N\times 3} \rightarrow \mathbb{R}^{N\times 3}$ that take as input a point cloud x and return a weakly augmented (x') and a strongly augmented point cloud (x'') respectively. Then, we adopt a self-distillation paradigm, where we train a student encoder Φ to match the output of a teacher encoder $\tilde{\Phi}$. In particular, we match two global shape descriptors, $\tilde{g} = \tilde{\Phi}(x')$ and $g = \Phi(x'')$, computed by feeding a weakly augmented point cloud x' to the teacher and the strongly augmented version x'' to the student.

By taking inspiration from [202], we design the student and the teacher to output probability distributions over D dimensions, denoted by q and $\tilde{q}$, respectively. These probabilities can be obtained by normalizing the output of the two encoders, *i.e.*, g and $\tilde{g}$, with a softmax function:

$$\begin{aligned} q(g,\tau) &= \frac{\exp(g/\tau)}{\sum_{d=1}^{D}\exp\left(g^{(d)}/\tau\right)}, \\ \tilde{q}(\tilde{g},\tilde{\tau}) &= \frac{\exp(\tilde{g}/\tilde{\tau})}{\sum_{d=1}^{D}\exp\left(\tilde{g}^{(d)}/\tilde{\tau}\right)} \end{aligned} \tag{8.2}$$

where $\tau > 0$ and $\tilde{\tau} > 0$ are the two temperature parameters which control the sharpness of the output distributions for the student and the teacher, respectively. Differently from [202], we don't require any complex scheduling for the temperature parameters, and we just empirically set them to 0.5 by observing the model performance on the source domain for the ModelNet→ScanNet experiment and set it to the same value for all the others. To force the embedding of the augmented point cloud to match that computed for the original one, we minimize the cross-entropy:

$$\mathcal{L}_{sd}(\tilde{g}, g) = -\tilde{q}(\tilde{g},\tilde{\tau}) \log q(g,\tau) \tag{8.3}$$

by running backprop on the student network Φ, while the weights of the teacher are updated by computing an exponential moving average of those of the student. Please note that both networks share the same architecture but have different weights. We employ an EMA as a teacher network since it is a convenient way to provide robust and stable features throughout the training process without the need of training another network [191, 213].

Data augmentation and transformation functions. To implement f' and f'', we use a set of common data augmentation techniques for point clouds such as: jittering, elastic deformation [214], scaling along the three axes. More specifically, to obtain the weakly augmented point cloud x', we only use jittering, while for the strongly augmented x'', we employ all the above transformations. Additionally, when performing synthetic-to-real adaptation, we also include random point removal [195].

Interestingly, the same 3D transformations can be used to simulate the target distribution given source data. In fact, although it is not possible to exactly predict the shift between two domains, one can approximate the nuisances that affect the target data through aggressive data augmentation. For example, when performing UDA between different synthetic domains, shapes may have similar geometric elements but with different styles [215, 216], which can be mimicked by object distortions or elongation and scaling. Similarly, when moving from a synthetic domain to a real one, it is reasonable to assume that shapes within the same class will appear similar to CAD models but will have missing components due to occlusions, and point coordinates will be affected by the noise-originated in the acquisition process. Therefore, as shown in Fig. 8.2 (left), at training time we augment the source data by re-utilizing the transformation function f'', with the goal of minimising the gap between the two domains in the input space and seamlessly obtain a better alignment also in the feature space. Applying such well-designed augmentations to source data combined with our distillation technique is beneficial to the student model. Intuitively, by distillation we aim at clustering target samples, while by data augmentation we force source clusters, naturally obtained with a classification loss, to be aligned with the target ones.

8.1.3 Pseudo-labels initialization

In the first step of our method, we exploit the self-distillation module presented in the previous section to obtain an initial set of pseudo-labels for the target domain. In particular, as shown in Fig. 8.2 (left), we train a classifier $\Omega = \Phi \circ \Psi$ on top of the student feature extractor and fed with augmented source data. We use the cross entropy loss:

$$\mathcal{L}_{ce}(x''_s, y_s) = -y_s \log \Omega(x''_s) \tag{8.4}$$

Simultaneously, we feed to the encoder Φ batches of source and target point clouds strongly augmented with the transformation function f'', while $\tilde{\Phi}$ receives the weakly augmented versions, and minimize Eq. (8.3) to learn the desired clustered feature space for the data of the target domain. After training, the initial set of pseudo-labels is computed by feeding each target sample x_t^i into Ω and selecting the class with the highest confidence score: $\hat{y}_t^i = \Lambda(\Omega(x_t^i))$.

8.1.4 Self-training and pseudo-labels refinement

In the second step, we exploit and refine the previously obtained pseudo-labels. We do this by alternating self-training and refinement in an iterative procedure.

Self-training. In this step we train our classifier $\Omega = \Phi \circ \Psi$ leveraging pseudo-labels, starting from scratch if it is the first iteration. To do so, we first split the pairs of target samples and associated pseudo-labels $(x_t^i, \hat{y}_t^i)$ into two disjoint sets, *i.e.*, $\hat{\mathcal{Y}}_{tc}$ and $\hat{\mathcal{Y}}_{tn}$, associated with confident and non-confident pseudo-labels, respectively, and with the former initialized to the empty set. The sets will be useful to realize the iterative procedure outlined at the end of the section. We then train Φ and Ψ using self-distillation and supervision for both domains, with supervision for the target coming from pseudo-labels:

$$\begin{aligned} \mathcal{L} &= -\mathcal{L}_{ce}(x_s'', y_s) - \lambda\mathcal{L}_{ce}(x_t', \hat{y}_t) - \mathcal{L}_{sd}(x'', x') \\ \text{where} \quad \lambda &= \begin{cases} 1, & \hat{y}_t \in \hat{\mathcal{Y}}_{tc} \\ 0.2, & \hat{y}_t \in \hat{\mathcal{Y}}_{tn} \end{cases} . \end{aligned} \tag{8.5}$$

Note that, as in the previous step, $\mathcal{L}_{sd}$ acts on both domains.

Refinement. Naively using pseudo-labels as done in the previous step typically leads to ignoring the classes that are underrepresented in the source domain and to obtain sub-optimal performance on the target domain due to noise in the pseudo-labels [217, 218]. Hence, we run self-training only for a few epochs and then refine the pseudo-labels exploiting a GCN. Our intuition is that, by leveraging on a global view of the target dataset, the GCN can better disambiguate hard cases compared to the initial pseudo-labels provided by the classifier, which, on the other hand, takes its decision on each input sample in isolation. For instance, even if few samples of a rare class are tightly connected (i.e. node with high degrees), it is likely for their confidence to be high as in their neighbourhood only nodes with the same class are present. The role of the GCN is therefore twofold: it corrects pseudo-labels; it decides which pseudo-labels should be considered confident and thereby moved from $\hat{\mathcal{Y}}_{tn}$ into $\hat{\mathcal{Y}}_{tc}$. We obtain the graph $\mathcal{G}$ by considering all samples in the target domain, as shown in Fig. 8.2 (right), and we build the adjacency matrix A based on the cosine similarity between the global shape embeddings g:

$$A_{i,j} = \begin{cases} 1, & \frac{\langle g_i, g_j \rangle}{\|g_i\| \|g_j\|} > \epsilon \\ 0, & otherwise \end{cases} \tag{8.6}$$

with ϵ being a similarity threshold empirically set to 0.95 so that the node degree (the average number of neighbours for each node of the graph) is roughly 10. This is necessary for memory constraints, as the required memory to train a GCN is highly affected by this hyper-parameter. Inspired by [219], we equip each node in $\mathcal{G}$ with the embedding g as well as with the prediction provided by the classifier Ω, *i.e.*, the vector $\hat{p}$. These two pieces of information provide the GCN with cues concerning both the geometric structure as well as the semantic class of the object. For example, it may be the case that two point clouds have similar embeddings and yet belong to different classes. This occurs frequently when considering a real domain, where an occluded chair with a missing back could easily be misclassified as a table or the back itself with missing legs can be confused for a monitor. Hence, providing the GCN with the additional information on the probability distribution among the K classes can help it attain more accurate pseudo-labels for target samples featuring similar embeddings. Then, we compute the input to the GCN as

$$H^{(0)} = \Phi(X_t) + \Omega(X_t)W_D \tag{8.7}$$

where X_t is the set of all target samples and $W_D \in R^{K \times D}$ is a learnable projection matrix that projects the output distribution over K classes in a D-dimensional space.

Afterward, following Eq. (8.1), we stack three graph convolutional layers where the last acts as a node classifier that returns a matrix of size $n_t \times K$. The GCN is optimized with a classical cross-entropy loss computed over all target samples, $\hat{\mathcal{Y}}_{tn} \cup \hat{\mathcal{Y}}_{tc}$, without taking into account the confidence on their pseudo-labels. It is worth noticing that, the predictions $\Omega(X_t)$, *i.e.*, part of the input to the GCN, do not necessarily match the $\hat{\mathcal{Y}}_{tn} \cup \hat{\mathcal{Y}}_{tc}$ pseudo-labels. However, the GCN can just learn to output the same probability vector $\Omega(X_t)$, discarding part of the input features [219], and consequently failing in generalizing at test time due to label leakage. Hence, we randomly mask (*i.e.*, set to zero) 20% of the inputs $\Omega(X_t)$ at training time.

Finally, after training, we exploit the GCN to extract confident samples, *i.e.*, the top θ predictions for each class, update the corresponding pseudo-labels with the output of the GCN, and move them from $\hat{\mathcal{Y}}_{tn}$ into $\hat{\mathcal{Y}}_{tc}$.

Iterative training. We argue that the topology of the graph highly influences the output of the GCN. As the encoder improves its embeddings with multiple rounds of self-training, also thanks to the self-distillation process, pseudo-labels become better and better since the graph structure improves. Hence, we plug the previous steps into an iterative learning process, where we repeat:

a) self-train with Eq. (8.5) Φ and Ψ for e epochs using self-distillation and supervision for both domains, with supervision for the target coming from pseudo-labels;

b) build $\mathcal{G}$ and train the GCN to refine pseudo-labels;

Method	ModelNet to ShapeNet	ModelNet to ScanNet	ShapeNet to ModelNet	ShapeNet to ScanNet	ScanNet to ModelNet	ScanNet to ShapeNet	Avg
No Adaptation	80.5	41.6	75.8	40.0	60.5	63.6	60.3
PointDAN	80.2	45.3	71.2	46.9	59.8	66.2	61.6
DefRec+PCM	81.1	50.3	54.3	52.8	54.0	69.0	60.3
3D Puzzle	81.6	49.7	73.6	41.9	65.9	68.1	63.5
RefRec	81.4	56.5	**85.4**	53.3	73.0	73.1	70.5
(Ours)	**83.4**	**61.6**	77.3	**57.7**	**78.6**	**79.8**	**73.1**
Oracle	93.2	66.2	95	66.2	95.0	93.2	

TABLE 8.1. Shape classification accuracy (%) on the PointDA-10 dataset with PointNet. For each method, we report the average results on three runs. Best result on each column is in bold.

c) update current pseudo-labels, moving the top θ predictions of the GCN for each class from $\hat{\mathcal{Y}}_{tn}$ to $\hat{\mathcal{Y}}_{tc}$.

To gradually increase the size of $\hat{\mathcal{Y}}_{tc}$, θ starts from 0 and grows to 1 to include more and more samples during training. At test time, the GCN as well as the teacher encoder $\tilde{\Phi}$ can be simply discarded, with Ω being the only network required to perform inference. Although the GCN can potentially be used at test time to obtain better performance, we discard it as this would introduce additional requirements such as keeping the whole training set in memory, and computing the neighborhood of each test sample.

8.2 Experiments

To show the effectiveness of our method, we compare it against state-of-the-art methods for UDA for point cloud classification such as [206, 200, 207], using two different backbones for our feature extractors: PointNet [160] and DGCNN [167]. Furthermore, we compare with a baseline *i.e.*, a simple model trained only on the source domain without any adaptation, and an oracle model, which instead assumes to have all target data available. The former constitutes the lower-bound in terms of performance, while the latter is considered as the upper-bound since all target data can be utilized. Finally, we also conducted an experiment on the challenging task of part segmentation to show how our method can be extended to different tasks than point cloud classification. In this case, we adopt the setting introduced in [29], which is the only method performing adaptation on such a task for the synthetic-to-real scenario.

8.2.1 Results

Classification. We report in Tab. 8.1 and Tab. 8.2 our results with PointNet and DGCNN, respectively. For PointNet, we establish overall the new state-of-the-art with 73.1% in terms

Method	**ModelNet to ShapeNet**	**ModelNet to ScanNet**	**ShapeNet to ModelNet**	**ShapeNet to ScanNet**	**ScanNet to ModelNet**	**ScanNet to ShapeNet**	**Avg**
No Adaptation	83.3	43.8	75.5	42.5	63.8	64.2	62.2
PointDAN	83.9	44.8	63.3	45.7	43.6	56.4	56.3
DefRec+PCM	81.7	51.8	78.6	54.5	73.7	71.1	68.6
GAST †	**84.8**	59.8	**80.8**	56.7	81.1	74.9	73.0
GLRV	85.4	60.4	78.8	57.7	77.8	76.2	72.7
ImplicitPCDA	86.2	58.6	81.4	56.9	81.5	74.4	73.2
(Ours)	83.9	**61.1**	80.3	**58.9**	**85.5**	**80.9**	**75.1**
Oracle	93.9	78.4	96.2	78.4	96.2	93.9	80.5

TABLE 8.2. Shape classification accuracy (%) on the PointDA-10 dataset with DGCNN. For each method, we report the average results on three runs. Best result on each column is in bold. † Denotes a more powerful variant of DGCNN and results are obtained by performing checkpoint selection on the test set.

of accuracy. We also note that our framework achieves the best results in 5 out of 6 settings, with a big gap in ModelNet→ScanNet and ShapeNet→ScanNet (+5.1% and +4.4%) that are the most challenging scenarios as they involve synthetic-to-real UDA. In particular, we highlight the result obtained in ModelNet→ScanNet (61.6%), which is, roughly, only 5% less than the oracle. We also observe remarkable improvements when addressing the opposite case *i.e.*, real-to-synthetic (last two columns). This demonstrates the capability of our framework to deal with large domain shifts. As regards as synthetic-to-synthetic UDA, we observe good performance in ModelNet→ShapeNet, while we are the second-best model in ShapeNet→ModelNet. We attribute the gap with RefRec to the peculiarity of ShapeNet→ModelNet, where the source domain is a complex dataset while the target is a simple one with shapes clearly distinguishable among classes *i.e.*, objects with similar shapes do belong to the same class. In such specific scenarios, reconstruction-based approaches such as RefRec shine since the auxiliary task of reconstructing a point cloud naturally tends to form well-shaped clusters in feature space that are amenable for classification.

Furthermore, we repeat the same experiments using DGCNN as our main backbone. We achieve again state-of-the-art results (75.1%), showing the generality of our approach towards other architectures. Overall, we observe a similar trend w.r.t. Tab. 8.1, with an increase in performance in almost all configurations w.r.t. previous works.

Part Segmentation. Although our main goal is to propose a method that aims at solving UDA for point cloud classification, our method can be easily extended to more challenging tasks such as part segmentation, which consists of assigning to each vertex of the shape one object category. As done for point cloud classification, we perform a first step of self-distillation to distill good features for the target domain unsupervisedly. We then simply adapt the self-training step by considering each vertex of the input shape as a node in the

Part Segmentation: ShapeNetPart → ScanOBJ_BG					
Method	Seat	Back	Base	Arm	Avg.
Source only	67.85	45.60	84.89	14.87	53.30
3D Puzzle [29]	65.70	49.11	85.91	21.40	55.53
Self-dist (ours)	71.1	79.3	65.2	37.0	63.2
(ours)	**74.7**	**82.7**	**67.9**	**37.7**	**65.7**

TABLE 8.3. Per part and average mIoU (%) of chair segmentation for ShapeNetPart to ScanOBJ-BG.

Step	ce	sd	kd	**ModelNet to ShapeNet**	**ModelNet to ScanNet**	**ShapeNet to ModelNet**	**ShapeNet to ScanNet**	**ScanNet to ModelNet**	**ScanNet to ShapeNet**	**Avg**
	✓			80.5	41.6	75.8	40.0	60.5	63.6	60.3
PL init	✓	✓		**82.1**	**57.2**	77.6	**55.0**	**71.0**	**72.1**	**69.2**
	✓		✓	79.6	54.0	**79.2**	53.2	53.9	70.0	65.0

TABLE 8.4. Ablation study for the first step of our framework. ce: cross-entropy loss on source domain sd: self-distillation loss Eq. (8.3) in feature space used to train the pseudo-labels model; kd: standard knowledge distillation loss [220] in output space. We report the average results on three runs.

graph. In this case, the node representation consists of a local feature vector extracted from the main backbone, which is a PointNet as in [29]. The whole graph is theoretically composed of all points of all shapes in the dataset. However, keeping all vertices in memory would be impractical and we perform the procedure explained in Sec. 8.1.2 by considering 20000 points of the whole dataset for each refinement iteration. Results are reported in Tab. 8.3. The evaluation metric is the mean Intersection-over-Union (mIoU), which is computed for each part Q for all the samples of the *chair* class. Then, the average across parts is reported. First, we observe that our full framework (last row) surpasses by more than 10% the previous method (second row). Furthermore, we highlight the effectiveness of self-distillation for the part segmentation task. Indeed, when only performing the first step of our pipeline (third row of Tab. 8.3), we already overcome [29] by 7.7%.

Self-distillation vs knowledge distillation. In Tab. 8.4, we ablate our self-distillation strategy and also compare it to an obvious alternative, *i.e.*, applying Eq. (8.3) in output space. In this case, the self-distillation loss in Eq. (8.3) is applied on the output of the classifier rather than the feature vector of the backbone. This protocol is similar to the knowledge distillation paradigm [220] that uses soft pseudo-labels. While we observe in both cases an improvement over the baseline trained only on source data (first row), the improvement is twice as large when self-distillation is deployed, which demonstrates the importance of working in feature space. Moreover, the large improvement in absolute terms (+8.9% on average) attained by using self-distillation shows its effectiveness in reducing the domain gap, validating our intuition to use it to tackle UDA. Interestingly, we observe a different behaviour for ShapeNet→ModelNet. This is again likely due to the peculiarity of the setting. With the source domain being much larger and richer than the target one, it is plausible that pseudo-labels in output space are quite accurate, and therefore more effective in this case.

Step	st	ref	sd	ModelNet to ShapeNet	ModelNet to ScanNet	ShapeNet to ModelNet	ShapeNet to ScanNet	ScanNet to ModelNet	ScanNet to ShapeNet	Avg
Adaptation	✓			82.7	59.3	74.9	56.4	77.1	77.8	71.4
	✓	✓		**83.4**	60.9	**78.2**	56.3	77.9	79.4	72.7
	✓	✓	✓	**83.4**	**61.6**	77.3	**57.7**	**78.6**	**79.8**	**73.1**

TABLE 8.5. Ablation for the second step of our algorithm. st: self-training with pseudo-labels of the last row of Tab. 8.4 model; sd: self-distillation loss in the adaptation step; ref: refinement of pseudo-labels with GCN. We average results on three runs.

The model trained with self-distillation is used to extract the initial set of pseudo-labels for our method, as well as for all the self-training variants compared in Tab. 8.5. Finally, we highlight how the results obtained with self-distillation are clearly superior in all scenarios on average to those attained by competitors based on self-supervised learning tasks, *e.g.*, row 2 (DefRec) and 3 (3D puzzle) of Tab. 8.1, that are based on a reconstruction and a 3D puzzle pretext task, respectively. This provides empirical support for our claim on the higher effectiveness of self-distillation with respect to auxiliary tasks for 3D UDA.

Self-training strategies. In Tab. 8.5, we perform an ablation study on the second step of our pipeline. We start by applying the simplest strategy to perform self-training (first row), *i.e.*, using all pseudo-labels of the target domain together with the labels from the source domain to train a single classifier. This provides competitive results (71.4%), which is already better than the previous state-of-the-art model (70.5%), again showcasing the effectiveness of self-distillation to obtain pseudo-labels for UDA. When activating also the proposed online refinement that iteratively improves pseudo-labels thanks to the global reasoning of the GCN (second row), we appreciate another large improvement compared to the naive self-training, which validates the importance of the proposed iterative refinement. Finally, in the last row, we report the results attained by activating self-distillation also in the adaptation step, which leads to the best performance and is the model used in all other experiments. As a further validation of the importance of the design decisions in our framework, we plot the training curves of the synthetic-to-real ModelNet→ScanNet in Fig. 8.3. The curves represent the test accuracy on the target domain during training. The red plot shows the behaviour of the naive self-training, which corresponds to row 1 of Tab. 8.5. On the other hand, the blue lines represent the training curves obtained with our full model, *i.e.*, last row of Tab. 8.5. We can appreciate that, after a certain number of steps, the blue line is always above the red line. This is clear evidence that in our full model, pseudo-labels are improved over time, while in the naive case, the model starts to overfit, leading to a plateau. We also wish to point out that such behaviour is key to a good UDA method because, in the absence of target labels to perform validation, it is basically impossible to decide when to stop the training process.

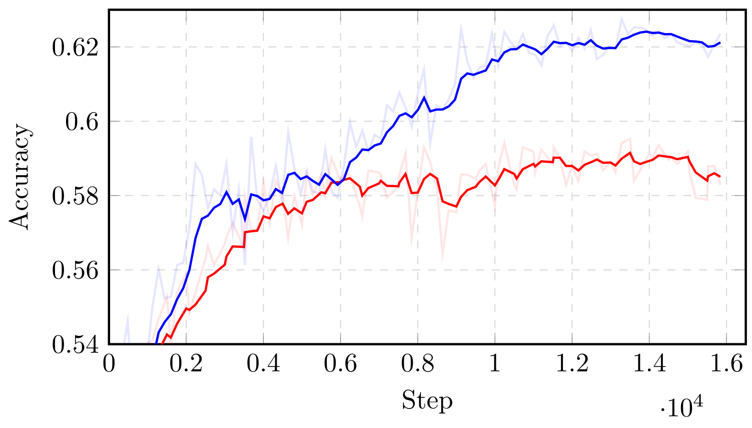


FIGURE 8.3. Test accuracy on target domain during training on ModelNet→ScanNet. Our model (Blue) consistently improves pseudo-labels during training differently from a simple self-training strategy in which pseudo-labels are fixed (Red).

8.3 Conclusions

In this chapter, we explored a novel strategy to learn features on the target domain without the need of annotations. We first proposed to guide the network to learn a clustered feature space for the target domain and preserve discriminability suitable for classification. In addition, we introduced a novel refinement strategy that is able to globally reason on the target domain by means of GNN and to correct misclassified samples during training. Combining the two contributions, allowed to establish the state-of-the-art in the reference benchmarks. Finally, we showed how these contributions can be used for more challenging tasks such as part segmentation.

Part III

Domain Adaptation for multi-modal Data

Chapter 9

Initial Remarks

LiDAR semantic segmentation is the task of assigning a class label to each point of a 3D scan gathered by Light Detection and Ranging (LiDAR) sensors. These devices can record accurate depth information regardless of the lighting conditions, making them a reliable source of information for autonomous driving. However, LiDAR data is colorless, unstructured, and sparse. Consequently, scene understanding using only LiDARs is extremely challenging. Yet, nowadays, autonomous vehicles are commonly equipped also with other sensors, such as RGB cameras. For this reason, the research community has recently developed multi-modal approaches [221] exploiting both these modalities. However, akin to other tasks, multi-modal LiDAR segmentation networks suffer from the domain shift problem, i.e. models struggle to generalize to environments different from the training one. A straightforward solution would be to gather more and more annotated data in many scenarios. However, this process is cumbersome and time-consuming. As an example, annotating a point cloud acquired in an urban environment of $100m^3$ needs from 1.5 to 4.5 hours by human annotators with 3D expertise [222]. Once again, we can rely on UDA techniques to address this problem. However, only a few proposals deal directly with LiDAR semantic segmentation task [223], and even fewer try to exploit multiple modalities such as RGB images and LiDAR point clouds [31, 33] in the UDA scenario. In the latter setup, referred to as multi-modal UDA for LiDAR segmentation, one can leverage both modalities as sources of information. The standard approach processes them by means of two networks, one processing the 2D images and the other the 3D point clouds. To this end, XMUDA [31] proposed a benchmark and a two-branches architecture that uses a cross-modal loss on each domain independently forcing predictions extracted from 3D points and the corresponding 2D pixels to be similar in the same domain. Despite the effectiveness of this approach, we argue that it mainly focuses on the alignment across modalities rather than the actual alignment across domains. DsCML [33] performs a step forward in domain alignment, employing adversarial training to align multi-modal features also between domains. However, adversarial learning is notoriously unstable in segmentation tasks, especially when applied to deep features, leading to variable performance. In Chapter 10, we follow closely the research line of this dissertation and explore how auxiliary tasks such as depth completion, can be effectively utilized for multi-modal UDA for LiDAR segmentation. In Chapter 11 instead, we look at the same problem from a different perspective. More specifically, we propose to exploit the multi-modal nature of the input data to make the two branches proposed in [31] more robust to the domain shift.

9.1 Related Works

Point Cloud Semantic Segmentation. 3D data can be represented in several ways such as point clouds, voxels, and meshes, each with its pros and cons. Similarly to pixels in 2D, voxels represent 3D data as a discrete grid of the 3D space. This representation allows using convolutions as done for images. However, performing a convolution over the whole 3D space is memory-intensive, and it does not consider that many voxels are usually empty. Some 3D CNNs [224, 225] rely on OctTree [226] to reduce the memory footprint but without addressing the problem of manifold dilation. SparseConvNet [227] and similar implementations [228] address this problem by using hash tables to convolve only on active voxels, allowing the processing of high-resolution point clouds with only one point per voxel. Aside from cubic discretization, some approaches [229, 230] employ cylindrical voxels. Other methods address the problem with sparse point-voxel convolutions [231]. Differently, point-based networks process directly each point of a point cloud. PointNet++ [161] extract features from each point, and then extract global and local features by means of max-pooling in a hierarchical way. Many improvements have been proposed in this direction, such as continuous convolutions [168] or lightweight alternatives [232]. In this work, we select SparseConvNet [227] as our 3D network as done by other works in the field [31, 33, 221, 233] since it is suitable for 3D semantic segmentation of large scenes.

Multi-Modal Learning for 3D UDA. While the majority of works consider UDA for semantic segmentation of images, fewer approaches have been proposed for the 3D counterpart, with only a limited number of works addressing the problem of UDA for LiDAR Segmentation [234, 235]. Very recently, some works have addressed the challenging multi-modal segmentation task from LiDARs and RGB sensors [31, 33]. XMUDA [31] is the first work that focuses on UDA in the above setting, defining a benchmark and presenting a baseline approach that employs a loss to align features across modalities. DsCML [33] is the first to explicitly address alignment across domains in this setup employing an adversarial loss, which however may lead to extremely variable performances.

9.2 Multi-Modal Datasets for 3D Semantic Segmentation

We evaluate our proposal using two different versions of the same dataset. More precisely, in Chapter 10, we follow the setting established in xMUDA [31] and test our method on three different scenarios: day-to-night, country-to-country, and dataset-to-dataset. The first two settings leverage the NuScenes [236] dataset which consists of 1000 driving scenes in total, each of 20 seconds, with 40k annotated point-wise frames taken at 2Hz. The former exhibits severe light changes between the source and the target domain, while the latter covers changes in the scene layout. For the day-to-night, the RGB images exhibit a severe gap due to the different lighting conditions, while the LiDAR shows small differences being the same sensor. For the country-to-country scenario, the sensor setup is the same, but objects may have different appearances as two different cities are involved. In both settings, adaptation is performed on five categories: *vehicle, pedestrian, bike, traffic boundary, background*.

The third and most difficult scenario is the dataset-to-dataset case, which is realized by adapting from A2D2[237] to SemanticKITTI [222] and comprises both a large change in the sensors setup and in appearance. The A2D2 dataset is composed of 20 drives, with a total of 28,637 frames. As the LiDARs sensor is very sparse (16 layers), all three front LiDARs are used. All frames of all sequences are used for training, except for the sequence 20180807_145028 which is left out for testing. The SemanticKITTI dataset features a large-angle front camera and a 64-layer LiDAR. Scenes from 0, 1, 2, 3, 4, 5, 6, 9, 10 are used for training, scene 7 as validation, and 8 as a test set. For this adaptation setup, only the ten classes that are in common along the two datasets are used: *car, truck, bike, person, road, parking, sidewalk, building, nature, other-objects*. In Chapter 11 instead, we follow the benchmark introduced in [32] which introduced a refined version of the NuScenes dataset mentioned above and it includes one more additional interesting scenario. More precisely, for the day-to-night and country-to-country scenarios, each framework is now evaluated under 6 different classes *vehicle, driveable_surface, sidewalk, terrain, manmade, vegetation*. Finally, in addition to the same country-to-country scenario detailed above, the authors of [32] also introduce a fourth challenging scenario that foresees adaptation from synthetic to real data, and it is implemented by adapting from VirtualKITTI [238] to SemanticKITTI. VirtualKITTI consists of 5 driving scenes obtained with Unity and simulating the SemanticKITTI dataset. Following [32], we also test on 6 shared classes between the 2 datasets which are: *vegetation terrain, building, road, object, truck, car*. It is important to note in all cases, LiDAR point clouds and camera are synchronized and calibrated so that the projection between a 3D point and its corresponding 2D image pixel can always be computed. Furthermore, only 3D points visible from the camera are used for both training and testing.

Chapter 10

Enanching Multi-Modal 3D Semantic Segmentation with Depth Completion

10.1 Introduction

In this chapter, we explore how depth completion can be used as an auxiliary task to make the features of the 2D network more similar between domains and as a powerful data augmentation technique for the 3D network. To this end, we argue that to complete sparse depth inputs, a network needs to infer the geometrical structure of the scene, e.g., understand the shape of cars or that the road is flat. Unlike 2D appearance, which may be extremely different across domains due to environmental variables such as light and weather, or 3D scans, which may differ because of LiDAR patterns and densities, the 2D depth structure is similar, e.g., roads appear in the bottom part of the image and are flat independently of the domain, as it can be seen in the Completed Depth row of Fig. 10.1, left column. Following the above reasoning, a depth completion network trained jointly on the source and target data should extract features robust to the domain shift. Moreover, the geometrical structures are tightly linked to the semantics of the scene [239], thus training a network for depth completion should also push the features to be discriminative for the semantic segmentation task. We leverage these intuitions and we project the 3D points to the image plane to obtain a sparse depth map. Then, we train a multi-task 2D network to jointly segment the RGB source images and complete the sparse depths on both domains, forcing target features to be robust to the domain shift and discriminative for semantic segmentation. However, to the best of our knowledge, no depth completion model can be trained solely on the same LiDAR input without additional data such as ground truth dense depth maps or video sequences. Thus, we propose a simple yet effective self-supervised technique to train a depth completion network without external data. Finally, we propose to exploit the estimated dense depth maps as a powerful data augmentation technique in the source domain to boost the performance of the 3D network. To do so, we project each pixel back to the 3D space and assign the most confident 2D predictions as proxy labels to the corresponding 3D points. We add new annotated points to the source LiDAR point clouds by looking at class-specific confidence

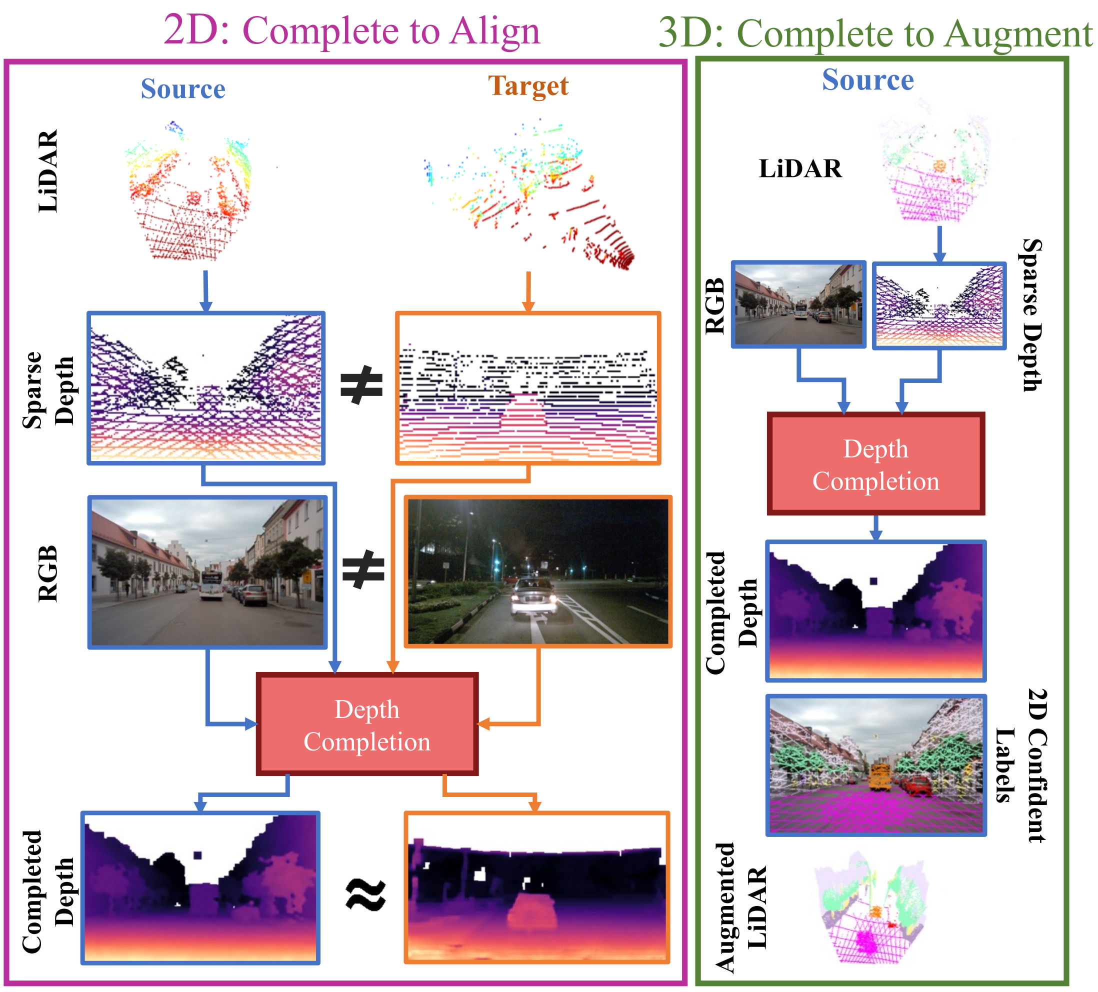


FIGURE 10.1. Our multi-modal UDA framework for LiDAR segmentation exploits self-supervised depth completion as an auxiliary task. As completed depths are similar between domains, training a network for 2D segmentation and depth completion pushes source and target features to be more robust to domain shift. Moreover, we can use these depths as a data augmentation for the labeled LiDAR point clouds.

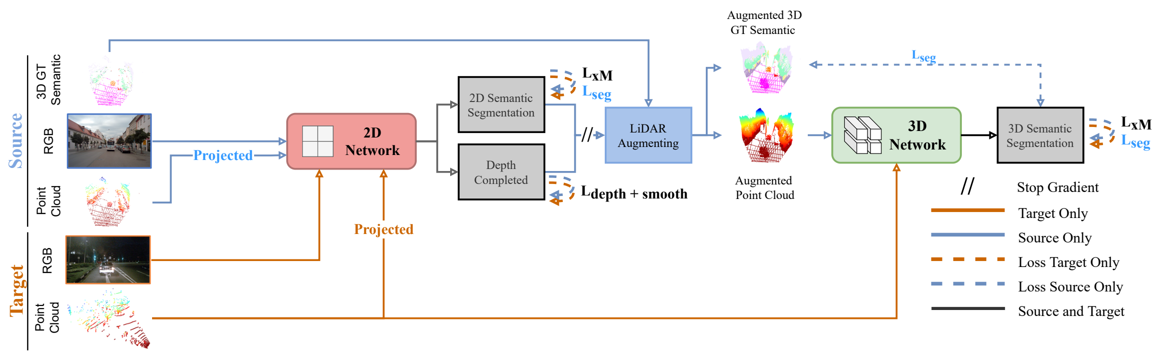


FIGURE 10.2. **Framework Overview.** First, the 2D network outputs a densified depth map and 2D semantic labels, then these data are used to augment the ground truth on the source domain to improve the performance of the 3D network.

percentiles, thereby obtaining much denser 3D clouds with annotations, as shown in the bottom-right part of Fig. 10.1.

10.2 Method

Given a LiDAR scan and the corresponding RGB images for both domains, our goal is to solve 3D semantic segmentation on the target domain. Supervision is provided only for sparse 3D points of the source domain. Our framework dubbed Complete to Segment (CtS), is depicted in Fig. 10.2.

10.2.1 Preliminaries

Given a 2D image, $\boldsymbol{x}^{2D}$, and a corresponding 3D point cloud, $\boldsymbol{x}^{3D}$, we define as $\boldsymbol{y}^{3D}$ the semantic label for each 3D point. Assuming LiDARs measurements to be expressed in the camera reference frame and the availability of the intrinsic camera matrix, we can project each 3D point into the image plane. A sparse depth map, $\boldsymbol{D}^{3D\rightarrow 2D}$, can be easily obtained by assigning the value of z to each corresponding pixel (u, v). Then, we can assign the 3D label to the corresponding 2D pixels obtaining a sparse 2D semantic map, $\boldsymbol{y}^{3D\rightarrow 2D}$. We denote as $\mathcal{S}$ the *source* domain, for which annotations are available, and as $\mathcal{T}$ the *target* domain, where no annotations are accessible. Thus, we specify by subscripts s and t whether the data belong to $\mathcal{S}$ or $\mathcal{T}$ respectively. Our full dataset is composed of: i) images, $\boldsymbol{x}_s^{2D}$ and $\boldsymbol{x}_t^{2D}$; ii) point clouds, $\boldsymbol{x}_s^{3D}$ and $\boldsymbol{x}_t^{3D}$; iii) semantic labels for points clouds, $\boldsymbol{y}_s^{3D}$.

Two-streams architecture. Following the standard approach in this setup [31, 33], we deploy a two-streams architecture that processes 2D and 3D data independently. As argued in [31], having two networks is important to obtain modality-specific predictions which can be fused together effectively. Indeed, the final predictions are obtained by averaging the predictions of the 2D and 3D branches. The proposed multi-task 2D network is described in Sec. 10.2.2. Regarding the 3D network, similarly to [31, 33], we use SparseConvNet [227], with voxel size 5 cm to ensure that at most one point is inside each voxel.

Supervised Learning. We supervise both 2D and 3D networks using the cross-entropy loss on the source domain:

$$\mathcal{L}_{\text{seg}}(\boldsymbol{x}_s, \boldsymbol{y}_s) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} \boldsymbol{y}_s^{(n,c)} \log \boldsymbol{P}_{\boldsymbol{x}_s}^{(n,c)} \tag{10.1}$$

with $(\boldsymbol{x}_s, \boldsymbol{y}_s)$ being either $(\boldsymbol{x}_s^{2D}, \boldsymbol{y}_s^{3D\to 2D})$ or $(\boldsymbol{x}_s^{3D}, \boldsymbol{y}_s^{3D})$, C denoting the number of classes, N the number of labeled points in a mini-batch, and $\boldsymbol{P}_{\boldsymbol{x}_s}$ the prediction of the 2D or 3D semantic network depending on the modality of $\boldsymbol{x}_s$.

Cross Modal Learning. As highlighted in [31] it is important that the two branches communicate, so that each of the two modalities can take advantage of the other. Given a pair of corresponding 2D-3D points, we apply a mechanism similar to [33], though without deformable convolution. In particular, given a squared patch centered in a 2D point, we force the predictions of each pixel in the patch to be similar to that of the corresponding 3D point with a KL loss. This cross-modal loss denoted as $\mathcal{L}_{\text{xM}}$, is applied by means of auxiliary heads that are trained to mimic the output of the main classifier of the other modality. In this way, the main classifier is simultaneously influenced by the features learned by the other network while keeping its strength. We rely on this simple mechanism to establish a strong baseline as a starting point on which we develop.

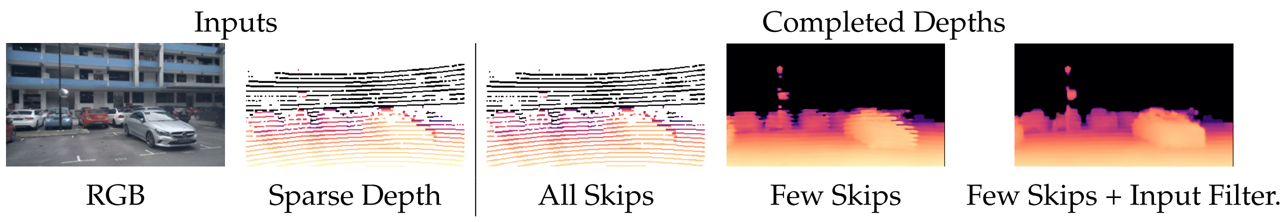


FIGURE 10.3. Depth Completion Ablation. From left to right: the RGB image, the input sparse depth, the depth completed using all the skip connections from the depth encoder, the depth completed using only skip connections $\frac{1}{8}$ and $\frac{1}{16}$, the depth completed applying also input filtering. Sparse depth maps are dilated for visualization purposes.

10.2.2 Depth Completion

Our proposal is based on the following considerations. First, 2D depth maps are similar across domains, e.g., the bottom part of the image is smooth, cars have the same 3D shapes regardless of the time of the day, etc. Second, depth structures such as edges or blobs are tightly correlated to semantic segmentation, indeed we can easily recognize that a car is in the scene only by looking at the depth map. Finally, correlations between depth and semantics are similar across domains, e.g., a road is typically a plane or the sky is far away. Based on the above intuitions, in our work, we consider depth completion as an auxiliary task to make the features of the 2D network similar between domains, while at the same time preserving discriminability for semantic segmentation for the target domain. Specifically, given a 3D scan, we project the 3D points into the image plane to obtain a sparse depth map. Then, we train a multi-task 2D network jointly to segment the source images and complete the sparse depth map on both domains, naturally forcing target features to be robust to the domain shift and discriminative for semantic segmentation. Unluckily, current state-of-the-art techniques for depth completion [240, 241] all require either to be trained with dense depth ground truth or auxiliary information such as video sequences[242, 243]. As we can leverage only single-view sparse depths as supervision, we propose a novel technique to achieve this goal. In the next sections, we first define our multi-task architecture for semantic and depth completion, then we describe the training protocol to pursue self-supervised depth completion.

2D Depth Completion and Semantic Network. We modify a standard U-Net [38] with backbone ResNet34 for 2D semantic segmentation by introducing a multi-scale depth encoder and decoder. The latter takes in input depth features at $\frac{1}{8}$, $\frac{1}{16}$ scales, and RGB features at $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ and $\frac{1}{16}$ to output a dense depth map. Multi-scale depth feature maps from the depth decoder are then leveraged by the semantic segmentation decoder to output semantic classes. A schematic visualization of the 2D network is shown in Fig. 10.4.

Depth supervision. We supervise the depth branch by means of the LiDAR depth points provided as input. We employ the L1 loss alongside edge-aware smoothness [100], described by Eq. 10.2 and Eq. 10.3, respectively, where $\boldsymbol{D}$ is the dense output depth map, $M_{uv} = \boldsymbol{D}_{uv}^{3D\to 2D} > 0$ is a mask of valid reprojected depth points and N_m is the number of valid points in M.

$$L_{\text{depth}}(\boldsymbol{x}^{\text{2D}}, \boldsymbol{D}^{3D\to 2D}) = \frac{1}{N_m}\sum_{u,v}^{N_m} |\boldsymbol{D}_{uv} - \boldsymbol{D}_{uv}^{3D\to 2D}| \cdot M_{uv} \tag{10.2}$$

$$L_{\text{smooth}}(\boldsymbol{x}^{\text{2D}}) = \frac{1}{N_m}\sum_{u,v}^{N_m}(|\delta_u \boldsymbol{D}|e^{-|\delta_u \boldsymbol{x}^{\text{2D}}|} + |\delta_v \boldsymbol{D}|e^{-|\delta_v \boldsymbol{x}^{\text{2D}}|}) \tag{10.3}$$

We penalize abrupt depth changes in areas other than RGB edges through the L_{smooth} term. This strategy has been proven to be effective in self-supervised depth-from-mono [100]. However, by supervising with the LiDAR depth points provided in input, the network can simply learn the identity function. As described before, we limit the usage of high-resolution skip connections from the depth encoder to prevent this behaviour, i.e., we use only skip connections at a $\frac{1}{8}$ and $\frac{1}{16}$ of the input resolution. By comparing columns 3 and 4 of Fig. 10.3 we note the importance of this choice.

Input filtering. Depth completion frameworks are usually trained without any kind of filtering of the sparse depth provided in input [244, 245, 240]. However, when the sparse

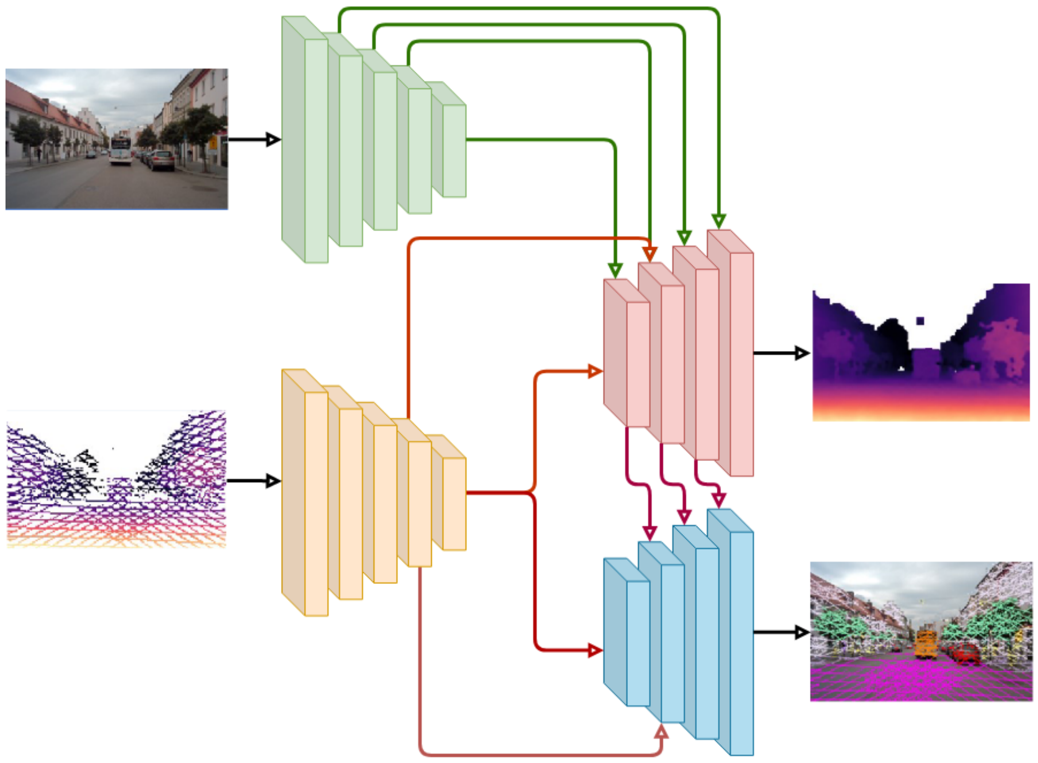


FIGURE 10.4. 2D Network for Depth Completion and Semantic Segmentation. It is composed of a sparse depth encoder and an RGB encoder, the depth decoder takes features at multiple scales from both to output a densified depth map. The segmentation head leverages the multi-scale features of the depth decoding step to output semantic segmentation labels.

input depths are obtained through re-projection from a LiDAR sensor, large areas may be affected by errors due to occlusion between the LiDAR sensor and the RGB camera. This issue yields regions where depth measurements of occluded objects mix together, typically at the borders of objects standing in front of a background far behind, as shown in Fig. 10.5 (middle column). Usually, depth completion networks can learn to cope with this issue, if not excessively prominent, when a cleaned and denser depth ground truth is available. However, when self-supervising the network by the LiDAR itself, these inconsistencies worsen the completion performance. To filter out the occluded depth points, we follow the coarse yet simple and fast approach proposed by [246]: for each depth point d of the projected LiDAR $\boldsymbol{D}^{3D \to 2D}$, we take into account the other valid depth points inside a patch $L_F(d)$ of size $F \times F$ and compute the minimum $m(d) = \min\{y : y \in L_F(d), y > 0\}$, then we apply a threshold over the error between the minimum and the depth value to filter out the outliers $|m(d) - d|/m(d) < \lambda_f$, with $\lambda_f = 0.1$, obtaining a filtered depth, as shown in Fig. 10.5 (3rd column). We set $F = 9$ in this work. Applying this filtering step to the sparse input depths improves the densified outputs provided by our depth completion network, as shown in Fig. 10.3 (last column).

Output filtering. Finally, we argue that the densified depth map really depends on the

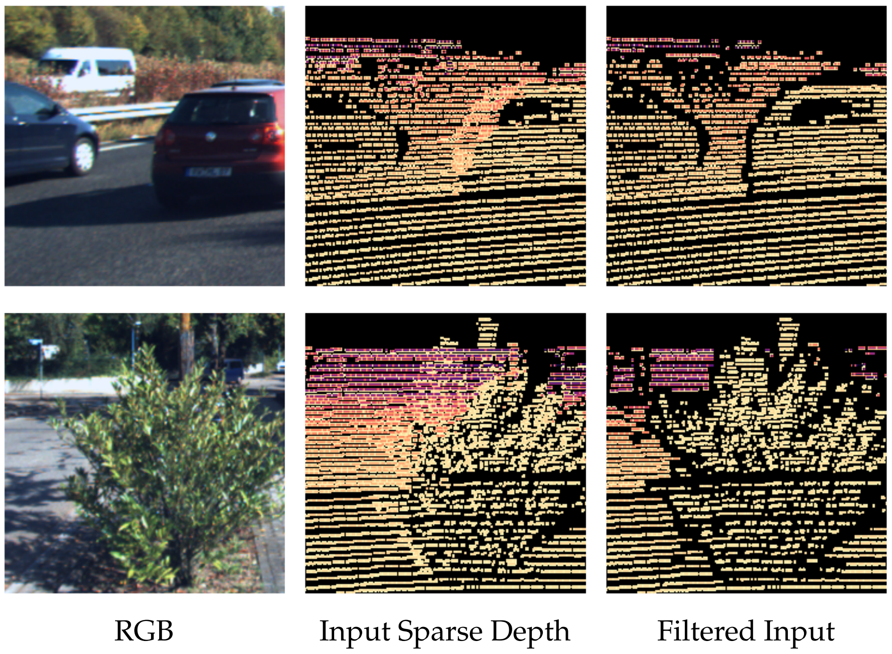


FIGURE 10.5. Example of input filtering to remove occluded pixels. From left to right: RGB image, sparse depth obtained from input LiDAR, filtered depth in input to our 2D network.

spatial distribution of the valid depth measurements. Even though LiDAR sensors usually output an almost homogeneous distribution of sparse points, large areas of the image can lack them at all, e.g., the sky, reflective or absorbent surfaces, as well as objects too far away. In these regions, the depth completion network will likely yield wrong predictions that are not good to be projected back into the 3D point cloud, which is needed for the LiDAR data augmentation strategy described in the next section. To filter these regions out, we employ the following strategy. First, we set pixels with invalid depth measurements in the input LiDAR to zero (white pixels in the top-right image of Fig. 10.6). Then, we apply a max pool with a large kernel size of size λ_p and stride 1 to the input sparse LiDAR obtaining a dilated depth map. We use λ_p equal to 17. In the dilated depth map, pixels with a large invalid neighborhood will have a depth equal to zero. We then select pixel coordinates with a depth equal to zero, and we filter out pixels at the same coordinates from the completed depth map produced by our 2D network (white part of the bottom-right figure in Fig. 10.6).

10.2.3 LiDAR Data Augmentation

Thanks to depth completion, we obtain dense depth maps that can be exploited to boost the performance of the 3D network. Assuming that LiDAR measurements are in the camera reference frame and intrinsic parameters of the RGB camera are known, we can project back each 2D pixel into the corresponding 3D point. Potentially, we can use all these 3D points to increase the number of samples in input to the 3D network. In this way, we alleviate the severe sparsity problem of LiDAR point clouds, and at the same time, we reduce the

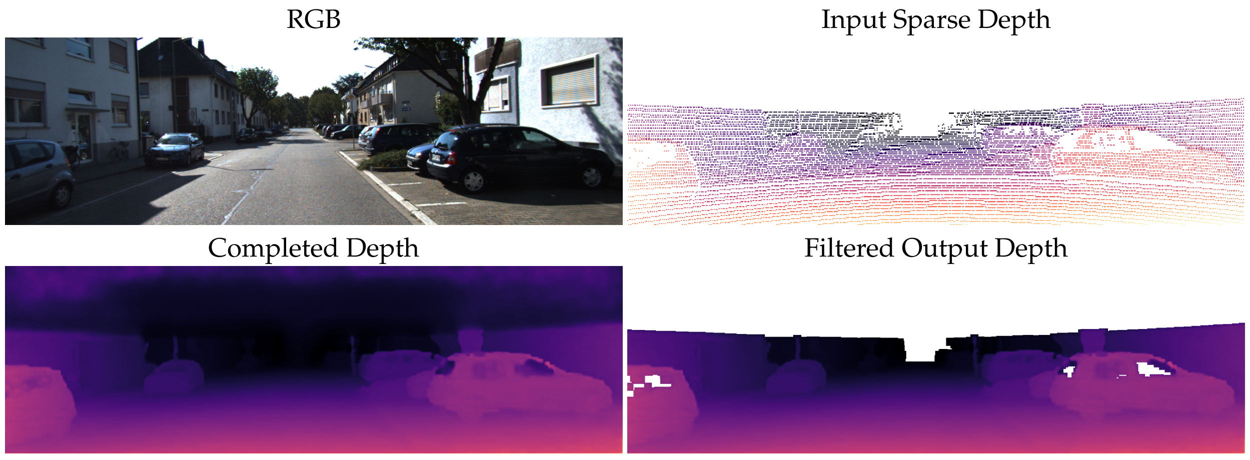


FIGURE 10.6. Output filtering to remove large areas without valid LiDAR measurements. From top to bottom, from left to right: RGB image, sparse input depth, completed depth map, filtered output.

overfitting on the source input scanning pattern that can be different from the target one. However, to fully exploit the potential of the completed depth map, we ought to be able to assign a label to each new point. We do this by relying on the output of the 2D backbone as the 2D network has an inductive bias that pushes pixels in the same neighbourhood to be classified similarly, even with sparse supervision, thus producing dense semantic predictions. However, naively projecting all pixels to obtain a 3D point cloud leads to a huge input that would make the training impractical. Moreover, not all semantic predictions are correct, especially for the target domain. For these reasons, we lift proxy labels from 2D to 3D only for data from the source domain, where the network is trained supervisedly. Then, we select points based on a class-wise confidence-level strategy. Given a 2D dense semantic prediction $P^{2D}_{x_s}$, for each pixel location, we apply the argmax operator to obtain the predicted semantic class, and we use the max operator on the logits after softmax to obtain a per-pixel confidence map as done in several other works [110, 22]. Then, for each class c, we sort predictions based on their confidence scores and we maintain a random 2% among the 10% most confident pixels. In this way, we take into account the class distribution and select pixels for all classes, including rarer ones. Thus, we generate new points and proxy labels only for the source domain, respectively $\tilde{\boldsymbol{x}}_s^{3D}$ and $\tilde{\boldsymbol{y}}_s^{3D}$. A visualization of this augmentation is illustrated in Fig. 10.7. The original labeled LiDAR (left column) only covers a small fraction of the whole image, while our method allows us to synthesize new correctly labeled points (right column).

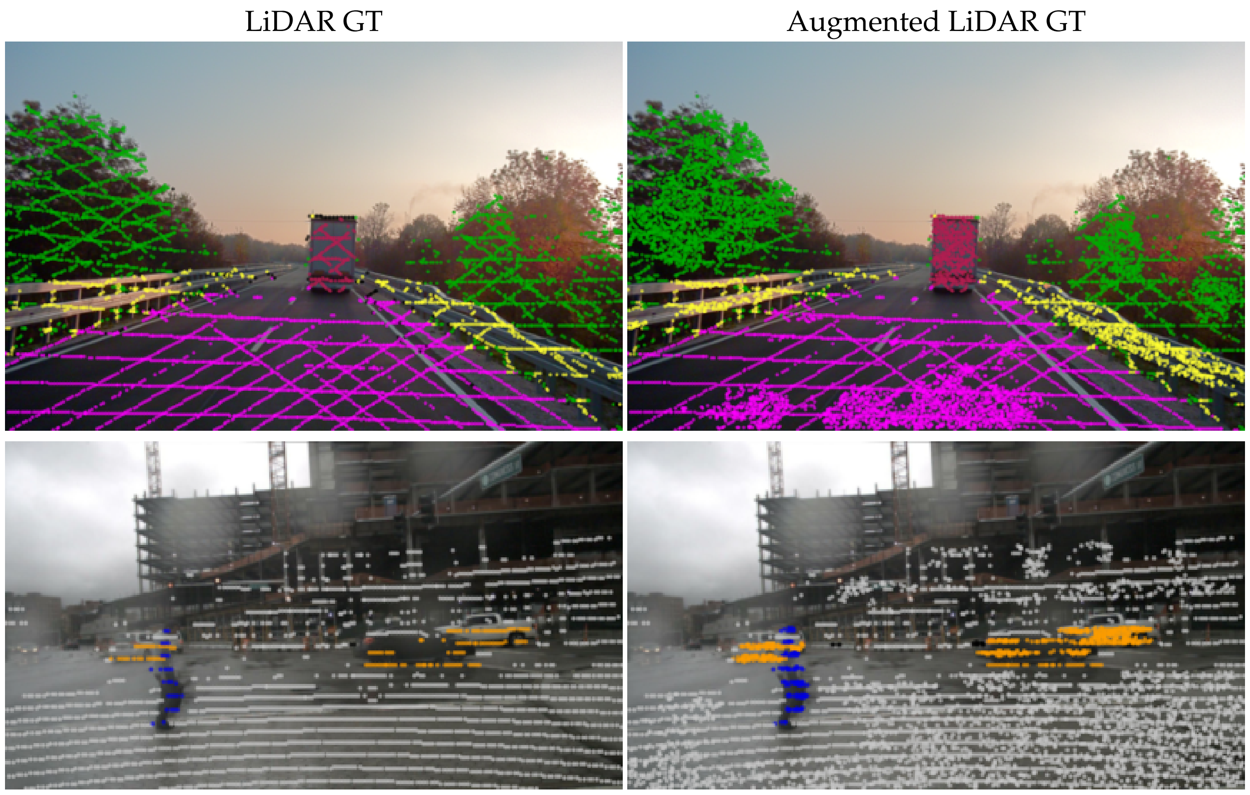


FIGURE 10.7. Effect of our LiDAR augmentation on A2D2 (top) and Nuscenes-USA (bottom). Left: GT LiDAR projected in 2D. Right: Augmented LiDAR with new labeled points, projected over 2D images. Colors indicate semantic class.

10.2.4 Learning Process

The framework is optimized in an end-to-end manner by the following objective function:

$$\begin{aligned}\mathcal{L} = &\mathcal{L}_{\text{seg}}(\boldsymbol{x}_s^{2\text{D}}, \boldsymbol{y}_{\boldsymbol{s}}^{3D\to 2D}) \\ &+ \mathcal{L}_{\text{seg}}(\boldsymbol{x}_s^{3\text{D}}, \boldsymbol{y}_s^{3\text{D}}) + \mathcal{L}_{\text{seg}}(\boldsymbol{x}_s^{\tilde{3}\text{D}}, \boldsymbol{y}_s^{\tilde{3}\text{D}}) \\ &+ \lambda_s \mathcal{L}_{\text{xM}}(\boldsymbol{x}_s^{2D}, \boldsymbol{x}_s^{3D}) + \lambda_t \mathcal{L}_{\text{xM}}(\boldsymbol{x}_t^{2D}, \boldsymbol{x}_t^{3D}) \\ &+ \lambda_d L_{\text{depth}}(\boldsymbol{x}_s^{2D}, \boldsymbol{D}_s^{3D\to 2D}) + \lambda_g L_{\text{smooth}}(\boldsymbol{x}_s^{2D}) \\ &+ \lambda_d L_{\text{depth}}(\boldsymbol{x}_t^{2D}, \boldsymbol{D}_t^{3D\to 2D}) + \lambda_g L_{\text{smooth}}(\boldsymbol{x}_t^{2D})\end{aligned} \tag{10.4}$$

Where λ parameters are the weights applied to each loss component. We keep these hyper-parameters fixed for all settings. Note that, $\mathcal{L}_{\text{seg}}(\boldsymbol{x}_s^{\tilde{3}\text{D}}, \boldsymbol{y}_s^{\tilde{3}\text{D}})$ is only activated after a certain amount of steps N_{aug}, as we need depth completion to be strong enough to reach a low error in its predictions and the 2D segmentation to be reasonably accurate in the source domain. Moreover, when synthesizing the new 3D points $\boldsymbol{x}_s^{\tilde{3}\text{D}}$ from the completed depths, we avoid gradient propagation back to the 2D network from the 3D network.

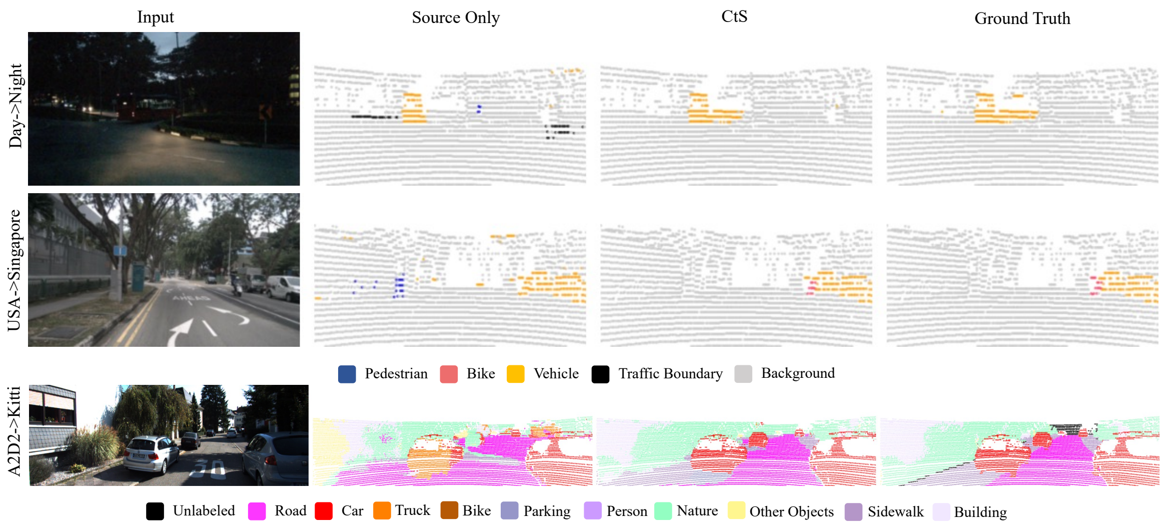


FIGURE 10.8. Qualitatives Adaptation Results. Left to right: Input images, source-only predictions, CtS predictions, and GT.

10.3 Experiments

10.3.1 Implementation Details and Datasets

We use the same data augmentation pipeline as our competitors, i.e., random horizontal flipping and color jittering for 2D images, vertical axis flipping, random scaling, and random 3D rotations for the 3D scans. Augmentations are done independently for each branch. We train with batch size 8, alternating batches of source and target domain. The smallest dataset is repeated to match the length of the other. We use Adam optimizer, we initialize the learning rate at 0.001 and divide by 10 at the iterations 80k and 90k. We train for 100k steps. We use λ_s, λ_t, λ_d, λ_g equals to 0.8, 0.1, 0.1, 0.01 respectively. We selected these parameters based on training loss values, without performing a grid search. We use the same values for all our experiments. We evaluate our framework in the same way as our two competitors xMUDA[31] and DsCML[33] on three standard benchmarks used for multi-modal domain adaptation that provide three different scenarios: day-to-night, country-to-country, and dataset-to-dataset. The first two settings leverage the NuScenes [236] dataset by means of the Day/Night and USA/Singapore splits. In the former, the RGB images exhibit a severe gap due to the different lighting conditions, while the LiDAR shows small differences being the same sensor. For the latter, the sensor setup is the same, but objects may have different appearances as two different cities are involved. The dataset-to-dataset case is realized by adapting from A2D2[237] to SemanticKITTI [222], which comprises both a large change in the sensors setup and in appearance.

10.3.2 UDA Results

We report in Tab. 10.1 our results. We detail for each method the mean Intersection over Union (mIoU) for each modality independently (2D and 3D), and we also show the score obtained by averaging the 2D and 3D scores after Softmax as done by our competitors (Avg). We also report results from [33] for uni-modal domain adaptation techniques applied to each modality independently as a reference. To provide more reliable results in this multi-modal setup, we report the average of three different runs, using the official code [1] provided by the authors. We highlight that we used the same number of steps, optimizers, and hyper-parameters for our competitors as well as for our method to be fair. Trainings require approximately one day for the USA → Singapore and Day → Night setups, and three days for A2D2 → SemanticKITTI on an NVIDIA 3090 RTX GPU. Model selection is done as in our competitors by selecting the best on the validation domain of the target domain, and reporting results on the test set. Our method achieves state-of-the-art performance across all scenarios and modalities. In particular, CtS shines in the Day → Night adaptation scenario, where the RGB domain gap is larger. In this setting, in fact, we improve by 1.9% for the 2D branch and by 1.6% in terms of mIoU when comparing with the best previous model. When averaging the predictions from both 2D and 3D branches, which is the real and final objective, we observe an even larger improvement of 5,3% (Avg column). We attribute this to the completion auxiliary task, which is able to guide the network to classify each pixel by also reasoning on the 3D cues learned by solving the depth completion task. On USA → Singapore, we also observe good results, especially for the 2D branch where we obtain a large 3.4% improvement. This means that the proposed depth completion auxiliary task is also beneficial in the presence of a smaller RGB domain gap. As regards A2D2 → SemanticKITTI, we improve by 0.6%, 2.6%, and 0.9% the previous best multi-modal framework for 2D, 3D, and Avg respectively. In this setting, where the LiDAR sensor is completely different across domains, we substantially improve the performance of the 3D network. This is due to the proposed 3D augmentation, which is indeed able to avoid overfitting of the source pattern and at the same time reduce the sparsity of the LiDAR input.

10.3.3 Additional Studies

Ablation of contributions. In Tab. 10.2 we ablate the effect of our contributions considering both USA → Singapore and Day → Night. In the first row, we report the results of our baseline architecture, where we employ the cross-modal loss L_{xM} between the two modalities as done in [33] but without the deformable convolutions. In the second row, we activate depth completion as an auxiliary task, and we observe a large boost for the 2D network for both scenarios: +5,1% and +2,3% respectively. This confirms that forcing the network to

[1]https://github.com/leolyj/DsCML, https://github.com/valeoai/xmuda

Modality	Method	USA → Singapore			Day → Night			A2D2 → SemanticKITTI		
		2D	3D	Avg	2D	3D	Avg	2D	3D	Avg
	Baseline (Source only)	53.2	46.8	61.2	41.8	41.4	47.6	36.4	37.3	42.2
Uni-modal	MinEnt [152]	53.4	47.0	59.7	44.9	43.5	51.3	38.8	38.0	42.7
	CyCADA [152]	54.9	48.7	61.4	45.7	45.2	49.7	38.2	43.9	43.9
	AdaptSegNet [23]	56.3	47.7	61.8	45.3	44.6	49.6	38.8	44.3	44.2
	CLAN [247]	57.8	51.2	62.5	45.6	43.7	49.2	39.2	44.7	44.5
Multi-modal*	xMUDA [31]	57.2	51.6	61.1	48.9	45.6	52.9	39.0	43.4	44.9
	DsCML [33]	58.5	52.3	62.3	47.5	45.2	53.0	38.9	40.1	43.2
	DsCML + CMAL [33]	57.5	51.0	61.9	46.9	36.2	49.2	27.4	33.3	33.6
	CtS (Ours)	**61.9**	**52.4**	**63.6**	**50.8**	**47.2**	**58.3**	**39.6**	**46.0**	**45.8**

TABLE 10.1. Results for 3D semantic segmentation with both uni-modal and multi-modal adaptation methods. We report performance for each network stream in terms of mIoU. 'Avg' column denotes the obtained by taking the mean of the 2D and 3D predictions. * indicates the mean of three different runs with different seeds.

		USA → Singapore			Day → Night		
2D-C	3D-A	mIoU			mIoU		
		2D	3D	Avg	2D	3D	Avg
		56.2	51.3	61.8	48.2	42.8	52.8
✓		61.3	51.0	62.1	50.5	45.4	54.2
	✓	56.5	**52.6**	60.5	49.5	45.8	53.0
✓	✓	**61.9**	52.4	**63.6**	**50.8**	**47.2**	**58.3**

TABLE 10.2. Ablation studies for the proposed contributions. 2D-C: 2D completion, 3D-A: Augmentation for 3D network.

reason about the depth structures of the input image helps to generalize better. Moreover, we can observe that our first contribution already improves the overall performance (Avg) by 0.3% and 1.4% respectively. When only activating the LiDAR augmentation, we expect the 3D branch to observe a larger improvement as we are specifically tackling the 3D modality. Indeed, we note a +1.6% for USA → Singapore and +0.4% for Day → Night in terms of mIoU when comparing the performance of this model (third row) with our baseline. Since this augmentation step needs a dense depth map, in this case, we exploit a separate depth completion network pre-trained with our self-supervised methodology. Thereby, the 2D semantic network is not multi-task. When activating all contributions, we obtain the best average results. In Fig. 10.8, we depict a qualitative comparison of a source-only model with our proposal.

Auxiliary tasks alternatives. A plausible alternative to injecting 3D cues into the learning process is to use monocular depth estimation as an auxiliary task. To compare depth completion with this solution, we implement a network with a single encoder that processes RGB images and two decoders, one that predicts each pixel semantic label, and one to estimate depth as done for the depth completion task. The model is then optimized in the same way i.e., by applying Eq. (10.2) and Eq. (10.3) on both domains and Eq. (10.1) only for the source one. We compare this variant with the proposed auxiliary task in Tab. 10.3. We observe that

	USA → Singapore			Day → Night		
2D architecture	mIoU			mIoU		
	2D	3D	Avg	2D	3D	Avg
Depth from Mono	57.4	49.7	60.2	38.4	45.1	43.7
Completion	**61.3**	**51.0**	**62.1**	**50.5**	**45.4**	**54.2**

TABLE 10.3. Comparison with different auxiliary tasks.

Supervision		Method	RMSE ↓ (mm)	MAE ↓
GT Depth	‡	NLSPN [245]	788.00	199.50
	‡	PENet [244]	791.62	242.25
	‡	PackNet [249]	1027.32	356.04
		StD [242]	878.56	260.90
Photometric + LiDAR		StD [242]	1384.85	358.92
Photometric		StD [242]	1901.16	658.13
LiDAR		CTS (depth only)	1788.37	506.86

TABLE 10.4. Results on the validation split of KITTI Depth Completion. GT: ground truth, Photometric: photometric loss on videos, LiDAR: sparse depths from input LiDAR. ‡: evaluated using officially released weights.

depth completion performs better across all modalities in both Day → Night and USA → Singapore. This is due to the fact that to solve the task of monocular depth estimation, the network has to rely on RGB features that do not provide any additional improvements if the gap in the RGB space is too large. On the other hand, we argue that depth completion networks can focus also on the geometry of the scene in input and not only on RGB images to solve the task, and this is important to improve generalization on the target domain.

Quantitative results on depth completion. Although we mainly employ depth completion as an auxiliary task, we investigate the quality of completed depths also from a quantitative point of view. In Table 10.4, we compare our self-supervised approach with state-of-the-art supervised depth completion methods that leverage the dense ground-truth of the KITTI-Depth-Completion split[248], and with methods that exploit video sequences [242]. Despite being trained with the input LiDAR only, our performances are still comparable. Indeed, our method has a Mean Absolute Error (MAE) only 300mm higher than the state-of-the-art supervised method [245] (row 1 vs 7), and performs better than [242] when using only the photometric loss on video sequences (row 6 vs 7). The quality of our completed depth maps can be assessed by looking at the results in Fig. 10.9.

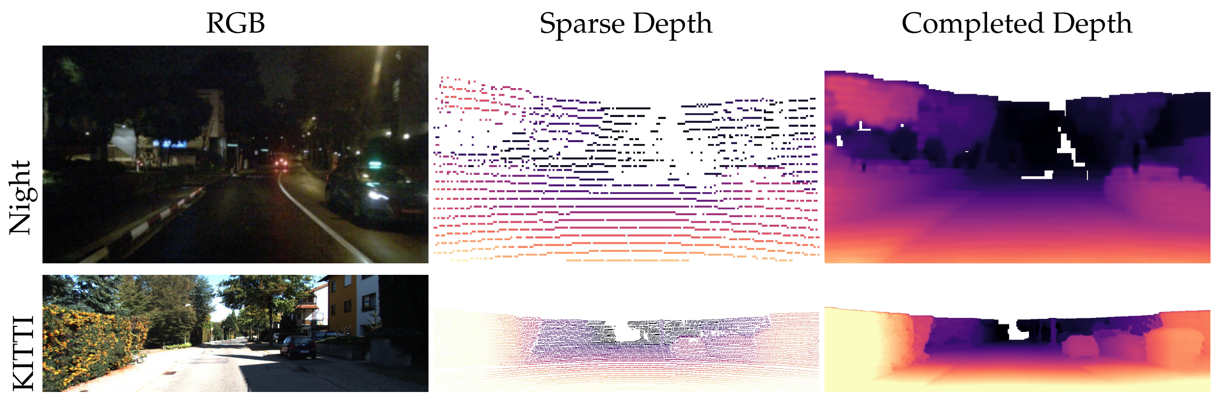


FIGURE 10.9. Depth completion qualitative results. From left to right, RGB, sparse depth from LiDAR, and completed depth.

10.4 Conclusions

We have shown that depth completion is an effective auxiliary task to improve generalization for the 2D network. Furthermore, we propose to exploit completed depths to augment the source LiDAR to achieve better results. We believe that this task could be even more useful when applied to online adaptation, where video sequences can be available and could be used to obtain better 3D geometries, and consequently a better semantic understanding.

Chapter 11

On the Complementarity of 2D and 3D Networks for Multi-Modal Semantic Segmentation

11.1 Introduction

As we have seen from the previous chapter, all the approaches for multi-modal 3D semantic segmentation [31, 32, 33, 221, 233] leverage a peculiar two-branch 2D-3D architecture, in which images are processed by a 2D convolutional network, e.g., ResNet [14], while point clouds by a 3D convolutional backbone, e.g., SparseConvNet [227]. By processing each modality independently, each of the two branches focuses on extracting features from its specific signal (RGB colors or 3D structure information) that can be fused effectively due to their inherent complementarity in order to produce a better segmentation score. Indeed, averaging logits from the two branches provides often an improvement in performance, e.g., a mIoU gain from 2% to 4% in almost all experiments in [31]. Although we agree that each modality embodies specific information, such as color for images and 3D coordinates for point clouds, we argue that the complementarity of the features extracted by the two branches is also tightly correlated to the different information processing machinery, i.e., 2D and 3D convolutions, which makes networks focusing on different areas of the scene with different receptive fields. Indeed, in Fig. Fig. 11.1, given a point belonging to the red car, we visualize the effective receptive field [250] of the 2D and 3D networks (red ellipses). As we can clearly see from the receptive fields in the right part of the figure, the features extracted by the 3D network mainly leverage points in a 3D neighborhood, i.e., include points of the car surface. In contrast, the features extracted by the 2D network look at a neighborhood in the 2D projected space, and thus they depend also on pixels of the building behind the car, which are close in image space but far in 3D. We argue that this is one of the main reasons why the features from the two branches can be fused so effectively. Based on the above intuition, we propose to feed 3D and RGB signals to *both* networks as this should not hinder the complementarity of their predictions, with the goal of making the network more

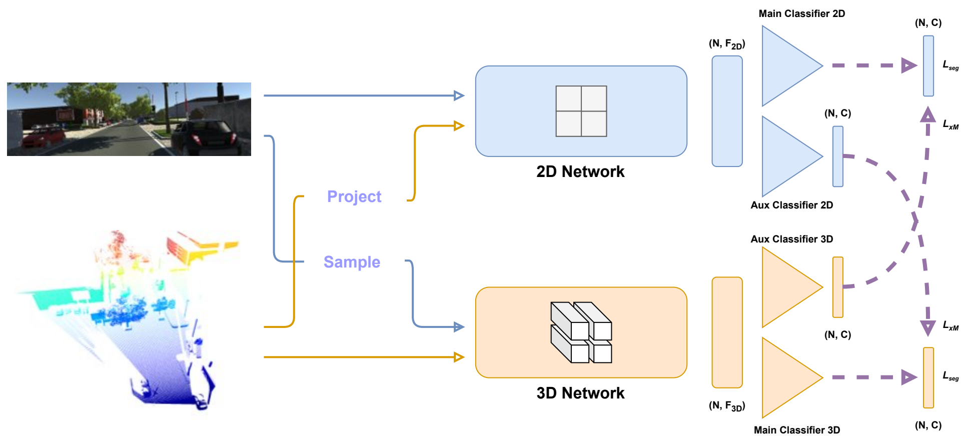


FIGURE 11.1. **Framework overview.** The RGB image and the sparse depth map obtained from the projection of the corresponding point cloud are fed to a custom 2D architecture to extract point-wise features. The same point cloud and sampled colors from the RGB image are given in input to the 3D Network. Then, two main classifiers output the main predictions to be used at test time. Moreover, two auxiliary classifiers are used at training time only to allow the exchange of information across branches.

robust to the change of distributions between the training and the test scenarios. Feeding both branches with both modalities would make: i) the 2D network more robust to domain shifts, as depth information (z coordinates of point clouds projected into image space) is more similar across different domains, as shown in several papers [105, 251, 107, 104, 48, 252]; ii) the 3D network more capable of adapting to new domains thanks to RGB information associated with each point which allows learning better semantic features for the target domain, when this is available, using Unsupervised Domain Adaptation (UDA) approaches. Thus, we propose a simple architecture for multi-modal 3D semantic segmentation consisting of a 2D-3D architecture with each branch fed with both RGB and 3D information. Despite its simplicity, our proposal achieves state-of-the-art results in multi-modal UDA benchmarks, surpassing competitors by large margins, as well as significantly better domain generalization compared to a standard 2D-3D architecture [32].

11.2 Method

Setup and Notation. The notation is very similar to the one in the previous chapter as we tackle the same problem. We define input source samples $\{x_s^{2D}, x_s^{3D}\} \in \mathcal{S}$ and target samples $\{x_t^{2D}, x_t^{3D}\} \in \mathcal{T}$, with x^{2D} being the 2D RGB image and x^{3D} the corresponding point cloud, with 3D points in the camera reference frame. Note that x^{3D} contains only points visible from the RGB camera, assuming that the calibration of the two sensors is available for both domains and does not change over time. We assume the availability of annotations y_s^{3D} only for the source domain for each 3D point. We also have at our disposal the unlabeled samples

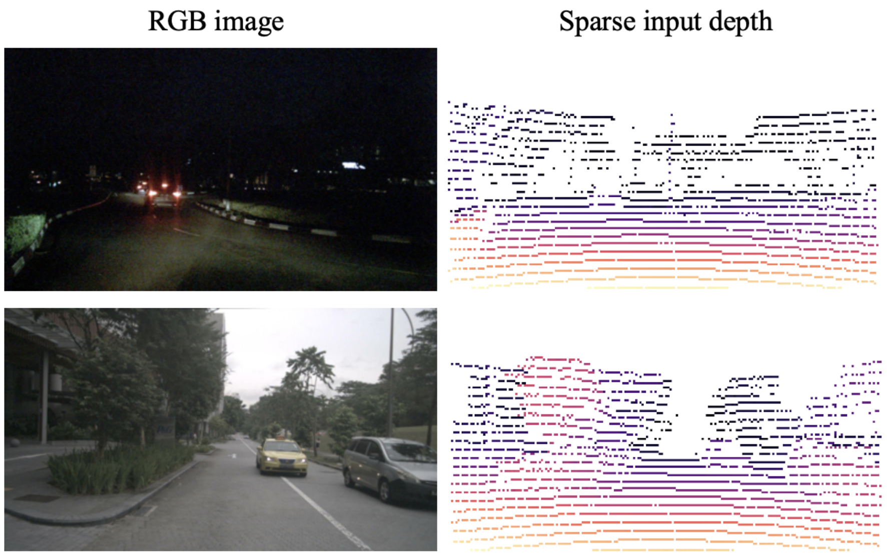


FIGURE 11.2. Depth comparison during daylight or night. Differently, from the RGB image (left column), a sparse depth map obtained by projecting a LiDAR scan into the image plane is not affected by the light conditions.

from the target domain. Our goal is to obtain a point-wise prediction $N \times C$ for $\boldsymbol{x}_t^{3D}$, with N and C being the number of points of the target point cloud and the number of classes, respectively.

11.2.1 Base 2D/3D Architecture

We build our contributions upon the two independent branches (2D and 3D) architecture proposed in [32]. The 2D branch processes images to obtain a pixel-wise prediction given $\boldsymbol{x}^{2D}$ and it consists of a standard 2D U-Net[38]. On the other hand, the 3D branch takes in input point clouds to estimate the class of each point of $\boldsymbol{x}^{3D}$ and it is implemented as a 3D sparse convolutional network [227]. Thanks to the fact that 2D-3D correspondences are known, 3D points can be projected into the image plane to supervise the 2D branch, as supervision is provided only for the sparse 3D points. We denote the 3D semantic labels projected into 2D with the symbol $\boldsymbol{y}^{3D \to 2D}$. As argued by [32], such design choice allows one to take advantage of the strengths of each input modality, and final predictions can be obtained by averaging the outputs of the two branches to achieve an effective ensemble. In our work, we adopt the same framework, and we give an intuitive explanation of why this design choice is particularly effective. In particular, we reckon that the two predictions are complementary not only for the input signals being different but also for the fact the two branches focus on different things to determine their final predictions. Indeed, 3D convolutions produce features by looking at points that are close in the 3D space, while the 2D counterparts focus on neighboring pixels in the 2D image plane. Therefore, given corresponding 2D and 3D points, the two mechanisms implicitly produce features containing complementary information. In the right part of Fig.

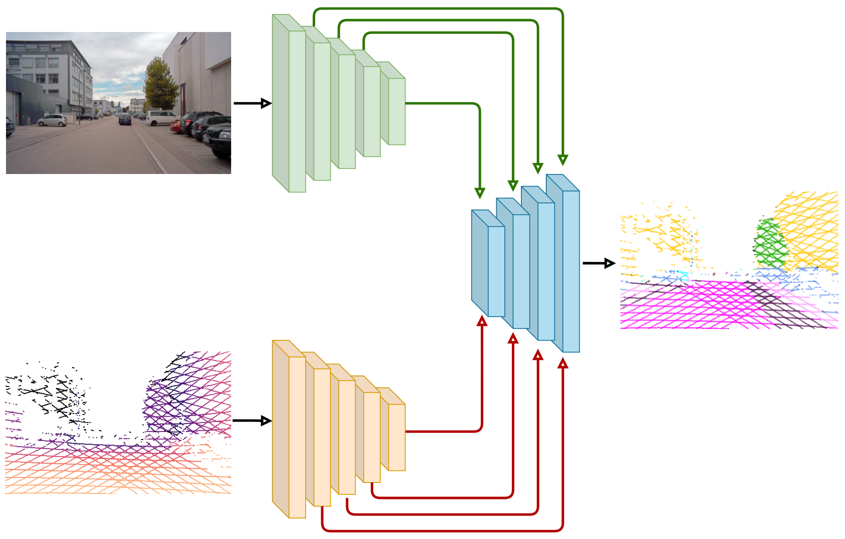


FIGURE 11.3. 2D Network of our framework. It is composed of a depth encoder and an RGB encoder to process the two inputs independently. The segmentation decoder leverages the multi-scale features of both encoders to predict semantic segmentation labels.

Fig. 11.1 we visualize the Effective Receptive Fields (ERF) [250] of a 2D U-Net with backbone ResNet34[14] and of a 3D U-Net with backbone SparseConvNet[227]. It is worth highlighting that we do not focus on the theoretical but on the effective receptive field, which is computed by analyzing the real contribution of each input point to the final prediction (the hotter the color intensity in the visualization, the larger the point contribution). Comparing the re-projected 2D ERF into 3D and the 3D ERF we can clearly appreciate that the 2D network focuses on sparse 3D regions, i.e., from the car to the building in the background, while the 3D counterpart reasons on a local 3D neighborhood (only car points). With this intuition in mind, we argue that by feeding the RGB signal to the 3D network, and the 3D information to the 2D backbone, we would still obtain complementary features that can be effectively fused together. Moreover, it is well-known that employing depth information as input to 2D segmentation networks can make it more robust to domain shift [253, 252]. At the same time, we posit that the 3D network with RGB information may be able to extract better semantic features. Differently from previous approaches that employ two independent architectures, based on the above considerations, we propose our multi-modal, two-branch framework named MM2D3D. In Sec. 11.2.2 we show how a point cloud can be used to obtain a stronger and more suitable input signal for the 2D network. Similarly, in Sec. 11.2.3 we describe our multi-modal 3D network.

11.2.2 Depth-based 2D Encoder

In this section, we focus on how we can use point clouds to make a 2D segmentation architecture more robust to domain shift. Inspired by [253, 252], we propose to use depth maps as an input signal that is less influenced by the domain gap. As we can observe from the two depth maps in Fig. 11.2, it is hard to understand which one was captured during day or night. At the same time, some objects such as the car can be distinguished by only looking at depths (bottom right of the second depth map). Thus, depth maps provide useful hints to solve the task of semantic segmentation. Given these considerations, we argue that exploiting such invariant information may alleviate the domain shifts and can be used to extract discriminative features for the segmentation task. At a first glance, injecting 3D cues into the 2D branch may seem redundant as the 3D network already has the capability to reason on the full 3D scene. However, given that the two networks have very different receptive fields, we can exploit such additional and useful information without the risk of hindering the complementarity of the two signal streams. Assuming point clouds expressed in the camera reference frame and the availability of the intrinsic camera matrix, we can project the original 3D point cloud to obtain a sparse depth map. In practice, the value of the z axis is assigned to the pixel coordinate (u, v) obtained by projecting a 3D point into the image plane. Similarly to [253], to process both inputs, we modify the 2D encoder of the 2D U-Net architecture by including an additional encoder to process the sparse depth maps obtained from the point cloud. As can be seen in Fig. 11.3, the two streams i.e. one for the RGB image and the other for the sparse depth map, are processed independently. Then, the concatenated depth and RGB features are processed by a decoder, composed of a series of transposed convolutions and convolutions in order to obtain semantic predictions of the same size as the input image. Moreover, features from layers of $\frac{1}{2}$ to $\frac{1}{16}$ of the input resolution are concatenated using skip connections with the corresponding layer of the decoder. This simple design choice allows semantic predictions to be conditioned also on the input depth signal, without altering the RGB encoder that provides useful classification features. Furthermore, without altering the RGB encoder, we can take advantage of a pre-trained architecture on ImageNet [47] as done by our competitors.

11.2.3 RGB Based 3D Network

In this work, we focus on the 3D convolutional network, SparseConvNet [227], as it can segment large scenes efficiently. In this network, the initial point cloud is first voxelized such that each 3D point is associated with only one voxel. Then, rather than processing the entire voxel grid, these models work with a sparse tensor representation ignoring empty voxels for the sake of efficiency. The network associates a feature vector to each voxel, and convolutions calculate their results based on these features. A standard choice for the voxel features is to simply assign to it a constant value, i.e., 1. Although these strategies have been shown to be effective [221, 254], the feature vector can be enriched to make it even more suitable for semantic segmentation. Based on our intuition of the different receptive fields, we can borrow information from the other modality to improve the performance of each branch, still preserving 2D-3D feature complementarity. Thus, we use RGB colors directly as features for each voxel of the SparseConvNet. Moreover, we design a simple yet effective strategy to let the 3D network decide whether to use or not this information. More specifically, the original RGB pixel values are fed to a linear layer that predicts a scalar value α to be multiplied by the color vector. For instance, learning this scaling could be useful in the UDA scenario, where we can train on unlabelled target samples, to discard RGB colors in case they do not provide any useful information, e.g., dark pixels in images acquired at night time.

11.2.4 Learning Scheme

Supervised Learning. Given the softmax predictions of the 2D and 3D networks, P_{2D} and P_{3D}, we supervise both branches using the cross-entropy loss on the source domain:

$$\mathcal{L}_{\text{seg}}(\boldsymbol{x}_s, \boldsymbol{y}_s) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} \boldsymbol{y}_s^{(n,c)} \log \boldsymbol{P}_{\boldsymbol{x}_s}^{(n,c)} \tag{11.1}$$

with $(\boldsymbol{x}_s, \boldsymbol{y}_s)$ being either $(\boldsymbol{x}_s^{\text{2D}}, \boldsymbol{y}_s^{3D\rightarrow 2D})$ or $(\boldsymbol{x}_s^{\text{3D}}, \boldsymbol{y}_s^{3D})$.

Cross-Branch Learning. To allow an exchange of information between the two branches, [32] and [33] add an auxiliary classification head to each one. The objective of these additional classifiers is to mimic the other branch output. The two auxiliary heads estimate the other modality output: 2D mimics 3D ($P_{\text{2D}\rightarrow\text{3D}}$) and 3D mimics 2D ($P_{\text{3D}\rightarrow\text{2D}}$). In practice, this is achieved with the following objective:

$$\begin{aligned}\mathcal{L}_{\text{xM}}(\boldsymbol{x}) &= \boldsymbol{D}_{\text{KL}}(\boldsymbol{P}_{\boldsymbol{x}}^{(n,c)} || \boldsymbol{Q}_{\boldsymbol{x}}^{(n,c)}) \\ &= -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} \boldsymbol{P}_{\boldsymbol{x}}^{(n,c)} \log \frac{\boldsymbol{P}_{\boldsymbol{x}}^{(n,c)}}{\boldsymbol{Q}_{\boldsymbol{x}}^{(n,c)}}\end{aligned} \tag{11.2}$$

with $(\boldsymbol{P}, \boldsymbol{Q}) \in \{(\boldsymbol{P}_{\text{2D}}, P_{\text{3D}\rightarrow\text{2D}}), (\boldsymbol{P}_{\text{3D}}, P_{\text{2D}\rightarrow\text{3D}})\}$ where $\boldsymbol{P}$ is the distribution from the main classification head which has to be estimated by $\boldsymbol{Q}$. Note that in Eq. (11.2), x can belong to either $\mathcal{T}$ or $\mathcal{S}$. This means that, in the UDA scenario, Eq. (11.2) can also be optimized for $\mathcal{T}$, forcing the two networks to have consistent behavior across the two modalities for the target domain as well without any labels.

Self-Training. Only in the UDA scenario, where unlabelled target samples are available, as done by [32], we perform one round of Self-Training [110] using pseudo-labels [255]. Specifically, after training the model with Eq. (11.1) for the source domain and Eq. (11.2) on both domains, we generate predictions on the unlabeled target domain dataset to be used as pseudo ground truths, $\hat{\boldsymbol{y}}_t$. Following [32], we filter out noisy pseudo-labels by considering only the most confident predictions for each class. Then, we retrain the framework from scratch the model minimizing the following objective function:

$$\begin{aligned}\mathcal{L} = \mathcal{L}_{\text{seg}}(\boldsymbol{x}_s, \boldsymbol{y}_s) + \lambda_t \mathcal{L}_{\text{seg}}(\boldsymbol{x}_t, \hat{\boldsymbol{y}}_t) \\ + \lambda_{xs} \mathcal{L}_{\text{xM}}(\boldsymbol{x_s}) + \lambda_{xt} \mathcal{L}_{\text{xM}}(\boldsymbol{x_t})\end{aligned} \tag{11.3}$$

11.3 Experiments

11.3.1 Implementation details

We use the same data augmentation pipeline as our competitors, which is composed of random horizontal flipping and color jittering for 2D images, while vertical axis flipping, random scaling, and random 3D rotations are used for the 3D scans. It is important to note that augmentations are done independently for each branch. We implement our framework in PyTorch using two NVIDIA 3090 GPU with 24GB of RAM. We train with a batch size of 16, alternating batches of source and target domain for the UDA case and source only in DG. The smaller dataset is repeated to match the length of the other. We rely on the AdamW optimizer [198] and the One Cycle Policy as a learning rate scheduler [128]. We train for 50, 35, 15, and 30 epochs for USA $\rightarrow$ Singapore, Day $\rightarrow$ Night, v. KITTI $\rightarrow$ Sem. KITTI, and A2D2 $\rightarrow$ Sem. KITTI respectively. As regards the hyper-parameters, we follow [32] and set $\lambda_s = 0.8, \lambda_t = 0.1, \lambda_{xs} = 0.1, \lambda_{xt} = 0.01$ in all settings without performing any fine-tuning on these values.

11.3.2 UDA results

Following previous works in the field [32, 33], we evaluate the performance of a model on the target test set using the standard Intersection over Union (IoU) and select the best checkpoint according to a small validation set on the target domain. In Tab. 11.1, we report our results on the four challenging UDA benchmarks explained in **??**. For each experiment, we report two reference methods: a model trained only on the source domain, named *Baseline (Source Only)*; a model trained only on the target data using annotations, representing the upper bound that can be obtained with real ground-truth, namely *Oracle*. We note that these two models employ the two independent stream architecture of [32]. In the columns *Avg*, we report the results obtained by the mean of the 2D and 3D outputs after softmax which is the final output of our multi-modal framework. For the sake of completeness, we also report the results of each individual branch (2D and 3D only). We compare our method with both Uni-modal and Multi-Modal approaches. In particular, we mainly focus on a

Modality	Method	USA → Singapore			Day → Night			v.KITTI → Sem.KITTI			A2D2 → Sem.KITTI		
		2D	3D	Avg	2D	3D	Avg	2D	3D	Avg	2D	3D	Avg
	Baseline (Source only)	58.4	62.8	68.2	47.8	68.8	63.3	26.8	42.0	42.2	34.2	35.9	40.4
Uni-modal	MinEnt [152]	57.6	61.5	66.0	47.1	68.8	63.6	39.2	43.3	47.1	37.8	39.6	42.6
	Deep logCORAL [256]	64.4	63.2	69.4	47.7	68.7	63.7	41.4	36.8	47.0	35.1	41.0	42.2
	PL [257]	62.0	64.8	70.4	47.0	69.6	63.0	21.5	44.3	35.6	34.7	41.7	45.2
Multi-modal	xMUDA [31]	64.4	63.2	69.4	55.5	69.2	67.4	42.1	46.7	48.2	38.3	46.0	44.0
	DsCML* [33]	52.9	52.3	56.9	51.2	61.4	61.8	31.8	32.8	34.8	25.4	32.6	33.5
	MM2D3D (Ours)	**71.7**	**66.8**	**72.4**	**70.5**	**70.2**	**72.1**	**53.4**	**50.3**	**56.5**	**42.3**	**46.1**	**46.2**
	Oracle	75.4	76.0	79.6	61.5	69.8	69.2	66.3	78.4	80.1	59.3	71.9	73.6

TABLE 11.1. **Results for UDA for 3D semantic segmentation with both uni-modal and multi-modal adaptation methods**. We report performance for each network stream in terms of mIoU. 'Avg' column denotes the obtained by taking the mean of the 2D and 3D predictions. * indicates trained by us using official code.

comparison with xMUDA[32] and DsCML[33], as they are the current s.o.t.a. methods for UDA in our multi-modal setting. In particular, for the latter, we use the official code provided by the authors [1] to retrain the model on the new more exhaustive benchmark defined by [32]. Overall, we note how our contributions largely improve results over competitors across all settings and modalities. In USA → Singapore, we observe a large boost in both branches, and on average we report a +3% (third row of the Multi-modal section). The large improvement (+7.3%) for the 2D model, suggests that the depth cues injected into a common 2D decoder can be quite useful even if the light conditions are similar across domains. In Day → Night, we observe a remarkable +15% for the 2D branch, which in turn raises the average score to +4.7% when compared with the previous best model. We attribute this boost in performance to the depth encoder, which is able to provide useful hints when the RGB encoder has to deal with large changes in light conditions. Remarkably, our network surpasses even the performance of the two independent streams *Oracle*. Indeed, as discussed in Sec. 11.2.2, the sparse depth is able to give useful details for the task of semantic segmentation. Moreover, thanks to the fact that the cross-modal loss Sec. 11.2.4 is optimized for both domains, the network lean to use both encoders to make the final predictions, leading to more robust performance when the encoder receives a less informative RGB signal. In the challenging synthetic-to-real case (v. KITTI → Sem. KITTI), we also notice consistent improvements in both branches. We highlight that even though RGB colors are likely the main source of the domain gap, they are still useful to obtain a stronger 3D model (+3.6%). In the A2D2 → Sem. KITTI setting, where the sensors setup is different, we still benefit from the depth hints provided to the 2D network, and on average, our method surpasses by 2.2% xMUDA. In general, we highlight that though we employed both modalities in the 2D and 3D branches, the Avg performances are better than those of each individual branch, supporting our core intuition. In Fig. 11.4, we report some qualitative results obtained with our framework.

[1] https://github.com/leolyj/DsCML

Method	USA → Singapore			Day → Night			v.KITTI → Sem.KITTI			A2D2 → Sem.KITTI		
	2D	3D	Avg	2D	3D	Avg	2D	3D	Avg	2D	3D	Avg
xMUDA* [31]	58.7	**62.3**	68.6	43.0	**68.9**	59.6	25.7	37.4	39.0	34.9	**36.7**	41.6
MM2D3D (Ours)	**69.7**	**62.3**	**70.9**	**65.3**	63.2	**68.3**	**37.7**	**40.2**	**44.2**	**39.6**	35.9	**43.6**

TABLE 11.2. **Results for 3D for semantic segmentation in the Domain Generalization setting.** We report performance for each network stream in terms of mIoU. 'Avg' column denotes the obtained by taking the mean of the 2D and 3D predictions. * indicates trained by us using official code.

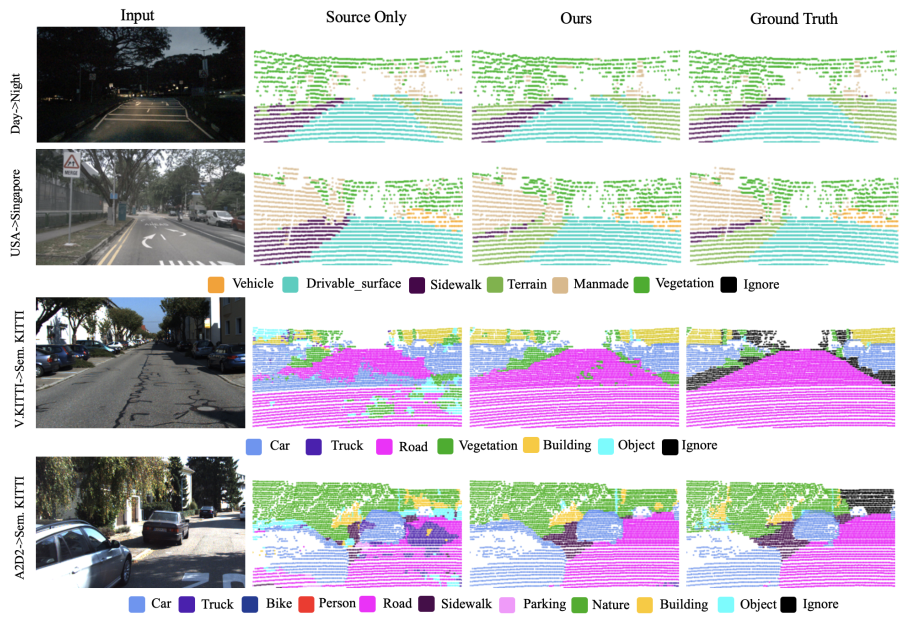


FIGURE 11.4. **Qualitative examples of the proposed framework in the UDA scenario.** From left to right: RGB images, point cloud segmentations projected into 2D for visualization purpose of the baseline source only model, our method, and the ground truth respectively. From top to bottom: the four different adaptation scenarios. Comparisons are provided for the target domain.

11.3.3 Domain Generalization results

In this section, we test our contributions in the Domain Generalization setting, in which the target data cannot be used at training time. For this study we consider XMUDA [32] as our baseline two-branch 2D-3D method, and we show that our simple contribution can boost generalization performances. Results are reported in Tab. 11.2. To implement this experiment we keep the same hyper-parameters as used in the UDA scenario. We retrain [31] using the official code, but without the target data. Also in this setting, we observe overall large improvements. We believe that this can be ascribed especially to the introduction of the depth encoder, which helps to achieve a better generalization. Evidence of this is well observable in the Day → Night, where the 2D performance increases from 43% to 65.3% in terms of mIoU, but also for USA → Singapore and (v. KITTI → Sem. KITTI), where we achieve +11% and +12 respectively. In the Day → Night scenario, the 3D branch experiences a drop in performance. We think that it is related to the large domain shift of RGB images. Differently from the adaptation scenario in which we can train directly on the unlabeled

Method	Depth	RGB	USA → Singapore			Day → Night			v.KITTI → Sem.KITTI			A2D2 → Sem.KITTI		
			2D	3D	Avg	2D	3D	Avg	2D	3D	Avg	2D	3D	Avg
xMUDA [32]			64.4	63.2	69.4	55.5	69.2	67.4	42.1	46.7	48.2	38.3	46.0	44.0
MM2D3D (Ours)	✓		69.5	64.0	69.6	**71.3**	69.9	**72.8**	52.6	40.3	53.7	41.7	44.8	45.9
MM2D3D (Ours)	✓	✓	**71.7**	**66.8**	**72.4**	70.5	**70.2**	72.1	**53.4**	**50.3**	**56.5**	**42.3**	**46.1**	**46.2**

TABLE 11.3. **Modality-wise ablation of the proposed framework in the UDA scenario**. *Depth* indicates the usage of the additional sparse depth encoder, while *RGB* denotes the introduction of the RGB information in the 3D network.

MM2D3D (Ours)	71.7	66.8	72.4	70.5	**70.2**	72.1	53.4	50.3	56.5	42.3	46.1	46.2
xMUDA [32] + PL	67.0	65.4	71.2	57.6	69.6	64.4	45.8	51.4	52.0	41.2	**49.8**	47.5
MM2D3D (Ours) + PL	**74.3**	**68.3**	**74.9**	**71.3**	69.6	**72.2**	**55.4**	**55.0**	59.7	**46.4**	48.7	**50.7**
MM2D3D (Ours) + Fusion	x	x	74.0	x	x	71.0	x	x	**60.4**	x	x	48.8

TABLE 11.4. **Self-training Analysis.** Results with different self-training strategies in the UDA scenario.

target data to counteract this problem, in the generalization scenario, it influences badly the 3D performance. However, we note that our final Avg prediction still outperforms xMUDA.

11.3.4 Ablation Studies

Modality-wise analysis. In Tab. 11.3, we ablate our contributions starting from the model proposed by [31] in the UDA scenario. We start by activating our depth-based network, introduced in Sec. 11.2.3. The performance boost given by our proposal is remarkable across all settings. In cases such as Day → Night, where the RGB gap is larger, the depth cues injected with skip connections to the semantic decoder greatly enhance performances in the target domain (+15.8% for 2D and +5.4% in "Avg). We note also a consistent improvement for the remaining settings, in particular, we highlight a +10.5% for the 2D scores on the challenging synthetic-to-real adaptation benchmark (v. KITTI → Sem. KITTI). Furthermore, when feeding RGB colors to the 3D network (last row of Tab. 11.3), we observe improved performances in almost all settings. The largest improvement is observed in the synthetic-to-real setting, where we achieve a +10% in terms of mIou for the 3D, which in turn increased the average score from 53.7% to 56.5%. Better performance is also achieved for both the 3D network and the average score for A2D2 → Sem. KITTI.

Self-Training. In this section, we compare different self-training strategies and report results in Tab. 11.4. As explained in Sec. 11.2.4, for the self-training protocol we first need a model trained on the source domain to produce the pseudo-labels for the target domain in the second round. We report in the first row of Tab. 11.4 the performance of this starting model to better appreciate the effectiveness of self-training. First, we note how thanks to our contributions, for USA → Singapore, Day → Night, and v. KITTI → Sem. KITTI we already surpass xMUDA[31] on the *Avg* column even without the usage of pseudo-labels. When pseudo-labels from the 2D and the 3D branches are used to supervise the 2D and the 3D network respectively, we establish new state-of-the-art performances for all four settings in the average predictions (third row). Furthermore, in the fourth row of Tab. 11.4, we deploy the strategy proposed in [31], where point-wise features from the two networks are concatenated and used to train a unique classifier). In this case, we observe mixed results, indicating that this self-training strategy is not necessarily better across all settings when compared to the standard self-training protocol.

11.4 Conclusions

We shed some light on the complementarity of recent and emerging 3D-2D architectures for 3D semantic segmentation. We provide an intuitive explanation based on the notion of the effective receptive field of why processing data with these two networks grants orthogonal predictions that can be effectively fused together. Based on this, we propose to feed both modalities to both branches. Despite the simplicity of our approach, we establish new state-of-the-art results in four common UDA scenarios and demonstrate superior generalization performance over the baseline 2D-3D architecture. A limitation of our work is that our method is purely multi-modal, and it requires both modalities and a valid calibration across sensors at test time. An interesting future direction is to investigate how our approach may generalize to other multi-modal 2D-3D architectures for semantic segmentation.

Part IV

Neural Fields for 3D data

Chapter 12

Initial Remarks

In accordance with the introductory objectives of this thesis, our primary focus centers on Transfer Learning, with a specific emphasis on Unsupervised Domain Adaptation. However, we aspire to enhance the breadth of this work by incorporating supplementary chapters that delve into seminal studies on Neural Fields. These chapters, although somewhat far from the core theme of Transfer Learning, are included to provide the reader with a comprehensive understanding of the potential intersections between Neural Fields and our primary research focus. Recognizing that Neural Fields represent a relatively recent development within the vision community, especially for 3D applications, we acknowledge that the direct link between Neural Fields and transfer learning may appear somewhat loose. Nevertheless, given the escalating interest in this field, we are confident that the value of Neural Fields will become increasingly evident over time also for other topics such as Transfer Learning.

In Chapter 13, we begin with a thorough introduction to the general concept of Implicit Neural Representation (INR), which is a specific parametrization of a Neural field that uses a single Multi-Layer-Perceptron (MLP) to represent a field. Subsequently, we introduce a novel framework for representation learning for the processing of INRs of 3D shapes. Finally, in Chapter 14, we propose to use a different parametrization and a new approach to process Neural Fields. In this chapter, we also showcase preliminary experiments within the realm of Transfer Learning, specifically emphasizing the transference of knowledge across modalities, such as from point clouds to meshes.

12.1 Related Works

12.1.1 Deep learning on 3D shapes.

Due to their regular structure, voxel grids have always been appealing representations for 3D shapes and several works proposed to use 3D convolutions to perform both discriminative [258, 259, 260] and generative tasks [261, 262, 263, 264, 265, 182]. The huge memory requirements of voxel-based representations, though, led researchers to look for less demanding alternatives, such as point clouds. Processing point clouds, however, is far from straightforward because of their unorganized nature. As a possible solution, some works projected the original point clouds to intermediate regular grid structures such as voxels [266] or images [267, 268]. Alternatively, PointNet [160] proposed to operate directly on raw coordinates by means of shared multi-layer perceptrons followed by max pooling to aggregate point features. PointNet++ [161] extended PointNet with a hierarchical feature learning paradigm to capture the local geometric structures. Following PointNet++, many works focused on designing new local aggregation operators [162], resulting in a wide spectrum of specialized methods based on convolution [166, 269, 270, 271, 272, 273, 168], graph [274, 275, 167], and attention [276, 277] operators. Yet another completely unrelated set of deep learning methods have been developed to process surfaces represented as meshes, which differ in the way they exploit vertices, edges and faces as input data [278]. *Vertex-based* approaches leverage the availability of a regular domain to encode the knowledge about points neighborhoods through convolution or kernel functions [279, 280, 281, 282, 283, 284, 285, 286]. *Edge-based* methods take advantages of these connections to define an ordering invariant convolution [287], to construct a graph on the input meshes [288] or to navigate the shape structure [289]. Finally, *Face-based* works extract information from neighboring faces [290, 291, 292, 293]. In Chapter 13, we explore INRs as a unified representation for 3D shapes and propose a framework that enables the use of the same standard deep learning machinery to process them, independently of the INRs underlying signal. Next, in Chapter 14, we go beyond the concept of INR and use a different parameterization for Neural Fields that enables easier processing and consequently leads to better performance.

12.1.2 Neural fields.

Recent approaches have shown the ability of MLPs to parameterize fields representing any physical quantity of interest [294]. The works focusing on representing 3D data with MLPs rely on fitting functions such as the unsigned distance [295], the signed distance [296, 297, 298, 299, 300], the occupancy [301, 302], or the scene radiance [303]. Among these approaches, SIREN [304] uses periodic activation functions to capture high-frequency details. In this thesis, we focus on 3D data represented as voxel grids, meshes and point clouds and we adopt SIREN in Chapter 13 to demonstrate for the first time how downstream tasks can be solved starting from this representation. More recently however, hybrid representations, in which the MLP is paired with a discrete data structure, have been introduced within the vision and graphic communities motivated by faster inference [305], better use of network capacity [306] and suitability to editing tasks [307]. These data structures decompose the input coordinate space, either regularly, such as for voxel grids [305, 308, 307], tri-planes [309, 310, 311, 312], and 4D tensors [313], or irregularly, such as for point clouds [314], and meshes [315]. Following the success of this parameterization of Neural Fields, in Chapter 14 we investigate how to directly process hybrid fields to solve tasks such as shape classification and 3D part segmentation and even NeRFs classification. We focus on tri-planes due to their regular grid structure and compactness, which enable standard neural networks to process them seamlessly and effectively.

Neural functionals. Several very recent approaches aim at processing functions parameterized as MLPs by employing other neural networks. MLPs are known to exhibit weight space symmetries [316], *i.e.*, hidden neurons can be permuted across layers without changing the function represented by the network. Works such as DWSNet [317], NFN [9], and NFT [318] leverage weight space symmetries as an inductive bias to develop novel architectures designed to process MLPs. Both DWSNet and NFN devise neural layers equivariant to the permutations arising in MLPs. In contrast, NFT builds upon the intuition of achieving permutation equivariance by removing positional encoding from a Transformer architecture. The framework proposed in Functa [319] relies on learning priors on the whole dataset with a shared network and then encoding each sample in a compact embedding. In this case, each neural field is parameterized by the shared network plus the embedding. In particular, Functa [319] leverages meta-learning techniques to learn the shared network, which is modulated with latent vectors to represent each data point. These vectors are then used to address both discriminative and generative tasks. It is worth pointing out that, though not originally proposed as a framework to process neural fields, DeepSDF [296] learns dataset priors by optimizing a reconstruction objective through a shared auto-decoder network conditioned on a shape-specific embedding. Thus, the embeddings learned by DeepSDF may be used for neural processing tasks similar to Functa's. All these previous works emerged after we developed inr2vec which we will introduce in the next chapter. Therefore, in Chapter 14 we introduce a thorough benchmark assessing all the pros and cons of each methodology.

Chapter 13

Deep Learning on Implicit Neural Representations of Shapes

13.1 Introduction

Since the early days of computer vision, researchers have been processing images stored as two-dimensional grids of pixels carrying intensity or color measurements. But the world that surrounds us is three-dimensional, motivating researchers to try to process also 3D data sensed from surfaces. Unfortunately, the representation of 3D surfaces in computers does not enjoy the same uniformity as digital images, with a variety of discrete representations, such as voxel grids, point clouds and meshes, coexisting today. Besides, when it comes to processing by deep neural networks, all these kinds of representations are affected by peculiar shortcomings, requiring complex ad-hoc machinery [161, 167, 278] and/or large memory resources [258]. Hence, no standard way to store and process 3D surfaces has yet emerged.

Recently, a new kind of representation has been proposed, which leverages on the possibility of deploying a Multi-Layer Perceptron (MLP) to fit a continuous function that represents *implicitly* a signal of interest [294]. These representations, usually referred to as Implicit Neural Representations (INRs), have been proven capable of encoding effectively 3D shapes by fitting *signed distance functions (sdf)* [296, 320, 297], *unsigned distance functions (udf)* [295] and *occupancy fields (occ)* [301, 300]. Encoding a 3D shape with a continuous function parameterized as an MLP decouples the memory cost of the representation from the actual spatial resolution, *i.e.*, a surface with arbitrarily fine resolution can be reconstructed from a fixed number of parameters. Moreover, the same neural network architecture can be used to fit different implicit functions, holding the potential to provide a unified framework to represent 3D shapes.

Due to their effectiveness and potential advantages over traditional representations, INRs are gathering ever-increasing attention from the scientific community, with novel and striking results published more and more frequently [321, 322, 320, 323]. This lead us to conjecture that, in the forthcoming future, INRs might emerge as a standard representation to store and

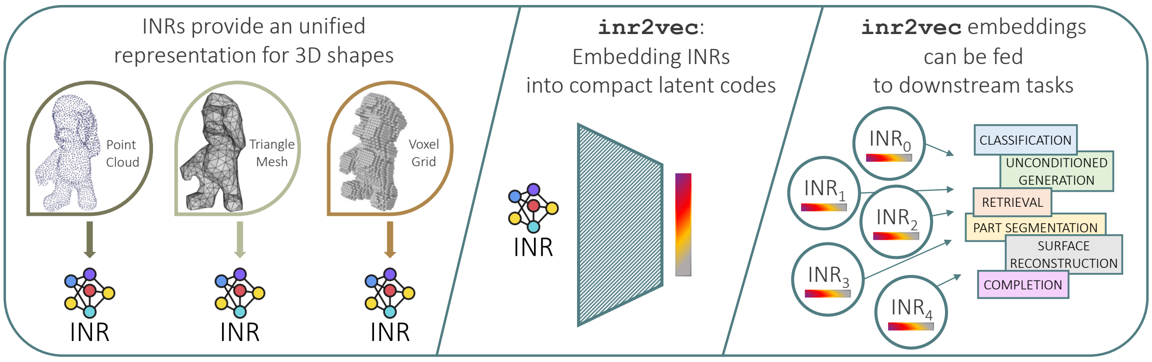


FIGURE 13.1. **Overview of our framework. Left**: INRs hold the potential to provide an unified representation for 3D shapes. **Center**: Our framework, dubbed inr2vec, produces a compact representation for an input INR by looking only at its weights. **Right**: inr2vec embeddings can be used with standard deep learning machinery to solve a variety of downstream tasks.

communicate 3D shapes, with repositories hosting digital twins of 3D objects realized only as MLPs becoming commonly available.

An intriguing research question does arise from the above scenario: beyond storage and communication, would it be possible to *process* directly INRs of 3D shapes with deep learning pipelines to solve downstream tasks as it is routinely done today with discrete representations like point clouds or meshes? In other words, would it be possible to process an INR of a 3D shape to solve a downstream task, *e.g.*, shape classification, without reconstructing a discrete representation of the surface?

Since INRs are neural networks, there is no straightforward way to process them. Earlier work in the field, namely OccupancyNetworks [301] and DeepSDF [296], fit the whole dataset with a shared network conditioned on a different embedding for each shape. In such formulation, the natural solution to the above mentioned research problem could be to use such embeddings as representations of the shapes in downstream tasks. This is indeed the approach followed by contemporary work [319], which addresses such research problem by using as embedding a latent modulation vector applied to a shared base network. However, representing a whole dataset by a shared network sets forth a difficult learning task, with the network struggling in fitting accurately the totality of the samples (as we show in Appendix A.1). Conversely, several recent works, like SIREN [304] and others [324, 325, 326, 327, 328] have shown that, by fitting an individual network to each input sample, one can get high-quality reconstructions even when dealing with very complex 3D shapes or images. Moreover, constructing an individual INR for each shape is easier to deploy in the wild, as the availability of the whole dataset is not required to fit an individual shape. Such works are gaining ever-increasing popularity and we are led to believe that fitting an individual network is likely to become the common practice in learning INRs.

Thus, we investigate how to perform downstream tasks with deep learning pipelines on shapes represented as individual INRs. However, a single INR can easily count hundreds of thousands of parameters, though it is well known that the weights of a deep model provide a vastly redundant parametrization of the underlying function [329, 330]. Hence, we settle on investigating whether and how an answer to the above research question may be provided by a representation learning framework that learns to squeeze individual INRs into compact and meaningful embeddings amenable to pursuing a variety of downstream tasks.

Our framework, dubbed inr2vec and shown in Fig. 13.1, has at its core an encoder designed to produce a task-agnostic embedding representing the input INR by processing only the INR weights. These embeddings can be seamlessly used in downstream deep learning pipelines, as we validate experimentally for a variety of tasks, like classification, retrieval, part segmentation, unconditioned generation, surface reconstruction and completion. Interestingly, since embeddings obtained from INRs live in low-dimensional vector spaces regardless of the underlying implicit function, the last two tasks can be solved by learning a simple mapping between the embeddings produced with our framework, *e.g.*, by transforming the INR of a udf into the INR of an sdf. Moreover, inr2vec can learn a smooth latent space conducive to interpolating INRs representing unseen 3D objects. Additional details and code can be found at `https://cvlab-unibo.github.io/inr2vec`. Our contributions can be summarised as follows:

- we propose and investigate the novel research problem of applying deep learning directly on individual INRs representing 3D shapes;
- to address the above problem, we introduce inr2vec, a framework that can be used to obtain a meaningful compact representation of an input INR by processing only its weights, without sampling the underlying implicit function;
- we show that a variety of tasks, usually addressed with representation-specific and complex frameworks, can indeed be performed by deploying simple deep learning machinery on INRs embedded by inr2vec, the same machinery regardless of the INRs underlying signal.

13.2 Learning to Represent INRs

The research question we address with this work is whether and how can we process directly INRs to perform downstream tasks. For instance, can we classify a 3D shape that is implicitly encoded in an INR? And how? As anticipated in Sec. 13.1, we propose to rely on a representation learning framework to squeeze the redundant information contained in the weights of INRs into compact latent codes that could be conveniently processed with standard deep learning pipelines.

Our framework, dubbed inr2vec, is composed of an encoder and a decoder. The encoder, detailed in Fig. 13.2, is designed to take as input the weights of an INR and produce a compact

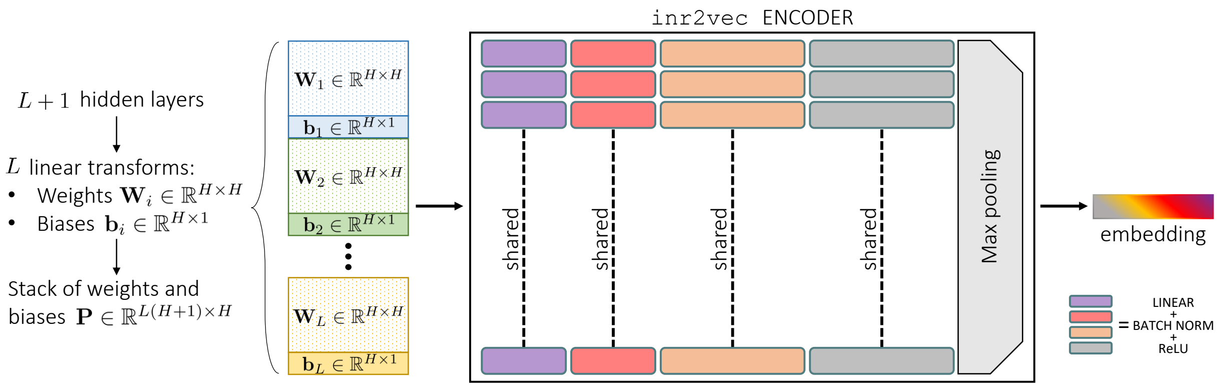

$L+1$ hidden layers
L linear transforms:
• Weights $\mathbf{W}_i \in \mathbb{R}^{H\times H}$
• Biases $\mathbf{b}_i \in \mathbb{R}^{H\times 1}$
Stack of weights and biases $\mathbf{P} \in \mathbb{R}^{L(H+1)\times H}$
$\mathbf{W}_1 \in \mathbb{R}^{H\times H}$
$\mathbf{b}_1 \in \mathbb{R}^{H\times 1}$
$\mathbf{W}_2 \in \mathbb{R}^{H\times H}$
$\mathbf{b}_2 \in \mathbb{R}^{H\times 1}$
$\mathbf{W}_L \in \mathbb{R}^{H\times H}$
$\mathbf{b}_L \in \mathbb{R}^{H\times 1}$
inr2vec ENCODER
shared
shared
shared
shared
Max pooling
embedding
= LINEAR + BATCH NORM + ReLU

FIGURE 13.2. **inr2vec encoder architecture**.

embedding that encodes all the relevant information of the input INR. A first challenge in designing an encoder for INRs consists in defining how the encoder should ingest the weights as input, since processing naively all the weights would require a huge amount of memory (see Appendix A.5). Following standard practice [304, 324, 325, 326, 327], we consider INRs composed of several hidden layers, each one with H nodes, *i.e.*, the linear transformation between two consecutive layers is parameterized by a matrix of weights $\mathbf{W}_i \in \mathbb{R}^{H\times H}$ and a vector of biases $\mathbf{b}_i \in \mathbb{R}^{H\times 1}$. Thus, stacking $\mathbf{W}_i$ and $\mathbf{b}_i{}^T$, the mapping between two consecutive layers can be represented by a single matrix $\mathbf{P}_i \in \mathbb{R}^{(H+1)\times H}$. For an INR composed of $L+1$ hidden layers, we consider the L linear transformations between them. Hence, stacking all the L matrices $\mathbf{P}_i \in \mathbb{R}^{(H+1)\times H}, i = 1, \dots, L$, between the hidden layers we obtain a single matrix $\mathbf{P} \in \mathbb{R}^{L(H+1)\times H}$, that we use to represent the INR in input to inr2vec encoder. We discard the input and output layers in our formulation as they feature different dimensionality and their use does not change inr2vec performance, as shown in Appendix A.9.

The inr2vec encoder is designed with a simple architecture, consisting of a series of linear layers with batch norm and ReLU non-linearity followed by final max pooling. At each stage, the input matrix is transformed by one linear layer, that applies the same weights to each row of the matrix. The final max pooling compresses all the rows into a single one, obtaining the desired embedding. It is worth observing that the randomness involved in fitting an individual INR (weights initialization, data shuffling, etc.) causes the weights in the same position in the INR architecture not to share the same role across INRs. Thus, inr2vec encoder would have to deal with input vectors whose elements capture different information across the different data samples, making it impossible to train the framework. However, the use of a shared, pre-computed initialization has been advocated as a good practice when fitting INRs, *e.g.*, to reduce training time by means of meta-learned initialization vectors, as done in MetaSDF [324] and in the contemporary work exploring processing of INRs [319], or to obtain desirable geometric properties [297]. We empirically found that following such a practice, *i.e.*, initializing all INRs with the same random vector, favors the alignment of weights across INRs and enables the convergence of our framework. In order to guide the

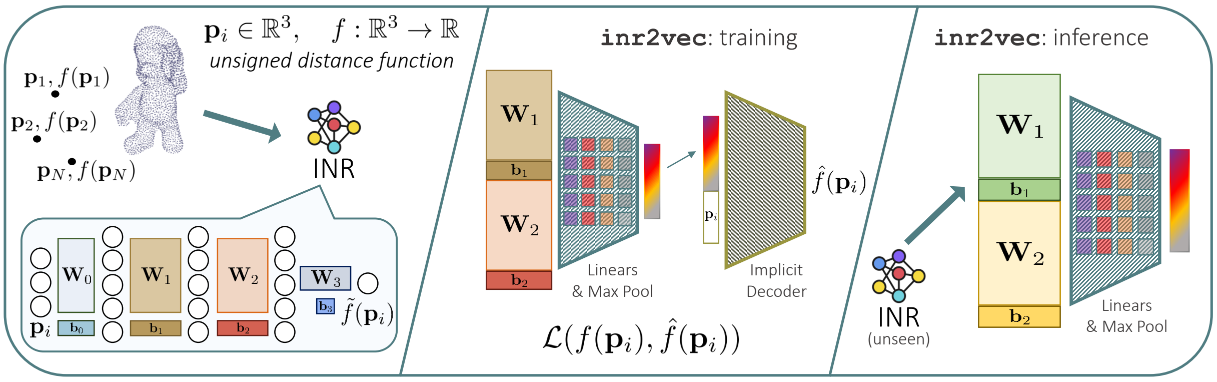
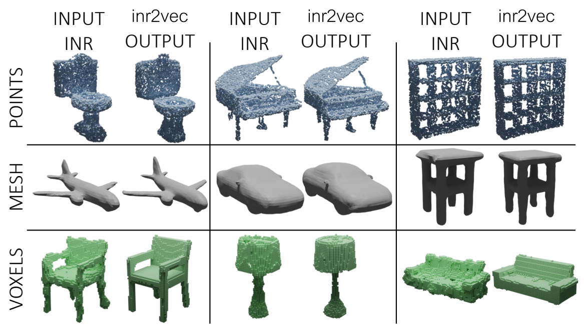

$\mathbf{p}_i \in \mathbb{R}^3, \quad f: \mathbb{R}^3 \to \mathbb{R}$
unsigned distance function
$\mathbf{p}_1, f(\mathbf{p}_1)$
$\mathbf{p}_2, f(\mathbf{p}_2)$
$\mathbf{p}_N, f(\mathbf{p}_N)$
INR
$\mathbf{W}_0$ $\mathbf{W}_1$ $\mathbf{W}_2$ $\mathbf{W}_3$
$\mathbf{b}_0$ $\mathbf{b}_1$ $\mathbf{b}_2$ $\mathbf{b}_3$
$\mathbf{p}_i$ $\tilde{f}(\mathbf{p}_i)$
inr2vec: training
Linears & Max Pool
Implicit Decoder
$\hat{f}(\mathbf{p}_i)$
$\mathcal{L}(f(\mathbf{p}_i), \hat{f}(\mathbf{p}_i))$
inr2vec: inference
INR (unseen)
Linears & Max Pool

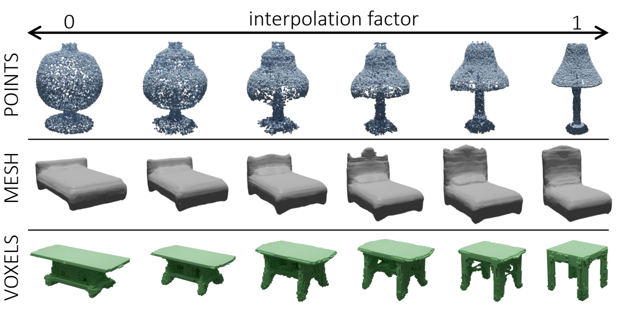
FIGURE 13.3. **Training and inference of our framework. Left:** We consider shapes represented as INRs. As an example, we show an INR fitting the *udf* of a surface. **Center:** inr2vec encoder is trained together with an implicit decoder to replicate the underlying 3D signal of the input INR. **Right:** At inference time, the learned encoder can be used to obtain a compact embedding from unseen INRs.

inr2vec encoder to produce meaningful embeddings, we first note that we are not interested in encoding the values of the input weights in the embeddings produced by our framework, but, rather, in storing information about the 3D shape represented by the input INR. For this reason, we supervise the decoder to replicate the function approximated by the input INR instead of directly reproducing its weights, as it would be the case in a standard auto-encoder formulation. In particular, during training, we adopt an implicit decoder inspired by [296], which takes in input the embeddings produced by the encoder and decodes the input INRs from them (see Fig. 13.3 center). More specifically, when the inr2vec encoder processes a given INR, we use the underlying signal to create a set of 3D queries $\mathbf{p}_i$, paired with the values $f(\mathbf{p}_i)$ of the function approximated by the input INR at those locations (the type of function depends on the underlying signal modality, it can be *udf* in case of point clouds, *sdf* in case of triangle meshes or *occ* in case of voxel grids). The decoder takes in input the embedding produced by the encoder concatenated with the 3D coordinates of a query $\mathbf{p}_i$ and the whole encoder-decoder is supervised to regress the value $f(\mathbf{p}_i)$. After the overall framework has been trained end to end, the frozen encoder can be used to compute embeddings of unseen INRs with a single forward pass (see Fig. 13.3 right) while the implicit decoder can be used, if needed, to reconstruct the discrete representation given an embedding.

In Fig. 13.4a we compare 3D shapes reconstructed from INRs unseen during training with those reconstructed by the inr2vec decoder starting from the latent codes yielded by the encoder. We visualize point clouds with 8192 points, meshes reconstructed by marching cubes [331] from a grid with resolution 128^3 and voxels with resolution 64^3. We note that, though our embedding is dramatically more compact than the original INR, the reconstructed shape resembles the ground-truth with a good level of detail. Moreover, in Fig. 13.4b we linearly interpolate between the embeddings produced by inr2vec from two input shapes and show the shapes reconstructed from the interpolated embeddings. Results highlight that the latent



(A) inr2vec reconstructions.



(B) inr2vec interpolations.

FIGURE 13.4. **Properties of inr2vec latent space.**

space learned by inr2vec enables smooth interpolations between shapes represented as INRs. *Additional details on inr2vec training and the procedure to reconstruct the discrete representations from the decoder are in the Appendices.*

13.3 Deep Learning on INRs

In this section, we first present the set-up of our experiments. Then, we show how several tasks dealing with 3D shapes can be tackled by working only with inr2vec embeddings as input and/or output. *Additional details on the architectures and on the experimental settings are in Appendix A.6*.

General settings. In all the experiments reported in this section, we convert 3D discrete representations into INRs featuring 4 hidden layers with 512 nodes each, using the SIREN activation function [304].

We train inr2vec using an encoder composed of four linear layers with respectively 512, 512, 1024 and 1024 features, embeddings with 1024 values and an implicit decoder with 5 hidden layers with 512 features. In all the experiments, the baselines are trained using standard data augmentation (random scaling and point-wise jittering), while we train both inr2vec and the downstream task-specific networks on datasets augmented offline with the same transformations.

Point cloud retrieval. We first evaluate the feasibility of using inr2vec embeddings of INRs to solve tasks usually tackled by representation learning, and we select 3D retrieval as a benchmark. We follow the procedure introduced in [171], using the Euclidean distance to measure the similarity between embeddings of unseen point clouds from the test sets of ModelNet40 [182] and ShapeNet10 (a subset of 10 classes of the popular ShapeNet dataset [171]). For each embedded shape, we select its k-nearest-neighbours and compute a Precision Score comparing the classes of the query and the retrieved shapes, reporting the mean Average Precision for different k (mAP@k). Besides inr2vec, we consider three baselines to embed point clouds, which are obtained by training the PointNet [160], PointNet++ [161]

	ModelNet40			ShapeNet10			ScanNet10		
Method	mAP@1	mAP@5	mAP@10	mAP@1	mAP@5	mAP@10	mAP@1	mAP@5	mAP@10
PointNet [160]	80.1	91.7	94.4	90.6	96.6	98.1	65.7	86.2	92.6
PointNet++ [161]	85.1	93.9	96.0	92.2	97.5	98.6	71.6	89.3	93.7
DGCNN [167]	83.2	92.7	95.1	91.0	96.7	98.2	66.1	88.0	93.1
inr2vec	81.7	92.6	95.1	90.6	96.7	98.1	65.2	87.5	94.0

TABLE 13.1. **Point cloud retrieval quantitative results.**

	Point Cloud			Mesh	Voxels
Method	ModelNet40	ShapeNet10	ScanNet10	Manifold40	ShapeNet10
PointNet [160]	88.8	94.3	72.7	–	–
PointNet++ [161]	89.7	94.6	76.4	–	–
DGCNN [167]	89.9	94.3	76.2	–	–
MeshWalker [289]	–	–	–	90.0	–
Conv3DNet [258]	–	–	–	–	92.1
inr2vec	87.0	93.3	72.1	86.3	93.0

TABLE 13.2. **Results on shape classification across representations.**

and DGCNN [167] encoders in combination with a fully connected decoder similar to that proposed in [332] to reconstruct the input cloud. Quantitative results, reported in Tab. 13.1, show that, while there is an average gap of 1.8 mAP with PointNet++, inr2vec is able to match, and in some cases even surpass, the performance of the other baselines. Moreover, it is possible to appreciate in Fig. 13.5 that the retrieved shapes not only belong to the same class as the query but present also the same coarse structure. This finding highlights how the pretext task used to learn inr2vec embeddings can summarise relevant shape information effectively.

Shape classification. We then address the problem of classifying point clouds, meshes and voxel grids. For point clouds we use three datasets: ShapeNet10, ModelNet40 and ScanNet10 [174]. When dealing with meshes, we conduct our experiments on the Manifold40 dataset [278]. Finally, for voxel grids, we use again ShapeNet10, quantizing clouds to grids with resolution 64^3. Despite the different nature of the discrete representations taken into account, inr2vec allows us to perform shape classification on INRs embeddings, augmented online with E-Stitchup [333], by the very same downstream network architecture, *i.e.*, a simple fully connected classifier consisting of three layers with 1024, 512 and 128 features. We consider as baselines well-known architectures that are optimized to work on the specific input representations of each dataset. For point clouds, we consider PointNet [160], PointNet++ [161] and DGCNN [167]. For meshes, we consider a recent and competitive baseline that processes directly triangle meshes [289]. As for voxel grids, we train a 3D CNN classifier that we implemented following [258] (Conv3DNet from now on). Since only the train and test splits are released for all the datasets, we created validation splits from the training sets in order to follow a proper train/val protocol for both the baselines and inr2vec. As for the test shapes, we evaluated all the baselines on the discrete representations reconstructed from the INRs fitted on the original test sets, as these would be the only data available at test time in a scenario where INRs are used to store and communicate 3D data. We report the results of

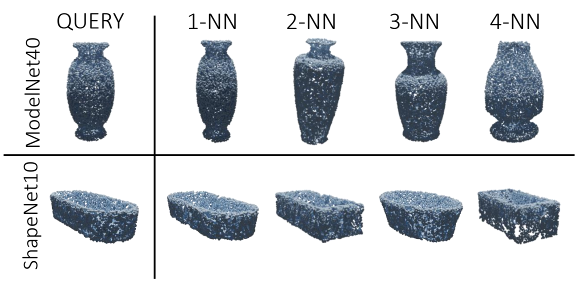


FIGURE 13.5. **Point cloud retrieval qualitative results.** Given the inr2vec embedding of a query shape, we show the shapes reconstructed from the closest embeddings (L2 distance).

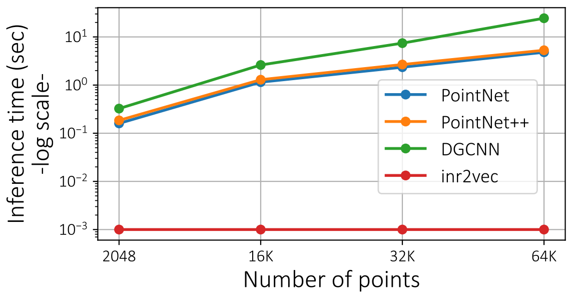


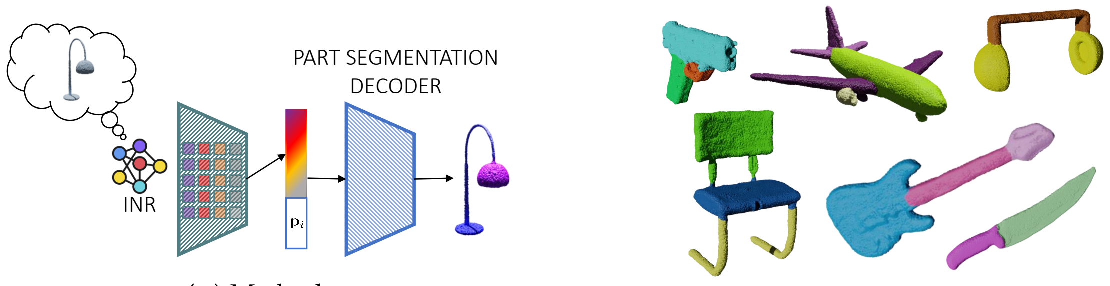
FIGURE 13.6. **Time required to classify INRs encoding udf.** For point cloud classifiers, the time to reconstruct the discrete point cloud from the INR is included in the chart.

the baselines tested on the original discrete representations available in the original datasets in Appendix A.8: they are in line with those provided here. The results in Tab. 13.2 show that inr2vec embeddings deliver classification accuracy close to the specialized baselines across all the considered datasets, regardless of the original discrete representation of the shapes in each dataset. Remarkably, our framework allows us to apply the same simple classification architecture on all the considered input modalities, in stark contrast with all the baselines that are highly specialized for each modality, exploit inductive biases specific to each such modality and cannot be deployed on representations different from those they were designed for. Furthermore, while presenting a gap of some accuracy points with respect to the most recent architectures, like DGCNN and MeshWalker, the simple fully connected classifier that we applied on inr2vec embeddings obtains scores comparable to standard baselines like PointNet and Conv3DNet. We also highlight that, should 3D shapes be stored as INRs, classifying them with the considered specialized baselines would require recovering the original discrete representations by the lengthy procedures described in Appendix A.3. Thus, in Fig. 13.6, we report the inference time of standard point cloud classification networks while including also the time needed to reconstruct the discrete point cloud from the input INR of the underlying *udf* at different resolutions. Even at the coarsest resolution (2048 points), all the baselines yield an inference time which is one order of magnitude higher than that required to classify directly the inr2vec embeddings. Increasing the resolution of the reconstructed clouds makes the inference time of the baselines prohibitive, while inr2vec, not requiring the explicit clouds, delivers a constant inference time of 0.001 seconds.

Point cloud part segmentation. While the tasks of classification and retrieval concern the possibility of using inr2vec embeddings as a compact proxy for the global information of the input shapes, with the task of point cloud part segmentation we aim at investigating whether inr2vec embeddings can be used also to assess upon local properties of shapes. The part segmentation task consists in predicting a semantic (*i.e.*, part) label for each point of

Method	instance mIoU	class mIoU	airplane	bag	cap	car	chair	earphone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skateboard	table
PointNet [160]	83.1	78.96	81.3	76.9	79.6	71.4	89.4	67.0	91.2	80.5	80.0	95.1	66.3	91.3	80.6	57.8	73.6	81.5
PointNet++ [161]	84.9	82.73	82.2	88.8	84.0	76.0	90.4	80.6	91.8	84.9	84.4	94.9	72.2	94.7	81.3	61.1	74.1	82.3
DGCNN [167]	83.6	80.86	80.7	84.3	82.8	74.8	89.0	81.2	90.1	86.4	84.0	95.4	59.3	92.8	77.8	62.5	71.6	81.1
inr2vec	81.3	76.91	80.2	76.2	70.3	70.1	88.0	65.0	90.6	82.1	77.4	94.4	61.4	92.7	79.0	56.2	68.6	78.5

TABLE 13.3. **Part segmentation quantitative results.** We report the IoU for each class, the mean IoU over all the classes (class mIoU) and the mean IoU over all the instances (instance mIoU).



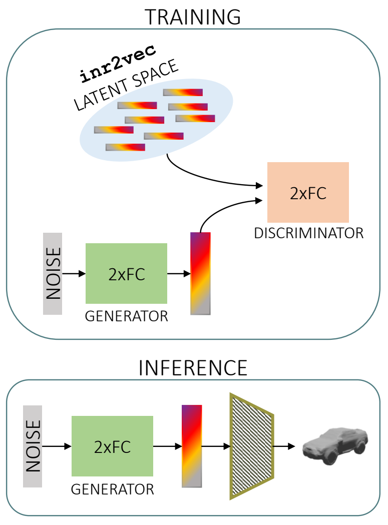
(A) Method.

(B) Qualitative results.

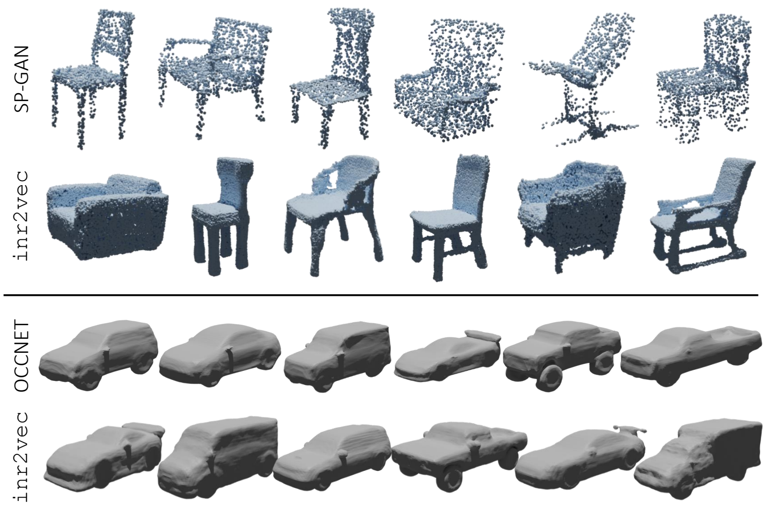
FIGURE 13.7. **Point cloud part segmentation.**

a given cloud. We tackle this problem by training a decoder similar to that used to train our framework (see Fig. 13.7a). Such decoder is fed with the inr2vec embedding of the INR representing the input cloud, concatenated with the coordinate of a 3D query, and it is trained to predict the label of the query point. We train it, as well as PointNet, PointNet++ and DGCNN, on the ShapeNet Part Segmentation dataset [334] with point clouds of 2048 points, with the same train/val/test of the classification task. Quantitative results reported in Tab. 13.3 show the possibility of performing also a local discriminative task as challenging as part segmentation based on the task-agnostic embeddings produced by inr2vec and, in so doing, to reach performance not far from that of specialized architectures. Additionally, in Fig. 13.7b we show point clouds reconstructed at 100K points from the input INRs and segmented with high precision thanks to our formulation based on a semantic decoder conditioned by the inr2vec embedding.

Shape generation. With the experiments reported above we validated that, thanks to inr2vec embeddings, INRs can be used as input in standard deep learning machinery. In this section, we address instead the task of shape generation in an adversarial setting to investigate whether the compact representations produced by our framework can be adopted also as medium for the output of deep learning pipelines. For this purpose, as depicted in Fig. 13.8a, we train a Latent-GAN [335] to generate embeddings indistinguishable from those produced by inr2vec starting from random noise. The generated embeddings can then be decoded into discrete representations with the implicit decoder exploited during inr2vec training. Since our framework is agnostic with respect to the original discrete representation of shapes used to fit INRs, we can train Latent-GANs with embeddings representing point



(A) Method.

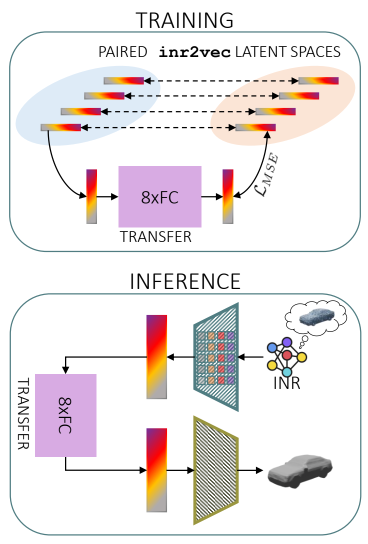


(B) Qualitative results.

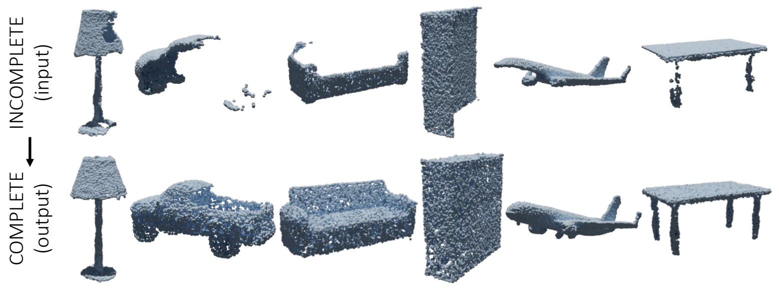
FIGURE 13.8. **Learning to generate shapes from inr2vec latent space.**

clouds or meshes based on the same identical protocol and architecture (two simple fully connected networks as generator and discriminator). For point clouds, we train one Latent-GAN on each class of ShapeNet10, while we use models of cars provided by [301] when dealing with meshes. In Fig. 13.8b, we show some samples generated with the described procedure, comparing them with SP-GAN [336] on the *chair* class for what concerns point clouds and Occupancy Networks [301] (VAE formulation) for meshes. Generated examples of other classes for point clouds are shown in Appendix A.10. The shapes generated with our Latent-GAN trained only on inr2vec embeddings seem comparable to those produced by the considered baselines, in terms of both diversity and richness of details. Additionally, by generating embeddings that represent implicit functions, our method enables sampling point clouds at any arbitrary resolution (*e.g.*, 8192 points in Fig. 13.8b) whilst SP-GAN would require a new training for each desired resolution since the number of generated points must be set at training time.

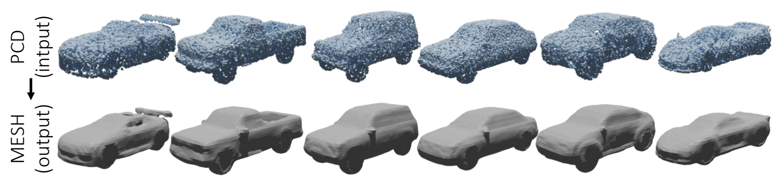
Learning a mapping between inr2vec embedding spaces. We have shown that inr2vec embeddings can be used as a proxy to feed INRs in input to deep learning pipelines, and that they can also be obtained as output of generative frameworks. In this section we move a step further, considering the possibility of learning a mapping between two distinct latent spaces produced by our framework for two separate datasets of INRs, based on a *transfer* function designed to operate on inr2vec embeddings as both input and output data. Such transfer function can be realized by a simple fully connected network that maps the input embedding into the output one and is trained by a standard MSE loss (see Fig. 13.9a). As inr2vec generates compact embeddings of the same dimension regardless of the input INR modality, the transfer function described here can be applied seamlessly to a great variety of tasks, usually tackled with ad-hoc frameworks tailored to specific input/output modalities.



(A) Method.



(B) Point cloud completion.



(C) Surface reconstruction.

FIGURE 13.9. **Learning a mapping between inr2vec latent spaces.**

Here, we apply this idea to two tasks. Firstly, we address point cloud completion on the dataset presented in [337] by learning a mapping from inr2vec embeddings of INRs that represent incomplete clouds to embeddings associated with complete clouds. Then, we tackle the task of surface reconstruction on ShapeNet cars, training the transfer function to map inr2vec embeddings representing point clouds into embeddings that can be decoded into meshes. As we can appreciate from the samples in Fig. 13.9b and Fig. 13.9c, the transfer function can learn an effective mapping between inr2vec latent spaces. Indeed, by processing exclusively INRs embedding, we can obtain output shapes that are highly compatible with the input ones whilst preserving the distinctive details, like the pointy wing of the airplane in Fig. 13.9b or the flap of the first car in Fig. 13.9c.

13.4 Concluding Remarks

We have shown that it is possible to apply deep learning directly on individual INRs representing 3D shapes. Our formulation of this novel research problem leverages on a task-agnostic encoder which embeds INRs into compact and meaningful latent codes without accessing the underlying implicit function. Our framework ingests INRs obtained from different 3D discrete representations and performs various tasks through standard machinery. However, we point out two main limitations: i) Although INRs capture continuous geometric cues, inr2vec embeddings achieve results inferior to state-of-the-art solutions ii) There is no obvious way to perform online data augmentation on shapes represented as INRs by directly altering their weights. In the future, we plan to investigate these shortcomings as well as applying inr2vec to other input modalities like images, audio or radiance fields. We will also investigate weight-space symmetries [338] as a different path to favour alignment of weights across INRs despite the randomness of training. We reckon that our work may foster the adoption of INRs as a unified and standardized 3D representation, thereby overcoming the current fragmentation of discrete 3D data and associated processing architectures.

Chapter 14

Neural Processing of Tri-Plane Hybrid Neural Fields

14.1 Introduction

A world of neural fields. Neural fields [294] are functions defined at all spatial coordinates, parameterized by a neural network such as a Multi-Layer Perceptron (MLP). They have been used to represent different kinds of data, like image intensities, scene radiances, 3D shapes, etc. In the context of 3D world representation, various types of neural fields have been explored, such as the signed/unsigned distance field (*SDF*/*UDF*) [296, 297, 320], the occupancy field (*OF*) [301, 300], and the radiance field (*RF*) [339]. Their main advantage is the ability to obtain a continuous representation of the world, thereby providing information at every point in space, unlike discrete counterparts like voxels, meshes, or point clouds. Moreover, neural fields allow for encoding a 3D geometry at arbitrary resolution while using a finite number of parameters, *i.e.*, the weights of the MLP. Thus, the memory cost of the representation and its spatial resolution are decoupled.

Recently, hybrid neural fields [294], which combine continuous neural elements (*i.e.*, MLPs) with discrete spatial structures (*e.g.*,voxel grids [300], point clouds [314], etc.) that encode local information, are gaining popularity due to faster inference [305], better use of network capacity [306] and suitability to editing tasks [307]. In particular, the community has recently investigated tri-planes [310], a type of hybrid representation whose discrete components are three feature planes (xy, yz, xz), due to its regular grid structure and compactness. Tri-planes have been deployed for *RF* [312] and *SDF* [309].

Neural processing of neural fields. As conjectured in Chapter 13, due to their advantages and increasing adoption in recent years, neural fields may become one of the standard methods for storing and communicating 3D information, *i.e.*, repositories of digital twins of real objects stored as neural networks will become available. In such a scenario, developing strategies to solve tasks such as classification or segmentation by directly processing neural fields becomes relevant to utilize these representations in practical applications. For instance, given a NeRF of a chair, classifying the weights of the MLP without rendering and processing

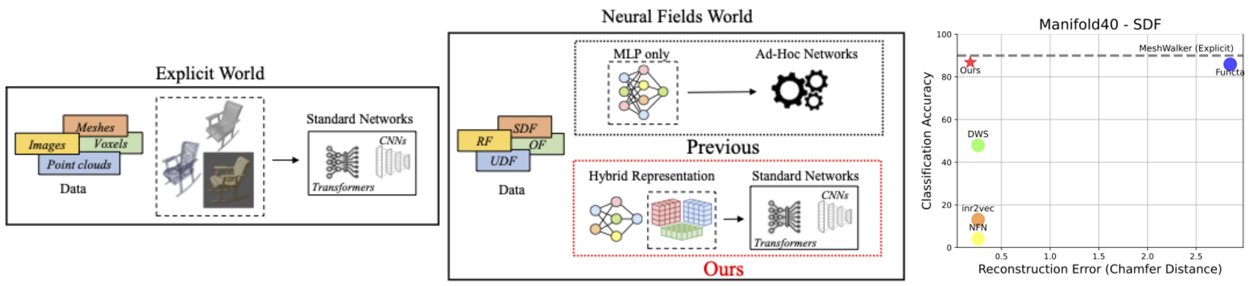


FIGURE 14.1. **Left:** Neural processing of hybrid neural fields allows us to employ well-established architectures to tackle deep learning tasks while avoiding problems related to processing MLPs, such as the high-dimensional weight space and the random initialization. **Right:** We achieve performance better than other works on this topic, close to methods that operate directly on explicit representations. without sacrificing the reconstruction quality of neural fields.

images would be faster, less computationally demanding, and more straightforward, *e.g.*,there is no need to understand where to sample the 3D space as *there is no sampling at all*.

Earlier methods on the topic, such as Functa [319], approached this scenario with shared networks trained on the whole dataset conditioned on a different global embedding for each object. In this case, a neural field is realized by the shared network plus the embedding, which is then processed for downstream tasks. However, representing a whole dataset with a shared network is difficult, and the reconstruction quality of neural fields inevitably drops (see the plot in Fig. 14.1). For this reason, later approaches such as inr2vec, NFN [9], NFT [318], and DWSNet [317] propose to process neural fields consisting of a single large MLP, such as SIREN [304], for each object. Although this strategy effectively maintains the reconstruction capabilities of neural fields, task performance suffers due to the challenges introduced by the need to handle MLPs, such as the large number of weights and the difficulty of embedding inductive biases into neural networks aimed at processing MLPs.

Moreover, randomly initialized MLPs trained on the same input data can converge to drastically different regions of the weight space due to the non-convex optimization problem and the symmetries of neural weight spaces [338, 340]. Thus, identifying a model capable of processing MLPs and generalizing among all possible initializations is not straightforward. Previous works partially address these problems: inr2vec proposes an efficient and scalable architecture, and bypasses the initialization problem by fixing it across MLPs; NFN, NFT, and DWSNet design networks that are equivariant to weight symmetries. Nonetheless, all previous methods processing neural fields realized as single MLPs achieve unsatisfying performance, far from established architectures that operate on explicit representations, *e.g.*,point clouds or meshes, as shown in Fig. 14.1 right.

Neural processing of tri-plane neural fields. To overcome the limitations of previous approaches and given the appealing properties of hybrid representations, in this paper, we explore the new research problem of tackling common 3D tasks by directly processing tri-plane neural fields. To this end, we analyze the information stored in the two components of this representation, which comprises a discrete feature space alongside a small MLP, and find out that the former contains rich semantic and geometric information. Based on this finding, we propose to process tri-plane neural fields by seamlessly applying, directly on the discrete feature space, standard neural architectures that have been developed and engineered over many years of research, such as CNNs [14] or, thanks to tri-plane compactness, even Transformers [15] (Fig. 14.1 left). Moreover, we note empirically that the same geometric structures are encoded in tri-planes fitted on the same shape from different initializations up to a permutation of the channels. Thus, we exploit this property to achieve robustness to the random initialization problem by processing tri-planes with standard architectures that are made invariant to permutation of the channels. We achieve much better performance than all previous methods in classifying and segmenting objects represented as neural fields, almost on par with established architectures that operate on explicit representations, without sacrificing the representation quality (Fig. 14.1). Finally, we observe that tri-plane neural fields for individual objects have a much better representation quality than shared network frameworks, comparable to the use of a large MLP (see Fig. 14.1 and Sec. 14.2.2).

Summary of our contributions. Code can be found at `https://github.com/CVLAB-Unibo/triplane_processing`.

• We set forth the new research problem of solving tasks by directly processing tri-plane neural fields. We show that the discrete features encode rich semantic and geometric information, which can be elaborated by applying well-established architectures. Moreover, we note how similar information is stored in tri-planes with different initializations of the same shape. Yet, the information is organized with different channel orders.

• We show that applying well-established architectures on tri-planes achieves much better results than processing neural fields realized as a large MLP. Moreover, we reveal that employing architectures made invariant to the channel order improves performance in the challenging but more realistic scenario of randomly initialized neural fields. In this way, we almost close the gap between methods that operate on explicit representations and those working directly on neural representations.

• To validate our results, we build a comprehensive benchmark for tri-plane neural field classification. We test our method by classifying neural fields that model various fields (*UDF*, *SDF*, *OF*, *RF*). In particular, to the best of our knowledge, we are the first to classify NeRFs without explicitly reconstructing the represented signal.

• Finally, as the tri-plane structure is independent of the represented field, we train a single network to classify diverse neural fields. Specifically, we show promising preliminary results of a unique model capable of classifying *UDF*, *SDF*, and *OF*, showcasing some preliminary experiments in the realm of Transfer Learning as this is the main topic of this dissertation.

14.2 Tri-plane hybrid neural fields

14.2.1 Preliminaries

Neural fields. A field is a physical quantity defined for all domain coordinates. We focus on fields describing the 3D world, and thus on $\mathbb{R}^3$ coordinates $\mathbf{p} = (x, y, z)$. We consider the 3D fields commonly used in computer vision and graphics, *i.e.*, the *SDF* [296] and *UDF* [295], which map coordinates to the signed an unsigned distance from the closest surface, respectively, the *OF* [301], which computes the occupancy probability, and the *RF* [339], that outputs (R, G, B) colors and density σ. A field can be modeled by a function, Φ, parameterized by θ. Thus, for any point $\mathbf{p}$, the field is given by $\hat{\mathbf{q}} = \Phi(\mathbf{p}; \theta)$. If parameters θ are the weights of a neural network, Φ is said to be a neural field. On the other hand, if some of the parameters are the weights of a neural network, whereas the rest encode local information within a discrete spatial structure, Φ is a hybrid neural field [294].

Tri-plane representation. A special case of hybrid neural fields, originally proposed in [310], is parameterized by a discrete tri-plane feature map, T, and a small MLP network, M (Fig. 14.2, left). T consists of three orthogonal 2D feature maps, $T = (\mathbf{F}_{xy}, \mathbf{F}_{xz}, \mathbf{F}_{yz})$, with $\mathbf{F}_{xy}, \mathbf{F}_{xz}, \mathbf{F}_{yz} \in \mathbb{R}^{C \times H \times W}$, where C is the number of channels and W, H are the spatial dimensions of the feature maps. The feature vector associated with a 3D point, $\mathbf{p}$, is computed by projecting the point onto the three orthogonal planes so to get the 2D coordinates, $\mathbf{p}_{xy}$, $\mathbf{p}_{xz}$, and $\mathbf{p}_{yz}$, relative to each plane. Then, the four feature vectors corresponding to the nearest neighbours in each plane are bi-linearly interpolated to calculate three feature vectors, $\mathbf{f}_{xy}$, $\mathbf{f}_{xz}$, and $\mathbf{f}_{yz}$, which are summed up element-wise to obtain $\mathbf{f} = \mathbf{f}_{xy} + \mathbf{f}_{xz} + \mathbf{f}_{yz}$, $\mathbf{f} \in \mathbb{R}^C$.

Finally, we concatenate $\mathbf{f}$ with a positional encoding [339], **PE**, of the 3D point $\mathbf{p}$ and feed it to the MLP, which in turn outputs the field value at $\mathbf{p}$: $\hat{\mathbf{q}} = \Phi(\mathbf{p}; \theta) = M([\mathbf{f}, \mathbf{PE}])$. We implement M with *sin* activation functions [304] to better capture high-frequency details.

Learning tri-planes. To learn a field, we optimize a (T, M) pair *for each 3D object*, starting from randomly initialized parameters, θ, for both M and T. We sample N points $\mathbf{p}_i$ and feed them to T and M to compute the corresponding field quantities $\hat{\mathbf{q}}_i = \Phi(\mathbf{p}_i; \theta)$. Then, we optimize θ with a loss, $\mathcal{L}$, capturing the discrepancy between the predicted fields $\hat{\mathbf{q}}_i$ and the ground truth $\mathbf{y}_i$, applying an optional mapping between the output and the available supervision if needed (*e.g.*,volumetric rendering in case of *RF*). An overview of this procedure is shown on the left of Fig. 14.2 and described in detail in Appendix B.1. We repeat this process for each 3D shape of a dataset, thereby creating a dataset of tri-plane hybrid neural fields (Fig. 14.2, right). We set C to 16 and both H and W to 32. We use MLPs with three hidden layers, each having 64 neurons. We note that our proposal is independent of the learning procedure, and, in a scenario in which neural fields are a standard 3D data representation, we would already have datasets available.

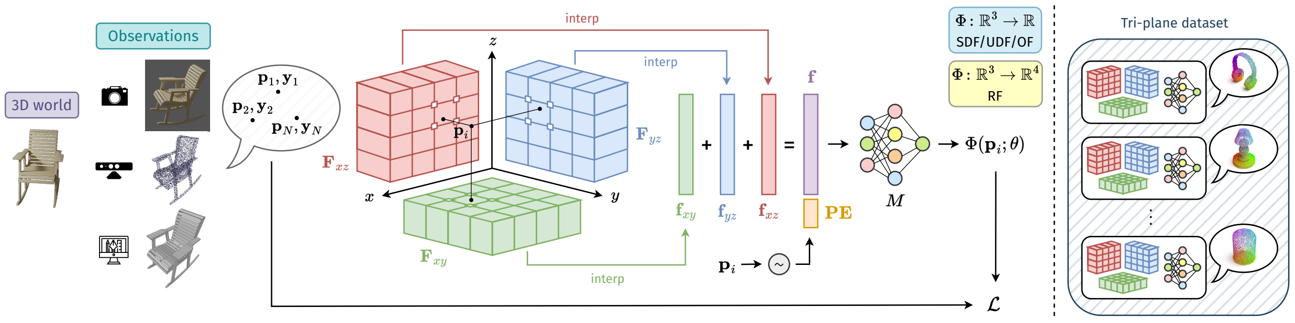

3D world
Observations
$\mathbf{p}_1, \mathbf{y}_1$
$\mathbf{p}_2, \mathbf{y}_2$
$\mathbf{p}_N, \mathbf{y}_N$
interp
interp
interp
$\mathbf{F}_{xz}$
$\mathbf{F}_{yz}$
$\mathbf{F}_{xy}$
$\mathbf{p}_i$
$\mathbf{f}_{xy}$ + $\mathbf{f}_{yz}$ + $\mathbf{f}_{xz}$ = $\mathbf{f}$
PE
M
$\Phi(\mathbf{p}_i;\theta)$
$\mathcal{L}$
$\Phi: \mathbb{R}^3 \to \mathbb{R}$ SDF/UDF/OF
$\Phi: \mathbb{R}^3 \to \mathbb{R}^4$ RF
Tri-plane dataset

FIGURE 14.2. **Left:** Tri-plane representation and learning of each neural field. **Right:** Datasets are composed of many independent tri-plane hybrid neural fields, each representing a 3D object.

14.2.2 Tri-plane analysis

We investigate here the benefits of tri-planes for 3D data representation and neural processing. Firstly, we assess their reconstruction capability, which is crucial in a world where neural fields may be used as a standard way to represent 3D assets. Secondly, we analyze the information learned in the 2D planes and how to be robust to the random initialization problem when handling tri-planes.

Reconstruction quality.

We assess the tri-plane reconstruction performance by following the benchmark introduced in inr2vec. In Tab. 14.1, we present the quantitative outcomes obtained by fitting *SDF*s and *UDF*s from meshes and point clouds of the Manifold40 dataset [278]. We compare with neural fields employed in inr2vec inr2vec and alternatives based on a shared architecture, such as DeepSDF [296] and Functa [319]. Given the *SDF* and *UDF* fields learned by each framework, we reconstruct the explicit meshes and point clouds as described in Appendix B.2.1 and evaluate them against the ground-truths. To conduct this evaluation, we sample dense point clouds of 16,384 points from both the reconstructed and ground-truth shapes. We employ the Chamfer Distance [332] and the F-Score [341] to evaluate fidelity to ground-truths. As for meshes, the tri-planes representation stands out with the lowest Chamfer Distance (CD) (0.18 mm), indicating its excellent reconstruction quality despite the relatively small number of parameters (only 64K). For point clouds, tri-planes produce reconstructions slightly worse than inr2vec but still comparable, *i.e.*, 0.21m vs 0.24mm CD. In Appendix B.2.2 (Fig. B.7), we show reconstructions attained from tri-plane representations for various types of fields. Moreover, in agreement with the findings of inr2vec, Tab. 14.1 shows that shared network frameworks such as DeepSDF and Functa yield significantly worse performance in terms of reconstruction quality. We finally point out how sharing the MLP for all tri-planes is not as effective as learning individual neural fields (third vs second row). These results support our intuition that reconstruction quality mandates hybrid neural fields optimized individually on each data sample and highlight the importance of investigating the direct neural processing of these representations. In Appendix B.2.3 (Fig. B.2, Fig. B.3),

			Mesh from *SDF*		Point Cloud from *UDF*	
Method	Type	# Params (K)	CD (mm)	F-score (%)	CD (mm)	F-score (%)
inr2vec	Single	800	0.26	69.7	0.21	65.5
Tri-plane	Single	64	0.18	68.6	0.24	60.7
Tri-plane	Shared	64	1.57	42.9	3.45	33.3
DeepSDF [296]	Shared	2400	6.6	25.1	5.6	5.7
Functa [319]	Shared	7091	2.85	21.3	12.8	5.8

TABLE 14.1. **Results of mesh and point cloud reconstruction on the Manifold40 test set.** "Single" and "Shared" indicate neural fields trained on each shape independently or on the whole dataset.

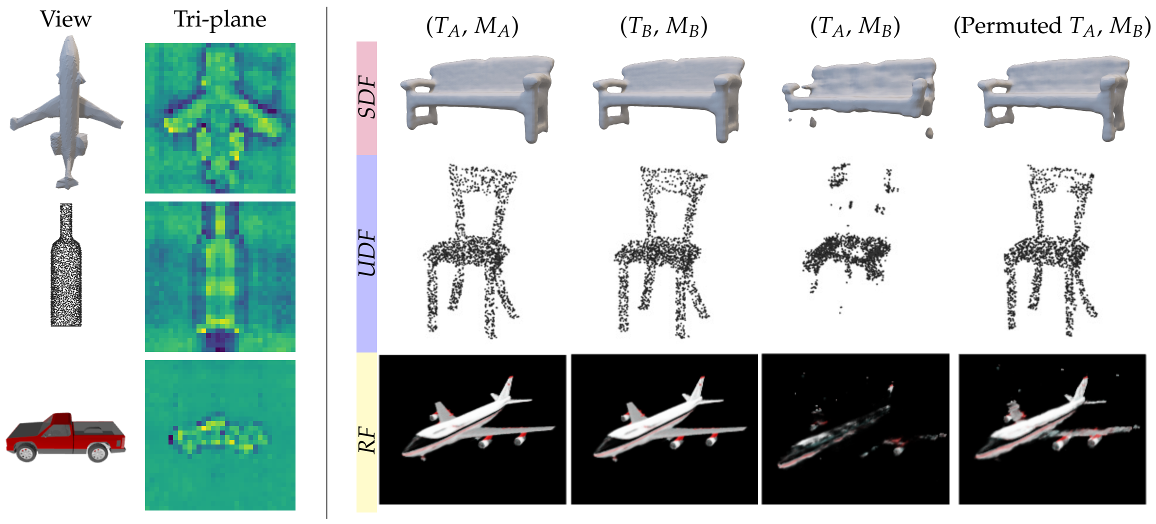


FIGURE 14.3. **Left:** For three different hybrid neural fields (from top to bottom: *SDF*, *UDF*, *RF*) we render a view of the reconstructed 3D object alongside the corresponding tri-plane feature map. **Right:** From left to right, reconstructions of two (tri-plane, MLP) pairs with different initializations, namely (T_A, M_A) and (T_B, M_B); the mixed pair (T_A, M_B); a channel permutation of T_A and M_B.

we show the reconstructions obtained by tri-planes and the other approaches considered in our evaluation.

Tri-plane content. To investigate how to directly process tri-plane neural fields, we inspected the content of their discrete spatial structure by visualizing the features stored in a plane alongside the view of the object rendered from the vantage point corresponding to the plane. Examples of these visualizations are depicted in Fig. 14.3 (left) for various objects such as a car, an airplane, and a bottle. To visualize features as a single image, displayed by a *viridis* colormap, we take a sum across the feature channels at each spatial location. These visualizations show clearly that the tri-plane spatial structure learns the object shape, *i.e.*, it contains information about its geometry. For this reason and further investigations reported in Appendix B.4, we conjecture, and demonstrate empirically in subsequent sections, that to tackle tasks such as classification and segmentation we can discard the MLPs and process only the tri-plane structure of the neural fields. Remarkably, the regular grid structure of tri-planes allows us to deploy popular and effective neural architectures, such as CNNs and

Transformers. On the contrary, direct ingestion of MLPs for neural processing is problematic and leads to sub-optimal task performance.

Random initialization. Furthermore, we investigate the effect of random initializations on tri-plane neural fields. We note here that by "random initialization" we mean that each $(\text{tri-plane}, \text{MLP})$ pair adheres to the same initialization scheme but has a different random seed (see Appendix B.5.5). We find out empirically that the main difference between tri-plane structures learned from different optimizations of the same shape lies in the channel order within a feature plane. Indeed, we conducted experiments where we fit the same 3D shape twice (see Fig. 14.3 (right)), starting from two random initializations of both the tri-plane structure and the MLP weights. Although the geometric content of the two tri-planes is similar, due to the different initialization, the tri-plane learned in the first run cannot be used with the MLP obtained in the second (third column of Fig. 14.3, right side), and vice-versa. However, it is always possible to find a suitable permutation of the channels of the first tri-plane such that the second MLP can correctly decode its features (fourth column of Fig. 14.3, right side), and vice-versa. We found the right permutation by a brute-force search based on maximizing reconstruction quality. To make the search feasible, we used a smaller number of channels, *i.e.*, $C = 8$ rather than $C = 16$. Still, the experimental results in Sec. 14.3.1 support our belief that the main source of variance across randomly initialized tri-plane optimizations of the same shape consists of a permutation of the channel order.

Thus, unlike neural fields realized as MLPs, with tri-planes, it is straightforward to counteract the nuisances due to random initialization by adopting standard architectures made invariant to the channel order.

14.2.3 Architectures for neural processing of tri-plane neural fields

Based on the above analysis, we propose to process tri-planes with Transformers [15]. In particular, we propose to rely on a Transformer encoder without positional encoding, which is equivariant to token positions. By tokenizing tri-planes so that each token represents a channel of a plane, such architecture seamlessly computes representations equivariant to the order of the channels. Specifically, we unroll each channel of size $H \times W$, to obtain a token of dimension HW within a sequence of length $3C$ tokens. These tokens are then linearly projected and fed into the Transformer. The output of the encoder is once again a sequence of $3C$ tokens.

For global tasks like classification, the output sequence is subsequently subjected to a max pool operator to obtain a global embedding that characterizes the input shape. In our experiments, this embedding is then processed through a stack of fully connected layers to compute the logits. The way the tokens are defined, the absence of positional encoding, and the final max pool operator allow for achieving invariance to the channel order. For dense tasks like part segmentation, instead, we also utilize the decoder part of Transformers. More specifically, we treat the coordinates queries to segment as a sequence of input tokens to the decoder. Each point $\mathbf{p}$ with coordinates (x, y, z) undergoes positional encoding [339] and is then projected to a higher-dimensional space using a linear layer.

By leveraging the cross-attention mechanisms within the decoder, each input token representing a query point can globally attend to the most relevant parts of the tri-planes processed by the encoder to produce its logits. Additional details about the architectures, including block diagrams, are reported in Appendix B.5.3.

14.3 Tasks on Neural fields

14.3.1 Neural field classification

Benchmark.

We perform extensive tests to validate our approach. In so doing, we build the first neural field classification benchmark, where we compare all the existing proposals for neural field processing on the task of predicting the category of the objects represented within the field without recreating the explicit signal. Specifically, we test all methods on *UDF* fields obtained from point clouds of ModelNet40 [182], ShapeNet10 [28], and ScanNet10 [28]; *SDF* fields learned from meshes of Manifold40 [278]; *OF* fields obtained from voxels grids of ShapeNet10. In addition, we provide for the first time classification results on neural radiance fields (*RF*), learned from ShapenetRender [342]. See Appendix B.5.1 for more details on the benchmark. Besides a simple MLP baseline, we compare with frameworks designed to process neural fields realized as MLPs, *i.e.*, inr2vec, NFN [9], NFT [318], and DWSNet [317]. These methods process single MLP neural fields, which we implement as SIREN networks [304]. Differently from inr2vec, the MLPs in our benchmark are *randomly initialized* to simulate real-world scenarios. Unlike all previous methods, ours processes individual tri-plane neural fields, which are also randomly initialized. Moreover, we compare with frameworks where neural fields are realized by a shared network and a small latent vector or modulation, *i.e.*, DeepSDF [296] and Functa [319]. Whenever possible, we use the official code released by the authors to run the experiments. Note that not all frameworks can be easily extended to all fields. Therefore, we only test each framework in the settings that are compatible with our resources and that do not require fundamental changes to the original implementations (see Appendix B.5.2 for more details).

Results. As we can observe in Tab. 14.2, overall, shared architecture frameworks (DeepSDF and Functa) outperform previous methods that directly operate on neural fields represented as a single neural network.

			UDF			*SDF*	*OF*	*RF*
Method	Type	Input	ModelNet40	ShapeNet10	ScanNet10	Manifold40	ShapeNet10	ShapeNetRender
DeepSDF [296]	Shared	Latent vector	41.2	76.9	51.2	64.9	–	–
Functa [319]	Shared	Modulation	**87.3**	83.4	56.4	85.9	36.3	–
inr2vec	Single	MLP	10.6	42.0	40.9	13.1	38.6	–
MLP	Single	MLP	3.7	28.8	36.7	4.2	29.6	22.0
NFN [9]	Single	MLP	9.0	9.0	45.3	4.1	33.8	87.0
NFT [318]	Single	MLP	6.9	6.9	45.3	4.1	33.8	85.3
DWSNet [317]	Single	MLP	56.3	78.4	62.2	47.9	79.1	83.1
Ours	Single	Tri-plane	87.0	**94.1**	**69.1**	**86.8**	**91.8**	**92.6**

TABLE 14.2. **Test set accuracy for shape classification across neural fields.** We compare several frameworks capable of processing neural fields

However, we point out again that the reconstruction capability of such frameworks is poor, as shown in Sec. 14.2.2. Conversely, previous methods that utilize individual neural fields demonstrate superior reconstruction quality but struggle to perform effectively in real-world scenarios where shapes need to be fitted starting from arbitrary initialization points. inr2vec makes the assumption of learning all MLPs starting from the same initialization, and it does not work when this initialization schema is not applied. Among the family of methods that adopt layers equivariant and invariant to permutations of the neurons, only DWSNet works on the large MLPs constituting our benchmark, though performance tends to be worse than shared network approaches. Our method delivers the best of both worlds: it ingests tri-planes neural fields, which exhibit excellent reconstruction quality while achieving the best performance overall, often surpassing by a large margin all other methods, including those relying on a shared neural field, *e.g.*,the accuracy on ScanNet10 is 56.4 for Functa vs 69.1 for our method. Hence, we can state confidently that our approach achieves the best trade-off between classification accuracy and reconstruction quality. Finally, we highlight that our proposal is effective with all the datasets and kinds of fields addressed in the experiments.

Comparison with explicit representations. In Tab. 14.3, we compare our method against established architectures specifically designed to process explicit representations. For a fair comparison, we reconstruct the explicit data from each field so that exactly the same shapes are used in each experiment. Practically, we reconstruct point clouds, mesh, and voxel grids from *UDF*, *SDF*, and *OF*, respectively. Then, we process them with specialized architectures, *i.e.*, PointNet [160] for point clouds, MeshWalker [289] for meshes, and Conv3DNet [258] for voxel grids. As for *RF*, we render a multi-view dataset with 36 views for each object. Then, we train 36 per-view ResNet50 [14] so as to ensemble the predictions at test time. We highlight how our proposal, which can classify every neural field with the same standard architecture, almost closes the performance gap with respect to *specialized* architectures designed to process explicit representations. Noticeably, we show that NeRFs can be classified accurately from the features stored in a tri-plane structure without rendering any images.

Towards universal tri-plane classification. Finally, to the best of our knowledge, we implement for the first time a *universal tri-plane classifier*, *i.e.*, a model which can be trained

Method	Input	ModelNet40	ShapeNet10	ScanNet10	Manifold40	ShapeNet10	ShapeNetRender
Ours	Tri-plane	87.0	94.1	69.1	86.8	91.8	92.6
PointNet [160]	Point Cloud	88.8	94.3	72.7	–	–	–
MeshWalker [289]	Mesh	–	–	–	90.0	–	–
Conv3DNet [258]	Voxel	–	–	–	–	92.1	–
ResNet50 [14]	Images	–	–	–	–	–	94.0

TABLE 14.3. **Comparison with explicit representations. Top:** Test set accuracy of our neural field processing method. **Bottom:** Standard networks trained and tested on explicit representations.

and tested with any kind of tri-plane hybrid neural field. Indeed, since the tri-plane structure, as well as the neural processing architecture, are just the same, regardless of the kind of field, we can seamlessly learn a unified model able to classify a variety of fields. For example, we start from the meshes of the Manifold40 dataset and obtain the corresponding point clouds and voxel grids so as to fit three different fields (*SDF*, *UDF*, and *OF*). Accordingly, we build training, validation, and test sets with samples drawn from all three fields. More precisely, if a shape appears in a set represented as an *SDF*, it also appears in that set as a *UDF* and *OF*. Then, as reported in Tab. 14.4, we run classification experiments by training models on each of the individual fields as well as on all three of them jointly. The results show that when a classifier is trained on only one field, it may not generalize well to others. On the other hand, a single model trained jointly on all fields not only works well with test samples coming from each one, but it also outperforms the models trained individually on a single kind of field.

14.3.2 Neural field 3D part segmentation

We explore here the potential of our method in tackling dense prediction tasks like part segmentation, where the goal is to predict the correct part label for any given 3D point.

In Tab. 14.5, we compare our method to inr2vec, which was trained on fields generated from random initialization and is the only competitor capable of addressing the part segmentation task. Our experiments were conducted by fitting *UDF* fields from point clouds of 2048 points from the ShapeNetPart dataset [334]. As a reference, we present the results obtained using specialized architectures commonly used for point cloud segmentation, like PointNet, PointNet++, and DGCNN. As in inr2vec, all models are trained on the point clouds reconstructed from the fitted fields.

We observe that our proposal outperforms inr2vec by a large margin, with improvements of 20% and 16.7% for instance and class mIoU, respectively. Moreover, Tab. 14.5 demonstrates once again that tri-planes are effective in substantially reducing the performance gap between processing neural fields and explicit representations.

14.3.3 Different architectures for tri-plane processing

In Tab. 14.6, we compare several plausible alternatives to Transformers for processing tri-planes, which have roughly the same number of parameters and have been trained with

Train			Test		
UDF	*SDF*	*OF*	*UDF*	*SDF*	*OF*
✓			84.7	78.4	15.6
	✓		67.3	86.8	11.9
		✓	49.3	46.9	77.7
✓	✓	✓	**87.4**	**87.8**	**80.3**

TABLE 14.4. **Universal tri-plane classifier.** Test set accuracy on Manifold40.

Method	Input	instance mIoU	class mIoU	airplane	bag	cap	car	chair	earphone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skateboard	table
inr2vec	MLP	64.2	64.5	57.9	72.9	67.8	56.4	67.6	48.4	81.6	70.6	55.5	88.8	51.5	87.2	64.7	40.1	58.4	62.5
Ours	Tri-plane	**84.2**	**81.3**	**83.0**	**80.2**	**87.4**	**76.6**	**90.2**	**68.2**	**91.6**	**85.9**	**82.1**	**95.0**	**70.7**	**94.4**	**81.9**	**59.0**	**73.4**	**80.9**
PointNet [160]	Point Cloud	83.1	78.96	81.3	76.9	79.6	71.4	89.4	67.0	91.2	80.5	80.0	95.1	66.3	91.3	80.6	57.8	73.6	81.5
PointNet++ [161]	Point Cloud	84.9	82.73	82.2	88.8	84.0	76.0	90.4	80.6	91.8	84.9	84.4	94.9	72.2	94.7	81.3	61.1	74.1	82.3
DGCNN [167]	Point Cloud	83.6	80.86	80.7	84.3	82.8	74.8	89.0	81.2	90.1	86.4	84.0	95.4	59.3	92.8	77.8	62.5	71.6	81.1

TABLE 14.5. **Part segmentation results. Top:** Implicit frameworks. **Bottom:** Methods on explicit representation. In **bold**, best results among frameworks processing neural fields.

the same hyperparameters. As discussed previously, since tri-planes contain an informative and regular discrete data structure and are compact, they can be processed with standard architectures. Hence, we test an MLP, a ResNet50 [14], and two variants of PointNet all with roughly the same parameters. A simple MLP that processes the flattened tri-planes (row 1) severely underperforms with respect to the alternatives, likely due to its inability to capture the spatial structures present in the input as well as its sensitivity to the channel permutation caused by random initializations. A standard CNN like ResNet50, processing tri-planes stacked together and treated as a multi-channel image of resolution $W \times H$, is instead equipped with the inductive biases needed to effectively process the spatial information contained in the tri-planes (Fig. 14.3) and already delivers promising performance, although it cannot cope with channel permutations.

In Tab. 14.6, we compare several plausible alternatives to Transformers for processing tri-planes, which have roughly the same number of parameters and have been trained with the same hyperparameters. As discussed previously, since tri-planes contain an informative and regular discrete data structure and are compact, they can be processed with standard architectures. Hence, we test an MLP, a ResNet50 [14], and two variants of PointNet all with roughly the same parameters. A simple MLP that processes the flattened tri-planes

		UDF			*SDF*	*OF*
Method	Input	ModelNet40	ShapeNet10	ScanNet10	Manifold40	ShapeNet10
MLP	Tri-plane	41.6	84.2	55.8	40.2	79.1
CNN	Tri-plane	82.2	92.1	63.4	82.5	88.4
PointNet	Tri-plane	85.8	93.4	**69.3**	85.6	91.5
Spatial PointNet	Tri-plane	32.3	65.4	51.3	37.0	54.7
Transformer	Tri-plane	**87.0**	**94.1**	69.1	**86.8**	**91.8**

TABLE 14.6. **Ablation study of architectures for tri-plane neural field classification**

(row 1) severely underperforms with respect to the alternatives, likely due to its inability to capture the spatial structures present in the input as well as its sensitivity to the channel permutation caused by random initializations. A standard CNN like ResNet50, processing tri-planes stacked together and treated as a multi-channel image of resolution $W \times H$, is instead equipped with the inductive biases needed to effectively process the spatial information contained in the tri-planes (Fig. 14.3) and already delivers promising performance, although it cannot cope with channel permutations.

14.4 Concluding remarks and limitations

We have shown that tri-plane hybrid neural fields are particularly amenable to direct neural processing without sacrificing representation quality. Indeed, by feeding only the tri-plane structure into standard architectures, such as Transformers, we achieve better classification and segmentation performance compared to previous frameworks aimed at processing neural fields and dramatically shrink the gap with respect to specialized architectures designed to process 3D data represented explicitly. To validate our intuitions, we propose the first benchmark for neural processing of neural fields, which includes the main kinds of fields used to model the 3D world as well as all the published methods that tackle this very novel research problem. Within our experimental evaluation, we show for the first time that NeRFs can be effectively classified without rendering any images.

Chapter 15

Final Remarks

In this thesis, we explored the problem of Transfer Learning from several points of view. In the first part, we mainly focused on 2D Semantic Segmentation and studied how its relationship with other tasks such as Depth Estimation and Edge Detection can be exploited to boost the model performance on unlabelled datasets.

We started from Chapter 3 by introducing a novel framework able to transfer knowledge by learning a transfer function in feature space between two networks. In particular, this mapping can be learned by means of a neural network and we proposed two effective ways to facilitate the learning process of this function. First, we found that it is important to align the features at the input level of the transfer function, and we implemented this strategy by introducing the NDA (Norm Discrepancy Alignment) loss. Then, we demonstrated how exploiting an additional auxiliary task, such as Edge Detection, that shares some fundamental properties with the other two main tasks can further enrich the feature learned by the two networks and ease the learning of the transfer function. As argued previously in Chapter 3, this approach is complementary to other common Domain Adaptation techniques. Therefore, we proposed in Chapter 4 an effective way to leverage this framework such that it can be plugged seamlessly into other Domain Adaptation methods. At its core, the main idea is to exploit Self-Supervised Monocular Depth Estimation to provide strong geometric priors to improve the predictions of a Semantic Segmentation network that already applies some alignment strategies but lacks geometric knowledge. Then, we implemented an effective data augmentation strategy for Self-training that is once again based on the geometric cues provided by the monocular depth estimation network. We believe that this synergy between Depth Estimation and Semantic Segmentation (or other tasks in general) is key for obtaining a model capable of working well in unseen scenarios. Indeed, many works followed this idea [343, 344, 345] and have been able to further improve performance. Finally, in Chapter 5, we focused on the core problem of Semantic Segmentation which is obtaining sharp segmentation masks even on unllabeled datasets. To this end, we have shown that domain-invariant features can be used to estimate a displacement map, that can be subsequently exploited to refine the segmentation mask. Combined with a novel Self-training strategy that is specifically designed to keep valuable information at the boundaries of an object, we have been able to considerably improve performance along boundaries with respect to previous methods.

In the second part of the thesis, we examined the Transfer Learning problem applied to 3D data such as point clouds. This field has emerged in the last years and it is relatively new compared to the much richer literature of UDA for 2D Semantic Segmentation. First, in Chapter 7, we largely improved upon state-of-the-art UDA for point cloud classification showing that Self-training can be quite effective in this scenario. In particular, we designed a specific 3D strategy based on point cloud reconstruction that is able to refine effectively pseudo-labels obtained from a previously pre-trained model. In Chapter 8, we went one step further and inspired by popular and successful Self-Supervised techniques for 2D data, we proposed a novel strategy to learn features on the target domain without the need for annotations that preserve feature discriminability. We also leveraged the tools provided by GNNs to implement a Self-training algorithm able to refine misclassified samples by reasoning globally on the target domain.

As new ways of representing 3D data such as Neural Fields are emerging in the vision community, it would be interesting to investigate whether these representations are more suitable to tackle Transfer Learning. Some early works, such as [200], already provided some hints on the usefulness of implicit representations for Domain Adaptation applied to 3D data, although more research should be conducted to clearly establish its advantages in this field. With the hope of inspiring the reader on this aspect, we also included in Part IV some recent developments on processing directly Neural Fields, which we believe could provide a unified and a standalone representation method for continuous signal and 3D more specifically. Hence, in Chapter 13, we presented inr2vec, a representation learning framework that allows encoding the weights of a network representing a 3D shape into a compact embedding, so that it can be used to solve common downstream tasks such as point cloud classification or part segmentation. Nonetheless, inr2vec requires all networks representing 3D shapes to be initialized in the same way to work correctly. For this reason, in Chapter 14, we proposed to adopt tri-plane hybrid neural fields instead, and we demonstrated that this representation is particularly amenable for direct processing without sacrificing representation quality. With these additional chapters, we hope to stimulate further research on investigating whether this representation can also be effective for Transfer Learning, and we included some preliminary results showing that Neural Fields representing different modalities such as point clouds, meshes and voxels can be used together to boost performance for 3D shape classification, although within the same domain. This can be however already seen as a form of Transfer Learning between different 3D representations. Indeed, thanks to the Neural Field representation, we have shown that a classifier trained on Neural Field obtained from point clouds can be used to classify Neural Fields representing meshes and vice-versa.

We concluded our journey on the classical Transfer Learning problem in the third part of this dissertation, where we merged the knowledge acquired in the two previous parts to tackle a much more challenging and realistic adaptation scenario: UDA for Multi-Modal 3D Semantic Segmentation. We exploited once again in Chapter 10 the synergy between depth and segmentation to improve existing works. In particular, given the availability of multiple sensors at the same time, we extracted depth maps from the LiDAR sensor and used Depth Completion as an auxiliary task to boost the performance of the 2D network responsible for estimating the segmentation mask given the corresponding RGB data. In turn, we exploited the completed depth map, to also improve the 3D network by means of a novel Self-training strategy. We further contribute to the field in Chapter 11 by studying the complementarity of the 2D and 3D networks. Indeed, as examined in Chapter 10, the de-facto architecture for addressing Multi-Modal Domain Adaptation foresees a two-branch model to process the two inputs independently, to then merge the final segmentation mask at the end. Thus we shed some light on this aspect and provide an intuitive explanation based on the notion of the effective receptive field of why processing data with these two networks grants orthogonal predictions that can be effectively fused together.

In conclusion, this thesis introduces various strategies to address the Transfer Learning problem in the presence of limited annotation. The fundamental insight derived from this dissertation is that, in such situations, exploring the relationships between different yet related tasks holds great promise for achieving robust performance, even when dealing with unlabeled datasets.

Looking ahead, we would like to suggest two potential avenues for further research in this dynamic field. Firstly, delving into recent advancements in Self-supervised learning techniques, such as [346, 347], presents an intriguing opportunity as these techniques showcase notable effectiveness in learning robust features, even for dense tasks such as Semantic Segmentation. Indeed, given that unlabeled data is frequently available in the target domain, recent advancements in Self-supervised could be used to obtain more robust models.

A second interesting and more ambitious future direction is to investigate whether implicit representations can be exploited to reduce the domain gap for 3D semantic segmentation. Indeed, as emerged from this thesis and previous work in the literature, one of the main problems in UDA for 3D Semantic Segmentation is the input representation. Although the input data always consists of point clouds with *xyz* coordinates, many factors may worsen the gap among the distributions. The most evident is probably the density of such points. To alleviate this problem, an exciting direction to pursue is the possibility of using implicit representations to decouple the input data (i.e. its content) from its representation. This opens up the possibility of sampling the input signal at the desired resolution and extracting representations that are domain-independent. A seminal work that can be used as a starting point is [348], in which the authors propose to fit a neural field from LiDAR scans. Following this line of research, another compelling direction would be to apply common domain adaptation algorithms directly to Neural Fields. This strategy would have the great advantage that it could be applied seamlessly to any kind of data since Neural Fields can potentially represent any continuous signal.

Part V

Appendices

Appendix A

Deep Learning on Implicit Neural Representations of Shapes

A.1 Individual INRs vs. Shared Network Frameworks

In this section, we aim to compare the representation power of individual INRs (*i.e.*, one network for each data point) with the one of frameworks adopting a single shared network for the whole dataset, like DeepSDF [296], OccupancyNetworks [301] or [319]. The important difference between such approaches and our method relies in the fact that in shared network frameworks, the shared network and the set of latent codes are the implicit representation, whose reconstruction quality is negatively affected by using a single network to represent the whole dataset, as shown below. In our framework, instead, we decouple the representations (INRs) from the embeddings used to process them in downstream tasks (yielded by inr2vec). The quality of the representation is then entrusted to the individual INRs and inr2vec does not influence it.

To compare the representation quality of individual INRs with the one of share network frameworks, we fitted the SDF of the meshes in the Manifold40 dataset with OccupancyNetworks [301], DeepSDF [296] and LatentModulatedSiren (*i.e.*, the architecture used by the contemporary work that addresses deep learning on INRs [319]). Then, we reconstructed the training discrete meshes from the three frameworks and we compared them with the ground-truth ones, performing the same comparison using the discrete shapes reconstructed from individual INRs. To perform the comparison, we first reconstructed meshes and then we sampled dense point clouds (16,384 points) from the reconstructed surfaces, doing the same for the ground-truth meshes. We report the quantitative comparisons in Tab. A.1, using two metrics: the Chamfer Distance as defined in [332] and the F-Score as defined in [341].

The comparison reported in the Tab. A.1a shows that both OccupancyNetworks and LatentModulatedSiren cannot represent the shapes of the training set with a good fidelity, most likely because of the single shared network that struggles to fit a big number of shapes with high variability (∼10K shapes, 40 classes). At the same time, DeepSDF obtains really poor scores, highlighting the difficulty of training an auto-decoder framework on a large and

Method	train set	
	CD (mm) ↓	F-Score ↑
OccupancyNetworks	0.8	0.44
DeepSDF	11.1	0.14
LatentModulatedSiren	0.7	0.37
Individual INRs	**0.3**	**0.50**

(A) Train set.

Method	test set	
	CD (mm) ↓	F-Score ↑
OccupancyNetworks	1.3	0.39
DeepSDF	6.6	0.25
LatentModulatedSiren	1.9	0.28
Individual INRs	**0.3**	**0.49**

(B) Test set.

TABLE A.1. **Individual INRs vs. shared network frameworks.** Comparison between discrete meshes reconstructed from individual INRs and from shared network frameworks on Manifold40.

varied dataset. Individual INRs, instead, produce reconstructions with good quality, even if we adopted a fitting procedure with only 500 steps for each shape.

Moreover, the approaches based on a conditioned shared network tend to fail in representing unseen samples that are out of the training distribution. Hence, in the foreseen scenario where INRs become a standard representation for 3D data hosted in public repositories, leveraging on a single shared network may imply the need to frequently retrain the model upon uploading new samples which, in turn, would change the embeddings of all the previously stored data. On the contrary, uploading the repository with a new shape would not cause any sort of issue with individual INRs, where one learns a network for each data point.

To better support our statements, in Tab. A.1b we report the comparison between shape reconstructed from OccupancyNetworks, DeepSDF, LatentModulatedSiren and individual INRs, using shapes from the test set of Manifold40, *i.e.*, shapes unseen at training time. The numbers show that both OccupancyNetworks and LatentModulatedSiren present a drop in the quality of the reconstructions, indicating that both frameworks struggle to represent new shapes not available at training time. Surprisingly, DeepSDF produces better scores on the test set with respect to the results on the train set, but still presents a quite poor performance in comparison with the other methods. Conversely, the problem of representing unseen shapes is inherently inexistent when adopting individual INRs, as shown by the numbers in the last row of Tab. A.1b, which are almost identical to the ones presented in Tab. A.1a.

We report in Fig. A.1 and Fig. A.2 the comparison described above from a qualitative perspective. It is possible to observe that the visualizations confirm the results posted in Tab. A.1, with shared network frameworks struggling to represent properly the ground-truth shapes, while individual INRs enable high-fidelity reconstructions.

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

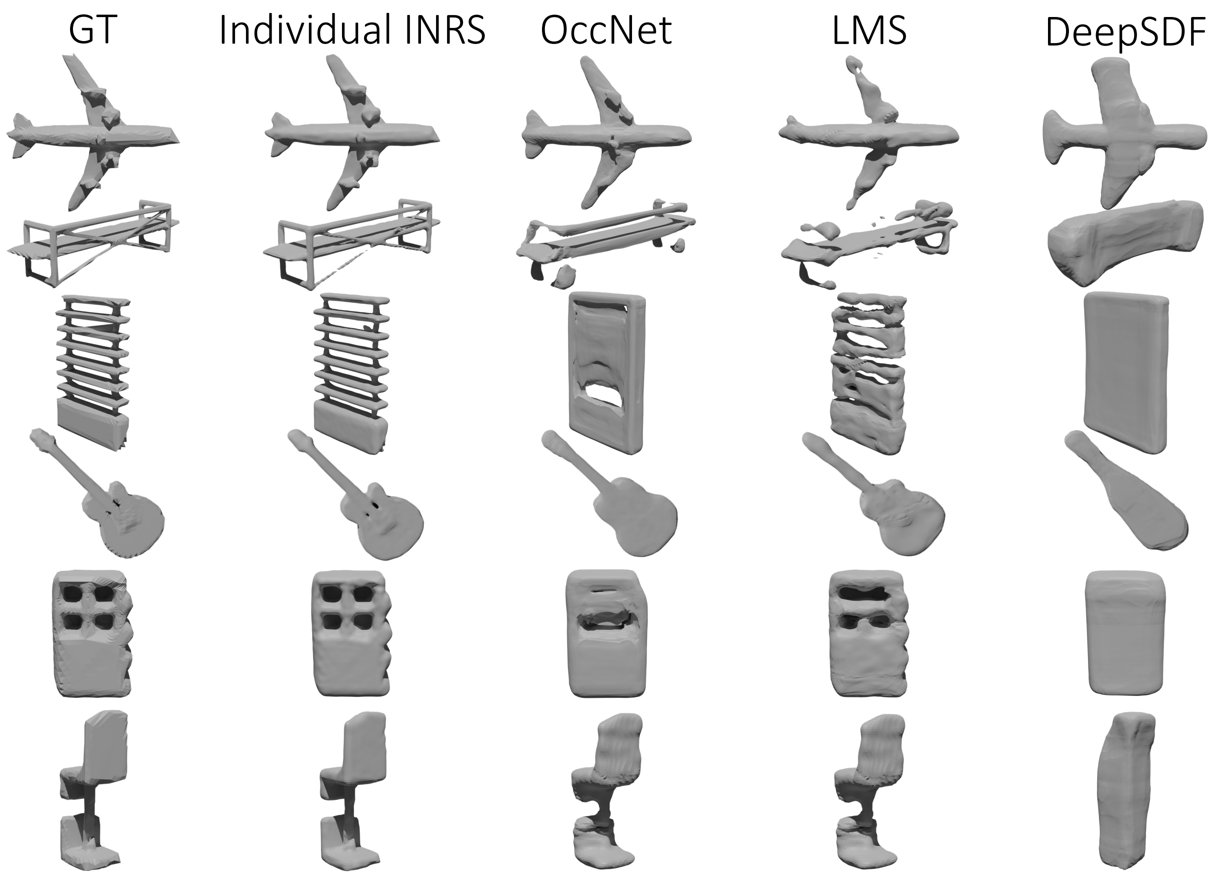


FIGURE A.1. **Individual INRs vs. shared network frameworks (train shapes).** Qualitative comparison of meshes from Manifold40 reconstructed from individual INRs or from shared network frameworks, when dealing with training shapes. OccNet stands for OccupancyNetworks, LMS stands for LatentModulatedSiren.

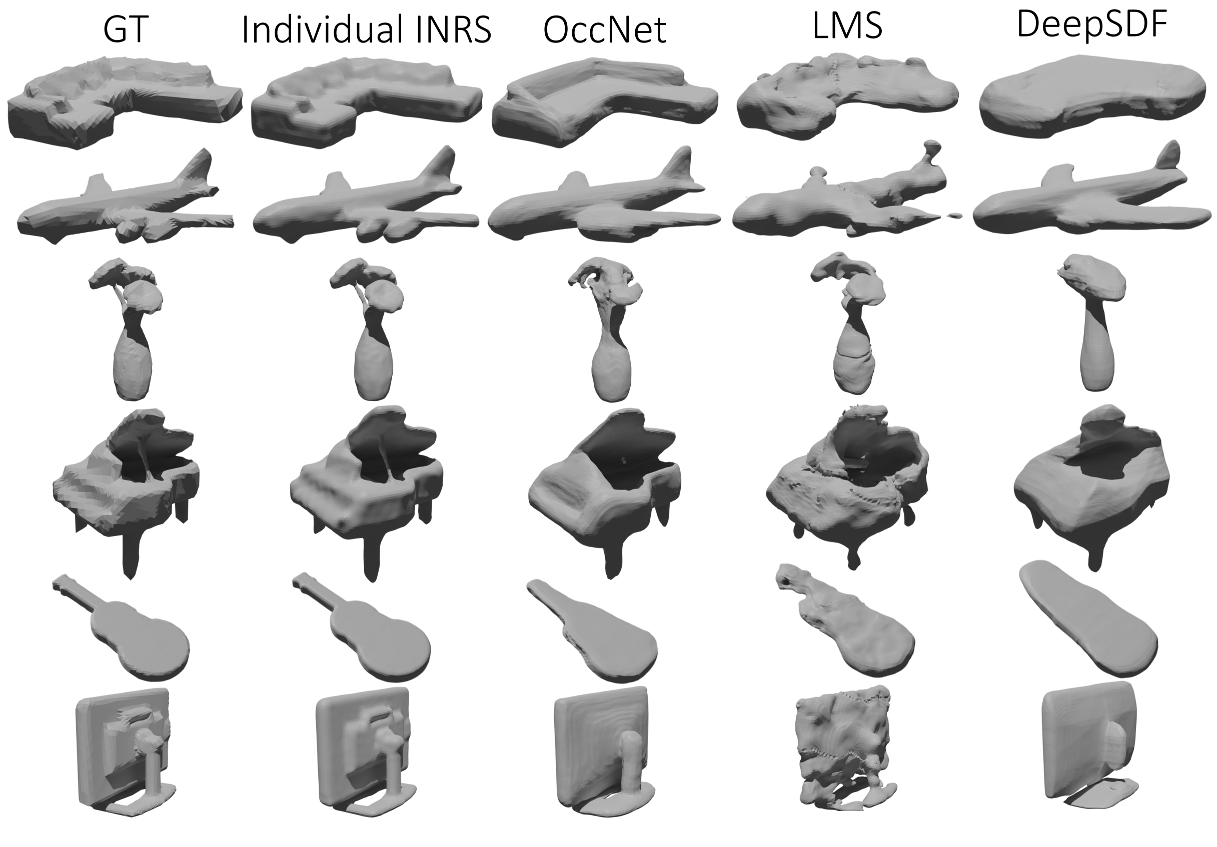


FIGURE A.2. **Individual INRs vs. shared network frameworks (test shapes).** Qualitative comparison of meshes from Manifold40 reconstructed from individual INRs or from shared network frameworks, when dealing with shapes unseen during training. OccNet stands for OccupancyNetworks, LMS stands for LatentModulatedSiren.

A.2 Obtaining INRs from 3D Discrete Representations

In this section, we detail the procedure used when fitting INRs to create the datasets used in this work. Given a dataset of 3D shapes we train a set of the same number of MLPs, fitting each one on a single 3D shape. Every MLP is thus trained to approximate a continuous function that describes the represented shape, the nature of the function being chosen according to the discrete representation in which the shape is provided. We adopt MLPs with multiple hidden layers of the same dimension as done in [304, 324, 325, 326, 327], interleaved by the sine activation function, as proposed in [304], to enhance the capability of the MLPs to fit the high frequency details of the input signal.

In its general formulation, an INR can be used to fit a continuous function $f : \mathbb{R}^{in} \rightarrow \mathbb{R}^{out}$. To do so, a training set composed of N points $\mathbf{x}_i \in \mathbb{R}^{in}$ with $i = 1, 2, ..., N$, paired with values $\mathbf{y}_i = f(\mathbf{x}_i) \in \mathbb{R}^{out}$, is exploited to find the optimal parameters θ^* for the MLP that implements the INR, by solving the optimization problem:

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{y}_i, f_\theta(\mathbf{x}_i)), \tag{A.1}$$

where f_θ represents the function f approximated by the MLP with parameters θ and ℓ is a loss function that computes the error between predicted and ground-truth values.

The output value $f_\theta(\mathbf{x}_i)$ is computed as a series of linear transformations, each one followed by a non-linear activation function (*i.e.*, the sine function in our case), except the last one. Considering a MLP m, the mapping between its layers $L-1$ and L consists in a linear transformation that maps the values $\mathbf{h}_m^{L-1} \in \mathbb{R}^{D_{L-1}}$ from the layer $L-1$ into the values $\mathbf{h}_m^L = \phi(\mathbf{W}_m^L \mathbf{h}_m^{L-1} + \mathbf{b}_m^L) \in \mathbb{R}^{D_L}$ of the layer L, with $\mathbf{W}_m^L$ being the matrix of weights $\in \mathbb{R}^{D_L \times D_{L-1}}$, $\mathbf{b}_m^L$ being the biases vector $\in \mathbb{R}^{D_L}$, and $\phi(\cdot)$ the non-linearity [304]. If we now consider M MLPs used to fit M different INRs and the mapping between the layers $L-1$ and L, we can easily compute such mapping simultaneously for all the MLPs on modern GPUs thanks to tensor programming frameworks. The mapping consists indeed in a straightforward tensor contraction operation, where the values $\mathbf{h}_{L-1} \in \mathbb{R}^{M \times D_{L-1}}$ of the layer $L-1$ are mapped to the values $\mathbf{h}_L = \mathbf{W}_L \mathbf{h}_{L-1} + \mathbf{b}_L \in \mathbb{R}^{M \times D_L}$ of the layer L, with $\mathbf{W}^L \in \mathbb{R}^{M \times D_L \times D_{L-1}}$ and $\mathbf{b}^L \in \mathbb{R}^{M \times D_L}$. Extending this formulation to all the layers of the chosen MLP architecture allows to fit multiple INRs in parallel.

In the following, we describe how we train MLPs to obtain INRs starting from point clouds, triangle meshes and voxel grids.

Point clouds. The INR for a 3D shape represented by a point cloud $\mathcal{P}$ encodes the *unsigned distance function* (udf) of the point cloud $\mathcal{P}$. Given a point $\mathbf{p} \in \mathbb{R}^3$, the value $udf(\mathbf{p})$ is defined as $\min_{q \in \mathcal{P}} \|\mathbf{p} - \mathbf{q}\|_2$, *i.e.*, the euclidean distance from $\mathbf{p}$ to the closest point $\mathbf{q}$ of the point cloud. After preparing a training set of N points $\mathbf{x}_i \in \mathbb{R}^3$ with $i = 1, 2, ..., N$, coupled with their udf values $y_i \in \mathbb{R}$, the INR of the underlying 3D shape is obtained by training a MLP to regress correctly the udf values, with the learning objective:

$$\mathcal{L}_{mse} = \frac{1}{N} \sum_{i=1}^{N} (y_i - f_\theta(\mathbf{x}_i))^2, \tag{A.2}$$

that consists in the mean squared error between ground-truth values y_i and the predictions by the MLP $f_\theta(\mathbf{x}_i)$. An alternative objective is converting the udf values y_i into values y_i^{bce} continuously spanned in the range $[0, 1]$, with 0 and 1 representing respectively the predefined maximum distance from the surface and the surface level set (*i.e.*, distance equal to zero). Then, the MLP optimizes the binary cross entropy between such labels and the predicted values, defined as:

$$\mathcal{L}_{bce} = -\frac{1}{N} \sum_{i=1}^{N} y_i^{bce} log(\hat{y}_i) + (1 - y_i^{bce}) log(1 - \hat{y}_i), \tag{A.3}$$

where $\hat{y}_i = \sigma(f_\theta(\mathbf{x}_i))$, with σ representing the sigmoid function. In our experiments, we found empirically that this second learning objective leads to faster convergence and more accurate INRs, and we decided to adopt this formulation when producing INRs from point clouds.

Triangle meshes. Triangle meshes are usually adopted to represent closed surfaces. This provides an additional information compared to the point clouds case, since the 3D space can be divided into the portion contained *inside* and *outside* the closed surface. Thus, the INR of a closed 3D surface represented by a triangle mesh can be obtained by fitting the *signed distance function* (sdf) to the surface defined by the mesh. Given a point $\mathbf{p} \in \mathbb{R}^3$, the value $sdf(\mathbf{p})$ is defined as the euclidean distance from $\mathbf{p}$ to the closest point of the surface, with positive sign if $\mathbf{p}$ is outside the shape and negative sign otherwise. Similarly to the point clouds case, an INR for a 3D shape represented by a triangle mesh can be obtained by pursuing the learning objective presented in Eq. (A.2), using a training set composed of 3D points paired with their sdf values. However, it possible to adopt a learning objective based on the binary cross entropy loss reported in Eq. (A.3) also for triangle meshes, and we empirically observed the same benefits. Hence, we adopt it also when fitting INRs on meshes. In this case, the sdf values y_i are converted into values $y_i^{bce} \in [0, 1]$, with 0 and 1 representing respectively the predefined maximum distance *inside* and *outside* the shape, *i.e.*, 0.5 represents the surface level set.

Voxel grids. A voxel grid is a 3D grid of V^3 cubes marked with label 1, if the cube is occupied, and label 0 otherwise. In order to fit an INR on voxels, it is possible to learn to regress the *occupancy function* (occ) of the grid itself. The training set, in this case, contains V^3 3D points that corresponds to the centroids of the cubes that compose the voxel grid.

Being each of such points $\mathbf{x}_i$ associated to a 0-1 label y_i, it is straightforward to use a binary classification objective to train the MLP that implement the desired INR. More specifically, we adopt the learning objective defined as:

$$\mathcal{L}_{focal} = -\frac{1}{N}\sum_{i=1}^{N} \alpha(1-\hat{y}_i)^{\gamma} y_i log(\hat{y}_i) + (1-\alpha)\hat{y}_i^{\gamma}(1-y_i)log(1-\hat{y}_i), \tag{A.4}$$

where $\hat{y}_i = \sigma(f_\theta(\mathbf{x}_i))$, while α and γ are respectively the balancing parameter and the focusing parameter of the focal loss proposed in [349]. We deploy a focal loss to alleviate the imbalance between the number of occupied and empty voxels.

A.3 Reconstructing Discrete Representations from INRs

In this section we discuss how it is possible to sample 3D discrete representations from INRs, which could be necessary to process the underlying shapes with algorithms that need an explicit surface (*e.g.*, Computational Fluid Dynamics [350, 351, 352]) or simply for visualization purposes.

Point clouds from udf. To sample a dense point cloud from an INR fitted on its udf, we use a slightly modified version of the algorithm proposed in [295]. The basic idea is to query the udf with points scattered all over the considered portion of the 3D space, *projecting* such points onto the isosurface according to the predicted udf values. In order to do that, let us define f_θ as the udf approximated by the INR with parameters θ. Given a point $\mathbf{p} \in \mathbb{R}^3$, it can be projected onto the isosurface by computing its updated position $\mathbf{p_s}$ as:

$$\mathbf{p_s} = \mathbf{p} - f_\theta(\mathbf{p}) \cdot \frac{\nabla_p f_\theta(\mathbf{p})}{\|\nabla_p f_\theta(\mathbf{p})\|}. \tag{A.5}$$

This can be intuitively understood by considering that the negative gradient of the udf indicates the direction of maximum decrease of the distance from the surface, pointing towards the closest point on it. Eq. (A.5), thus, can be interpreted as moving $\mathbf{p}$ along the direction of maximum decrease of the udf of a quantity defined by the value of the udf itself in $\mathbf{p}$, reaching the point $\mathbf{p}_s$ on the surface. One must consider, though, that f_θ is only an approximation of the real udf, which leads to two considerations. On a first note, the gradient of f_θ must be normalized (as done in Eq. (A.5)), while the gradient of the real udf has norm equal to 1 everywhere except on the surface. Secondly, the predicted udf value can be imprecise, implying that $\mathbf{p}$ can still be distant from the surface after moving it according Eq. (A.5). To address the second issue, the 3D position of p_s is refined repeating the update described in Eq. (A.5) several times. Indeed, after each update, the point gets closer and closer to the surface, where the values approximated by f_θ are more accurate, implying that the last updates should successfully place the point on the isosurface. Given an INR fitted on the udf of a point cloud, the overall algorithm to sample a dense point cloud from it is composed of the following steps. Firstly, we prepare a set of points scattered uniformly in the considered portion of the 3D space and we predict their udf value with the given INR. Then we filter out points whose predicted udf is greater than a fixed threshold (0.05 in our experiments). For the remaining points, we update their coordinates iteratively with Eq. (A.5) (we found 5 updates to be enough). Finally, we repeat the whole procedure until the reconstructed point cloud counts the desired number of points.

Triangle meshes from sdf. An INR fitted on the sdf computed from a triangle mesh allows to reconstruct the mesh by means of the Marching Cubes algorithm [331]. We refer the reader to the original paper for a detailed description of the method, but we report here a short presentation of the main steps, for the sake of completeness. Marching Cubes explores the considered 3D space by querying the sdf with 8 locations at a time, that are the vertices of an arbitrarily small imaginary cube. The whole procedure involves *marching* from one cube to the other, until the whole desired portion of the 3D space has been covered. For each cube, the algorithm determines the triangles needed to model the portion of the isosurface that passes through it. Then, the triangles defined for all the cubes are fused together to obtain the reconstructed surface. In order to determine how many triangles are needed for a single cube and how to place them, for each pair of neighbouring vertices of the cube, their sdf values are computed and one triangle vertex is placed between them if such values have opposite sign. Considering that the number of possible combinations of the sdf signs at the cube vertices is limited, it is possible to build a look-up table to retrieve the triangles configuration for the cube starting from the sdf signs at its eight vertices, combined in a 8-bit integer and used as key for the look-up table. After the triangles configuration for a cube has been retrieved, the vertices of the triangles are placed on the edges connecting the cube vertices, computing their exact position by linearly interpolating the two sdf values that are connected by each edge.

Voxel grids from occ. In order to reconstruct voxel grids from INRs, we adopt a straightforward procedure. Each INR has been trained to predict the probability of a certain voxel to be occupied, when queried with the 3D coordinates of the voxel centroid. Thus, a first step to reconstruct the fitted voxels consists in creating a grid of the desired resolution V. Then, the INR is queried with the V^3 centroids of the grid and predicts an occupancy probability for each of them. Finally, we consider as occupied only voxels whose predicted probability is greater than a fixed threshold, which we set to 0.4, as we found empirically that it allows for a good trade-off between scattered and over-filled reconstructions.

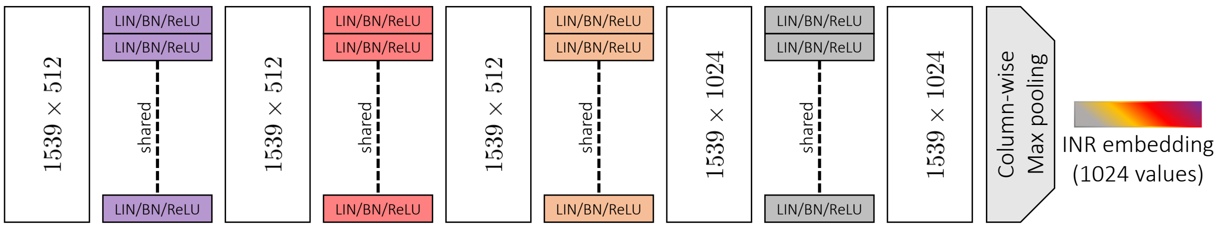


FIGURE A.3. **inr2vec encoder.** With a series of linear transformations and a final column-wise max pooling, the encoder maps the input weights matrix into a compact embedding. LIN/BN/ReLU stands for a linear transformation, followed by batch normalization and ReLU activation function.

A.4 inr2vec Encoder and Decoder Architectures

In this section, we describe the architecture of inr2vec encoder, along with the one of the implicit decoder used to train it (see Sec. 12.1.1).

Encoder. inr2vec encoder, detailed in Fig. A.3, consists in a series of linear transformations, that maps the input INR weights into features with higher dimensionality, before applying max pooling to obtain a compact embedding. More specifically, the input weights are rearranged in a matrix with shape $L(H+1) \times H$, where H is the number of nodes in the hidden layers of the MLP that implements the input INR and L is the number of linear transformations between such hidden layers (*i.e.*, the MLP has $L+1$ hidden layers). The matrix is obtained by stacking L matrices (one for each linear transformation), each one with shape $(H+1) \times H$, being composed of a matrix of weights with shape $H \times H$ and a row of H biases. In our setting, each MLP has 4 hidden layers with 512 nodes: the final matrix in input to inr2vec encoder has shape $3 \cdot (512+1) \times 512 = 1539 \times 512$. In the current implementation, the four linear mappings of the encoder transform each row of the input matrix into features with size 512, 512, 1024 and 1024, obtaining, at each step, features matrices with shape 1539×512, 1539×512, 1539×1024 and 1539×1024. Finally, the encoder applies column-wise max pooling to compress the final matrix into a single compact embedding composed of 1024 values. Between the linear mappings of the encoder, we adopt 1D batch normalization and ReLU activation functions.

Decoder. The implicit decoder that we adopt to train inr2vec is presented in Fig. A.4. We designed it taking inspiration from [296], since we need a decoder capable of reproducing the implicit function of input INR when conditioned on the embedding obtained by the encoder. Thus, the decoder takes in input the concatenation of the INR embedding with the coordinates of a given 3D query. We adopt the positional encoding proposed in [339] to embed the input 3D coordinates into a higher dimensional space to enhance the capability of the decoder to capture the high frequency variations of the input data. The query 3D coordinates are mapped into 63 values that, concatenated with the 1024 values that compose the INR embedding, result in a vector with 1087 values as input for inr2vec decoder. Internally, the implicit decoder is composed of 4 hidden layers with 512 nodes and of a skip connection that

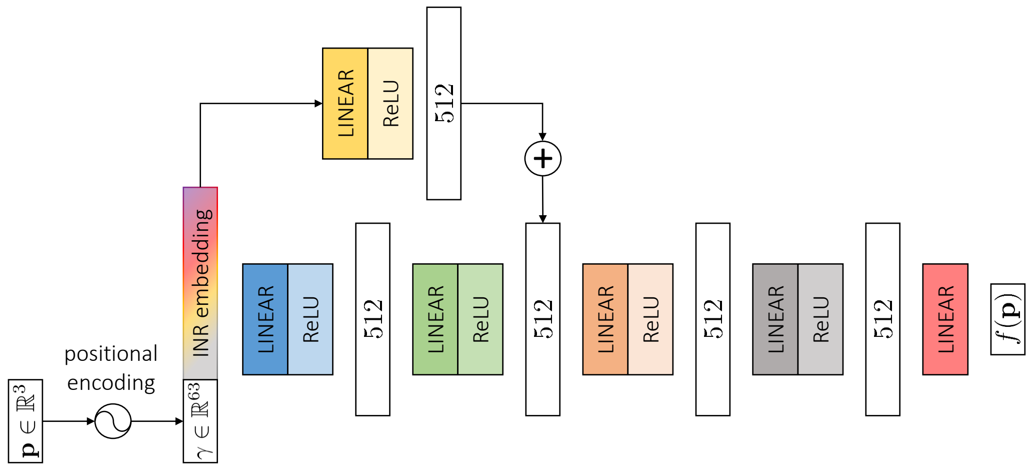


FIGURE A.4. **inr2vec decoder.** Our framework is trained with an implicit decoder, that maps an INR embedding concatenated with a 3D query into the value of the implicit function at the query coordinates.

projects the input 1087 values into a vector of 512 elements, that are summed to the features of the second hidden layer before being fed to the transformation that bridges the second and the third hidden layers. Finally, the features of the last hidden layer are mapped to a single output, which is compared to the ground-truth associated with the input 3D query to compute the loss. Each linear transformation of the decoder, except the output one, is paired with the ReLU activation function.

A.5 Motivation Behind inr2vec Encoder Design

We designed inr2vec encoder with the goal of obtaining a good scalability in terms of memory occupation. Indeed, a naive solution to process the weights of an input INR would consist in an MLP encoder mapping the flattened vector of weights to the embedding of the desired dimension. However, such approach would require a huge amount of memory resources, since an input INR of 4 layers of 512 neurons would have approximately 800K parameters. Thus, an MLP encoder going from 800K parameters to an embedding space of size 1024 would already have a totality of ~800M parameters. We report in Tab. A.2 a detailed analysis of the parameters of our encoder with respect to the ones of an MLP encoder by varying the input INR dimension. As we can notice the MLP encoder does not scale well, making this kind of approach very expensive in practice, while inr2vec encoder scales gracefully to bigger input INRs.

A.6 Experimental Settings

We report here a detailed description of the settings adopted in our experiments.

INR hidden dim.	INR #layers	INR #params	#params inr2vec encoder	#params MLP encoder
512	4	∼800K	∼3M	∼800M
512	8	∼2M	∼3M	∼2B
512	12	∼3M	∼3M	∼3B
512	16	∼4M	∼3M	∼4B
1024	4	∼3M	∼3.5M	∼3B
1024	8	∼7M	∼3.5M	∼7.5B
1024	12	∼11M	∼3.5M	∼12B
1024	16	∼15M	∼3.5M	∼16B

TABLE A.2. **Number of parameters of inr2vec encoder.** Comparison between the number of parameters of inr2vec encoder and the number of parameters of a generic MLP encoder.

INRs fitting. In every experiment, we fit INRs on 3D discrete representations using MLPs having 4 hidden layers with 512 nodes each. We implement MLPs using sine as a periodic activation function, as proposed in [304]. The procedure adopted to fit a single MLP consists in querying it with 3D points sampled properly in the space surrounding the underlying shape. The MLP predicts a value for each query and it's trained by computing a loss function between the predicted value and the ground-truth value of the fitted implicit function (*i.e.*, udf for point clouds, sdf for meshes and occ for voxels). The set of training queries is prepared according to different strategies, depending on the nature of the discrete representation being fitted. For voxel grids, the set of possible queries consists of the 3D coordinates of all the centroids of the grid. For point clouds and meshes, instead, queries are sampled with different densities in the volume containing the fitted shape: indeed, for each shape, we prepare 500K queries by taking 250K points close to the surface, 200K points at a medium-far distance for the surface, 25K far from the surface and other 25K scattered uniformly in the volume. The queries coordinates are computed by adding gaussian noise to the points of the fitted point cloud or to points sampled uniformly from the fitted mesh surface. More precisely, close queries are computed with noise sampled from the normal distribution $\mathcal{N}(0, 0.001)$, medium-far queries with noise from $\mathcal{N}(0, 0.01)$, far queries with noise from $\mathcal{N}(0, 0.1)$. The uniformly scattered queries are just computed by sampling each of their coordinates from the uniform distribution $\mathcal{U}(-1, 1)$, being the considered shapes normalized in such volume. As for the ground-truth values, for voxels they consist simply in the occupied/empty label of the voxel associated to the query. For point clouds, for each query we compute its udf value by building a KDTree on the fitted point cloud and looking for the closest point to the considered query (we used the Pytorch3D [353] implementation of the KDTree algorithm). For meshes, finally, we compute the sdf of queries with the functions provided in the Open3D library [354][1]. For each of the considered modalities, at each step of the fitting procedure, we randomly sample 10K pairs of queries/ground-truth values from the precomputed ones, performing a total of 500 steps for each shape. Thanks to the procedure detailed in Appendix A.2, we are able to fit up to 16 multiple MLPs in parallel,

[1]http://www.open3d.org/docs/latest/tutorial/geometry/distance_queries.html

using Adam optimizer [355] with learning rate set to 1e-4. On a final note, we fixed the weights initialization of the MLPs to be always the same, as we observed empirically this to be key to convergence of inr2vec. This choice poses no limitation to the practical use of our framework and has also been adopted in recent works [297, 324].

inr2vec training. According to what is described in Sec. 12.1.1, during training our framework takes in input the weights of a given INR and is asked to reproduce the implicit function fitted by the INR on a set of predefined 3D queries. Such queries are prepared with the same strategies described in the previous paragraph and, similarly to what is done while fitting INRs, at each step the training loss for inr2vec is computed on 10K queries randomly sampled from a set of precomputed ones. In every experiment, we train inr2vec with AdamW optimizer [356], learning rate 1e-4 and weight decay 1e-2 for 300 epochs, one epoch corresponding to processing all the INRs that compose the considered dataset, processing at each training step a mini-batch of 16 INRs. During training, we select the best model by evaluating its reconstruction capability on a validation set of INRs. When training on INRs obtained from point clouds, we compare the ground-truth set of points with the ones reconstructed by inr2vec decoder. For voxels, we compare the input and the output grid by comparing the point clouds composed by the centroids corresponding to occupied voxels. As for what concerns meshes, we compare the clouds containing input and output vertices. In all cases, the reconstruction quality is evaluated by computing the Chamfer Distance between ground-truth and output point clouds, as defined in [332]. See Appendix A.3 of this document for details on how to sample discrete 3D representations from the implicit functions fitted by INRs and that inr2vec is trained to reproduce.

Shape classification. The classifier that we deploy on inr2vec embeddings is composed of three linear transformations, mapping sequentially the input embedding with 1024 features to vectors of size 512, 256 and, finally, to a vector with a number of values corresponding to the number of classes of the considered dataset. The final vector is then transformed to a probability distribution with the softmax function. We use 1D batch normalization and the ReLU activation function between the classifier linear transformations. In all experiments, the classifier is trained for 150 epochs, with AdamW optimizer [356] and weight decay 1e-2. The learning rate is scheduled according to the OneCycle strategy [128], with maximum learning rate set to 1e-4. At each training step, the classifier processes a mini-batch counting 256 embeddings. During training, we select the best model by computing the classification accuracy on a validation set of embeddings. The best model is used after training to compute the classification accuracy on the test set, obtaining the numbers reported in the tables.

Point cloud part segmentation. In order to tackle point cloud part segmentation starting from inr2vec embeddings, we adopt a decoder similar to the one that we use for reconstruction during inr2vec training. The part segmentation decoder, depicted in Fig. A.5, is fed with the positional encoding of a 3D query together with the embedding of an input INR and predicts

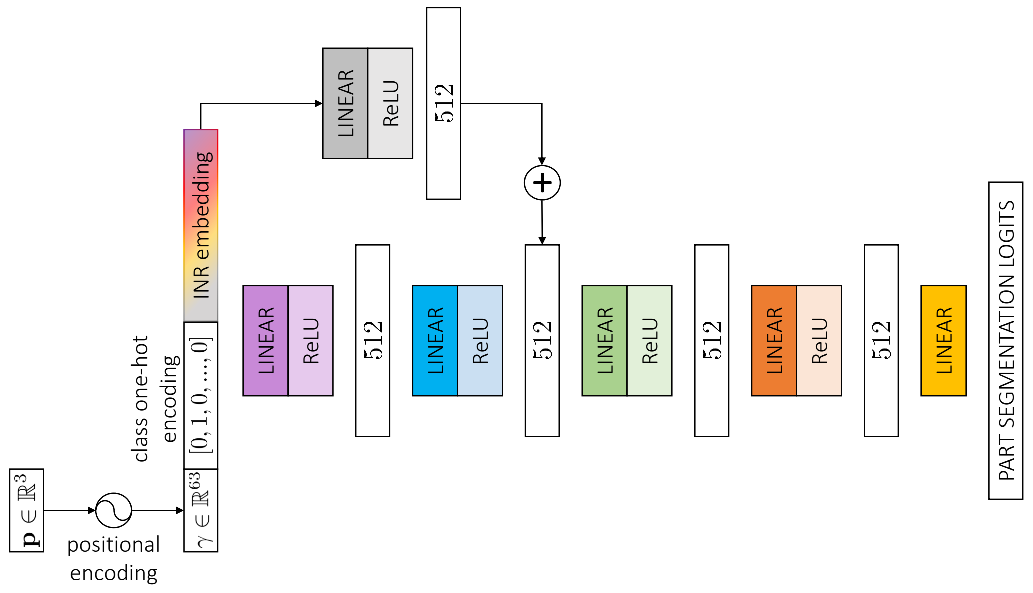


FIGURE A.5. **Part segmentation decoder.** We train a decoder to predict the part segmentation label of a given 3D query when conditioned on the embedding of the input INR and on the one-hot encoding of the INR class.

a K-dimensional vector of segmentation logits for the given query, with K representing the total number of parts of all the C available categories. Moreover, as done in previous work [160, 161, 167], we concatenate an additional C-dimensional vector to the input of the part segmentation decoder, conditioning the output of our decoder with the one-hot encoding of the input INR class. We conduct our experiments on the ShapeNet Part Segmentation dataset [334], which presents 16 categories labeled with two to five parts, for a total of 50 parts (*i.e.*, C=16 and K=50). According to a standard protocol [160, 161, 167], during training we compute the cross-entropy loss function on all the K logits predicted by our decoder, while, at test time, the final prediction is obtained considering only the subset of parts belonging to the specific class of the input INR. The part segmentation decoder is trained with the original point clouds available in the ShapeNet Part Segmentation dataset, where part labels are provided for each point of each cloud. At test time, though, we test both our decoder and the considered competitors on the point clouds reconstructed from the input INRs, since we want to simulate the scenario of 3D shapes being available exclusively in the form of INRs. Thus, the protocol to obtain a segmented point cloud starting from an input INR consists in reconstructing the cloud first (see Appendix A.3) and then in assigning a part label to each point of the reconstructed shape with our part segmentation decoder. When ground-truth labels are required to compute quantitative results, we obtain them by comparing the reconstructed cloud with the original one and assigning to each point of the reconstructed shape the part label of the closest point in the original cloud. Our part segmentation decoder is trained for 250 epochs with AdamW optimizer, OneCycle learning rate scheduling with maximum value set to 1e-4, weight decay equal to 1e-2 and mini-batches composed of 256 embeddings, each one paired with 3D queries from the original point clouds during training and from the ones reconstructed from the input INRs at test time. During training, we compute the class mIoU on the validation split and save the best model in order to compute the final metrics on the test set.

Shape generation. We perform unconditional shape generation by training Latent-GAN [335] to generate embeddings indistinguishable from the ones produced by inr2vec on a given dataset. This approach allows us to train a shape generation framework with the very same architecture to generate embeddings representing INRs with different underlying implicit functions, such as udf for the point clouds of ShapeNet10 and sdf for the models of cars provided by [301]. We conducted our experiments using the official implementation[2], setting all the hyperparameters to default. The generator network is implemented as a fully connected network with two layers and ReLU non linearity, that map an input noise vector with 128 values sampled from the normal distribution $\mathcal{N}(0, 0.2)$ to an intermediate hidden vector of the same dimension and then to the predicted embedding with 1024 values (we removed the final ReLU present in the original implementation). The discriminator is also a fully connected network, with three layers and ReLU non linearity. The first layer maps the embedding produced by the generator to a hidden vector with 256 values, which are then transformed by the second layer into a hidden vector with 512 values, that are finally used by the third layer, together with the sigmoid function, to predict the final score. According to the original implementation, we trained one separate Latent-GAN for each class of the considered datasets, using the Wasserstein objective with gradient penalty proposed in [357] and training each model for 2000 epochs.

Learning a mapping between inr2vec embedding spaces. The transfer function between inr2vec embedding spaces is implemented as a simple fully connected network, with 8 linear layers interleaved by 1D batch norm and ReLU activation functions. All the hidden features produced by the linear transformations present the same dimension of the input embedding, *i.e.*, 1024 values. The final linear layer predicts the output embedding, which is compared with the target one with a standard L2 loss. We train the transfer network with AdamW optimizer, constant learning rate and weight decay both set to 1e-4, stopping the training upon convergence, which we measure by comparing the shapes reconstructed by the predicted embeddings with the ground-truth ones on a predetermined validation split. Such validation metrics are used also to save the best model during training, which is finally evaluated on the test set.

[2]`https://github.com/optas/latent_3d_points`

	Point Cloud			Mesh	Voxels
Method	ModelNet40	ShapeNet10	ScanNet10	Manifold40	ShapeNet10
PointNet [160]	88.8	94.7	72.8	–	–
PointNet++ [161]	91.0	95.2	76.3	–	–
DGCNN [167]	91.6	94.0	75.1	–	–
MeshWalker [289]	–	–	–	90.6	–
Conv3DNet [258]	–	–	–	–	92.5
inr2vec	87.0	93.3	72.1	86.3	93.0

TABLE A.3. **Shape classification results.** We report here shape classification results when testing on the original discrete representations of the test sets instead of reconstructing them from the input INRs.

A.7 Implementation, Hardware and Timings

We implemented our framework with the PyTorch library, performing all the experiments on a single NVIDIA 3090 RTX GPU. We created an augmented version of each considered dataset, in order to obtain roughly ∼100K INRs, whose fitting requires around 4 days in the current implementation. Training inr2vec requires another 48 hours, while all the networks adopted to perform the downstream tasks on inr2vec embeddings can be trained in few hours.

A.8 Testing on Original Discrete 3D Representations

In the experiments "Shape classification" and "Point cloud part segmentation", we evaluated the competitors on the 3D discrete representations reconstructed from the INRs fitted on the test sets of the considered datasets, since these would be the only data available at test time in a scenario where INRs are used to store and communicate 3D data. For completeness, we report here the scores achieved by the baselines when tested on the original discrete representations, without reconstructing them from the input INRs. Such results are presented in Tab. A.3 for shape classification and in Tab. A.4 for part segmentation. We report in the tables also the results obtained with our framework: they are the same reported in Tab. 13.2 for what concerns shape classification, since our classifier processes exclusively inr2vec embeddings, while they are different for part segmentation, as we use as query points for our segmentation decoder those from the discrete point clouds reconstructed from input INRs in Tab. 13.3 while we use those from the original point clouds in the experiment reported here, as done for the competitors. The results reported in the tables show limited differences, either positive or negative, with the ones presented in Sec. 13.3, mostly within the range of variations due to the inherent stochasticity of training. There are few larger differences, like DGCNN on ModelNet40 (+1.7 when tested on the original discrete representations) or on ScanNet (-1.1 when tested on the original discrete representations), whose difference in sign however suggests neither of the two settings is clearly superior to the other.

Method	instance mIoU	class mIoU	airplane	bag	cap	car	chair	earphone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skateboard	table
PointNet [160]	83.0	78.8	80.5	77.9	78.3	74.4	89.0	68.3	90.1	82.2	80.7	94.7	63.1	91.7	79.3	58.2	72.7	81.0
PointNet++ [161]	84.4	82.8	81.7	86.5	85.2	78.6	90.2	77.9	91.2	84.4	83.2	95.4	72.0	94.6	83.3	64.2	75.6	80.9
DGCNN [167]	84.3	81.4	81.6	82.2	80.9	75.7	90.7	80.9	90.2	86.9	82.6	94.8	64.8	92.8	81.0	60.6	74.7	81.8
inr2vec	80.5	71.1	79.5	72.9	72.3	70.7	87.4	64.1	89.4	81.6	76.5	94.5	59.3	92.4	78.4	53.5	67.5	77.3

TABLE A.4. **Part segmentation results.** In this table we present part segmentation results when testing on the original discrete representations of the test sets instead of reconstructing them from the input INRs. We report the IoU for each class, the mean IoU over all the classes (class mIoU) and the mean IoU over all the instances (instance mIoU).

A.9 Alternative Architecture for inr2vec

As reported in Sec. 12.1.1, inr2vec encoder takes in input the weights of an INR reshaped in a suitable way, discarding the parameters of the first and of the last layers. In this section we consider the possibility of processing all the weights of the input INR, including the input/output ones. To this end, one must properly arrange the input/output parameters since they feature different dimensionality from the ones of the hidden layers and cannot be seamlessly stacked together with them. More specifically, the first layer of an INR consists in a matrix of weights $\mathbf{W}_{in} \in \mathbb{R}^{H \times D}$ and in a vector of biases $\mathbf{b}_{in} \in \mathbb{R}^{H \times 1}$, with H being the dimension of the hidden features of the INR and D being the dimension of the inputs (*i.e.*, 3 in our case of 3D coordinates). The output layer, instead, is responsible of transforming the final vector of hidden features to the predicted output, which is always a single value in the cases considered in our experiments (*i.e.*, udf, sdf and occ). Thus, the last layer presents a matrix of weights $\mathbf{W}_{out} \in \mathbb{R}^{1 \times H}$ and single bias $\mathbf{b}_{out}$. In order to include the input/output parameters in the matrix $\mathbf{P}$ presented in input to inr2vec encoder (see Sec. 12.1.1), $\mathbf{W}_{in}$ needs to be transposed, obtaining a matrix with shape $3 \times H$, $\mathbf{b}_{in}$ is transposed as done also for the biases of all the other layers, $\mathbf{W}_{out}$ doesn't need any manipulation and we decided to repeat the single-valued $\mathbf{b}_{out}$ H times. In this section, we compare the formulation presented in Sec. 12.1.1 (reported as "hidden layers") with the one proposed here (reported as "all layers"), looking at the reconstruction capabilities of the two variants of our framework when trained on ModelNet40. In Tab. A.5, we report both the F-score [341] and the Chamfer Distance (CD) [332] between the clouds used to obtain the INRs presented in input to inr2vec and the ones reconstructed from inr2vec embeddings, while in Fig. A.6 we show the same comparison from a qualitative perspective. Results show that processing all the INR weights doesn't produce any significant difference with respect to ingesting only the weights of the hidden layers. However, the latter variant provides a slight advantage in terms of F-score, simplicity and processing time, motivating our choice to adopt it as formulation for inr2vec.

Architecture	F-Score ↑	CD (mm) ↓
hidden layers	57.41	3.1
all layers	56.76	3.1

TABLE A.5. **Quantitative comparison between alternative inr2vec architectures**. We compare the reconstruction capability of inr2vec when processing only the weights of the hidden layers ("hidden layers") or all the weights ("all layers") of the input INRs.

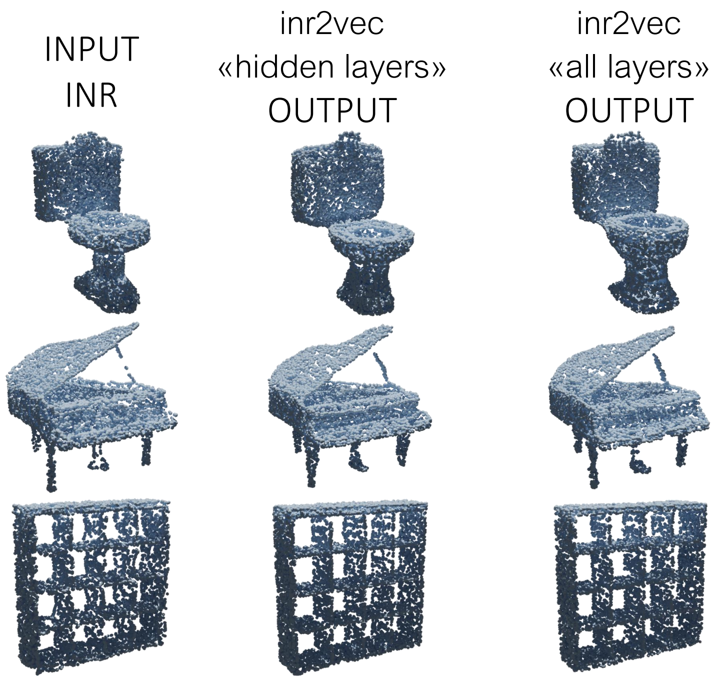


FIGURE A.6. **Qualitative comparison between alternative inr2vec architectures**. We compare the reconstruction capability of inr2vec when processing only the weights of the hidden layers ("hidden layers") or all the weights ("all layers") of the input INRs.

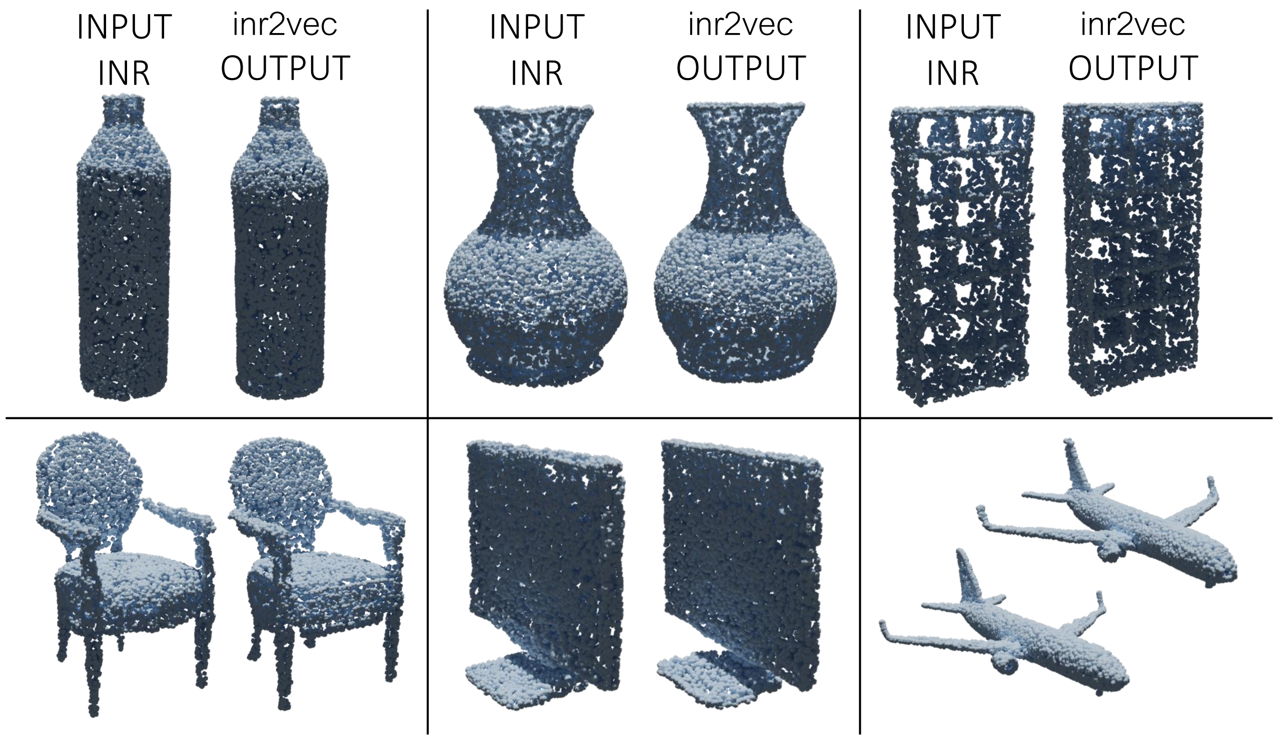


FIGURE A.7. **inr2vec reconstructions when dealing with INRs fitted on point clouds.** Comparison between discrete shapes reconstructed from the INRs presented in input to inr2vec (“INPUT INR”) and the ones reconstructed from inr2vec embeddings (“inr2vec OUTPUT”).

A.10 Additional Qualititative Results

We report here additional qualitative results. In Fig. A.7, Fig. A.8 and Fig. A.9 we show some comparisons between the discrete shapes reconstructed from input INRs and the ones reconstructed from inr2vec embeddings. Fig. A.10, Fig. A.11 and Fig. A.12 present smooth interpolations between inr2vec embeddings. In Fig. A.13 and Fig. A.14 we propose additional qualitative results for the point cloud retrieval experiments. Fig. A.15 shows qualitative results for point cloud part segmentation for all the classes of the ShapeNet Part Segmentation dataset. Fig. A.16, Fig. A.17 and Fig. A.18 report shapes generated with Latent-GANs [335] trained on inr2vec embeddings. Finally, Fig. A.19 and Fig. A.20 present additional qualitative results for the experiments of point cloud completion and surface reconstruction, tackled by learning a mapping between inr2vec latent spaces.

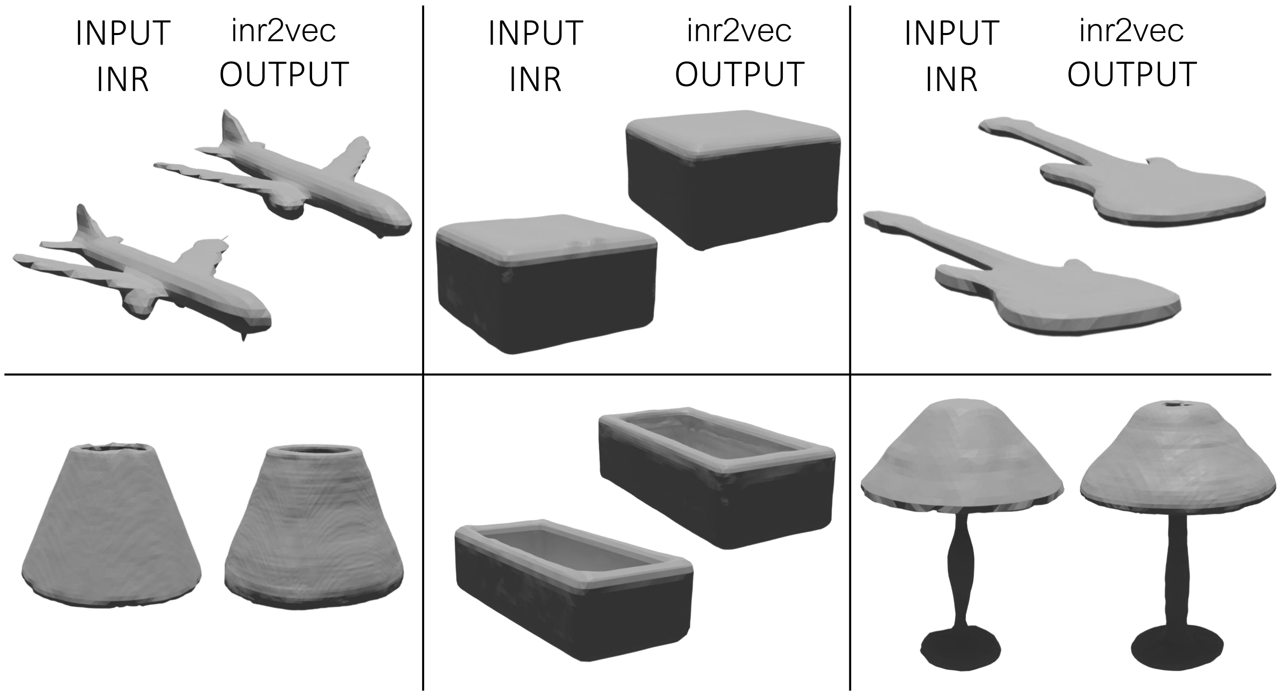


FIGURE A.8. **inr2vec reconstructions when dealing with INRs fitted on meshes.** Comparison between discrete shapes reconstructed from the INRs presented in input to inr2vec ("INPUT INR") and the ones reconstructed from inr2vec embeddings ("inr2vec OUTPUT").

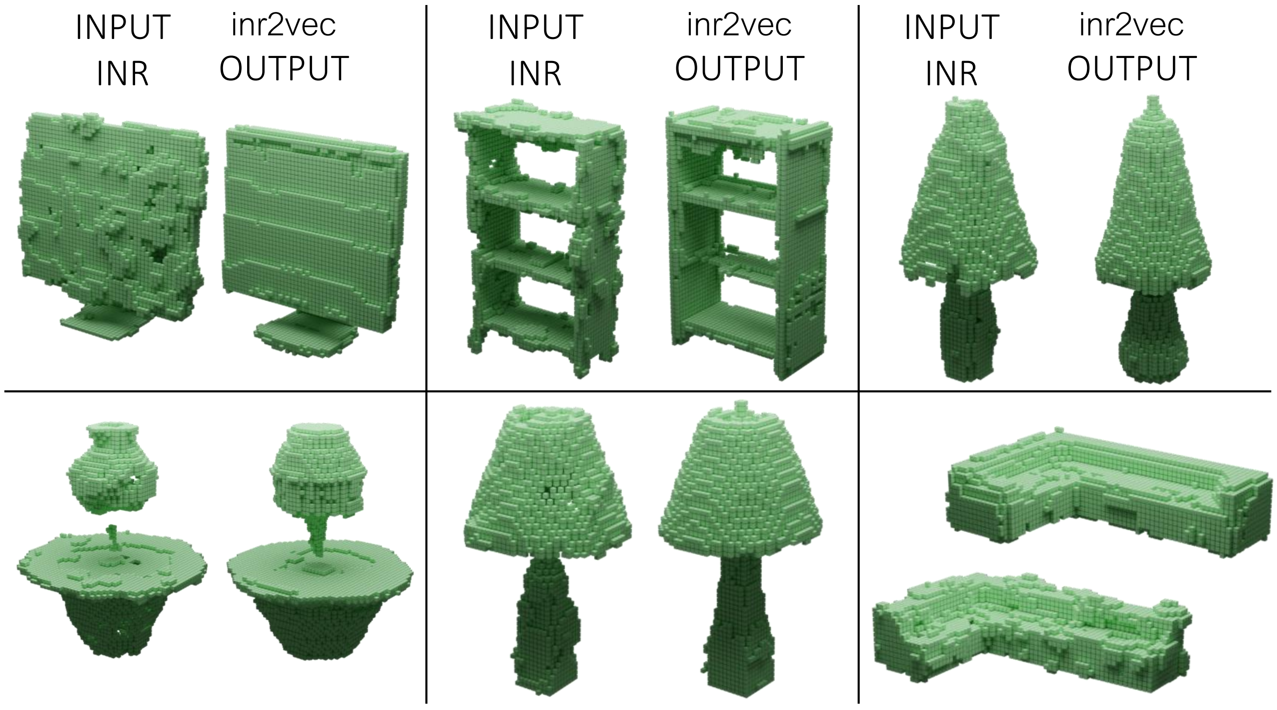


FIGURE A.9. **inr2vec reconstructions when dealing with INRs fitted on voxel grids.** Comparison between discrete shapes reconstructed from the INRs presented in input to inr2vec ("INPUT INR") and the ones reconstructed from inr2vec embeddings ("inr2vec OUTPUT").

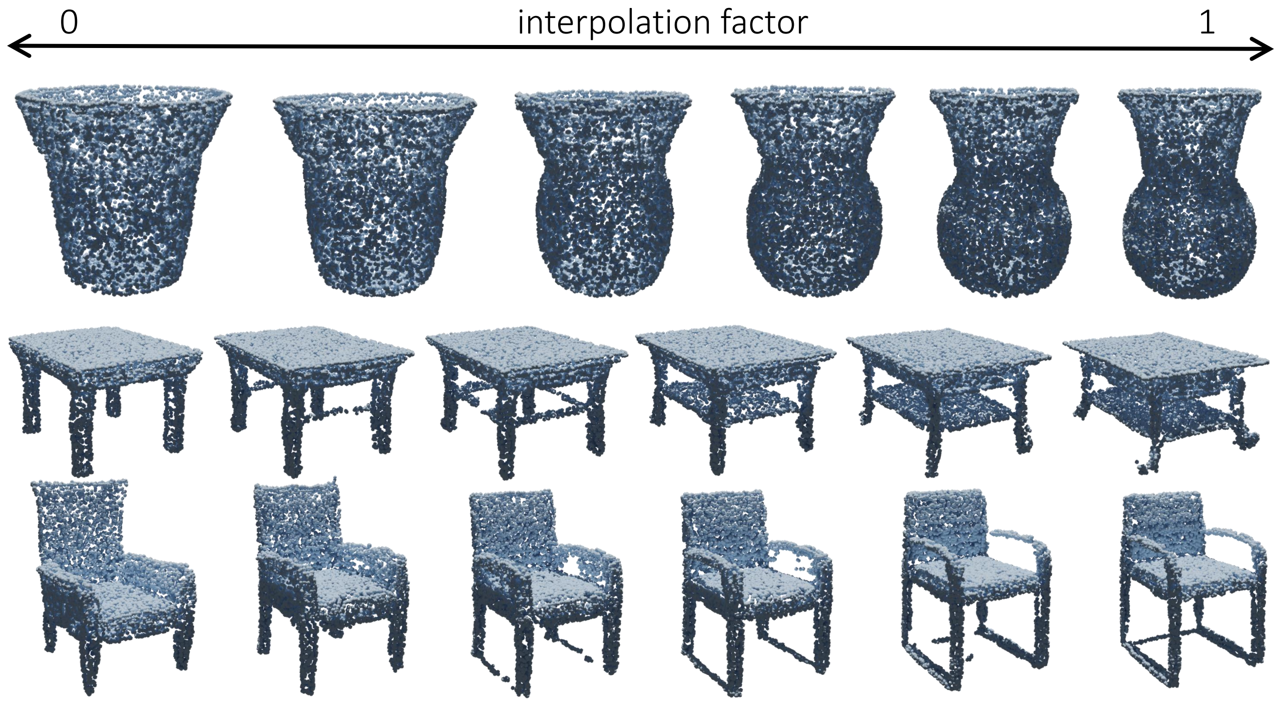


FIGURE A.10. **inr2vec latent space interpolation.** Given two inr2vec embeddings obtained from INRs fitted on point clouds, it is possible to linearly interpolate between them, producing new embeddings that represent unseen INRs of plausible shapes.

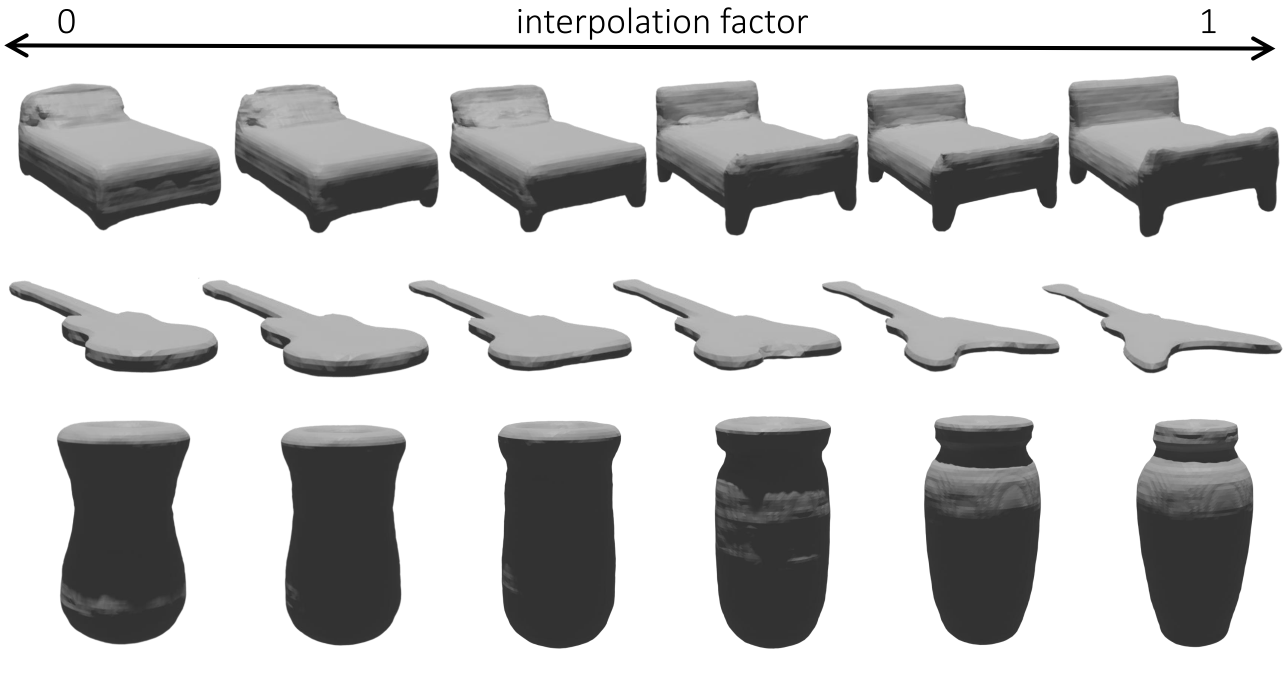


FIGURE A.11. **inr2vec latent space interpolation.** Given two inr2vec embeddings obtained from INRs fitted on meshes, it is possible to linearly interpolate between them, producing new embeddings that represent unseen INRs of plausible shapes.

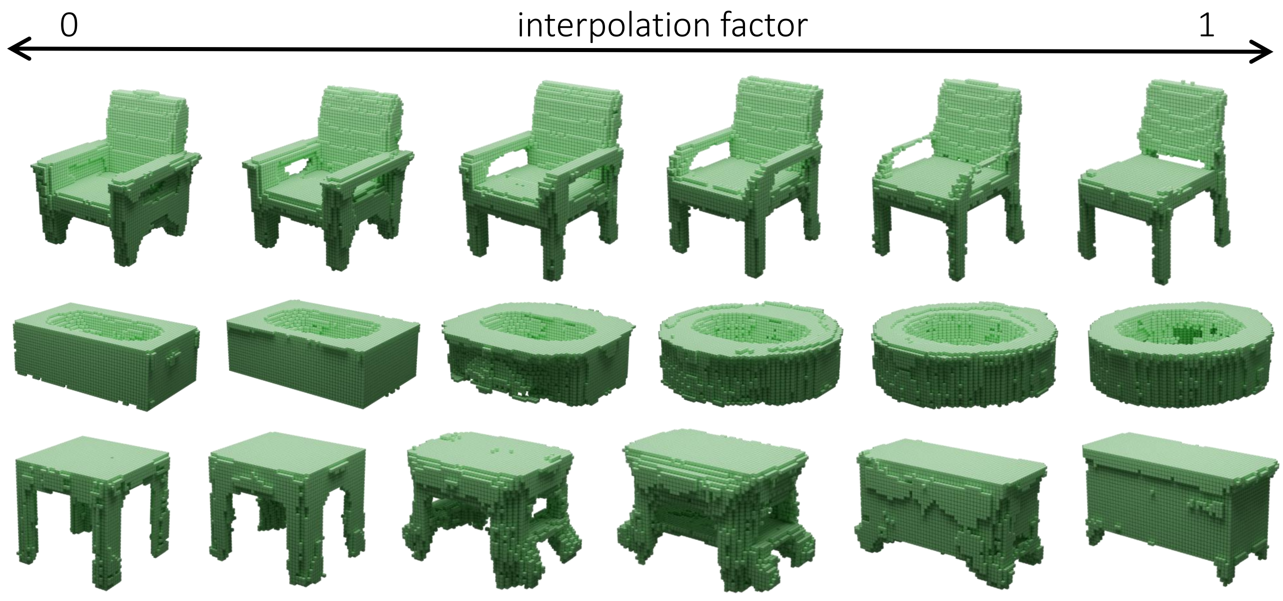


FIGURE A.12. **inr2vec latent space interpolation.** Given two inr2vec embeddings obtained from INRs fitted on voxel grids, it is possible to linearly interpolate between them, producing new embeddings that represent unseen INRs of plausible shapes.

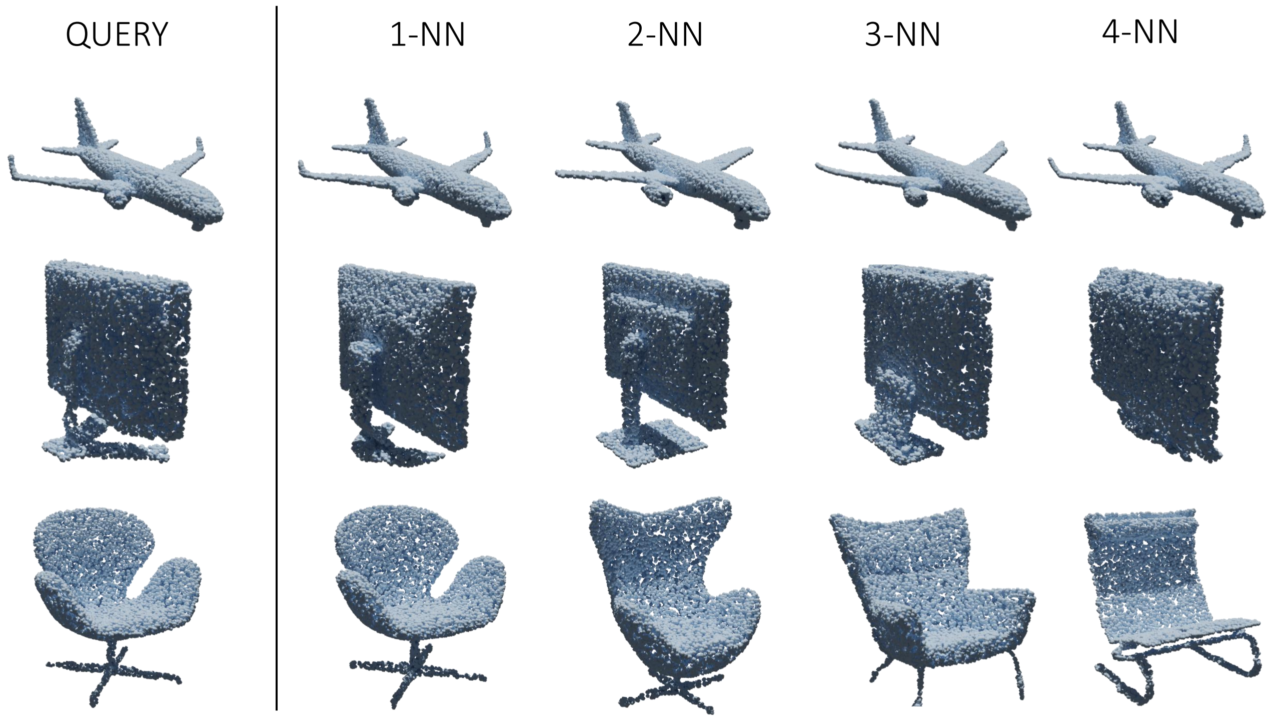


FIGURE A.13. **Point cloud retrieval (ModelNet40).** Qualitative results of the point cloud retrieval experiment conducted on inr2vec latent space. We show the discrete shape reconstructed from the query INR on the left and the discrete clouds reconstructed from the closest inr2vec embeddings in the columns 2-5.

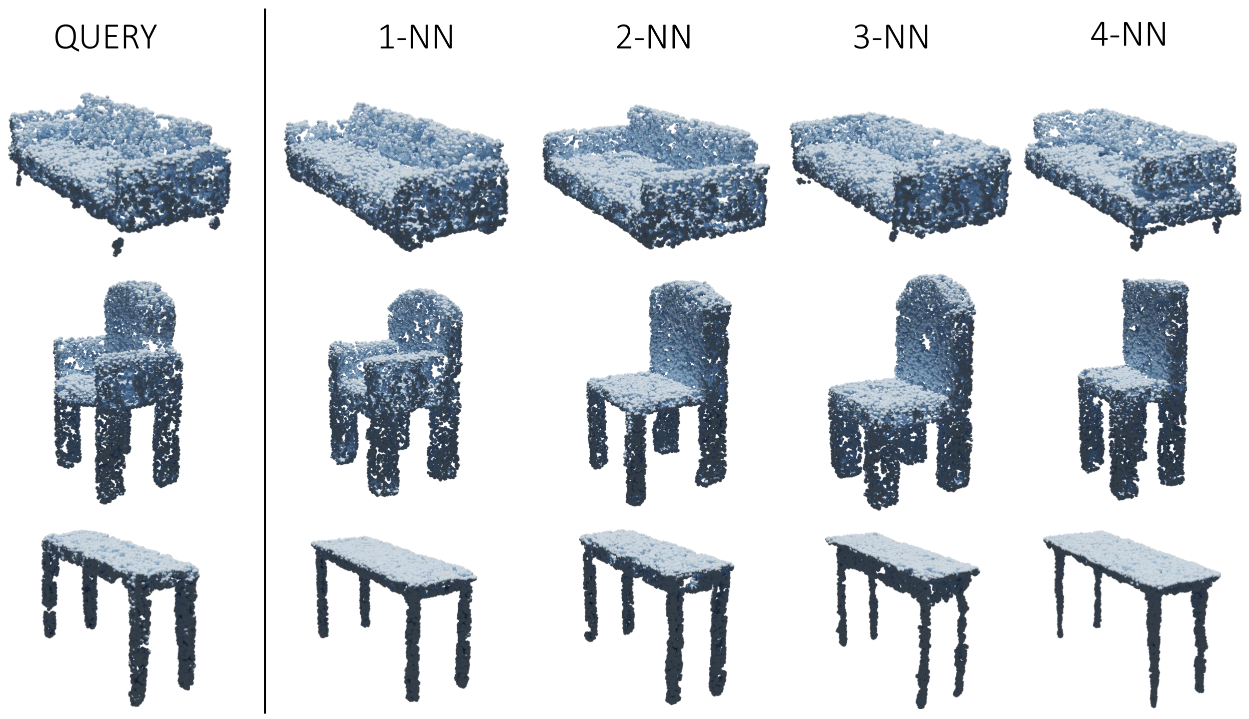


FIGURE A.14. **Point cloud retrieval (ShapeNet10).** Qualitative results of the point cloud retrieval experiment conducted on inr2vec latent space. We show the discrete shape reconstructed from the query INR on the left and the discrete clouds reconstructed from the closest inr2vec embeddings in the columns 2-5.

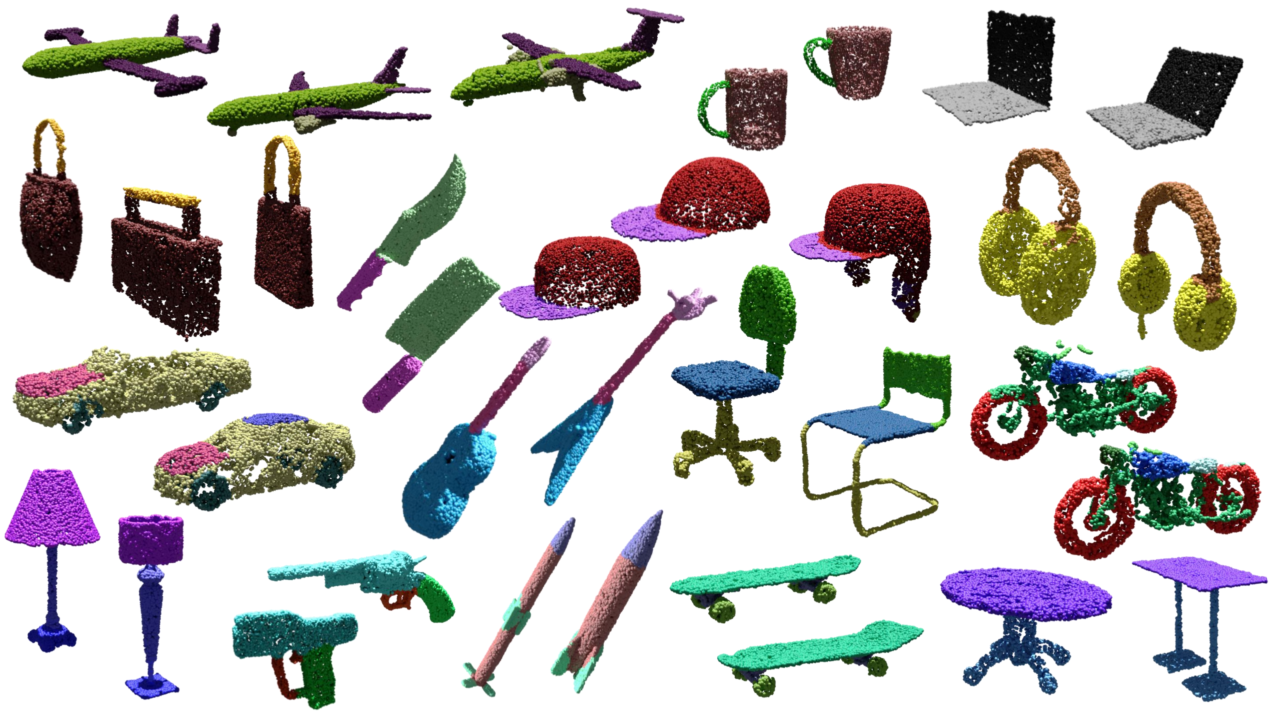


FIGURE A.15. **Point cloud part segmentation.** Qualitative results of the part segmentation experiment conducted with our segmentation decoder conditioned on inr2vec embeddings.

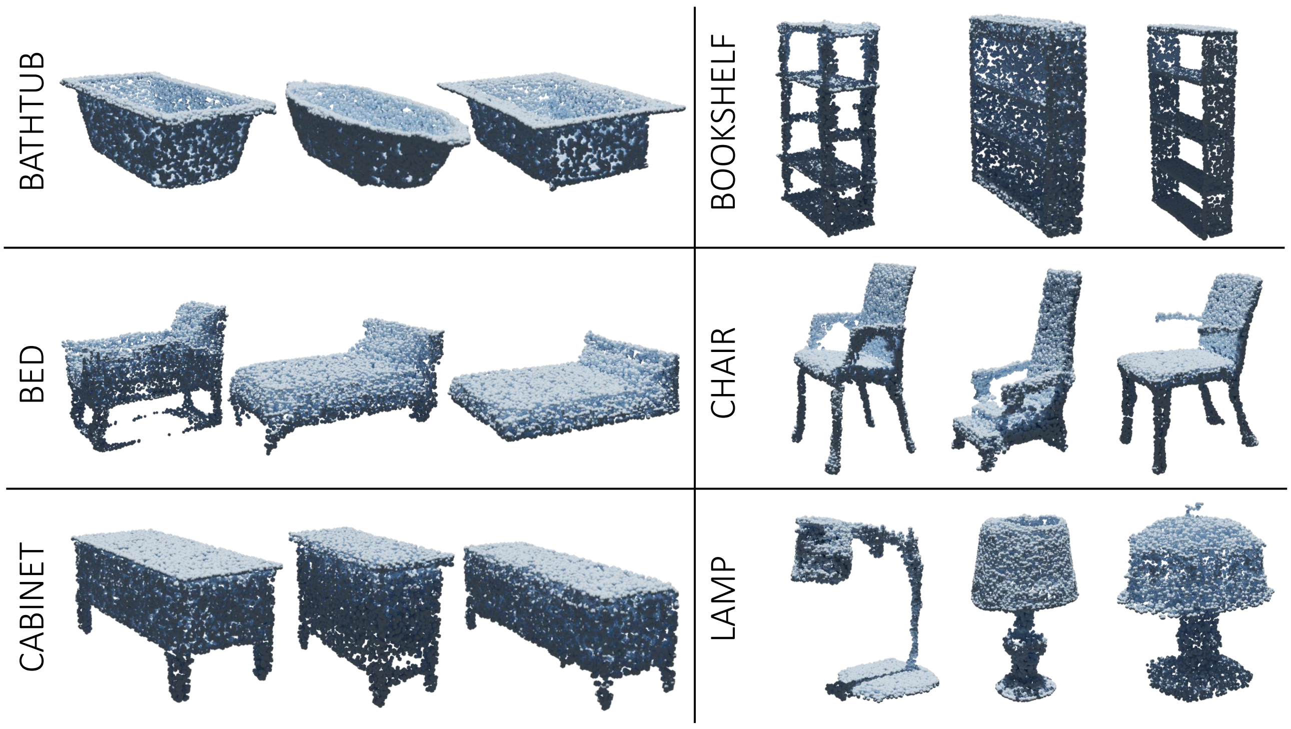


FIGURE A.16. **Shape generation (point clouds).** We show point clouds reconstructed from embeddings generated by a Latent-GAN trained on inr2vec embeddings (one model for each class).

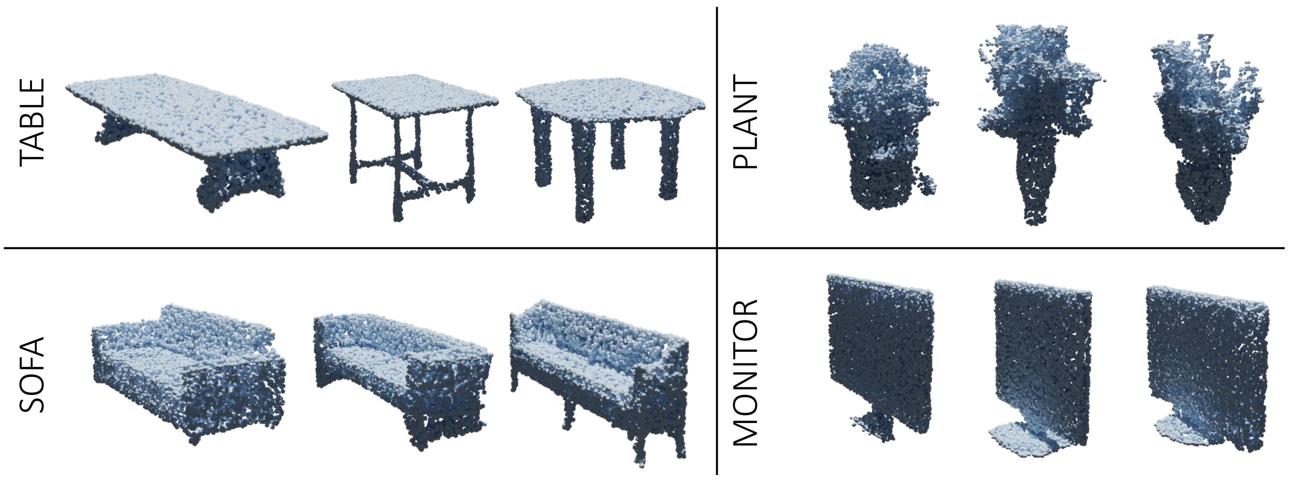


FIGURE A.17. **Shape generation (point clouds).** We show point clouds reconstructed from embeddings generated by a Latent-GAN trained on inr2vec embeddings (one model for each class).

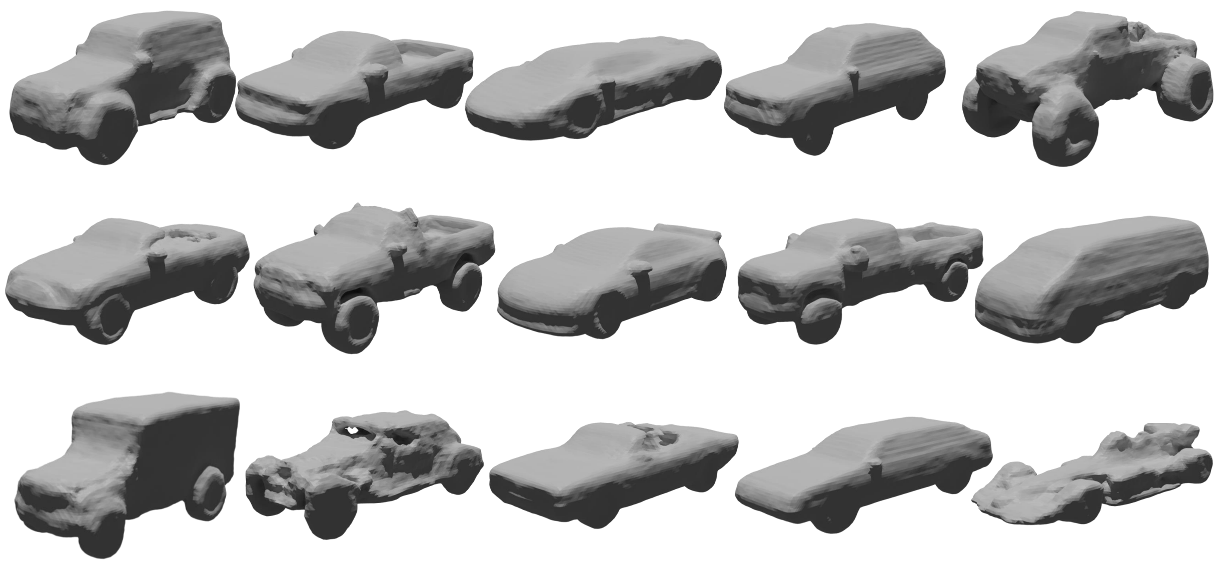


FIGURE A.18. **Shape generation (meshes).** We show meshes reconstructed from embeddings generated by a Latent-GAN trained on inr2vec embeddings.

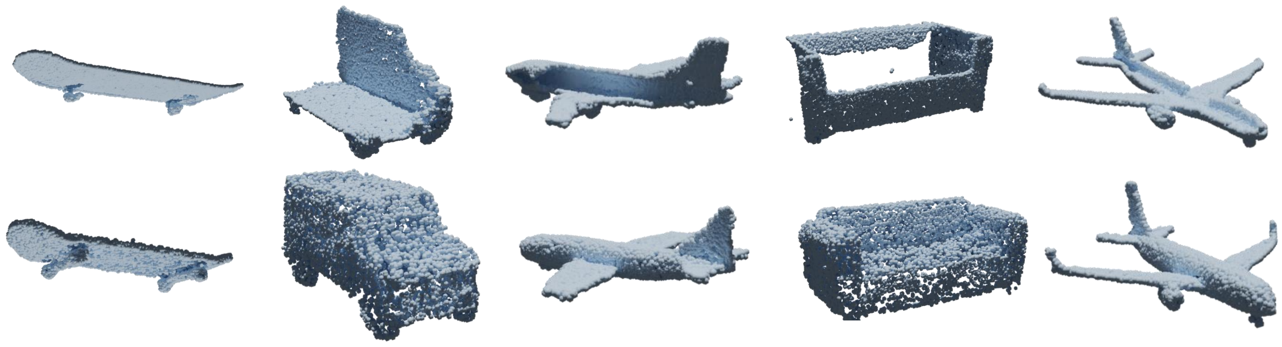


FIGURE A.19. **Point cloud completion.** Qualitative results of the point cloud completion experiment conducted with a transfer network that learns a mapping between inr2vec latent spaces.

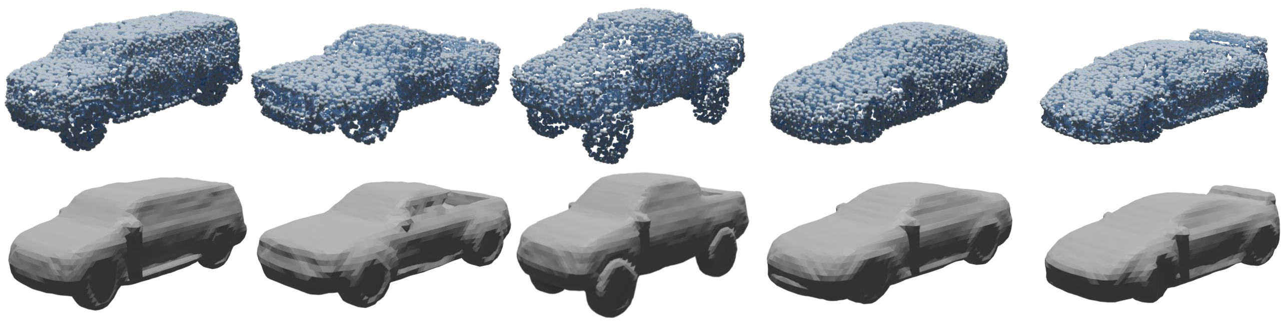


FIGURE A.20. **Surface reconstruction.** Qualitative results of the surface reconstruction experiment conducted with a transfer network that learns a mapping between inr2vec latent spaces.

A.11 Effectiveness of Using the Same Initialization for INRs

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

We report here an experiment to assess if order of data, or other sources of randomness arising while fitting INRs, do affect the repeatability of the embeddings computed by inr2vec. We fitted 10 INRs on the same discrete shape for 4 different chairs, *i.e.*, 40 INRs in total. Then, we embed all of them with the pretrained inr2vec encoder and compute the L2 distance between all pairs of embeddings. The block structure of the resulting distance matrix (see Fig. A.21) highlights how, under the assumption of shared initialization and hyperparameters, inr2vec is repeatable across multiple fittings.

Seeking for a proof with a stronger theoretical foundation, we turn our attention to the recent work *git re-basin* [340], where authors show that the loss landscape of neural networks contain (nearly) a single basin after accounting for all possible permutation symmetries of hidden units. The intuition behind this finding is that, given two neural networks that were trained with equivalent architectures but different random initializations, data orders, and potentially different hyperparameters or datasets, it is possible to find a permutation of the networks weights such that when linearly interpolating between their weights, all intermediate models enjoy performance similar to them – a phenomenon denoted as *linear mode connectivity*.

Intrigued by this finding, we conducted a study to assess whether initializing INRs with the same random vector, which we found to be key to inr2vec convergence, also leads to linear mode connectivity. Thus, given one shape, we fitted it with two different INRs and then we interpolated linearly their weights, observing at each interpolation step the loss value obtained by the *interpolated* INR on the same batch of points. For each shape, we repeated the experiment twice, once initializing the INRs with different random vectors and once initializing them with the same random vector.

The results of this experiment are reported for four different shapes in Fig. A.22. It is possible to note that, as shown by the blue curves, when interpolating between INRs obtained from the same weights initialization, the loss value at each interpolation step is nearly identical to those of the boundary INRs. On the contrary, the red curves highlight how there is no linear mode connectivity at all between INRs obtained from different weights initializations.

[340] propose also different algorithms to estimate the permutation needed to obtain linear mode connectivity between two networks. We applied the algorithm proposed in their paper in Section 3.2 (*Matching Weights*) to our INRs and observed the resulting permutations. Remarkably, when applied to INRs obtained from the same weights initialization, the retrieved permutations are identity matrices, both when the target INRs represent the same shape and when they represent different ones. The permutations obtained for INRs obtained from different initializations, instead, are far from being identity matrices.

All these results favor the hypothesis that our technique of initializing INRs with the same random vector leads to linear mode connectivity between different INRs. We believe that the possibility of performing meaningful linear interpolation between the weights occupying

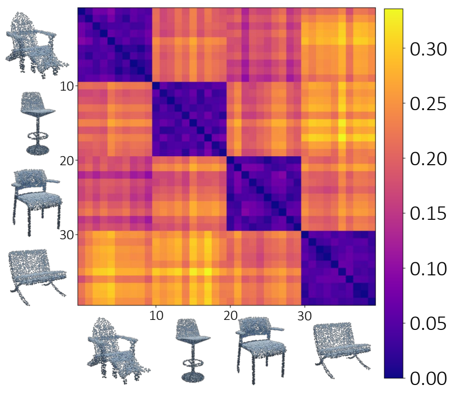


FIGURE A.21. **L2 distances between inr2vec embeddings.** For each shape, we fit 10 INRs starting from the same weights initialization (40 INRs in total). Then we plot the L2 distances between the embeddings obtained by inr2vec for such INRs.

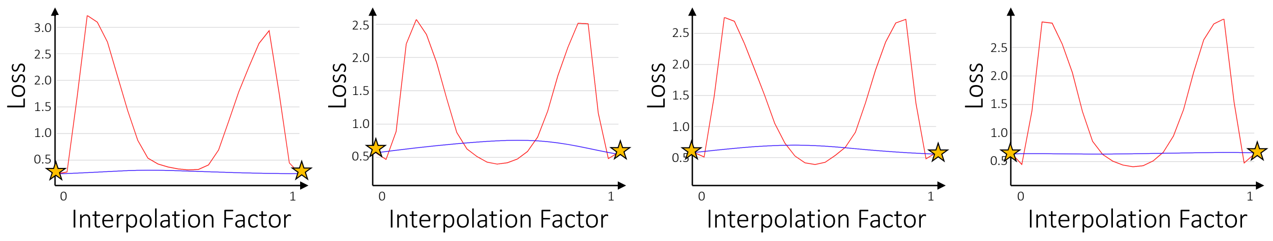


FIGURE A.22. **Linear mode connectivity study.** Each plot shows the variation of the loss function over the same batch of points when interpolating between two INRs representing the same shape. The red line describes the interpolation between INRs initialized differently, while the blue line shows the same interpolation between INRs initialized from the same random vector. The yellow stars represent the loss value of the boundary INRs.

the same positions across different INRs can be interpreted by considering corresponding weights as carrying out the same role in terms of feature detection units, explaining why the inr2vec encoder succeeds in processing the weights of our INRs.

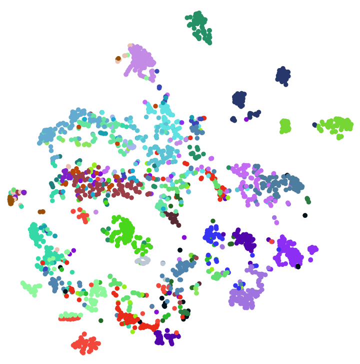
This intuition can be also combined with the finding of another recent work [358], that shows how the expressive power of SIRENs is restricted to functions that can be expressed as a linear combination of certain harmonics of the first layer, which thus serves as basis for the range of learnable functions.

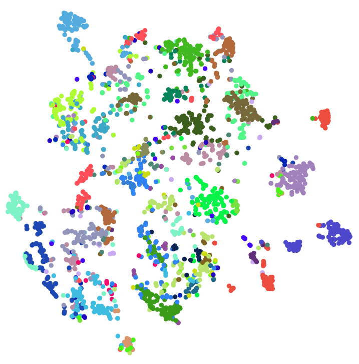
As stated above, the evidence of linear mode connectivity between INRs obtained from the same initialization leads us to believe that the weights of the first layer extract the same features across different INRs. Thus, we can think of using the same random initialization as a way to obtain the same basis of harmonics for all our INRs. We argue that this explains why it is possible to remove the first layer of the INRs presented in input to inr2vec (as empirically proved in Appendix A.9): if the basis is always the same, it is sufficient to process the layers from the second onwards, that represent the coefficients of the basis harmonics combination, as described in

A.12 t-SNE Visualization of inr2vec Latent Space

We provide in Fig. A.23 the t-SNE visualization of the embeddings produced by inr2vec when presented with the test set INRs of three different datasets. Fig. A.23a shows this visualization for INRs representing the point clouds from ModelNet40, Fig. A.23b for INRs representing meshes from Manifold 40, and Fig. A.23c for INRs obtained from the voxelized shapes in ShapeNet10.

The supervision signal adopted during the training of our framework does not entail any kind of constraints with respect to the organization of the learned latent space. Indeed, this was not necessary for our ultimate goal – *i.e.*, performing downstream tasks on the produced embeddings. However, it is interesting to observe from the t-SNE plots that inr2vec favors spontaneously a semantic arrangement of the embeddings in the learned latent space, with


(A) ModelNet40 (points)


(B) Manifold40 (meshes)

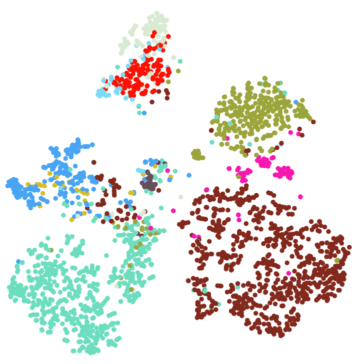

(C) ShapeNet10 (voxels)

FIGURE A.23. **t-SNE visualizations of inr2vec latent spaces.** We plot the t-SNE components of the embeddings produced by inr2vec on the test sets of three datasets, ModelNet40 (left), Manifold40 (center) and Shapenet10 (right). Colors represent the different classes of the datasets.

INRs representing objects of the same category being mapped into close positions – as shown by the colors representing the different classes of the considered datasets.

A.13 Ablation on INRs Size

Fig. A.24 reports a study that we conducted to determine the size of the INRs adopted throughout our experiments. More specifically, we considered three alternatives of SIREN, all featuring 4 hidden layers but different number of hidden features, namely 128, 256 and 512 respectively.

In the figure we show how the three SIREN variants perform in terms of being able of representing properly the underlying signal, which in this example is the orange plane on the left. Since we needed to create datasets comprising a huge number of INRs, we considered both the quality of the representation as well as the number of steps of the fitting procedure, with the goal of finding the best trade-off between quality and fitting time.

The results presented in the figure highlight how a SIREN with 512 hidden features can learn to represent properly the input shape in just 600 steps, while the other variants either take longer (as in the case of 256 hidden features) or not obtain at all the same quality (when using 128 hidden features).

This experiment enabled us to draw the conclusion that a SIREN with 4 hidden layers and 512 hidden features is the proper tool to obtain a good quality INR in short time.

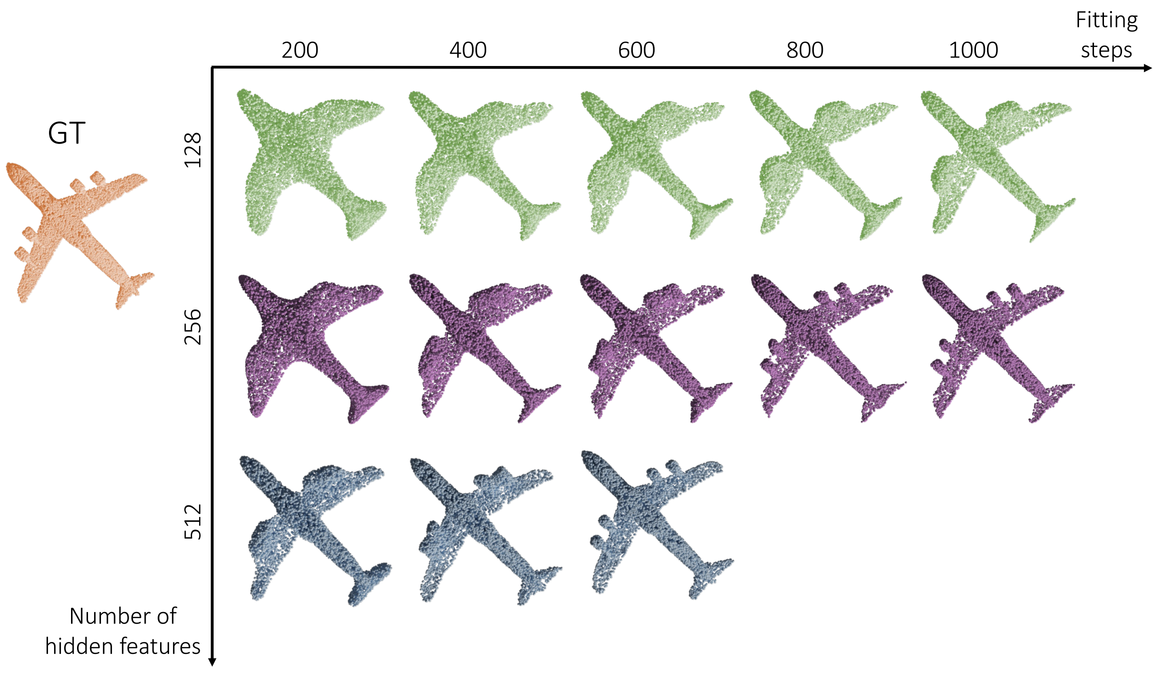


FIGURE A.24. **Comparison between different hidden sizes for INRs.** We report the fitting steps needed to obtain a good representation for INRs featuring different number of hidden features.

A.14 Shape Retrieval and Classification on DeepSDF Latent Codes

In Appendix A.1 we show that frameworks that adopt a shared network to produce INRs struggle to obtain a good representation quality, while individual INRs do not suffer of this problem by design. In this section, we go one step further and consider the possibility of peforming downstream tasks on the latent codes produced by DeepSDF [296].

In particular, we trained DeepSDF to fit the UDFs of our augmented version of ModelNet40, composed of ∼100K point clouds. For a fair comparison, we set the dimension of DeepSDF latent codes to 1024 – *i.e.*, the same used for inr2vec embeddings. Then we performed the experiments of shape retrieval and classification using DeepSDF latent codes, with the same settings presented in Sec. 13.3 for our framework.

The results reported in Tab. A.6 highlight that the poor representation quality obtained with DeepSDF – and shown to be an intrinsic problem with shared network frameworks in Appendix A.1 – has a detrimental effect on the quantitative results, proving once again that INR frameworks based on a shared network cannot be deployed as tool to obtain and process INRs effectively.

	ModelNet40		
Method	mAP@1	mAP@5	mAP@10
PointNet [160]	80.1	91.7	94.4
PointNet++ [161]	85.1	93.9	96.0
DGCNN [167]	83.2	92.7	95.1
inr2vec	81.7	92.6	95.1
DeepSDF	69.8	85.4	89.8

	ModelNet40
Method	Accuracy
PointNet [160]	88.8
PointNet++ [161]	89.7
DGCNN [167]	89.9
inr2vec	87.0
DeepSDF	64.9

TABLE A.6. **Comparison between inr2vec and DeepSDF.** We report results in shape retrieval (left) and shape classification (right) when using standard baselines, inr2vec embeddings or DeepSDF latent codes.

A.15 Shape Generation: Additional Comparison

In Fig. 13.8b we show a qualitative comparison between shapes generated with our framework (Sec. 13.3 - *Learning a mapping between inr2vec embedding spaces.*) and with competing methods, *i.e.*, SP-GAN [336] for point clouds and OccupancyNetworks [301] for meshes.

In Fig. A.25 we extend this comparison, by presenting samples obtained with our formulation applied to the voxelized chairs of ShapeNet10 and comparing them with samples produced by two additional methods that learn a manifold of individual INRs, namely GEM [359] and GASP [360], for which we used the original source code released by the authors. To generate the figure, despite the sampled shapes being voxel grids, we adopt the same procedure used by GEM and GASP and reconstruct meshes by applying Marching Cubes to extract the 0.5 isosurface.

Fig. A.25 show that all the considered methods can generate samples with a good variety in terms of geometry. However, it is possible to observe how the qualitative comparison favors the shape generated with inr2vec, which appear smoother than the ones generated by GASP and less noisy than the samples produced by GEM.

A.16 INR Classification Time: Extended Analysis

We report here the extended analysis of the inference times reported in Fig. 13.6, where we present the classification inference time needed to process udf INRs by standard point cloud baselines – PointNet [160], PointNet++ [161] and DGCNN [167] – and by inr2vec encoder paired with the fully-connected network that we adopt to classify the embeddings (see **??**).

The scenario that we had in mind while designing inr2vec is the one where INRs are the only medium to represent 3D shapes, with discrete point clouds not being available. Thus, in Fig. 13.6 for PointNet, PointNet++ and DGCNN we report the inference time including the time spent to reconstruct the discrete cloud from the input INR. In Fig. A.26 and Tab. A.7, for the sake of completeness, we report also the baselines inference times assuming the availability of discrete point clouds, stressing however that this is unlikely if INRs become a standalone format to represent 3D shapes.

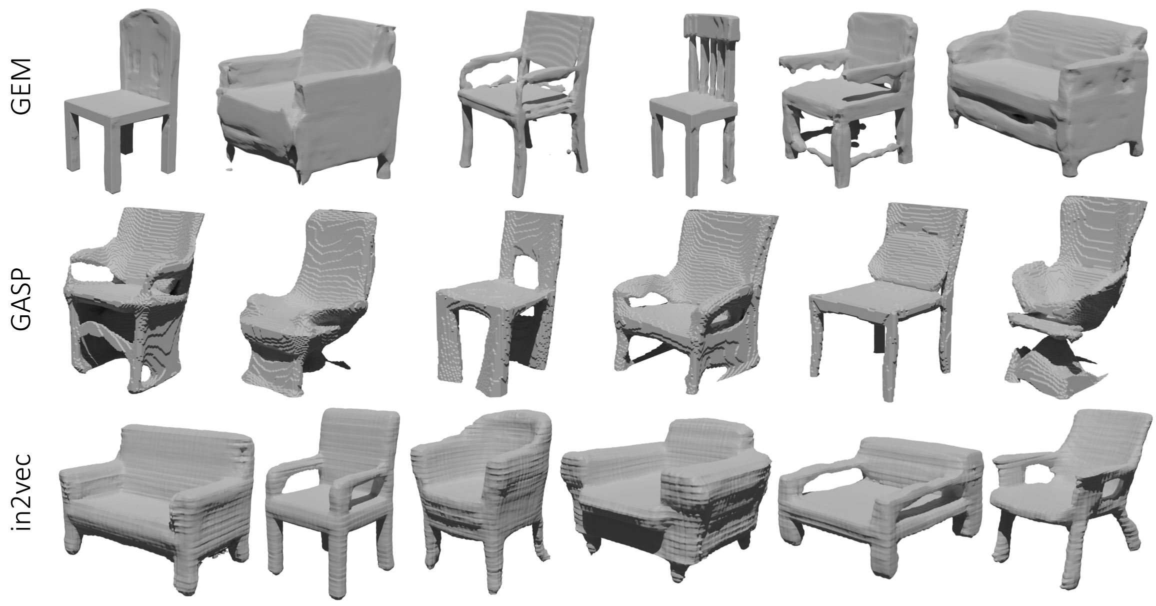


FIGURE A.25. **Shape generation: qualitative comparison.** We show samples generated with GEM [359], GASP [360] and with our method.

	Inference Time (seconds)			
Method	2048 pts	16K pts	32K pts	64K pts
PointNet	**0.001**	0.002	0.003	0.007
PointNet*	0.171	1.315	2.609	5.230
PointNet++	0.013	0.026	0.034	0.036
PointNet++*	0.185	1.293	2.672	5.287
DGCNN	0.158	1.285	4.788	19.26
DGCNN*	0.325	2.612	7.426	24.436
inr2vec	**0.001**	**0.001**	**0.001**	**0.001**

TABLE A.7. **Time required to classify INRs encoding udf.** All the times are computed on a gpu NVidia RTX 2080 Ti. * indicates that the time to reconstruct the discrete point cloud from the INR is included.

The numbers plotted in Fig. A.26 and reported in Tab. A.7 show clearly that our framework presents a big advantage with respect to the competitors. Indeed, by processing directly INRs – where the resolution of the underlying signal is theoretically infinite – inr2vec can classify INRs representing point clouds with different numbers of points with a constant inference time of 0.001 seconds.

The considered baselines, instead, are negatively affected by the increasing resolution of the input point clouds. While the inference time of PointNet and PointNet++ is still affordable even when processing 64K points, DGCNN gets drastically slow already at 16K points. Furthermore, if point clouds need to be reconstructed from the input INRs, the resulting inference time becomes prohibitive for all three baselines.

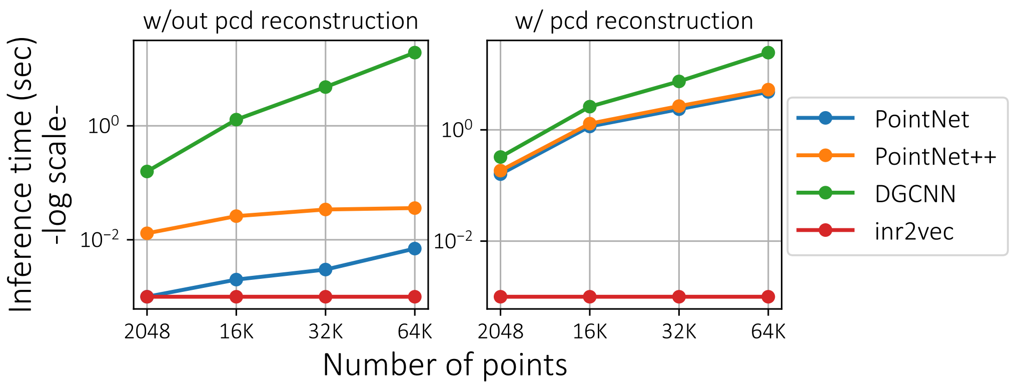


FIGURE A.26. **Time required to classify INRs encoding udf.** We plot the inference time of standard baselines and of our method, both considering the case in which discrete point clouds are available (left) and the one where point clouds must be reconstructed from the input INRs (right).

Appendix B

Neural Processing of Tri-Plane Hybrid Neural Fields

B.1 Learning tri-plane neural fields

In this section, we outline the procedure used to learn a single (tri-plane, MLP) pair, denoted as (T, M), to create the datasets of hybrid neural fields. We note that, nonetheless, our proposal is agnostic to how neural fields were trained, as, in the scenario we consider, neural fields would be available as a standard data representation and ready to be processed.

To learn a field, we optimize the parameters θ of both T and M with a loss between the reconstructed field $\hat{\mathbf{q}}_i$ and the sensor measurement $\mathbf{y}_i$. The optimized weights θ^* are obtained as follows:

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\alpha_1(\mathbf{y}_i), \alpha_2(\Phi_\theta(\mathbf{p}_i)))$$

where $\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^d$ represents the field function defined by both the tri-plane and the MLP, and $\mathcal{L}$ is a function that computes the error between predicted and ground-truth values, with $d = 1$ when supervising the fitting process of *UDF*, *SDF*, or *OF*, while $d = 4$ for *RF*. α_1 represents the mapping from the sensor domain to the field, *e.g.*,from a mesh to its *SDF*, which does not need to be differentiable. On the other hand, α_2 represents a *forward map* between the output of the field and the domain of the available supervisory signal and must be a differentiable function (*e.g.*,α_2 models volumetric rendering for *RF*). α_1 and α_2 can also be identity functions, *e.g.*,this is the case for α_2 when learning an *SDF* from a mesh, as supervise directly with the field values. In the remainder of this section, we describe the steps required to create our neural field datasets.

***UDF* from a point cloud.** Given a point $\mathbf{p} \in \mathbb{R}^3$, $UDF(\mathbf{p})$ is defined as $\min_{\mathbf{r} \in \mathcal{P}} \|\mathbf{p} - \mathbf{r}\|_2$, namely the Euclidean distance from $\mathbf{p}$ to the closest point $\mathbf{r}$ of the point cloud. For each shape, we first sample 600K points and compute the corresponding ground truth *UDF* values q_i. Between these points, 100K are sampled on the surface, 250K close to the surface, 200K points at a medium distance from the surface, 25K far from the surface, and an additional 25K scattered uniformly in the volume. More precisely, close points are computed by corrupting the point on the surface with noise sampled from the normal distribution $\mathcal{N}(0, 0.001)$, medium-distance ones with noise from $\mathcal{N}(0, 0.01)$, and far-away ones with noise from $\mathcal{N}(0, 0.1)$. Then, during the optimization process, we randomly select $N = 50$K points for each batch, apply the interpolation scheme explained in Sec. 14.2.1, retrieve the correct feature vector from the tri-plane, concatenate it with the positional encoding of $\mathbf{p}$, and feed it to the MLP to compute the loss. This procedure is repeated 1000 times. As for the training objective, we do as in inr2vec, *i.e.*,we scale the *UDF* ground truth labels into the $[0, 1]$ range, with 0 and 1 representing the maximum and minimum distance from the surface, respectively. We then constrain predictions to be in that range through a final sigmoid activation function. Finally, we optimize the weights of Φ using the binary cross entropy between the scaled ground truth labels q_i and the predicted field values $\hat{q}_i$:

$$\mathcal{L}_{\text{bce}} = -\frac{1}{N}\sum_{i=1}^{N} q_i \log(\hat{q}_i) + (1 - q_i)\log(1 - \hat{q}_i) \tag{B.1}$$

***SDF* from a mesh.** Given a point $\mathbf{p} \in \mathbb{R}^3$, $SDF(\mathbf{p})$ is defined as the Euclidean distance from $\mathbf{p}$ to the closest point of the surface, with positive sign if $\mathbf{p}$ is outside the shape and negative sign otherwise. For watertight meshes, we can easily understand whether a point lies inside or outside the surface by analyzing normals. We compute *SDF* ground truth values s_i for 600K sampled points. Then, we compute the binary cross entropy loss of Eq. (B.1) on $N = 50$K points. We found 600 optimization steps to be enough to achieve satisfying reconstruction quality. Similarly to the *UDF*, the *SDF* values s_i are scaled into the $[0, 1]$ range, with 0 and 1 representing the maximum and minimum distance from the surface, respectively and 0.5 representing the surface level set.

***OF* from a voxel grid.** Given a point $\mathbf{p} \in \mathbb{R}^3$, $OF(\mathbf{p})$ is defined as the probability o_i of $\mathbf{p}$ being occupied. For voxel grids, the fitting process is straightforward, as each cube c_i can either be occupied (value 1) or empty (value 0). Thus, we can directly apply the binary cross-entropy loss. We use the same protocol applied for point clouds and optimize for 1000 steps while sampling 50K points at each step. However, due to the high imbalance between empty and full cells, we follow inr2vec and employ a focal loss [349]:

$$\mathcal{L}_{\text{focal}} = -\frac{1}{N}\sum_{i=1}^{N} \beta(1 - o_i)^{\gamma} c_i \log(o_i) + (1 - \beta)o_i^{\gamma}(1 - c_i)\log(1 - o_i)$$

with β and γ representing the balancing and focusing parameters, respectively.

***RF* from images.** Given a point $\mathbf{p} \in \mathbb{R}^3$, $RF(\mathbf{p})$ [339] is a 4-dimensional vector containing the (R, G, B) color channels and density σ of the point. To fit a *RF*, we rely on the NerfAcc library [361]. Specifically, we implement a simplified version of NeRF that does not take into account the viewing direction and consists of a (tri-plane, MLP) pair that predicts the four field values. To render RGB images, we leverage the volumetric rendering implementation of [361]. In particular, at each iteration, for a given camera pose, a batch of rays (128 in our case) is selected with the corresponding RGB ground truth values and 3D points are sampled along these rays. The coordinates of these points are then interpolated, as described in Sec. 14.2.1, to extract features from the tri-plane and fed to the MLP, which in turn predicts the (R, G, B, σ) values for each of them. Finally, by means of volumetric rendering, a final RGB vector color is obtained and compared to the ground truth via a smooth L1 loss. Each *RF* is trained for 1500 steps.

B.2 Explicit reconstruction from neural fields

In this section, we discuss how to sample 3D explicit representations from neural fields and show some examples of such reconstructions.

B.2.1 Sampling explicit representations

Point cloud from *UDF*. We generate a point cloud based on the corresponding *UDF* using a slightly adapted version of the algorithm introduced by [295]. The fundamental concept involves querying the *UDF* with points distributed throughout the specific region of the 3D space under consideration. These points are then projected onto the isosurface based on their predicted *UDF* values. For a given point $\mathbf{p} \in \mathbb{R}^3$, its updated position $\mathbf{p}_{\text{new}}$ is determined as follows:

$$\mathbf{p}_{\text{new}} = \mathbf{p} - \Phi(\mathbf{p};\theta)\frac{\nabla_{\mathbf{p}}\Phi(\mathbf{p};\theta)}{\|\nabla_{\mathbf{p}}\Phi(\mathbf{p};\theta)\|} \tag{B.2}$$

where $\Phi(\mathbf{p};\theta)$ represents the field value at point $\mathbf{p}$ which is approximated using the (T, M) pair with parameters θ. It is important to note that the negative gradient of the *UDF* indicates the direction of the steepest decrease in distance from the surface, effectively pointing towards the nearest point on the isosurface. Eq. (B.2) can thus be interpreted as shifting point $\mathbf{p}$ along the direction of maximum *UDF* decrease, ultimately arriving at point $\mathbf{p}_{\text{new}}$ on the surface. However, it is crucial to observe that Φ serves as an approximation of the true *UDF*. This raises two key considerations:

1. the gradient of Φ must be normalized, as illustrated in Eq. (B.2), whereas the gradient of the actual *UDF* maintains norm 1 everywhere except on the surface;
2. the predicted *UDF* value can be imprecise, potentially resulting in point $\mathbf{p}$ remaining distant from the surface even after applying Eq. (B.2).

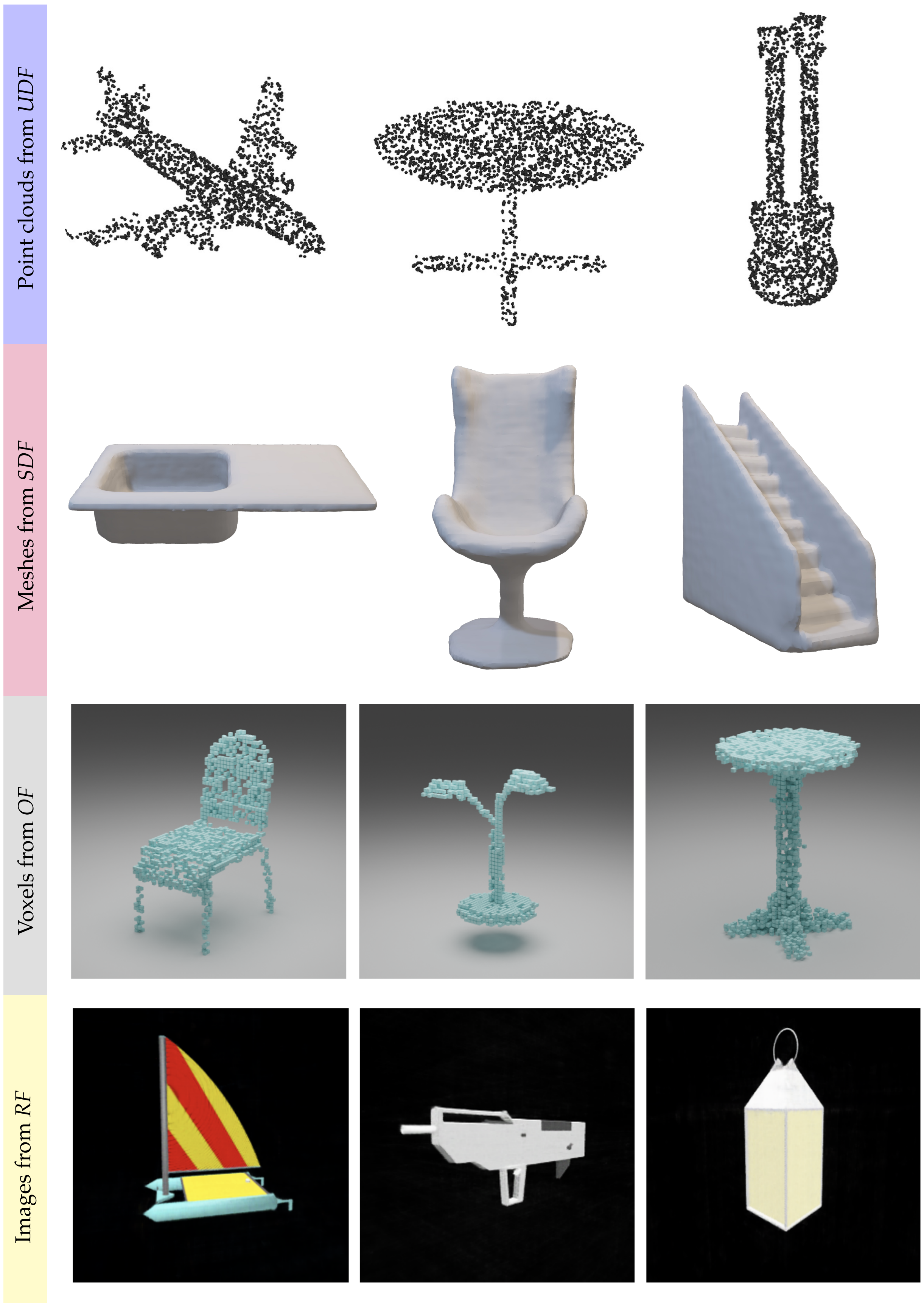


FIGURE B.1. **Tri-plane reconstruction examples of point clouds from *UDF*, meshes from *SDF*, voxels from *OF*, and images from *RF*** (from top to bottom)

To address the second point, we refine $\mathbf{p}_{\text{new}}$ by iteratively repeating the update described in Eq. (B.2). With each iteration, the point gradually approaches the surface, where the values approximated by Φ become more accurate, eventually placing the point precisely on the isosurface. The whole algorithm for sampling a dense point cloud from a given *UDF* entails the following steps:

1. generate a set of points uniformly scattered within the specified region of 3D space and predict their *UDF* values using the provided (T, M) pair;
2. discard points with predicted *UDF* values exceeding a fixed threshold (0.05 in our experiments). For the remaining points, update their coordinates iteratively using Eq. (B.2), typically requiring 5 updates for satisfactory results;
3. repeat the entire procedure until the reconstructed point cloud reaches the desired number of points.

Triangle mesh from *SDF*. We employ the Marching Cubes algorithm [331] to construct a mesh based on the corresponding *SDF*. The Marching Cubes process involves systematically traversing the 3D space by evaluating the *SDF* at 8 locations simultaneously, forming the vertices of a tiny virtual cube. This traversal continues until the entirety of the desired 3D region has been covered. For each cube, the algorithm identifies the triangles necessary to represent the portion of the isosurface passing through it. These triangles from all cubes are then integrated to create the final reconstructed surface. To determine the number and placement of triangles for an individual cube, the algorithm examines the *SDF* values at pairs of neighbouring vertices within the cube. A triangle vertex is inserted between two vertices with opposing *SDF* signs. Because the possible combinations of *SDF* signs at cube vertices are limited, a lookup table is generated to retrieve the triangle configuration for a given cube. This configuration is derived from the *SDF* signs at the eight vertices of the cube, which are combined into an 8-bit integer and used as a lookup table key. Once the triangle configuration for a cube is retrieved, the vertices of the triangles are positioned along the edges connecting the cube vertices. This positioning is accomplished through linear interpolation of the two *SDF* values associated with each edge.

Voxel grid from *OF*. To generate a voxel grid from its *OF*, we employ a straightforward procedure. Each neural field is trained to estimate the likelihood of a specific voxel being full when provided with the 3D coordinates of the voxel center. Consequently, the initial step in reconstructing the fitted voxels involves constructing a grid with the desired resolution, denoted as V. Then, the field is queried using the V^3 centroids of this grid, and it produces an estimated probability of occupancy for each centroid. Eventually, we designate voxels as occupied only if their predicted probability surpasses a predefined threshold, which we have empirically set to 0.4. This threshold selection has been determined through experimentation,

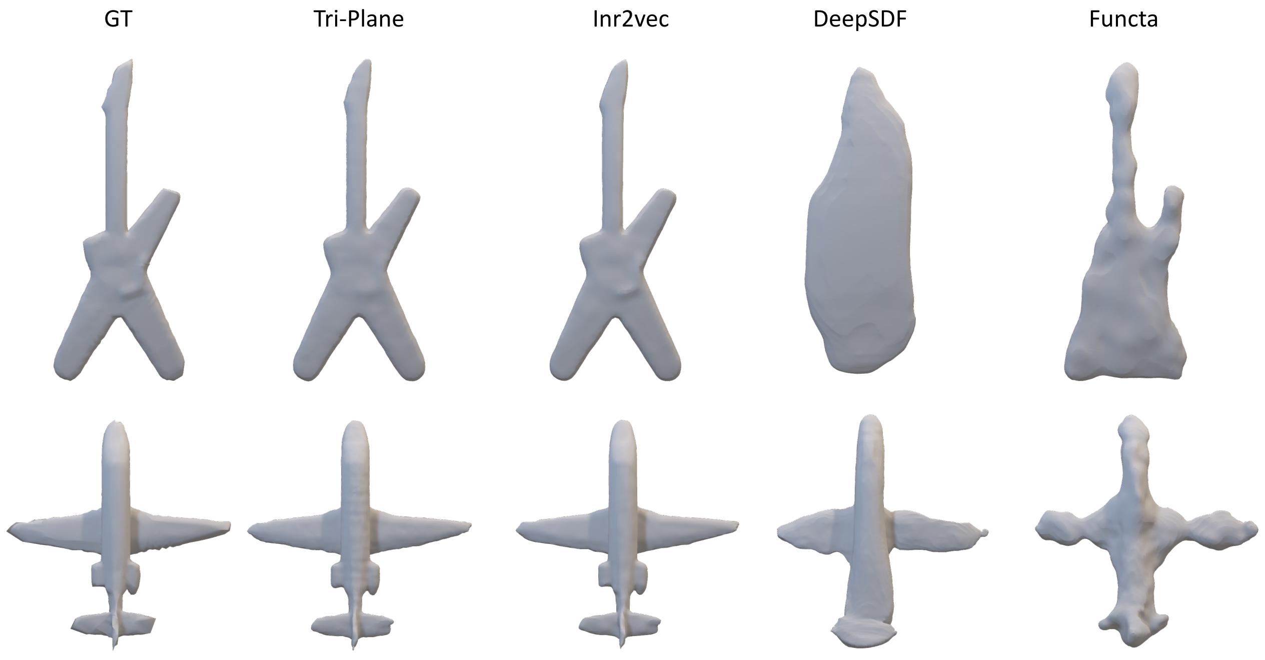


FIGURE B.2. **Reconstruction comparison for Manifold40 meshes obtained from *SDF***

as it strikes a suitable balance between creating reconstructions that are neither overly sparse nor excessively dense.

Images from *RF*. Given a camera pose and intrinsic parameters, to render images from a *RF*, we employ the same volumetric rendering scheme as in [361], outlined in Appendix B.1. An overview of the procedure is the following: for each pixel location, we cast the corresponding ray from the camera and sample 3D points along the ray. We feed these coordinates to a (T, M) pair, obtaining the corresponding (R, G, B, σ) field values. Then, we compute the volumetric rendering equation to calculate the final RGB value of the image pixel.

B.2.2 Examples of reconstructions by tri-planes

In Fig. B.1, we report some examples of point clouds, meshes, voxel grids and images reconstructed from tri-plane neural fields fitted to *UDF*s, *SDF*s, *OF*s and *RF*s, respectively. For all fields, we can observe a very good reconstruction quality.

B.2.3 Comparison between reconstructions by neural field processing frameworks

In Fig. B.2 and Fig. B.3, we show reconstructions of point clouds and meshes obtained by different frameworks used to process neural fields. We can notice that neural fields in which the neural component is a shared network trained on the whole dataset, i.e. Functa

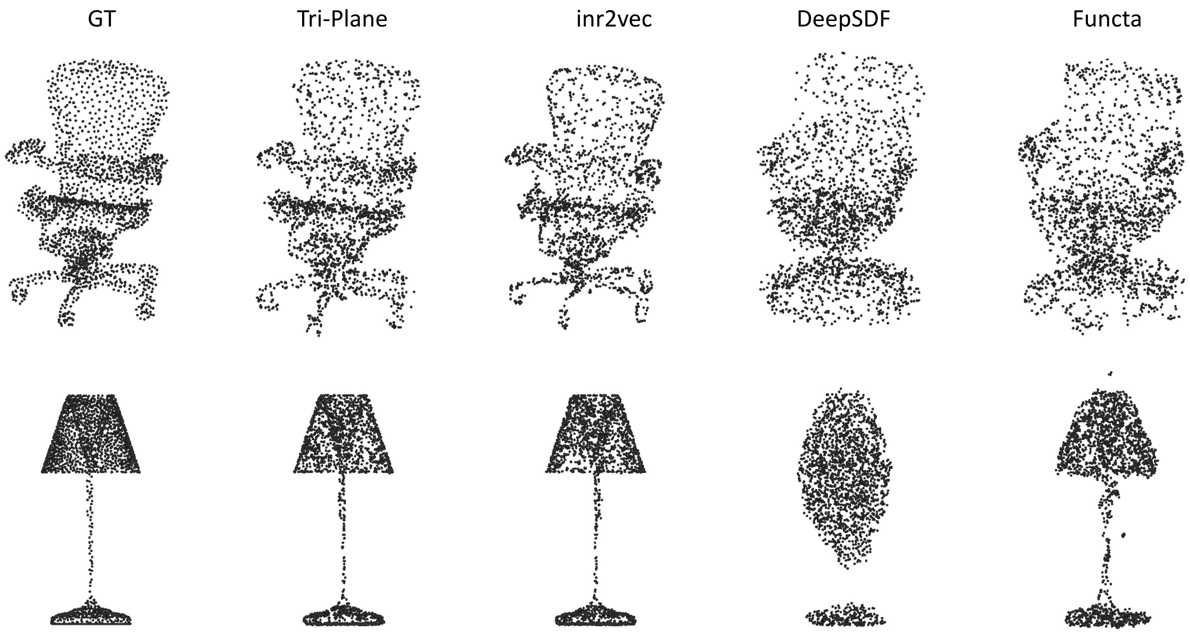


FIGURE B.3. **Reconstruction comparison for ModelNet40 point clouds obtained from *UDF***

			Mesh from *SDF*	
Method	Type	# Params (K)	CD (mm)	F-score (%)
Voxel	Single	554	0.19	69.1
Tri-plane	Single	64	0.18	68.6

TABLE B.1. **Mesh reconstruction results on the Manifold40 test set (voxel grid vs tri-plane).** We compare hybrid representations employing tri-planes or voxel grids as discrete data structures.

and DeepSDF, cannot properly reconstruct the original explicit data. Conversely, methods relying on fitting an individual network, either a large MLP (inr2vec) or a tri-plane and a small MLP (ours), provide high-quality reconstructions.

B.3 Voxel grid hybrid neural fields

In this paper, we use tri-plane neural fields. However, other kinds of hybrid neural fields may be considered plausible alternatives. Thus, we investigate the employment of voxel grid hybrid neural fields. First, we analyze their reconstruction quality and memory footprint. We report results on the Manifold40 test set in Tab. B.1. We observe that, compared to tri-planes, voxel-based hybrid fields achieve comparable reconstruction accuracy but at the cost of a much larger number of parameters, *i.e.*,554K vs 64K.

		UDF			*SDF*	*OF*
Method	Input	ModelNet40	ShapeNet10	ScanNet10	Manifold40	ShapeNet10
CNN	Tri-plane	82.2	92.1	63.4	82.5	88.4
3D CNN	Voxel	83.7	91.6	68.1	81.1	85.2
Transformer	Tri-plane	**87.0**	**94.1**	**69.1**	**86.8**	**91.8**

TABLE B.2. **Neural field classification results (voxel grid vs tri-plane).** We compare hybrid representations employing tri-planes or voxel grids as discrete data structures.

Then, we also try to classify voxel grid hybrid neural fields. With our resources (an RTX 3090 GPU), we were not able to train the Transformer architecture used for tri-planes. Indeed, using a voxel grid would require tokens of size 32^3 in our formulation (Transformer invariant to the channel order), leading to prohibitive training resources. Thus, we employ a 3D CNN (ResNet50-style). Results are reported in Tab. B.2. We can see that although processing voxels with a 3D CNN is more expansive in terms of both storage and computation than processing tri-planes with a 2D CNN, they achieve comparable classification performance (row 1 vs 2) without a clear winner between the two approaches. Yet, thanks to their high compactness, we can process tri-planes by Transformers, achieving significantly better performance (last row). Finally, we highlight that though the performances yielded in this experiment by a hybrid voxel-based approach are inferior to tri-planes, they are still much better than those achievable by previous proposals (see Tab. 14.2), which vouches for the effectiveness of hybrid approaches when it comes to processing neural fields without compromising reconstruction accuracy.

B.4 Deeper investigation on the tri-plane and MLP content

In this section, our objective is to gain a deeper understanding of the information stored in the tri-plane and the MLP.

B.4.1 Is the MLP alone enough for reconstruction?

The first question we aim to address is whether the MLP alone can serve as a neural field capable of representing 3D shapes without relying on the features provided by the tri-plane. As explained in Sec. 14.2, we utilize the 3D coordinates of a point to retrieve the corresponding feature vector from the tri-planes. This latter is then concatenated with the positional encoding of the coordinates. To examine whether the 3D coordinates and the MLP alone are sufficient to obtain accurate outputs, we conduct an experiment where we shuffle the tri-plane features along the spatial dimensions while preserving the channel orders. This means that each point $\mathbf{p}$ will be associated with a different yet meaningful feature vector

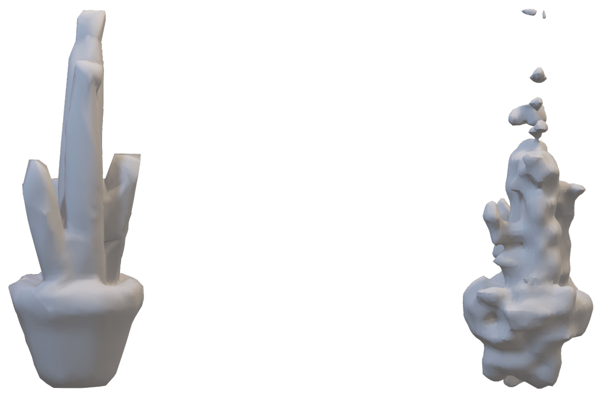


FIGURE B.4. **Tri-plane channel shuffling. Left:** Mesh reconstructed from a *SDF* tri-plane neural field. **Right:** Mesh reconstructed by spatially shuffling the features of the tri-plane while keeping the original MLP.

from another 3D point. If the MLP is still capable of providing the correct output in this scenario, it implies that all the geometric information of the underlying 3D shapes is likely contained within the MLP, and the tri-plane is actually not necessary. Thus, we conducted a reconstruction experiment of meshes from *SDF*, similar to that reported in Tab. 14.1. We compared the reconstructed mesh with the ground truth by calculating the Chamfer Distance and the F-score, using 16,384 points sampled from the surface. We note that when utilizing the MLP with features shuffled spatially, we get significantly worse values for both the Chamfer Distance and the F-score (2.9mm vs 0.16mm), which indicates poor reconstruction quality. This is clearly visible in Fig. B.4. This outcome highlights that the MLP alone, without the tri-plane features, is not capable of accurately reconstructing the original shape. Thus, we believe that the tri-plane provides essential geometric information about the represented 3D shape.

B.4.2 Is the MLP alone enough for classification?

We conducted additional experiments where we treated each MLP associated with a tri-plane as input to a classification pipeline. We utilize various methods, including a simple MLP classifier, as well as advanced frameworks such as NFT and inr2vec, which directly process MLPs. In this case, we completely disregarded the information provided by the tri-plane. We performed this experiment starting from random initialization. As shown in Tab. B.3, even though MLPs used in tri-planes are smaller than those used in other experiments, we note that directly classifying them leads to unsatisfying performances.

		SDF
Method	Input	Manifold40
MLP	MLP (Tri-plane)	4.3
NFN [9]	MLP (Tri-plane)	4.1
NFT [318]	MLP (Tri-plane)	4.1
inr2vec	MLP (Tri-plane)	7.4
Ours	Tri-plane	**86.8**

TABLE B.3. **Classification of MLPs of tri-plane neural fields on the Manifold40 test set.** Neural fields were randomly initialized.

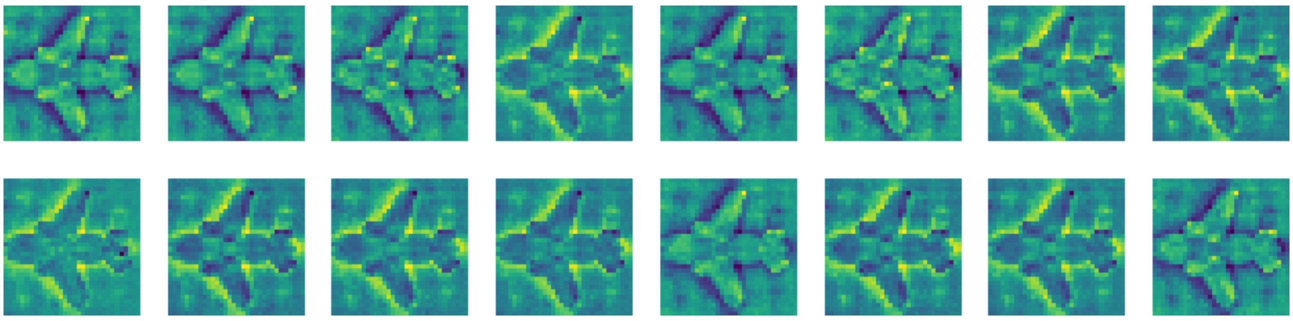


FIGURE B.5. **Channel visualizations.** We select one of the tri-planes and visualize all its 16 channels learned for an airplane.

B.4.3 Tri-plane channel visualizations

To understand what information is contained in different channels of a tri-plane feature map, we try to visualize all of them for a fixed shape. Fig. B.5 shows all 16 channels of a *SDF* tri-plane feature map of a *SDF* neural field representing an airplane. We can see that, although values can change across channels, the overall shape is outlined and repeated in each channel. This also motivates why a CNN network that is not invariant to the channel order can work on such kind of input.

B.4.4 Channel order investigation

In this section, we further analyze the tri-plane channel content to highlight that the main difference between two training episodes concerning the same shape boils down to a permutation of the channel order. Thus, we conduct a similar experiment to that illustrated in Fig. 14.3 with a tri-plane with only 8 channels. We visualize results in Fig. B.6, also reporting the tri-plane channel visualization for both training episodes, A and B, as well as the permutation that aligns the channels found in A to B. We discovered that permutation by minimizing the reconstruction error. We can appreciate that after the permutation, the corresponding channels contain similar features, resulting in the possibility of reconstructing the shape with the M_B.

B.4.5 Additional visualizations

In Fig. B.7, we show additional tri-plane visualization, similar to those of Fig. 14.3 (left).

B.5 Implementation details

B.5.1 Datasets

Regarding single MLP neural fields based on SIREN networks, we employ the same datasets used in inr2vec with the same train/val/test splits for *SDF*, *UDF*, and *OF*. When creating the tri-plane neural field datasets, we employ the original explicit datasets used to create the inr2vec neural fields. We also follow the same offline augmentation protocol that employs a non-uniform scaling along the three axes and then a re-normalization of the shape into the unit sphere to reach roughly 100K shapes in total. Thereby, the same explicit data is used to build the benchmark, while the only aspect that varies is the way to represent the neural field, *i.e.*,MLP only vs $(\text{tri-plane}, \text{MLP})$. Finally, to learn *RF*, we adopt the dataset introduced in [342]. This dataset contains renderings from 13 classes of ShapeNet, and for each shape, 36 views are generated around the object and used to fit the *RF*. In this case, we use the original dataset that accounts for 40511 shapes with no prior augmentation. For the train/val/test splits, we randomly sample 80% of the objects for the training set and select 10% shapes for both the validation and test splits.

We additionally note that the ShapeNet10 and ScanNet10 datasets mentioned in the main text are subsets of 10 (shared) classes of ShapeNet [171] and ScanNet [174], respectively, originally proposed in [28].

B.5.2 Benchmark

In this section, we detail some implementation details and choices behind the results reported in Tab. 14.2 by specifying how experiments were run for each one of our competitors.

DeepSDF. DeepSDF [296] was originally intended as a shared network architecture that learns an *SDF* from a dataset of meshes. The results reported in Tab. 14.2 for Manifold40 [278] were therefore obtained by running the code from the official repository to train the framework for 100 epochs and compute the 1024-dimensional embeddings of training, validation, and test set. A classifier (3 fully connected layers with ReLUs) was then trained on those embeddings. For *UDF* results, instead, we trained the framework on point cloud datasets by replacing the DeepSDF loss with the binary cross entropy of Eq. (B.1). Extensions to *OF* and *RF*, however, are not as straightforward and were thus not included as part of our work.

inr2vec. inr2vec was trained on each point cloud, mesh, and voxel dataset for 100 epochs via the official code and the resulting embeddings used to train a classifier (3 fully connected layers with ReLUs). inr2vec deals with *UDF*, *SDF*, and *OF*; an extension to *RF* would not be trivial thus it was not covered by our experiments.

DWSNet. [317] test their DWSNet architecture on neural fields representing grayscale images. We modified their official neural field classification code to work with *UDF*, *SDF*, *OF*, and *RF*, and trained their classifier (DWSNet + classification head) for 100 epochs.

NFN and NFT. We modified the official code for NFN [9] and NFT [318] to work with *UDF*, *SDF*, *OF*, and *RF*, by using two hidden equivariant layers (16 channels each) followed by an MLP classification head. We trained on each dataset for 80 epochs, by which time the validation loss had stopped improving.

Functa. To perform experiments with the Functa framework [319], we start from the original code provided by authors and adapt it to fit *UDF*, *SDF*, and *OF*. In particular, we use a latent modulation of size 512. At each iteration, we use 10k points to compute the loss. The inner loop of the meta-learning process is optimized for three iterations, and in total, we optimize for $5e^5$ steps with batch size 2. After the training phase, the weights of the shared network are frozen, and we execute the inner loop of the meta-learning protocol for three steps to obtain the latent modulation vector for each field.

B.5.3 Architectures

We provide here some additional details regarding the Transformer architectures used for classification and segmentation depicted in Fig. B.8. In both cases, the channels of the tri-planes are flattened and linearly projected to 512-dimensional vectors and fed to the Transformer encoder, which is the original Transformer encoder proposed in [15]. In our case, the encoder consists of 4 heads with 8 layers each. The encoder produces a single embedding for each tri-plane channel, and for classification, we use a max pool operator to achieve invariance to the channel order and obtain a single global embedding. The classifier consists of a single fully connected layer that yields the predicted probability distribution for the input field. In total, the number of parameters is roughly 25M, which is very similar to the number of parameters of a Resnet50. As regards part segmentation, we attach an additional Transformer decoder composed of 2 parallel heads with 4 layers each. The decoder takes in input the sequence of tokens from the encoder and the sequence of tokens obtained starting from the coordinates query to be segmented. More precisely, each point is first encoded using positional encoding from [339] that gives a 63-dimensional vector, and then we linearly project it into a 512-dimensional vector. Finally, given an object to segment, we also attach the one-hot encoding of its class (standard practice in part-segmentation). The decoder output is a sequence of transformed tokens that are given in input to a single layer to predict the final probability distribution. Note that we do not use the decoder auto-regressively as in the original Transformer. We just compute a forward pass on a batch of query points to segment all of them simultaneously. Hence, we also do not use masked attention at training time.

B.5.4 Training

We provide here additional training details. For classification, we use the same hyperparameters for all experiments. We adopt the AdamW optimizer [356], with the OneCycle scheduler [128] with maximum learning rate of 1e-4, and train for 150 epochs with batch size set to 256 using the cross entropy loss. We use a single NVIDIA RTX 3090 for all experiments. At training time, we apply a random crop of size 30×30 to the tri-planes that have a resolution of 32×32. As for part segmentation, we use the same configuration as for classification, although we train for 75 epochs with batch size 32. For training and testing, we use the same protocol used by our competitors. Indeed, at training time, we use the cross entropy loss function with all the object parts, although at test time, metrics are computed by selecting the highest scores among the correct object parts for a given class.

B.5.5 Random initialization

Throughout the main paper, we use the term "random initialization" to differentiate our setup from that considered in in2vec, where all neural fields are trained starting from the same, albeit random, set of parameters, i.e. initial values of weights and biases are sampled once and then used as the starting point to fit all MLPs. In our work, instead, we consider the more realistic scenario in which each individual neural field is trained starting from a random and different set of parameters. Nonetheless, when sampling initial values of weights and biases for each shape, we do indeed follow the specific initialization scheme suggested for SIRENs [304]. In other words, "random initialization" means that a different random set of parameters, sampled according to the SIREN rules, is used to initialize each MLP. Tri-planes, instead, are initialized with values sampled from a Gaussian distribution.

B.6 Training and inference time

In this section, we provide additional details on training and inference times, computed on a single e NVIDIA RTX 3090.

Tab. B.4 shows a comparison between the times required to fit approximately 100K shapes on the Manifold40 [278] training set for different methods. For neural fields consisting of an MLP only and a $(\text{tri-plane}, \text{MLP})$ pair, denoted by "MLP" and "Tri-plane", respectively, we fit each shape for 600 steps. Functa was trained with the original hyperparameters for 500K iterations for the meta-learning outer loop. DeepSDF was trained for 100 epochs. Notice how tri-plane hybrid neural fields are the ones requiring the least amount of time to fit the dataset.

	SDF
Method	Time (hours) ↓
DeepSDF [296]	61
Functa [319]	120
MLP	80
Tri-plane	**45**

TABLE B.4. **Training time comparison.** Time required to fit approximately 100K shapes on the Mandifold40 training set. Times are computed on a single NVIDIA RTX 3090.

	UDF			
	Time (seconds) ↓			
Method	2048 points	16K points	32K points	64K points
PointNet [160]	**0.002**	0.004	0.006	0.012
PointNet* [160]	0.099	0.764	1.388	2.793
Tri-plane	0.003	**0.003**	**0.003**	**0.003**

TABLE B.5. **Inference time to classify an input shape.** * indicates that the time to reconstruct the point cloud from the neural field is included. Times are computed on a single e NVIDIA RTX 3090.

Tab. B.5 compares the inference time required to classify an input shape with different strategies. The first row is the classification time of a PointNet processing a point cloud. In the second row, we report the inference time of PointNet, assuming we do not have an explicit point cloud available, namely PointNet*. In this case, the time includes reconstructing the point cloud from the tri-plane neural field, which would be the only data available in our scenario. Finally, the last row reports the time required by our method, *i.e.*,a Transformer processing an input hybrid tri-plane neural field. We show times for different point cloud resolutions (2048, 16K, 32K, 64K points). We notice that the inference time of our method is constant along the resolution axis; in particular, our method is comparable to the PointNet at low resolution, whereas it becomes increasingly faster as the resolution grows. Moreover, we highlight that our method inference times w.r.t. to those of a PointNet directly processing explicit point clouds are comparable at lower resolutions and even better at higher ones (*e.g.*,0.003 Tri-plane vs 0.012 PointNet for 16K points).

B.7 Tri-plane ablations

This section provides additional ablations concerning the tri-plane structure and hyper-parameters. Tab. B.6 shows that sharing the MLP across tri-planes leads to inferior results in both classification and reconstruction on Manifold40 [278]. It is also worth highlighting how sharing the MLP requires its availability at test time to create new neural fields, *i.e.*,to create new test data, limiting the deployment scenarios of our methodology, which are instead equivalent to those of discrete data structures when using a (MLP, triplane) pair for each sample. Tab. B.7 provides a comparative analysis of classification and reconstruction results on Manifold40 [278] between tri-planes with different resolution and/or number of channels. Interestingly, the classification accuracy and reconstruction error are quite robust to the number of channels and tri-plane resolution, which therefore are not critical design hyperparameters.

		SDF	Mesh from *SDF*	
		Classification	Reconstruction	
Method	Type	Accuracy (%) ↑	CD (mm) ↓	F-score (%) ↑
Tri-plane	Shared	84.7	1.57	42.9
Tri-plane	Single	**86.8**	**0.18**	**68.6**

TABLE B.6. **Shared vs individual MLP.** Comparison of classification and reconstruction results of tri-planes sharing all the same MLP vs when each tri-plane has its own individual MLP. Results were computed on the Manifold40 test set.

		SDF	Mesh from *SDF*	
Resolution	Channels	Accuracy (%) ↑	CD (mm) ↓	F-score (%) ↑
32×32	32	86.3	0.18	68.6
32×32	16	**86.8**	0.18	68.8
32×32	8	86.4	0.18	**69.2**
24×24	16	86.6	0.18	68.9
40×40	16	86.4	0.18	69.0

TABLE B.7. **Ablation study of tri-plane resolution and number of channels.** Second row is our choice of tri-plane size. Results were obtained on the Manifold40 test set.

B.8 Evaluating on the original discrete 3D representations

In Tab. 14.3 of the main paper, we evaluate the competitors on the data reconstructed from the tri-plane neural fields fitted on the test sets of the considered datasets since these would be the only data available at test time in the scenario described in Sec. 14.1, and as proposed in inr2vec, where neural fields are used to store and communicate 3D data. Nonetheless, we compare the classification performance of methods operating on discrete data both when the original data is used and when it is, instead, reconstructed from the corresponding neural field. Tab. 14.3 shows slightly better results when the original data is used. However, the

		Accuracy (%) ↑				
Method	Input	ModelNet40	ShapeNet10	ScanNet10	Manifold40	ShapeNet10
Ours	Tri-plane	87.0	94.1	69.1	86.8	91.8
PointNet [160]	Point Cloud	**88.8**	94.3	72.7	–	–
PointNet* [160]	Point Cloud	**88.8**	**94.7**	**72.8**	–	–
MeshWalker [289]	Mesh	–	–	–	90.0	–
MeshWalker* [289]	Mesh	–	–	–	**90.6**	–
Conv3DNet [258]	Voxel	–	–	–	–	92.1
Conv3DNet* [258]	Voxel	–	–	–	–	**92.5**

TABLE B.8. **Explicit methods on reconstructed data vs original discrete data.** Classification accuracy on the test set is reported. **Top:** our method, that processes (tri-plane) neural fields. **Bottom:** Methods that process explicit discrete 3D data, either reconstructed from the corresponding neural field (without *) or by operating directly on the original data (with *).

difference is small enough to prove that no significant loss of information occurs when storing data with tri-plane neural fields.

B.9 Study on the memory occupation of neural fields

Dataset	Representation	Num Shapes	Memory (GB)
ModelNet40	Point Cloud	110054	2.52
ModelNet40	Siren MLP (UDF)	110054	327.99
ModelNet40	Tri-plane (UDF)	110054	20.15
ShapeNet10	Point Cloud	103230	2.36
ShapeNet10	Siren MLP (UDF)	103230	307.65
ShapeNet10	Tri-plane (UDF)	103230	18.90
ScanNet10	Point Cloud	108360	2.48
ScanNet10	Siren MLP (UDF)	108360	322.94
ScanNet10	Tri-plane (UDF)	108360	19.84
Manifold40	Mesh	100864	10.79
Manifold40	Siren MLP (SDF)	100864	300.60
Manifold40	Tri-plane (SDF)	100864	18.47
ShapeNet10	Voxel	103230	25.20
ShapeNet10	Siren MLP (OF)	103230	307.65
ShapeNet10	Tri-plane (OF)	103230	18.90
ShapeNetRender	Training Images	40511	204.45
ShapeNetRender	Siren MLP (NeRF)	40511	120.73
ShapeNetRender	Tri-plane (NeRF)	40511	7.42

TABLE B.9. **Training dataset memory occupancy**

Nowadays, datasets are typically made of data represented explicitly, e.g., a point cloud dataset, a multi-view image dataset, and so on. However, in our vision, these datasets might be replaced by their neural field counterparts, *i.e.*,each element would be represented as a neural field. Thus, this section investigates the trade-off of using these novel representations regarding memory occupancy.

First, we investigate the memory in GB required by each raw dataset employed in our paper. We report the results in Tab. B.9 either by their original explicit representation or with neural fields. As neural field representation, we show the SIREN MLP adopted inr2vec or the tri-plane representation. We note that the tri-plane representation requires more memory than point clouds and meshes (*e.g.*,ModelNet40 and Manifold40). Yet, the advantages of using tri-plane representations stand out when compared with either voxels (ShapeNet10) or images (ShapeNetRender). Notably, tri-planes take significantly less memory than Siren MLPs.

However, we argue that the real advantages of using neural fields are related to the memory occupied by the data being decoupled from its spatial resolution. Thus, in Fig. B.9, we study the number of parameters required for different explicit representations compared to those of neural fields by varying the spatial resolution of the data. We include all variables required by an explicit representation in their parameter number, *e.g.*,each point of a point cloud has 3 parameters: its 3 coordinates x, y, and z. The blue, red, and green lines represent the parameters of the explicit, the SIREN MLP, and the tri-plane representation when changing the resolution. Regarding meshes and point clouds, we notice that the space occupied by tri-plane neural fields is slightly larger for the resolutions used in the datasets. However, even with a point cloud of 21,333 points and a mesh with 14,222 faces, using tri-plane neural fields to represent the data becomes advantageous. This is of utmost importance, considering that real datasets could contain point clouds or meshes with many more points or faces *e.g.*,Objaverse [362] features meshes with more than 10^7 polygons. The advantages are even more significant in the case of voxel grids, in which the memory occupancy scales cubically with the spatial resolution or NeRFs in which many training images per sample are required. Finally, we point out that the representation with tri-plane neural fields is advantageous compared to that with SIREN MLPs.

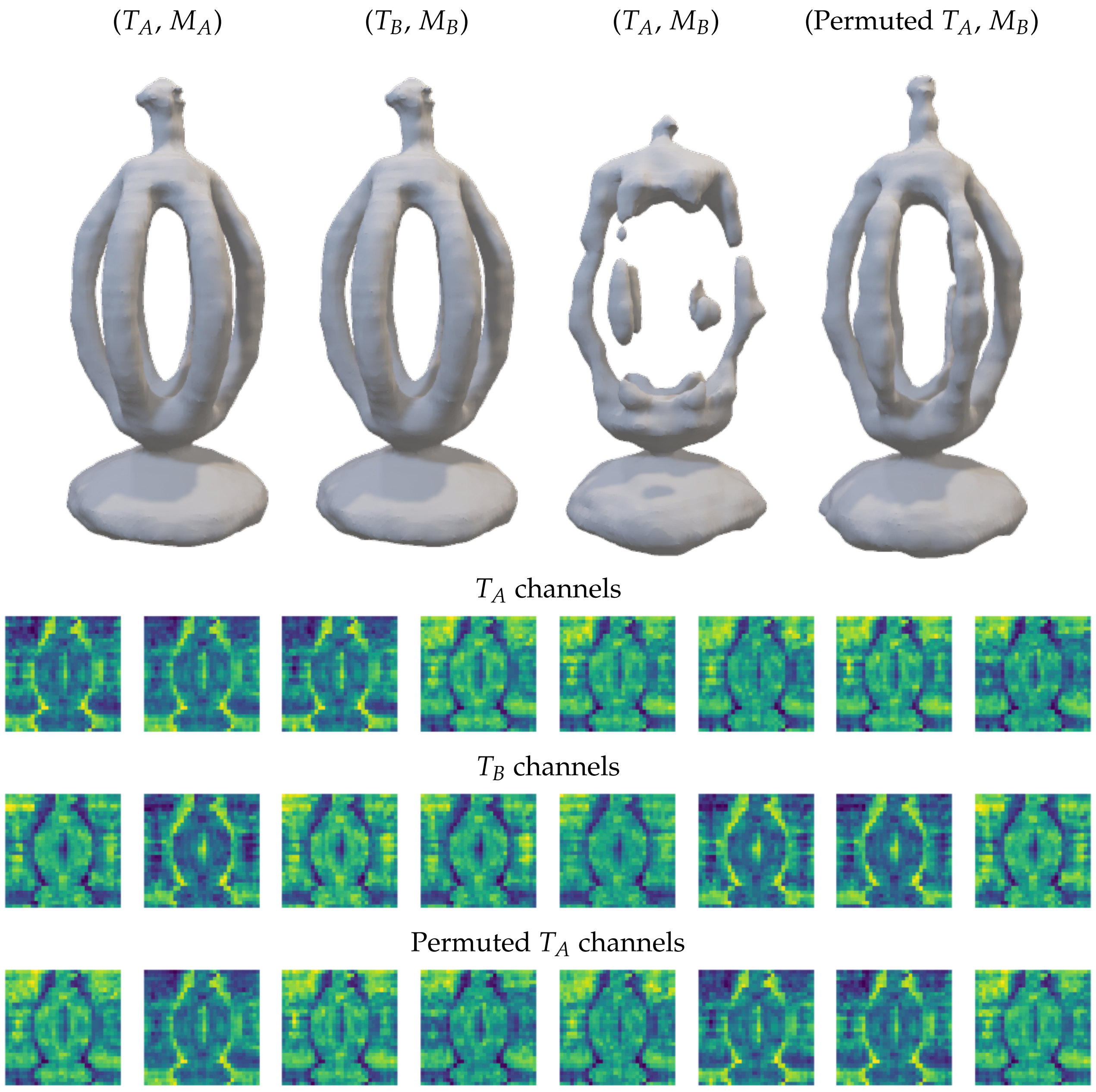


FIGURE B.6. **Row 1:** Reconstructions of different combinations of (tri-plane, MLP) pairs with different initializations. Permuted T_A means that the channels of T_A are permuted in order to minimize the reconstruction error when combined with M_B. **Rows 2–4:** Channel visualizations.

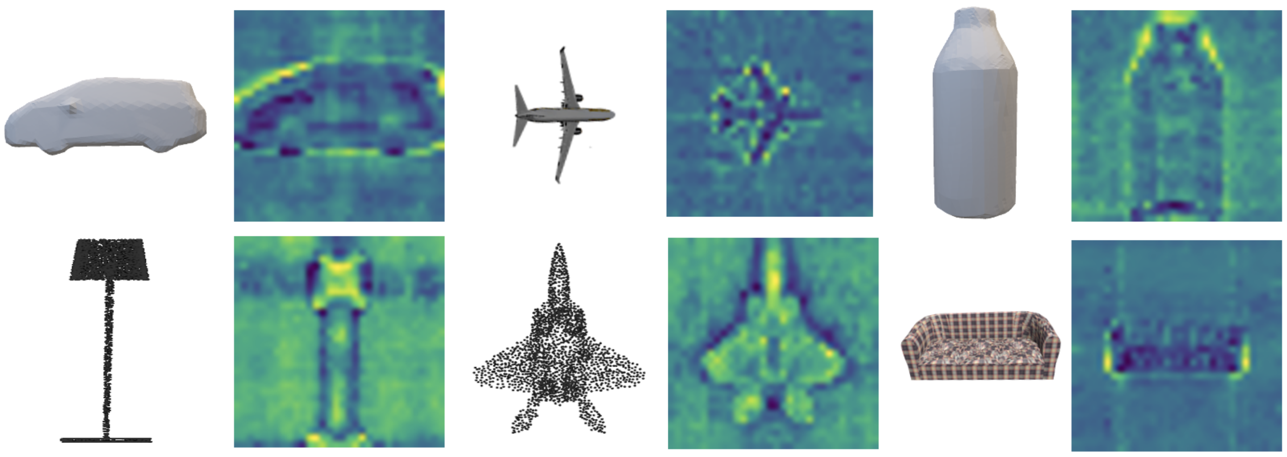


FIGURE B.7. **Additional visualizations.** For each object, we visualize the explicit reconstruction obtained from the hybrid neural field and one tri-plane feature map for three different fields, *SDF*, *UDF*, and *RF*. We visualize features as the normalized sum of all channels with a viridis colormap.

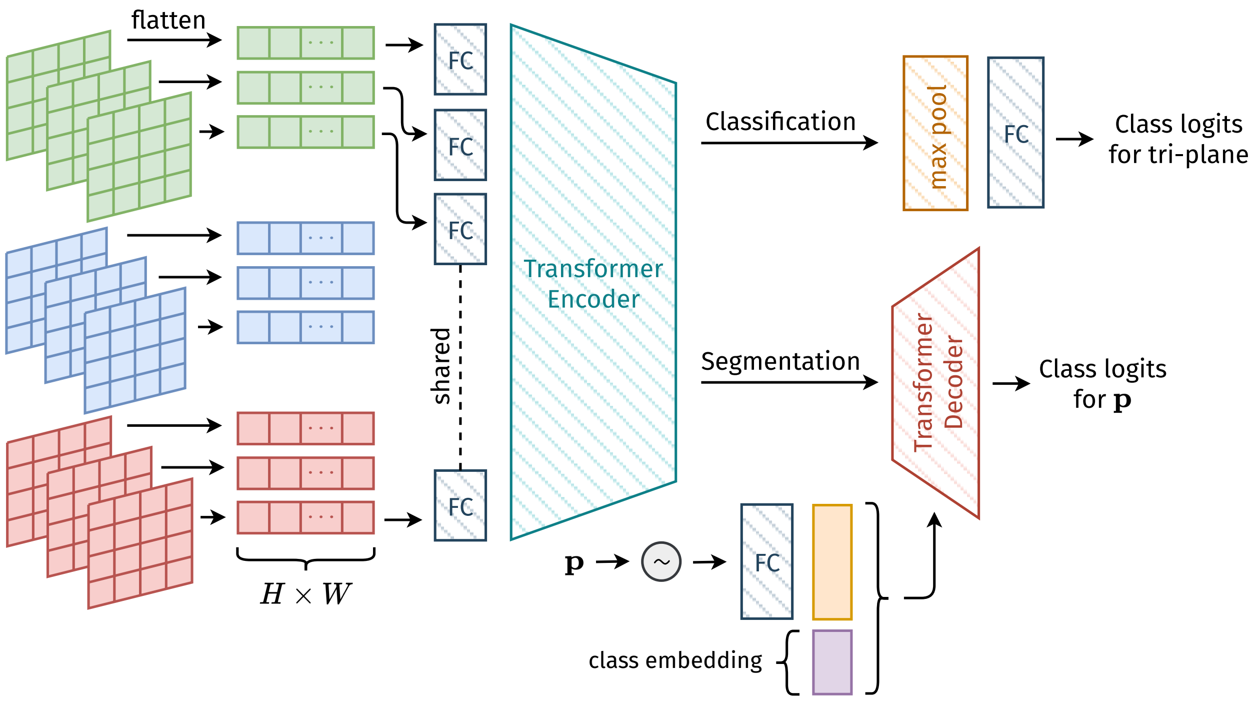


FIGURE B.8. **Architecture for tri-plane processing**

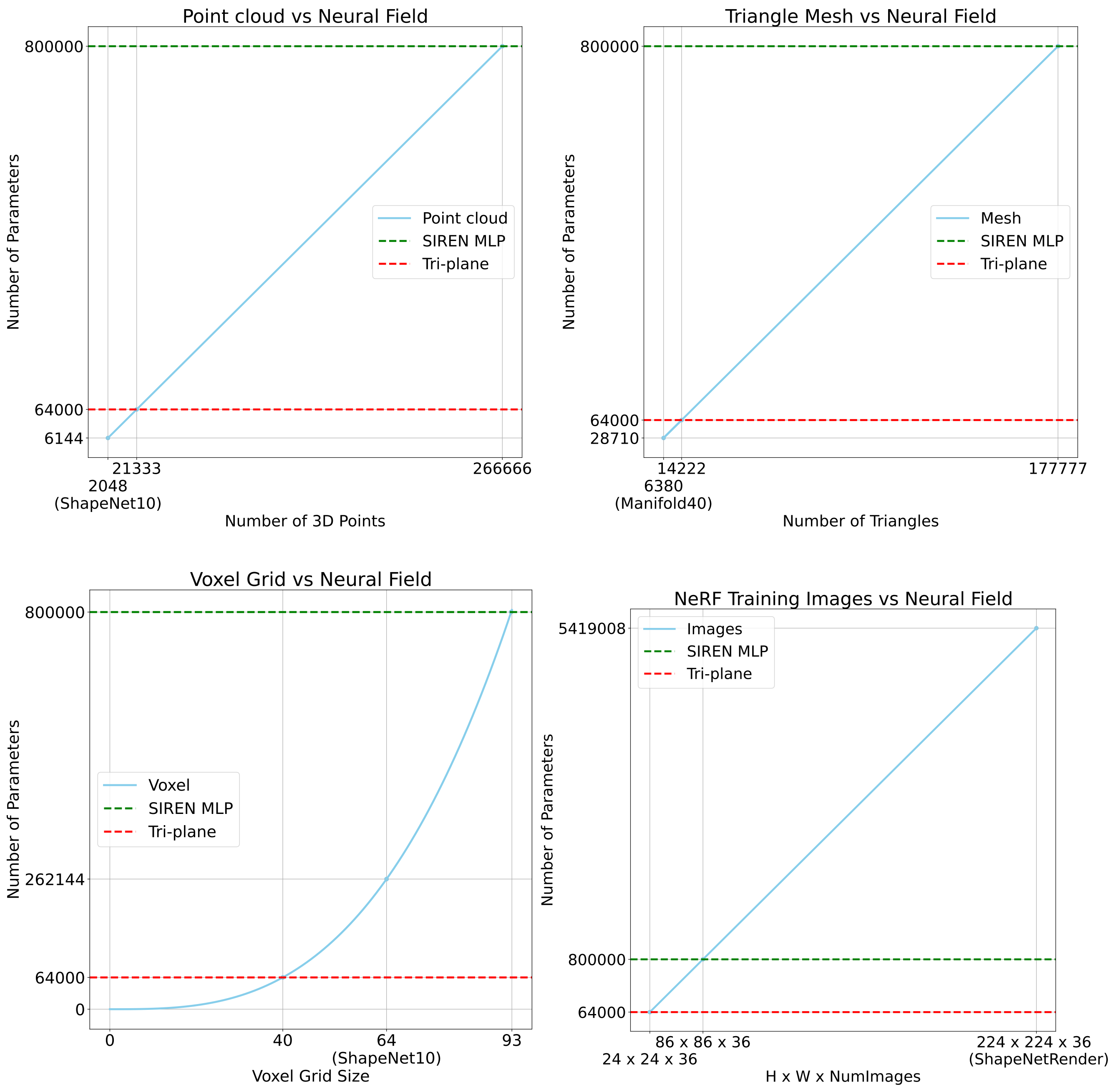


FIGURE B.9. **Number of parameters in relation to spatial resolution: neural fields vs explicit representations**

Bibliography

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.

[13] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations*. 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: 10.1109/cvpr.2016.90. URL: http://dx.doi.org/10.1109/cvpr.2016.90.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[16] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. “CNN Features off-the-shelf: an Astounding Baseline for Recognition”. In: *CoRR* abs/1403.6382 (2014). arXiv: 1403.6382. URL: http://arxiv.org/abs/1403.6382.

[17] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *CoRR* abs/1604.01685 (2016). arXiv: 1604.01685. URL: http://arxiv.org/abs/1604.01685.

[18] Hassan Alhaija, Siva Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. “Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes”. In: *International Journal of Computer Vision (IJCV)* (2018).

[19] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. “Taskonomy: Disentangling task transfer learning”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3712–3722.

[20] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Trans. on Knowl. and Data Eng.* 22.10 (Oct. 2010), pp. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191. URL: https://doi.org/10.1109/TKDE.2009.191.

[21] Mei Wang and Weihong Deng. “Deep Visual Domain Adaptation: A Survey”. In: *CoRR* abs/1802.03601 (2018). arXiv: 1802.03601. URL: http://arxiv.org/abs/1802.03601.

[22] Yunsheng Li, Lu Yuan, and Nuno Vasconcelos. “Bidirectional Learning for Domain Adaptation of Semantic Segmentation”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). DOI: 10.1109/cvpr.2019.00710. URL: http://dx.doi.org/10.1109/CVPR.2019.00710.

[23] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. “Learning to Adapt Structured Output Space for Semantic Segmentation”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). DOI: 10.1109/cvpr.2018.00780. URL: http://dx.doi.org/10.1109/CVPR.2018.00780.

[24] Minghao Chen, Hongyang Xue, and Deng Cai. “Domain Adaptation for Semantic Segmentation With Maximum Squares Loss”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). DOI: 10.1109/iccv.2019.00218. URL: http://dx.doi.org/10.1109/ICCV.2019.00218.

[25] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. “CyCADA: Cycle-Consistent Adversarial Domain Adaptation”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1989–1998. URL: http://proceedings.mlr.press/v80/hoffman18a.html.

[26] Zuxuan Wu, Xintong Han, Yen-Liang Lin, Mustafa Gökhan Uzunbas, Tom Goldstein, Ser Nam Lim, and Larry S. Davis. “DCAN: Dual Channel-Wise Alignment Networks for Unsupervised Scene Adaptation”. In: *Lecture Notes in Computer Science* (2018), pp. 535–552. ISSN: 1611-3349. DOI: 10.1007/978-3-030-01228-1_32. URL: http://dx.doi.org/10.1007/978-3-030-01228-1_32.

[27] Myeongjin Kim and Hyeran Byun. “Learning Texture Invariant Representation for Domain Adaptation of Semantic Segmentation”. In: (2020). DOI: 10.1109/cvpr42600.2020.01299. URL: http://dx.doi.org/10.1109/cvpr42600.2020.01299.

[28] Can Qin, Haoxuan You, Lichen Wang, C.-C. Jay Kuo, and Yun Fu. “PointDAN: A Multi-Scale 3D Domain Adaption Network for Point Cloud Representation”. In: *Advances in Neural Information Processing Systems*. 2019.

[29] Antonio Alliegro, Davide Boscaini, and Tatiana Tommasi. "Joint Supervised and Self-Supervised Learning for 3D Real World Challenges". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 6718–6725.

[30] Idan Achituve, Haggai Maron, and Gal Chechik. "Self-Supervised Learning for Domain Adaptation on Point Clouds". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 123–133.

[31] Maximilian Jaritz, Tuan-Hung Vu, Raoul de Charette, Emilie Wirbel, and Patrick Pérez. "xMUDA: Cross-Modal Unsupervised Domain Adaptation for 3D Semantic Segmentation". In: *CVPR*. 2020.

[32] Maximilian Jaritz, Tuan-Hung Vu, Raoul De Charette, Émilie Wirbel, and Patrick Pérez. "Cross-modal learning for domain adaptation in 3d semantic segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.2 (2022), pp. 1533–1544.

[33] Duo Peng, Yinjie Lei, Wen Li, Pingping Zhang, and Yulan Guo. "Sparse-to-dense Feature Matching: Intra and Inter domain Cross-modal Learning in Domain Adaptation for 3D Semantic Segmentation". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE, 2021.

[34] Adriano Cardace, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. "Shallow Features Guide Unsupervised Domain Adaptation for Semantic Segmentation at Class Boundaries". In: *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (2022). DOI: 10.1109/wacv51458.2022.00207. URL: http://dx.doi.org/10.1109/WACV51458.2022.00207.

[35] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651. ISSN: 2160-9292. DOI: 10.1109/tpami.2016.2572683. URL: http://dx.doi.org/10.1109/TPAMI.2016.2572683.

[36] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.4 (2018), pp. 834–848. ISSN: 2160-9292. DOI: 10.1109/tpami.2017.2699184. URL: http://dx.doi.org/10.1109/TPAMI.2017.2699184.

[37] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation". In: *CoRR* abs/1606.02147 (2016). URL: http://arxiv.org/abs/1606.02147.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (2015), pp. 234–241. ISSN: 1611-3349. DOI: 10.1007/978-3-319-24574-4_28. URL: http://dx.doi.org/10.1007/978-3-319-24574-4_28.

[39] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. "Playing for Data: Ground Truth from Computer Games". In: *Lecture Notes in Computer Science* (2016), pp. 102–118. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46475-6_7. URL: http://dx.doi.org/10.1007/978-3-319-46475-6_7.

[40] Yuhua Chen, Wen Li, and Luc Van Gool. "ROAD: Reality Oriented Adaptation for Semantic Segmentation of Urban Scenes". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). DOI: 10.1109/cvpr.2018.00823. URL: http://dx.doi.org/10.1109/CVPR.2018.00823.

[41] Yang Zhang, Philip David, and Boqing Gong. "Curriculum Domain Adaptation for Semantic Segmentation of Urban Scenes". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017). DOI: 10.1109/iccv.2017.223. URL: http://dx.doi.org/10.1109/ICCV.2017.223.

[42] Judy Hoffman, Dequan Wang, Fisher Yu, and Trevor Darrell. "Fcns in the wild: Pixel-level adversarial and constraint-based adaptation". In: *arXiv preprint arXiv:1612.02649* (2016).

[43] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[44] Mingxing Tan, Ruoming Pang, and Quoc V Le. "Efficientdet: Scalable and efficient object detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10781–10790.

[45] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818.

[46] Clement Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. "Digging Into Self-Supervised Monocular Depth Estimation". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). DOI: 10.1109/iccv.2019.00393. URL: http://dx.doi.org/10.1109/ICCV.2019.00393.

[47] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[48] Pierluigi Zama Ramirez, Alessio Tonioni, Samuele Salti, and Luigi Di Stefano. "Learning Across Tasks and Domains". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). DOI: 10.1109/iccv.2019.00820. URL: http://dx.doi.org/10.1109/ICCV.2019.00820.

[49] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. "A Comprehensive Survey on Transfer Learning". In: *arXiv preprint arXiv:1911.02685* (2019).

[50] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[51] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[52] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[53] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. ISSN: 2160-9292. DOI: 10.1109/tpami.2016.2577031. URL: http://dx.doi.org/10.1109/TPAMI.2016.2577031.

[54] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. "Mask R-CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017). DOI: 10.1109/iccv.2017.322. URL: http://dx.doi.org/10.1109/ICCV.2017.322.

[55] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[56] Arghya Pal and Vineeth N Balasubramanian. "Zero-Shot Task Transfer". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[57] Mei Wang and Weihong Deng. "Deep visual domain adaptation: A survey". In: *Neurocomputing* 312 (2018), pp. 135–153. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.05.083. URL: http://dx.doi.org/10.1016/j.neucom.2018.05.083.

[58] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. "Geodesic flow kernel for unsupervised domain adaptation". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 2066–2073.

[59] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. "Learning Transferable Features with Deep Adaptation Networks". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 97–105.

[60] Yaroslav Ganin and Victor Lempitsky. "Unsupervised Domain Adaptation by Backpropagation". In: *International Conference on Machine Learning*. 2015, pp. 1180–1189.

[61] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. "Domain-adversarial training of neural networks". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2096–2030.

[62] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. "Adversarial discriminative domain adaptation". In: *Computer Vision and Pattern Recognition (CVPR)*. 2017.

[63] Ruijia Xu, Guanbin Li, Jihan Yang, and Liang Lin. "Larger Norm More Transferable: An Adaptive Feature Norm Approach for Unsupervised Domain Adaptation". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1426–1435.

[64] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[65] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks". In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[66] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. "Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: 10.1109/cvpr.2017.18. URL: http://dx.doi.org/10.1109/CVPR.2017.18.

[67] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. "Image-To-Image Translation With Conditional Adversarial Networks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[68] Fabio Pizzati, Raoul de Charette, Michela Zaccaria, and Pietro Cerri. "Domain bridge for unpaired image-to-image translation and unsupervised domain adaptation". In: *The IEEE Winter Conference on Applications of Computer Vision*. 2020, pp. 2990–2998.

[69] Yang Zhang, Philip David, and Boqing Gong. "Curriculum domain adaptation for semantic segmentation of urban scenes". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2017.

[70] Pierluigi Zama Ramirez, Alessio Tonioni, and Luigi Di Stefano. "Exploiting semantics in adversarial training for image-level domain adaptation". In: *2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS)*. IEEE. 2018, pp. 49–54.

[71] Wei-Lun Chang, Hui-Po Wang, Wen-Hsiao Peng, and Wei-Chen Chiu. "All about structure: Adapting structural information across domains for boosting semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1900–1909.

[72] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. "CyCADA: Cycle-Consistent Adversarial Domain Adaptation". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 1989–1998.

[73] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. "Learning from simulated and unsupervised images through adversarial training". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[74] Yiheng Zhang, Zhaofan Qiu, Ting Yao, Dong Liu, and Tao Mei. "Fully Convolutional Adaptation Networks for Semantic Segmentation". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). DOI: `10.1109/cvpr.2018.00712`. URL: `http://dx.doi.org/10.1109/CVPR.2018.00712`.

[75] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser Nam Lim, and Rama Chellappa. "Learning from Synthetic Data: Addressing Domain Shift for Semantic Segmentation". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). DOI: `10.1109/cvpr.2018.00395`. URL: `http://dx.doi.org/10.1109/CVPR.2018.00395`.

[76] Fei Pan, Inkyu Shin, Francois Rameau, Seokju Lee, and In So Kweon. "Unsupervised Intra-domain Adaptation for Semantic Segmentation through Self-Supervision". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 3764–3773.

[77] Myeongjin Kim and Hyeran Byun. "Learning Texture Invariant Representation for Domain Adaptation of Semantic Segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 12975–12984.

[78] Alessio Tonioni, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. "Unsupervised Adaptation for Deep Stereo". In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[79] Chuanxia Zheng, Tat-Jen Cham, and Jianfei Cai. "T2Net: Synthetic-to-Realistic Translation for Solving Single-Image Depth Estimation Tasks". In: *The European Conference on Computer Vision (ECCV)*. Sept. 2018.

[80] Alessio Tonioni, Fabio Tosi, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. "Real-Time Self-Adaptive Deep Stereo". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[81] Iasonas Kokkinos. "UberNet: Training a Universal Convolutional Neural Network for Low-, Mid-, and High-Level Vision Using Diverse Datasets and Limited Memory". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: `10.1109/cvpr.2017.579`. URL: `http://dx.doi.org/10.1109/CVPR.2017.579`.

[82] Pierluigi Zama Ramirez, Matteo Poggi, Fabio Tosi, Stefano Mattoccia, and Luigi Di Stefano. "Geometry meets semantics for semi-supervised monocular depth estimation". In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 298–313.

[83] Roberto Cipolla, Yarin Gal, and Alex Kendall. "Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). DOI: `10.1109/cvpr.2018.00781`. URL: `http://dx.doi.org/10.1109/CVPR.2018.00781`.

[84] Fabio Tosi, Filippo Aleotti, Pierluigi Zama Ramirez, Matteo Poggi, Samuele Salti, Luigi Di Stefano, and Stefano Mattoccia. "Distilled semantics for comprehensive scene understanding from videos". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020.

[85] Amir R Zamir, Alexander Sax, Nikhil Cheerla, Rohan Suri, Zhangjie Cao, Jitendra Malik, and Leonidas J Guibas. "Robust Learning Through Cross-Task Consistency". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11197–11206.

[86] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. "Simultaneous deep transfer across domains and tasks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4068–4076.

[87] Jogendra Nath Kundu, Nishank Lakkakula, and R Venkatesh Babu. "UM-Adapt: Unsupervised Multi-Task Adaptation Using Adversarial Cross-Task Distillation". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1436–1445.

[88] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. “Playing for data: Ground truth from computer games”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 102–118.

[89] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. “The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3234–3243.

[90] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[91] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R Benenson, U. Franke, S. Roth, and B. Schiele. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[92] Yi-Hsin Chen, Wei-Yu Chen, Yu-Ting Chen, Bo-Cheng Tsai, Yu-Chiang Frank Wang, and Min Sun. “No More Discrimination: Cross City Adaptation of Road Scene Segmenters”. In: *ICCV*. 2017.

[93] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. “Learning to Adapt Structured Output Space for Semantic Segmentation”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018).

[94] Heiko Hirschmuller. “Accurate and efficient stereo processing by semi-global matching and mutual information”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. IEEE. 2005, pp. 807–814.

[95] Xavier Soria, Edgar Riba, and Angel Sappa. “Dense Extreme Inception Network: Towards a Robust CNN Model for Edge Detection”. In: *The IEEE Winter Conference on Applications of Computer Vision (WACV ’20)*. 2020.

[96] Zhonghao Wang, Mo Yu, Yunchao Wei, Rogerio Feris, Jinjun Xiong, Wen-mei Hwu, Thomas S. Huang, and Honghui Shi. “Differential Treatment for Stuff and Things: A Simple Unsupervised Domain Adaptation Method for Semantic Segmentation”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020). DOI: `10.1109/cvpr42600.2020.01265`. URL: `http://dx.doi.org/10.1109/cvpr42600.2020.01265`.

[97] Jianbo Jiao, Ying Cao, Yibing Song, and Rynson Lau. “Look deeper into depth: Monocular depth estimation with semantic booster and attention-driven loss”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 53–69.

[98] Marvin Klingner, Jan-Aike Termöhlen, Jonas Mikolajczyk, and Tim Fingscheidt. “Self-supervised Monocular Depth Estimation: Solving the Dynamic Object Problem by Semantic Guidance”. In: *Lecture Notes in Computer Science* (2020), pp. 582–600. ISSN: 1611-3349. DOI: 10.1007/978-3-030-58565-5_35. URL: http://dx.doi.org/10.1007/978-3-030-58565-5_35.

[99] Vitor Guizilini, Rui Hou, Jie Li, Rares Ambrus, and Adrien Gaidon. “Semantically-Guided Representation Learning for Self-Supervised Monocular Depth”. In: *Proceedings of the Eighth International Conference on Learning Representations (ICLR)*. 2020.

[100] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. “Unsupervised monocular depth estimation with left-right consistency”. In: *CVPR*. 2017.

[101] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. “Digging into self-supervised monocular depth estimation”. In: *ICCV*. 2019.

[102] Fabio Tosi, Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. “Learning monocular depth estimation infusing traditional stereo knowledge”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[103] Jamie Watson, Michael Firman, Gabriel J Brostow, and Daniyar Turmukhambetov. “Self-Supervised Monocular Depth Hints”. In: *ICCV*. 2019.

[104] Kohei Watanabe, Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada. “Multichannel Semantic Segmentation with Unsupervised Domain Adaptation”. In: *Computer Vision – ECCV 2018 Workshops* (2019), pp. 600–616. ISSN: 1611-3349.

[105] Tuan-Hung Vu, Himalaya Jain, Maxime Bucher, Matthieu Cord, and Patrick Perez Perez. “DADA: Depth-Aware Domain Adaptation in Semantic Segmentation”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). DOI: 10.1109/iccv.2019.00746. URL: http://dx.doi.org/10.1109/ICCV.2019.00746.

[106] Kuan-Hui Lee, German Ros, Jie Li, and Adrien Gaidon. “Spigan: Privileged adversarial learning from simulation”. In: *International Conference on Learning Representations*. 2019.

[107] Yuhua Chen, Wen Li, Xiaoran Chen, and Luc Van Gool. “Learning semantic segmentation from synthetic data: A geometrically guided input-output adaptation approach”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1841–1850.

[108] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. “Unsupervised cnn for single view depth estimation: Geometry to the rescue”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 740–756.

[109] Chao Zhou, Hong Zhang, Xiaoyong Shen, and Jiaya Jia. “Unsupervised Learning of Stereo Matching”. In: *ICCV*. 2017.

[110] Yang Zou, Zhiding Yu, Xiaofeng Liu, B. V. K. Vijaya Kumar, and Jinsong Wang. "Confidence Regularized Self-Training". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). DOI: 10.1109/iccv.2019.00608. URL: http://dx.doi.org/10.1109/ICCV.2019.00608.

[111] Yang Zou, Zhiding Yu, Xiaofeng Liu, B.V.K. Vijaya Kumar, and Jinsong Wang. "Confidence Regularized Self-Training". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2019.

[112] Zhedong Zheng and Yi Yang. "Rectifying Pseudo Label Learning via Uncertainty Estimation for Domain Adaptive Semantic Segmentation". In: *International Journal of Computer Vision (IJCV)* (2020). DOI: 10.1007/s11263-020-01395-y.

[113] Qing Lian, Lixin Duan, Fengmao Lv, and Boqing Gong. "Constructing Self-Motivated Pyramid Curriculums for Cross-Domain Semantic Segmentation: A Non-Adversarial Approach". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). DOI: 10.1109/iccv.2019.00686. URL: http://dx.doi.org/10.1109/ICCV.2019.00686.

[114] Fei Pan, Inkyu Shin, Francois Rameau, Seokju Lee, and In So Kweon. "Unsupervised Intra-Domain Adaptation for Semantic Segmentation Through Self-Supervision". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020). DOI: 10.1109/cvpr42600.2020.00382. URL: http://dx.doi.org/10.1109/cvpr42600.2020.00382.

[115] Ke Mei, Chuang Zhu, Jiaqi Zou, and Shanghang Zhang. "Instance Adaptive Self-Training for Unsupervised Domain Adaptation". In: *The European Conference on Computer Vision (ECCV)*. 2020.

[116] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[117] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation". In: *CoRR* abs/1606.02147 (2016). URL: http://arxiv.org/abs/1606.02147.

[118] Zhedong Zheng and Yi Yang. "Unsupervised Scene Adaptation with Memory Regularization in vivo". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (2020). DOI: 10.24963/ijcai.2020/150. URL: http://dx.doi.org/10.24963/ijcai.2020/150.

[119] Haoran Wang, Tong Shen, Wei Zhang, Lingyu Duan, and Tao Mei. "Classes Matter: A Fine-grained Adversarial Approach to Cross-domain Semantic Segmentation". In: *The European Conference on Computer Vision (ECCV)*. 2020.

[120] D. Lee. “Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks”. In: *Workshop on challenges in representation learning, ICML*. 2013.

[121] Viktor Olsson, Wilhelm Tranheden, Juliano Pinto, and Lennart Svensson. *ClassMix: Segmentation-Based Data Augmentation for Semi-Supervised Learning*. 2020. arXiv: 2007.07936.

[122] Wilhelm Tranheden, Viktor Olsson, Juliano Pinto, and Lennart Svensson. “DACS: Domain Adaptation via Cross-Domain Mixed Sampling”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2021, pp. 1379–1389.

[123] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[124] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. “Dilated Residual Networks”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2017.

[125] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 448–456. URL: http://dl.acm.org/citation.cfm?id=3045118.3045167.

[126] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[127] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[128] Leslie N. Smith and Nicholay Topin. “Super-convergence: very fast training of neural networks using large learning rates”. In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications* (2019). Ed. by TienEditor Pham. DOI: 10.1117/12.2520589. URL: http://dx.doi.org/10.1117/12.2520589.

[129] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. “CyCADA: Cycle-Consistent Adversarial Domain Adaptation”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1989–1998.

[130] Wei-Lun Chang, Hui-Po Wang, Wen-Hsiao Peng, and Wei-Chen Chiu. "All About Structure: Adapting Structural Information Across Domains for Boosting Semantic Segmentation". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). DOI: 10.1109/cvpr.2019.00200. URL: http://dx.doi.org/10.1109/CVPR.2019.00200.

[131] Yifu Chen, Arnaud Dapogny, and Matthieu Cord. "SEMEDA: Enhancing segmentation precision with semantic edge aware loss". In: *Pattern Recognition* 108 (2020), p. 107557. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2020.107557. URL: http://dx.doi.org/10.1016/j.patcog.2020.107557.

[132] Tsung-Wei Ke, Jyh-Jing Hwang, Ziwei Liu, and Stella X. Yu. "Adaptive Affinity Fields for Semantic Segmentation". In: *Lecture Notes in Computer Science* (2018), pp. 605–621. ISSN: 1611-3349. DOI: 10.1007/978-3-030-01246-5_36. URL: http://dx.doi.org/10.1007/978-3-030-01246-5_36.

[133] Liang-Chieh Chen, Jonathan T. Barron, George Papandreou, Kevin Murphy, and Alan L. Yuille. "Semantic Image Segmentation with Task-Specific Edge Detection Using CNNs and a Discriminatively Trained Domain Transform". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: 10.1109/cvpr.2016.492. URL: http://dx.doi.org/10.1109/CVPR.2016.492.

[134] Jianlong Yuan, Zelu Deng, Shu Wang, and Zhenbo Luo. "Multi Receptive Field Network for Semantic Segmentation". In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2020). DOI: 10.1109/wacv45572.2020.9093264. URL: http://dx.doi.org/10.1109/WACV45572.2020.9093264.

[135] Henghui Ding, Xudong Jiang, Ai Qun Liu, Nadia Magnenat Thalmann, and Gang Wang. "Boundary-Aware Feature Propagation for Scene Segmentation". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). DOI: 10.1109/iccv.2019.00692. URL: http://dx.doi.org/10.1109/ICCV.2019.00692.

[136] J Canny. "A Computational Approach to Edge Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (June 1986), pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851. URL: https://doi.org/10.1109/TPAMI.1986.4767851.

[137] Timnit Gebru, Judy Hoffman, and Li Fei-Fei. "Fine-grained recognition in the wild: A multi-task domain adaptation approach". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1349–1358.

[138] Jinwoo Choi, Gaurav Sharma, Samuel Schulter, and Jia-Bin Huang. "Shuffle and attend: Video domain adaptation". In: *European Conference on Computer Vision*. Springer. 2020, pp. 678–695.

[139] Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A. Efros. "Unsupervised Domain Adaptation through Self-Supervision". In: *arXiv preprint arXiv:1909.11825* (2019).

[140] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness." In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bygh9j09KX.

[141] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu koray. "Spatial Transformer Networks". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/33ceb07bf4eeb3da587e268d663aba1a-Paper.pdf.

[142] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. "Cutmix: Regularization strategy to train strong classifiers with localizable features". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 6023–6032.

[143] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. "Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[144] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. "Simple copy-paste is a strong data augmentation method for instance segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2918–2928.

[145] Yi-Hsuan Tsai, Kihyuk Sohn, Samuel Schulter, and Manmohan Chandraker. "Domain Adaptation for Structured Output via Discriminative Patch Representations". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). DOI: 10.1109/iccv.2019.00154. URL: http://dx.doi.org/10.1109/ICCV.2019.00154.

[146] Sujoy Paul, Yi-Hsuan Tsai, Samuel Schulter, Amit K. Roy-Chowdhury, and Manmohan Chandraker. "Domain Adaptive Semantic Segmentation Using Weak Labels". In: *European Conference on Computer Vision (ECCV)*. 2020.

[147] Jinyu Yang, Weizhi An, Chaochao Yan, Peilin Zhao, and Junzhou Huang. "Context-Aware Domain Adaptation in Semantic Segmentation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2021, pp. 514–524.

[148] Ke Mei, Chuang Zhu, Jiaqi Zou, and Shanghang Zhang. "Instance Adaptive Self-training for Unsupervised Domain Adaptation". In: *Lecture Notes in Computer Science* (2020), pp. 415–430. ISSN: 1611-3349. DOI: 10.1007/978-3-030-58574-7_25. URL: http://dx.doi.org/10.1007/978-3-030-58574-7_25.

[149] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. "Microsoft COCO: Common Objects in Context". In: *Lecture Notes in Computer Science* (2014), pp. 740–755. ISSN: 1611-3349. DOI: 10.1007/978-3-319-10602-1_48. URL: http://dx.doi.org/10.1007/978-3-319-10602-1_48.

[150] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[151] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

[152] Tuan-Hung Vu, Himalaya Jain, Maxime Bucher, Matthieu Cord, and Patrick Perez. "ADVENT: Adversarial Entropy Minimization for Domain Adaptation in Semantic Segmentation". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). DOI: 10.1109/cvpr.2019.00262. URL: http://dx.doi.org/10.1109/CVPR.2019.00262.

[153] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818.

[154] Philipp Krähenbühl and Vladlen Koltun. "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *Advances in Neural Information Processing Systems* 24 (2011), pp. 109–117.

[155] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[156] Pushmeet Kohli, Philip HS Torr, et al. "Robust higher order potentials for enforcing label consistency". In: *International Journal of Computer Vision* 82.3 (2009), pp. 302–324.

[157] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. "Rotation invariant spherical harmonic representation of 3 d shape descriptors". In: *Symposium on geometry processing*. Vol. 6. 2003, pp. 156–164.

[158] Kai O Arras, Oscar Martinez Mozos, and Wolfram Burgard. "Using boosted features for the detection of people in 2d range data". In: *Proceedings 2007 IEEE international conference on robotics and automation*. IEEE. 2007, pp. 3402–3407.

[159] Samuele Salti, Federico Tombari, and Luigi Di Stefano. "On the use of implicit shape models for recognition of object categories in 3d data". In: *Asian Conference on Computer Vision*. Springer. 2010, pp. 653–666.

[160] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[161] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *NIPS*. 2017.

[162] Ze Liu, Han Hu, Yue Cao, Zheng Zhang, and Xin Tong. "A closer look at local aggregation operators in point cloud analysis". In: *European Conference on Computer Vision*. Springer. 2020, pp. 326–342.

[163] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. "Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5589–5598.

[164] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. "Grid-gcn for fast and scalable point cloud learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5661–5670.

[165] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. "Relation-shape convolutional neural network for point cloud analysis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8895–8904.

[166] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. "Pointwise convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 984–993.

[167] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. "Dynamic graph cnn for learning on point clouds". In: *Acm Transactions On Graphics (tog)* 38.5 (2019), pp. 1–12.

[168] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. "Kpconv: Flexible and deformable convolution for point clouds". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6411–6420.

[169] Jiaxin Li, Ben M Chen, and Gim Hee Lee. "So-net: Self-organizing network for point cloud analysis". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9397–9406.

[170] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3d shapenets: A deep representation for volumetric shapes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[171] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012* (2015).

[172] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1588–1597.

[173] Ankit Goyal, Hei Law, Bowei Liu, Alejandro Newell, and Jia Deng. "Revisiting Point Cloud Shape Classification with a Simple and Effective Baseline". In: *International Conference on Machine Learning* (2021).

[174] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. "Scannet: Richly-annotated 3d reconstructions of indoor scenes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5828–5839.

[175] Binh-Son Hua, Quang-Hieu Pham, Duc Thanh Nguyen, Minh-Khoi Tran, Lap-Fai Yu, and Sai-Kit Yeung. "Scenenn: A scene meshes dataset with annotations". In: *2016 Fourth International Conference on 3D Vision (3DV)*. IEEE. 2016, pp. 92–101.

[176] Kuniaki Saito, Kohei Watanabe, Y. Ushiku, and T. Harada. "Maximum Classifier Discrepancy for Unsupervised Domain Adaptation". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 3723–3732.

[177] Panos Achlioptas, O. Diamanti, Ioannis Mitliagkas, and L. Guibas. "Learning Representations and Generative Models for 3D Point Clouds". In: *ICML*. 2018.

[178] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. "AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation". In: *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[179] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. "Foldingnet: Point cloud autoencoder via deep grid deformation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 206–215.

[180] Jiahao Pang, Duanshun Li, and Dong Tian. "Tearingnet: Point cloud autoencoder to learn topology-friendly representations". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7453–7462.

[181] Xuelin Chen, Baoquan Chen, and Niloy J Mitra. "Unpaired Point Cloud Completion on Real Scans using Adversarial Training". In: *International Conference on Learning Representations*. 2019.

[182] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3D ShapeNets: A deep representation for volumetric shapes". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1912–1920. DOI: 10.1109/CVPR.2015.7298801.

[183] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. *Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data*. 2019. arXiv: 1908.04616 [cs.CV].

[184] Wouter M Kouw and Marco Loog. "A review of domain adaptation without target labels". In: *IEEE transactions on pattern analysis and machine intelligence* 43.3 (2019), pp. 766–785.

[185] Angela Dai, Matthias Nießner, Michael Zollöfer, Shahram Izadi, and Christian Theobalt. "BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration". In: *ACM Transactions on Graphics 2017 (TOG)* (2017).

[186] D. Lee. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks". In: *International Conference on Machine Learning (ICML) Workshop*. 2013.

[187] Haowen Deng, Tolga Birdal, and Slobodan Ilic. "Ppf-foldnet: Unsupervised learning of rotation invariant 3d local descriptors". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 602–618.

[188] Riccardo Spezialetti, Samuele Salti, and Luigi Di Stefano. "Learning an effective equivariant 3d descriptor without supervision". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 6401–6410.

[189] Pedro H. O. Pinheiro. "Unsupervised Domain Adaptation with Similarity Learning". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 8004–8013.

[190] Guoliang Kang, Lu Jiang, Yi Yang, and Alexander G Hauptmann. "Contrastive Adaptation Network for Unsupervised Domain Adaptation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4893–4902.

[191] Antti Tarvainen and Harri Valpola. "Mean Teachers Are Better Role Models: Weight-Averaged Consistency Targets Improve Semi-Supervised Deep Learning Results". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1195–1204.

[192] Zhongyi Pei, Zhangjie Cao, Mingsheng Long, and Jianmin Wang. "Multi-Adversarial Domain Adaptation". In: *AAAI*. 2018.

[193] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I. Jordan. "Deep Transfer Learning with Joint Adaptation Networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 2208–2217.

[194] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "The earth mover's distance as a metric for image retrieval". In: *International journal of computer vision* 40.2 (2000), pp. 99–121.

[195] Riccardo Spezialetti, Federico Stella, Marlon Marcon, Luciano Silva, Samuele Salti, and Luigi Di Stefano. "Learning to Orient Surfaces by Self-supervised Spherical CNNs". In: *Advances in Neural Information Processing Systems* 33 (2020).

[196] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. "3dmatch: Learning local geometric descriptors from rgb-d reconstructions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1802–1811.

[197] Chuan-Xian Ren, Pengfei Ge, Peiyi Yang, and Shuicheng Yan. "Learning Target-Domain-Specific Classifier for Partial Domain Adaptation". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.5 (2021), pp. 1989–2001. ISSN: 2162-2388. DOI: 10.1109/tnnls.2020.2995648. URL: http://dx.doi.org/10.1109/TNNLS.2020.2995648.

[198] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[199] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=Skq89Scxx.

[200] Yuefan Shen, Yanchao Yang, Mi Yan, He Wang, Youyi Zheng, and Leonidas J. Guibas. "Domain Adaptation on Point Clouds via Geometry-Aware Implicits". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 7223–7232.

[201] Longkun Zou, Hui Tang, Ke Chen, and Kui Jia. "Geometry-Aware Self-Training for Unsupervised Domain Adaptation on Object Point Clouds". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 6403–6412.

[202] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. "Emerging Properties in Self-Supervised Vision Transformers". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2021.

[203] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. "Deep clustering for unsupervised learning of visual features". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 132–149.

[204] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. "Bootstrap your own latent: A new approach to self-supervised learning". In: *arXiv preprint arXiv:2006.07733* (2020).

[205] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. "Unsupervised feature learning via non-parametric instance discrimination". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3733–3742.

[206] Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. "RefRec: Pseudo-labels Refinement via Shape Reconstruction for Unsupervised 3D Domain Adaptation". In: *2021 International Conference on 3D Vision (3DV)*. IEEE. 2021.

[207] Hehe Fan, Xiaojun Chang, Wanyue Zhang, Yi Cheng, Ying Sun, and Mohan Kankanhalli. "Self-Supervised Global-Local Structure Modeling for Point Cloud Domain Adaptation With Reliable Voted Pseudo Labels". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 6377–6386.

[208] Zhedong Zheng and Yi Yang. "Unsupervised Scene Adaptation with Memory Regularization in vivo". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (2020).

[209] Yang Zou, Zhiding Yu, Xiaofeng Liu, B. V. K. Vijaya Kumar, and Jinsong Wang. "Confidence Regularized Self-Training". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019).

[210] Yang Zou, Zhiding Yu, BVK Vijaya Kumar, and Jinsong Wang. "Unsupervised Domain Adaptation for Semantic Segmentation via Class-Balanced Self-Training". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 289–305.

[211] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. ISSN: 2162-2388.

[212] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[213] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. "Momentum contrast for unsupervised visual representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.

[214] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).

[215] Zhaoliang Lun, Evangelos Kalogerakis, and Alla Sheffer. "Elements of style: learning perceptual shape style similarity". In: *ACM Transactions on graphics (TOG)* 34.4 (2015), pp. 1–14.

[216] Kai Xu, Honghua Li, Hao Zhang, Daniel Cohen-Or, Yueshan Xiong, and Zhi-Quan Cheng. "Style-content separation by anisotropic part scales". In: *ACM SIGGRAPH Asia 2010 papers*. 2010, pp. 1–10.

[217] Ke Mei, Chuang Zhu, Jiaqi Zou, and Shanghang Zhang. "Instance Adaptive Self-training for Unsupervised Domain Adaptation". In: *Lecture Notes in Computer Science* (2020), pp. 415–430. ISSN: 1611-3349.

[218] Inkyu Shin, Sanghyun Woo, Fei Pan, and In So Kweon. "Two-Phase Pseudo Label Densification for Self-training Based Domain Adaptation". In: *Lecture Notes in Computer Science* (2020), pp. 532–548. ISSN: 1611-3349.

[219] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*. 2021, pp. 1548–1554.

[220] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[221] Xu Yan, Jiantao Gao, Chaoda Zheng, Chao Zheng, Ruimao Zhang, Shuguang Cui, and Zhen Li. "2DPASS: 2D Priors Assisted Semantic Segmentation on LiDAR Point Clouds". In: *European Conference on Computer Vision*. Springer. 2022, pp. 677–695.

[222] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[223] Larissa T Triess, Mariella Dreissig, Christoph B Rist, and J Marius Zöllner. “A survey on deep domain adaptation for lidar perception”. In: *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*. IEEE. 2021, pp. 350–357.

[224] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. “Octnet: Learning deep 3d representations at high resolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3577–3586.

[225] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. “Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2088–2096.

[226] Donald Meagher. “Geometric modeling using octree encoding”. In: *Computer graphics and image processing* 19.2 (1982), pp. 129–147.

[227] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. “3D Semantic Segmentation with Submanifold Sparse Convolutional Networks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). DOI: 10.1109/cvpr.2018.00961. URL: http://dx.doi.org/10.1109/CVPR.2018.00961.

[228] Christopher Choy, JunYoung Gwak, and Silvio Savarese. “4d spatio-temporal convnets: Minkowski convolutional neural networks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3075–3084.

[229] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. “Cylindrical and asymmetrical 3d convolution networks for lidar segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 9939–9948.

[230] Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh. “Polarnet: An improved grid representation for online lidar point clouds semantic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9601–9610.

[231] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. “Searching efficient 3d architectures with sparse point-voxel convolution”. In: *European conference on computer vision*. Springer. 2020, pp. 685–702.

[232] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. "Randla-net: Efficient semantic segmentation of large-scale point clouds". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11108–11117.

[233] Inkyu Shin, Yi-Hsuan Tsai, Bingbing Zhuang, Samuel Schulter, Buyu Liu, Sparsh Garg, In So Kweon, and Kuk-Jin Yoon. "MM-TTA: multi-modal test-time adaptation for 3d semantic segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16928–16937.

[234] Christoph B Rist, Markus Enzweiler, and Dariu M Gavrila. "Cross-sensor deep domain adaptation for LiDAR detection and segmentation". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1535–1542.

[235] Mrigank Rochan, Shubhra Aich, Eduardo R Corral-Soto, Amir Nabatchian, and Bingbing Liu. "Unsupervised domain adaptation in lidar semantic segmentation with self-supervision and gated adapters". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 2649–2655.

[236] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. "nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[237] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, et al. "A2d2: Audi autonomous driving dataset". In: *arXiv preprint arXiv:2004.06320* (2020).

[238] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. "VirtualWorlds as Proxy for Multi-object Tracking Analysis". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: 10.1109/cvpr.2016.470. URL: http://dx.doi.org/10.1109/CVPR.2016.470.

[239] Pierluigi Zama Ramirez, Matteo Poggi, Fabio Tosi, Stefano Mattoccia, and Luigi Di Stefano. "Geometry meets semantics for semi-supervised monocular depth estimation". In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 298–313.

[240] Xinjing Cheng, Peng Wang, and Ruigang Yang. "Depth estimation via affinity learned with convolutional spatial propagation network". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 103–119.

[241] Sifei Liu, Shalini De Mello, Jinwei Gu, Guangyu Zhong, Ming-Hsuan Yang, and Jan Kautz. “Learning Affinity via Spatial Propagation Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/c22abfa379f38b5b0411bc11fa9bf92f-Paper.pdf.

[242] Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. “Self-supervised Sparse-to-Dense: Self-supervised Depth Completion from LiDAR and Monocular Camera”. In: *ICRA*. 2019.

[243] Alex Wong and Stefano Soatto. “Unsupervised Depth Completion with Calibrated Backprojection Layers”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 12747–12756.

[244] Mu Hu, Shuling Wang, Bin Li, Shiyu Ning, Li Fan, and Xiaojin Gong. “PENet: Towards Precise and Efficient Image Guided Depth Completion”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 13656–13662. DOI: 10.1109/ICRA48506.2021.9561035.

[245] Jinsun Park, Kyungdon Joo, Zhe Hu, Chi-Kuei Liu, and In So Kweon. “Non-local spatial propagation network for depth completion”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 120–136.

[246] Andrea Conti, Matteo Poggi, Filippo Aleotti, and Stefano Mattoccia. “Unsupervised confidence for LiDAR depth maps and applications”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IROS. 2022.

[247] Yawei Luo, Liang Zheng, Tao Guan, Junqing Yu, and Yi Yang. “Taking a closer look at domain shift: Category-level adversaries for semantics consistent domain adaptation”. In: *Proceedings of the Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 2507–2516.

[248] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. “Sparsity Invariant CNNs”. In: *2017 International Conference on 3D Vision (3DV)*. 2017.

[249] Vitor Guizilini, Rares Ambrus, Wolfram Burgard, and Adrien Gaidon. “Sparse Auxiliary Networks for Unified Monocular Depth Prediction and Completion”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[250] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. “Understanding the effective receptive field in deep convolutional neural networks”. In: *Advances in neural information processing systems* 29 (2016).

[251] Suman Saha, Anton Obukhov, Danda Pani Paudel, Menelaos Kanakis, Yuhua Chen, Stamatios Georgoulis, and Luc Van Gool. "Learning to Relate Depth and Semantics for Unsupervised Domain Adaptation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 8197–8207.

[252] Adriano Cardace, Luca De Luigi, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. "Plugging Self-Supervised Monocular Depth into Unsupervised Domain Adaptation for Semantic Segmentation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2022, pp. 1129–1139.

[253] Caner Hazirbas, Lingni Ma, Csaba Domokos, and Daniel Cremers. "Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture". In: *Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part I 13*. Springer. 2017, pp. 213–228.

[254] Cristiano Saltori, Fabio Galasso, Giuseppe Fiameni, Nicu Sebe, Elisa Ricci, and Fabio Poiesi. "Cosmix: Compositional semantic mix for domain adaptation in 3d lidar segmentation". In: *European Conference on Computer Vision*. Springer. 2022, pp. 586–602.

[255] Dong-Hyun Lee. "Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks". In: *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)* (July 2013).

[256] Pietro Morerio, Jacopo Cavazza, and Vittorio Murino. "Minimal-Entropy Correlation Alignment for Unsupervised Deep Domain Adaptation". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rJWechg0Z.

[257] Yunsheng Li, Lu Yuan, and Nuno Vasconcelos. "Bidirectional Learning for Domain Adaptation of Semantic Segmentation". In: *CVPR*. 2019.

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

[259] 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". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5648–5656.

[260] Shuran Song and Jianxiong Xiao. "Deep sliding shapes for amodal 3d object detection in rgb-d images". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 808–816.

[261] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction". In: *European conference on computer vision*. Springer. 2016, pp. 628–644.

[262] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. “Learning a predictable and generative vector representation for objects”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 484–499.

[263] Danilo Jimenez Rezende, SM Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. “Unsupervised learning of 3d structure from images”. In: *Advances in neural information processing systems* 29 (2016).

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

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

[266] 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”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10529–10538.

[267] Haoxuan You, Yifan Feng, Rongrong Ji, and Yue Gao. “Pvnet: A joint convolutional network of point cloud and multi-view for 3d shape recognition”. In: *Proceedings of the 26th ACM international conference on Multimedia*. 2018, pp. 1310–1318.

[268] Lei Li, Siyu Zhu, Hongbo Fu, Ping Tan, and Chiew-Lan Tai. “End-to-end learning local multi-view descriptors for 3d point clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1919–1928.

[269] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. “Spidercnn: Deep learning on point sets with parameterized convolutional filters”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 87–102.

[270] Matan Atzmon, Haggai Maron, and Yaron Lipman. “Point Convolutional Neural Networks by Extension Operators”. In: *ACM Transactions on Graphics* 37.4 (2018).

[271] Wenxuan Wu, Zhongang Qi, and Li Fuxin. “Pointconv: Deep convolutional networks on 3d point clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9621–9630.

[272] 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”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 14504–14513.

[273] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. "Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3173–3182.

[274] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. "Deepgcns: Can gcns go as deep as cnns?" In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 9267–9276.

[275] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. "Graph attention convolution for point cloud semantic segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10296–10305.

[276] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. "Pct: Point cloud transformer". In: *Computational Visual Media* 7.2 (2021), pp. 187–199.

[277] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. "Point transformer". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 16259–16268.

[278] Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Jun-Xiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R. Martin. "Subdivision-based Mesh Convolution Networks". In: *ACM Transactions on Graphics* 41.3 (Mar. 2022), pp. 1–16. ISSN: 1557-7368. DOI: 10.1145/3506694. URL: http://dx.doi.org/10.1145/3506694.

[279] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. "Geodesic convolutional neural networks on riemannian manifolds". In: *Proceedings of the IEEE international conference on computer vision workshops*. 2015, pp. 37–45.

[280] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. "Learning shape correspondence with anisotropic convolutional neural networks". In: *Advances in neural information processing systems* 29 (2016).

[281] 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". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4440–4449.

[282] Yuqi Yang, Shilin Liu, Hao Pan, Yang Liu, and Xin Tong. "PFCNN: Convolutional neural networks on 3D surfaces using parallel frames". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13578–13587.

[283] Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. "Surface networks via general covers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 632–641.

[284] Jonas Schult, Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. "Dualconvmesh-net: Joint geodesic and euclidean convolutions on 3d meshes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8612–8622.

[285] 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". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5115–5124.

[286] Dmitriy Smirnov and Justin Solomon. "HodgeNet: learning spectral geometry on triangle meshes". In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–11.

[287] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. "Meshcnn: a network with an edge". In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–12.

[288] Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. "Primal-dual mesh convolutional neural networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 952–963.

[289] Alon Lahav and Ayellet Tal. "Meshwalker: Deep mesh understanding by random walks". In: *ACM Transactions on Graphics (TOG)* 39.6 (2020), pp. 1–13.

[290] 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". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, pp. 837–845.

[291] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. "Meshnet: Mesh neural network for 3d shape representation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 8279–8286.

[292] Xianzhi Li, Ruihui Li, Lei Zhu, Chi-Wing Fu, and Pheng-Ann Heng. "DNF-Net: A deep normal filtering network for mesh denoising". In: *IEEE Transactions on Visualization and Computer Graphics* 27.10 (2020), pp. 4060–4072.

[293] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. "Deep Geometric Texture Synthesis". In: *ACM Trans. Graph.* 39.4 (2020). ISSN: 0730-0301. DOI: `10.1145/3386569.3392471`. URL: `https://doi.org/10.1145/3386569.3392471`.

[294] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. "Neural Fields in Visual Computing and Beyond". In: *arXiv preprint arXiv:2111.11426* (2021).

[295] Julian Chibane, Gerard Pons-Moll, et al. "Neural unsigned distance fields for implicit function learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21638–21652.

[296] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). DOI: 10.1109/cvpr.2019.00025. URL: http://dx.doi.org/10.1109/CVPR.2019.00025.

[297] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. "Implicit Geometric Regularization for Learning Shapes". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3789–3799.

[298] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. "Scene representation networks: Continuous 3d-structure-aware neural scene representations". In: *Advances in Neural Information Processing Systems* 32 (2019).

[299] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. "Local implicit grid representations for 3d scenes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 6001–6010.

[300] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. "Convolutional occupancy networks". In: *European Conference on Computer Vision*. Springer. 2020, pp. 523–540.

[301] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. "Occupancy networks: Learning 3d reconstruction in function space". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4460–4470.

[302] Zhiqin Chen and Hao Zhang. "Learning implicit fields for generative shape modeling". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5939–5948.

[303] 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". In: *ECCV*. 2020.

[304] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. "Implicit neural representations with periodic activation functions". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7462–7473.

[305] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. "KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021). DOI: 10.1109/iccv48922.2021.01407. URL: http://dx.doi.org/10.1109/ICCV48922.2021.01407.

[306] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. "Derf: Decomposed radiance fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 14153–14161.

[307] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. "Neural Sparse Voxel Fields". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 15651–15663. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/b4b758962f17808746e9bb832a6fa4b8-Paper.pdf.

[308] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. "Plenoxels: Radiance Fields without Neural Networks". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022). DOI: 10.1109/cvpr52688.2022.00542. URL: http://dx.doi.org/10.1109/CVPR52688.2022.00542.

[309] Yiqun Wang, Ivan Skorokhodov, and Peter Wonka. *PET-NeuS: Positional Encoding Tri-Planes for Neural Surfaces*. 2023. arXiv: 2305.05594 [cs.CV].

[310] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini de Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. "Efficient Geometry-aware 3D Generative Adversarial Networks". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022). DOI: 10.1109/cvpr52688.2022.01565.

[311] Rundi Wu and Changxi Zheng. "Learning to Generate 3D Shapes from a Single Example". In: *ACM Transactions on Graphics (TOG)* 41.6 (2022).

[312] Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. "Tri-MipRF: Tri-Mip Representation for Efficient Anti-Aliasing Neural Radiance Fields". In: *ICCV*. 2023.

[313] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. "TensoRF: Tensorial Radiance Fields". In: *European Conference on Computer Vision (ECCV)*. 2022.

[314] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Carsten Stoll, and Christian Theobalt. "Patchnets: Patch-based generalizable deep implicit 3d shape representations". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*. Springer. 2020, pp. 293–309.

[315] Sida Peng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. "Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 9054–9063.

[316] Robert Hecht-Nielsen. “On the algebraic structure of feedforward network weight spaces”. In: *Advanced Neural Computers*. Elsevier, 1990, pp. 129–135.

[317] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. “Equivariant Architectures for Learning in Deep Weight Spaces”. In: *International Conference on Machine Learning*. 2023.

[318] Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J Zico Kolter, and Chelsea Finn. “Neural Functional Transformers”. In: *Advances in neural information processing systems* 37 (2023).

[319] 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”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 5694–5725.

[320] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. “Neural geometric level of detail: Real-time rendering with implicit 3D shapes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 11358–11367.

[321] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *ACM Trans. Graph.* 41.4 (July 2022), 102:1–102:15. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127.

[322] Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. “ACORN: Adaptive coordinate networks for neural scene representation”. In: *ACM Trans. Graph. (SIGGRAPH)* 40.4 (2021).

[323] Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. “Learning Smooth Neural Functions via Lipschitz Regularization”. In: *arXiv preprint arXiv:2202.08345* (2022).

[324] Vincent Sitzmann, Eric Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. “Metasdf: Meta-learning signed distance functions”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 10136–10147.

[325] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. “Coin: Compression with implicit neural representations”. In: *arXiv preprint arXiv:2103.03123* (2021).

[326] Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. “Implicit Neural Representations for Image Compression”. In: *arXiv preprint arXiv:2112.04267* (2021).

[327] Yunfan Zhang, Ties van Rozendaal, Johann Brehmer, Markus Nagel, and Taco Cohen. “Implicit Neural Video Compression”. In: *arXiv preprint arXiv:2112.11312* (2021).

[328] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. "Fourier features let networks learn high frequency functions in low dimensional domains". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7537–7547.

[329] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[330] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. "A comprehensive survey on model compression and acceleration". In: *Artificial Intelligence Review* 53.7 (2020), pp. 5113–5155.

[331] William E Lorensen and Harvey E Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *ACM siggraph computer graphics* 21.4 (1987), pp. 163–169.

[332] Haoqiang Fan, Hao Su, and Leonidas J Guibas. "A point set generation network for 3d object reconstruction from a single image". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 605–613.

[333] Cameron R Wolfe and Keld T Lundgaard. "E-Stitchup: Data Augmentation for Pre-Trained Embeddings". In: *arXiv preprint arXiv:1912.00772* (2019).

[334] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. "A Scalable Active Framework for Region Annotation in 3D Shape Collections". In: *SIGGRAPH Asia* (2016).

[335] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. "Learning representations and generative models for 3d point clouds". In: *International conference on machine learning*. PMLR. 2018, pp. 40–49.

[336] Ruihui Li, Xianzhi Li, Ka-Hei Hui, and Chi-Wing Fu. "SP-GAN: Sphere-guided 3D shape generation and manipulation". In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–12.

[337] Liang Pan, Xinyi Chen, Zhongang Cai, Junzhe Zhang, Haiyu Zhao, Shuai Yi, and Ziwei Liu. "Variational relational point completion network". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 8524–8533.

[338] Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. "The Role of Permutation Invariance in Linear Mode Connectivity of Neural Networks". In: *International Conference on Learning Representations*. 2021.

[339] 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". In: *European conference on computer vision*. Springer. 2020, pp. 405–421.

[340] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. "Git re-basin: Merging models modulo permutation symmetries". In: *arXiv preprint arXiv:2209.04836* (2022).

[341] Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. "What do single-view 3d reconstruction networks learn?" In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3405–3414.

[342] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. "DISN: Deep Implicit Surface Network for High-quality Single-view 3D Reconstruction". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/39059724f73a9969845dfe4146c5660e-Paper.pdf.

[343] Suman Saha, Anton Obukhov, Danda Pani Paudel, Menelaos Kanakis, Yuhua Chen, Stamatios Georgoulis, and Luc Van Gool. "Learning to Relate Depth and Semantics for Unsupervised Domain Adaptation". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2021. DOI: 10.1109/cvpr46437.2021.00810. URL: http://dx.doi.org/10.1109/CVPR46437.2021.00810.

[344] Lukas Hoyer, Dengxin Dai, Qin Wang, Yuhua Chen, and Luc Van Gool. "Improving Semi-Supervised and Domain-Adaptive Semantic Segmentation with Self-Supervised Depth Estimation". In: *International Journal of Computer Vision* 131.8 (May 2023), pp. 2070–2096. ISSN: 1573-1405. DOI: 10.1007/s11263-023-01799-6. URL: http://dx.doi.org/10.1007/s11263-023-01799-6.

[345] Qin Wang, Dengxin Dai, Lukas Hoyer, Luc Van Gool, and Olga Fink. "Domain Adaptive Semantic Segmentation with Self-Supervised Depth Estimation". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2021. DOI: 10.1109/iccv48922.2021.00840. URL: http://dx.doi.org/10.1109/ICCV48922.2021.00840.

[346] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. "iBOT: Image BERT Pre-Training with Online Tokenizer". In: *International Conference on Learning Representations (ICLR)* (2022).

[347] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski.

"DINOv2: Learning Robust Visual Features without Supervision". In: *Transactions on Machine Learning Research* (2024). ISSN: 2835-8856. URL: https://openreview.net/forum?id=a68SUt6zFt.

[348] Shengyu Huang, Zan Gojcic, Zian Wang, Francis Williams, Yoni Kasten, Sanja Fidler, Konrad Schindler, and Or Litany. "Neural LiDAR Fields for Novel View Synthesis". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2023. DOI: 10.1109/iccv51070.2023.01672. URL: http://dx.doi.org/10.1109/ICCV51070.2023.01672.

[349] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[350] Pierre Baque, Edoardo Remelli, Francois Fleuret, and Pascal Fua. "Geodesic convolutional shape optimization". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 472–481.

[351] David JJ Toal and Andy J Keane. "Efficient multipoint aerodynamic design optimization via cokriging". In: *Journal of Aircraft* 48.5 (2011), pp. 1685–1695.

[352] Nobuyuki Umetani and Bernd Bickel. "Learning three-dimensional flow for interactive aerodynamic design". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–10.

[353] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. "Accelerating 3D Deep Learning with PyTorch3D". In: *arXiv:2007.08501* (2020).

[354] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).

[355] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[356] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[357] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. "Improved training of wasserstein gans". In: *Advances in neural information processing systems* 30 (2017).

[358] Gizem Yüce, Guillermo Ortiz-Jiménez, Beril Besbinar, and Pascal Frossard. "A Structured Dictionary Perspective on Implicit Neural Representations". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 19228–19238.

[359] Yilun Du, Katie Collins, Josh Tenenbaum, and Vincent Sitzmann. "Learning signal-agnostic manifolds of neural fields". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 8320–8331.

[360] Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. "Generative models as distributions of functions". In: *arXiv preprint arXiv:2102.04776* (2021).

[361] Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. "NerfAcc: Efficient Sampling Accelerates NeRFs." In: *arXiv preprint arXiv:2305.04966* (2023).

[362] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. "Objaverse: A universe of annotated 3d objects". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 13142–13153.