DEPARTMENT OF MATHEMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Mathematics in Data Science

Topological NeRF: 3D reconstruction with NeRFs reinforced with TDA

Topological NeRF: 3D-Rekonstruktion anhand NeRFs erweitert durch TDA

Author:	Yevhenii Sharapov
Supervisor:	Prof. Dr. Bastian Riek
Advisor:	Prof. Dr. Bastian Riek
Submission Date:	14.01.2015

I confirm that this master's thesis in mathematics in data science is my own work and I have documented all sources and material used.

Munich, Submission date   Yevhenii Sharapov

14.01.2025

[illegible]

Abstract

In the recent years topological data analysis (TDA) has found multiple applications in the field of deep learning and is being increasingly adopted for versatile tasks like reconstruction or graph neural networks. Surprisingly, many models can benefit from addition of topological supervision even if this is performed only as regularization routine. Neural radiance fields (NeRFs) are novel family of models, which aim to represent 3D scene by learning intensity and color information of a point in 3D space given location and angle of view. Original approach despite its expressiveness had several drawbacks, which were continuously addressed by recent papers. The way neural radiance fields process data collected from rays cast through the scene leaves a lot of space of application of TDA, which has not yet been elaborated by researchers. In this thesis we attempt to unify NeRFs with TDA and explore different ways of how this might be performed, possible constraints and effects on training and inference process. We also implement a novel Betti Curve Transform deep learning layer, which might be used in an end-to-end fashion on top of persistence diagram construction. Apart from this we implemented such algorithms as UnionFind and Betti Curve Alignment, which allow construction and comparison of Betti Curves. Our studies confirmed that this novel regularization term might be useful in many deep learning tasks. Additionaly we experimented with normalizing flows models endowed with topological features, which have proven to be helpful in capturing overall topology of the generated samples.

Kurzfassung

In letzten Jahren topologische Daten Analyse (topological data analysis TDA) wurde aktiv angewendet im Gebiet von Deep Learning und wird zunehmend für verschiedene Aufgaben wie die Rekonstruktion oder die Nutzung in Graph-Neuronalen Netzwerken eingesetzt. Erstaunlicherweise könnten viele Modelle davon profitieren, wenn man sie mit topologischer Begleitung ausrüstet, selbst wenn es sich lediglich um Regularisierung handelt. Neural Radience Fields (NeRFs) ist eine maßschneidende Familie von Modellen, die sich als Ziel setzen, 3D Szene durch Erlernen von Intensitäts- und Farbinformationen eines Punktes in 3D-Raum gegeben Position und Blickwinkel. Ursprunglicher Ansatz hatte einige Nachteile trotz seiner Kapazitet, die wurden dennoch in neusten Arbeiten angegangen. Die Art und Weise, wie die Strahlen, die durch den Raum geschoßen werden, von NeRF verarbeitet werden bietet vielseitige Möglichkeiten von Topologie Anwendung, die noch nicht eingehend von Forschern erarbeitet worden sind. In dieser Masterarbeit versuchen wir NeRFs mit TDA vereinen und erforschen daher wie man es erziehlen kann, wobei wir auch mögliche Beschränkungen und Wirkungen von Schullung und Inferenz berücksichtigen. Außerdem implementieren wir eine neuartige Deep-Learning-Schicht namens Betti Curve Transform, die auf persistente Diagramme in einer End-to-End-Architektur angewendet werden kann.Darüber hinaus haben wir solche Algorithmen wie UnionFind und Betti Kurve Zusammenstellung umgesetzt, was die Berechnung und Gegenpberstellung von Kurven ermöglicht. Unsere Ergebnisse haben bestätigt, dass diese neue Regularisierung in vielen Deep-Learning-Aufgaben nützlich sein könnte. Zusätzlich haben wir mit Normalizing-Flow-Modellen experimentiert, die mit topologischen Merkmalen ausgestattet sind. Diese haben sich als hilfreich erwiesen, um die Gesamt-Topologie der generierten Proben zu erfassen.

Contents

1 Introduction

In the recent years topological methods have been actively finding their way into the realm of machine learning, expanding the boundaries of the methodology employed. As a relatively young field, topological data analysis relies on the concept of persistent diagram, which was predated by Morse complexes [1, 2] and formally introduced in the early 2000s by Herbert Edelsbrunner et.al.[3]. Serving as a bridge this object paved the way for different computational methods revolving around this tool. Indeed, topology has always been striving to describe structure of spaces and yet has always been lacking application to real world problems, always remaining out of scope and nevertheless posing great interest for applied scientists.

As far as deep learning is concerned, representation learning plays a crucial role in systematizing knowledge about the domain, as it helps to obtain meaningful latent code, comprising the most important information about an object. Topology helped to propagate shape information contained in the original data to latent representations and thus proved to be essential in the field of neural networks. TDA finds application in multiple tasks, ranging from reconstruction and segmentation [4] to graph neural networks [5].

Despite high value of practical application, computational topology is still being a relatively novel field. As such, many of the scientific computing packages can not be used for direct integration into deep learning pipeline, as well as many topological metrics, kernels and models are lacking differentiable implementations allowing for end-to-end application in deep learning tasks. Apart from this TDA has been found to beneficial for numerous setups, where structure of data is of concern. [4, 6]

In 2020 neural radiance field (NeRF) [7] model altered the landscape of implicit 3D modelling [8, 9] from a set of 2D views introducing a novel methodology and shaping a new active field of research, which spans over versatile improvements over the original concept. While being expressive in the core, initial model suffered from multiple issues, for instance, dynamic illumination, occluded objects [10] over views or insufficient number of views for training process. Another problem posed by NeRF was computational efficiency and capability for acting as zero-shot model [11]. Among recent advances involving numerous techniques and approaches, topology has been employed only implicitly.

In this thesis we approach a problem of endowing a classical NeRF architecture with topological supervision and experiment extensively with possible ways of achieving this goal. Our focus is set on accelerating convergence of the original model and proving that topological methods can be applied in this novel field to approach this goal.

We structure our work in a coherent and logical way by starting from outlining basic theoretical foundations spanning elements of algebraic and computational topology and specifi architecture employed along the way. We then proceed with giving a brief outlook on related works in the areas of topological machine learning, implicit 3D reconstruction based on radiance fields and probabilistic models. In the methods chapter we explain, how we are going to approach reconstruction problem and finally move to basic experiments and main results, offering a deep look into our findings and insights for the future works.

2 Data

In this chapter, a brief discussion will be provided on the data that will be used to benchmark our algorithms. Since a novel differentiable implementation of a topological term based on *Betti Curves* has been introduced in this work, benchmarking of some of the algorithms on basic topological toy datasets will also be included. Information on these datasets will not be included here but will be mentioned in chapter6.

2.1 Spheres

Spheres mentioned in paper [12] is a benchmark dataset, used to validate results in *topological autoencoder* experiments section 6.2.1. It is represented by ten 101-dimensional spheres located inside a bigger sphere. This dataset is used for testing models for several purposes. Firstly it is high dimensional and therefore poses a challenge to most of machine learning algorithms, secondary it encodes important topological information, which can be learned and visualized effectively.[12]

2.2 FashionMNIST

FashionMNIST dataset was first introduced in [13] and was used as a benchmarking dataset for computer vision tasks. It consists of 70 000 images from fashion catalogue, downscaled to 28x28 pixels and featuring ten classes - t-shirts/tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots.

2.3 Nerf Synthetic Dataset

Original *NeRF* paper was benchmarked against several datasets.[7]. One them was collected from Blender via 3D modelling and represents some 3D objects available from multiple views, thus allowing to compare models on large amounts of data. This data is stored in the format of images and records, describing camera position (rotations, affine transform and file path). Dataset was sampled uniformly to capture multiple angles from upper hemisphere (and for some objects from whole sphere)

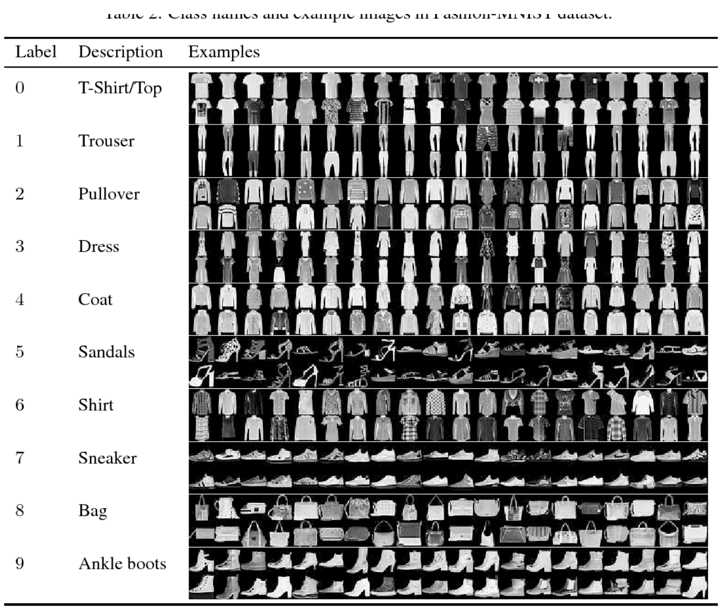


Table 2: Class names and example images in Fashion-MNIST dataset.

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 2.1: **Samples from FashionMNIST Dataset.** Dataset represents complex variaty of objects which serve as a baseline for versatile machine learning tasks.

and then split into training and evaluation datasets with 100 of images for each and test set with 200 samples. The images were captured at resolution of 800x800 pixels. In our experiments they were additionally downsampled to different dimensions to accelerate our study, specifically: 32x32, 96x96, 128x128, 512x512 pixels. Dataset also provides mask of the objects, as most of the canvas of such images is black.

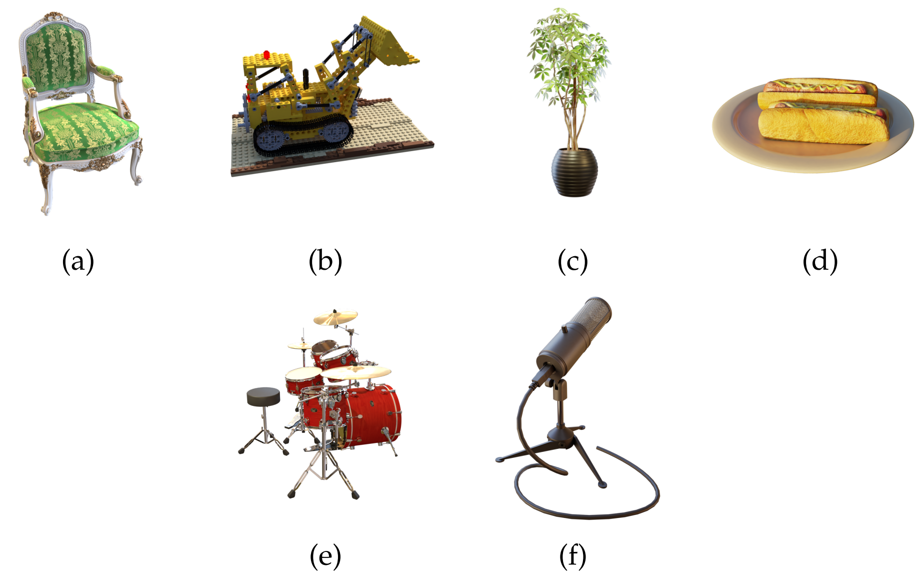


Figure 2.2: **NeRF synthetic dataset example objects in full resolution**. Here (a) - chair, (b) - lego, (c) - ficus, (d) - hotdog, (e) - drum and (f) - microphone.

3 Theoretical Background

In this section, the theoretical foundations of the methodology applied throughout the work are reviewed. The underlying concept of *persistent homology* and metrics over the space of persistence pairings are discussed, followed by the artificial neural networks employed in the research. In addition to the topological aspects of the toolbox, focus is placed on 3D Machine Learning, specifically a thorough analysis of the architectures and their potential applications in the realm of *Topological Data Analysis* (*TDA*). This enabled further exploration of the reconciliation between the two domains.

3.1 Fundamentals of Algebraic Topology

3.1.1 Simplicial and Geometric Complexes

TDA is a novel field of data analysis with the focus on topological properties of the underlying space the dataset is sampled from. Given a finite set of points sampled from Eucledian space one would like to reconstruct original topology of the underlying set they originate from. This target topology might be well approximated by discrete analogues. Let us start by introduction of the concept of an *abstract simplicial complex*:

Definition 3.1.1 (Abstract Simplicial Complex). $\mathbb{A} = \{A | A \in \mathbb{X}, \forall B \subset A, B \in \mathbb{A}, A \neq \emptyset\}$

Definition 3.1.2 (k-Skeleton of a simplex). k-Skeleton of an abstract simplicial complex $\mathbb{A}$ is defined as $Sk_k(\mathbb{A}) = \bigcup_{S \in \mathbb{A}, card(S) \leq k} S$

The dimension of an *abstract simplicial complex* is defined as the highest cardinality of the sets contained within it, minus one. The constituents of the complex are called simplices, and since any subset of a simplex is inherently included in the complex, it is referred to as a face of the simplex. The definition 3.1.1 of an *abstract simplicial complex* is purely combinatorial and extends the concept of graphs, which can be viewed as 2-dimensional *abstract simplicial complex*. However, *abstract simplicial complex* go beyond this by considering not just pairs of vertices, but tuples of vertices. This is why this structure is sometimes referred to as a hypergraph.

Abstract simplicial complex are not dealing with any topological structure and define a structure independent of the space, where the elements could be sampled from, hence necessitating the notion of *geometric simplicial complex*.

Definition 3.1.3 (Geometric Simplicial Complex)**.** Geometric simplicial complex $\mathbb{K}$ is called a collection of sets $\sigma \subset \mathbb{R}^n$ over a finite (or infinite) collection of points $X \subset \mathbb{R}^n$, where:

1. $\exists \overline{X} \subset X$ such that $\sigma = conv\{\overline{X}\}$ and $\overline{X}$ is affine independent.
2. $\forall \tau \subset \sigma, \tau \in \mathbb{K}$
3. $\forall \sigma, \tau \in \mathbb{K}, \sigma \cap \tau \neq \emptyset, \sigma \cap \tau \in \mathbb{K}$

Geometric simplicial complexes defined as in 3.1.3 can be viewed as a *geometric realization* of an *abstract simplicial complex*, indeed any *abstract simplicial complex* can be embedded into $\mathbb{R}^{n-1}$ or equally into $\mathbb{R}^{2d+1}$, where d - dimension of a complex and n - number of points, that the simplices span over. The last realization is possible is due to mapping points into general linear position. A mapping from *abstract simplicial complex* to its corresponding *geometric realization* could formally be defined as:

Definition 3.1.4 (Geometric Realization)**.** Let $\mathbb{A}$ be an abstract simplicial complex and $f : V(\mathbb{A}) \to \mathbb{R}^n$ - injective mapping, such that $f(\mathbb{A})$ - geometric simplicial complex, then f - geometric realization of abstract simplicial complex $\mathbb{A}$. We will also define the resulf of this mapping as $|\mathbb{A}|$

While being built on a fixed set of points, *geometric simplicial complex* might be refined using *barycentric subdivision* of a *geometric simplicial complex*:

Definition 3.1.5 (Barymetric Subdivision)**.** Let $X = \{x_i\}_{i=0}^{n}$ define a simplex $\sigma \in \mathbb{K}$ - geometric simplicial complex, let $bc(\sigma) = \{bc_I | I \subset 1 \dots n, bc_I = \frac{1}{|I|} \sum_{i \in I} x_i\}$ - set of all barycenters of subsets of σ, then $b_\delta(\sigma) = \{\tau | \exists \overline{X} \subset X, \overline{X} \neq \emptyset, \overline{bc} \subset bc(\sigma), \tau = conv\{\overline{X} \cup \overline{bc},\}, card(\overline{X} \cup \overline{bc}) = card(\sigma)\}$, this procedure is performed for all simplices, that are not faces for any other simplex.

Geometric simplicial complexes aim to describe the topology of the space the dataset is sampled from based on the finite dataset. However these approximations might be chosen in different manners, and a natural question arises - how to choose the one, that is guaranteed to capture the topology as the sampling becomes finer and more observations are available. For this purpose we will introduce several results of algebraic topology [14]:

Theorem 1 (Lesbegue number lemma)**.** Let $X \subset \mathbb{R}^n$ - compact space, then for any open cover $U = \{U_i\}$ there exists such number $\lambda > 0$, that $\forall \overline{X} \subset X$, that $diam(\overline{X}) < \lambda$ $\exists U_i \in U$ such that $\overline{X} \subset U_i$

Definition 3.1.6 (Nerve). Let F - collection of sets, then $Nrv(F) = \{G \subset F | \bigcap_{g \in G} g \neq \emptyset, card(G) < \infty\}$

We will also need a concept of *homotopy equivalence*, which plays a crucial role in comparing different topological spaces[15]:

Definition 3.1.7 (Homotopy). Given two continuous functions $f : X \to Y$ and $g : X \to Y$ on topological spaces X and Y, homotopy $H(t, x) : [0, 1] \times X \to X$ such that $\forall x \in X, H(0, x) = f(x), H(1, x) = g(x)$ and $H(t, x)$ is continuous in both variables, f and g are called homotopy equivalent.

Definition 3.1.8 (Retraction). Let $r : X \to A$ - continuous function on topological space X mapping to the set A, then r is called *retraction* of X to A if $r(X) = A, r|A = \mathbb{I}$

Definition 3.1.9 (Inclusion). Let $i : A \to X$ - continuous function mapping set $A \subset X$ into topological space X, then i is called *inclusion* of A into X if $\forall x \in A, i(x) = x$

Definition 3.1.10 (Homotopy equivalence). Two topological spaces X and Y are called homotopy equivalent if there such continuous mappings $f : X \to Y$ and $g : Y \to X$ such that $f \circ g : Y \to Y$ and $g \circ f : X \to X$ are *homotopy equivalent* to id_X and id_Y respectively.

Effectively *homotopy equivalence* defined in 3.1.10 means, that two topological spaces might be stretched or deformed in a continuous way, so that they can match each other. A notion of exact inverses is to restricting and leaves out many cases where no inverses exists.

We call a set contractible if it is *homotopy equivalent* to a point.

Definition 3.1.11 (Good open cover). Let F - cover of a set, such that $\forall F_i \in F, F_i$ is contractible.

Nerves are therefore *abstract simplicial complexes*, which describe how sets of a given family intersect with each other. They serve as a bridge from abstract algebraic topology to computational topology. From now on we consider *geometric realization* of *nerves* to reason about topological equivalence. This is highlighted by the following theorem given in two commonly employed variants[14]:

Theorem 2 (Nerve theorem (for open covers)). Let X - compact space (paracompact), and F - its open cover, such that $X = \bigcup_{F_i \in F} F_i$ such that $\forall G_I = \bigcap_{F_i, i \in I}, G \neq \emptyset, G-$ contractible, then $|Nrv(F)|$ if *homotopy equivalent* to X.

Theorem 3 (Nerve theorem (for compact convex covers)). Let X - compact space (paracompact), and F - its cover, such that $\forall F_i, F_i$ is compact and convex, $X = \bigcup_{F_i \in F}$, then $|Nrv(F)|$ if *homotopy equivalent* to X.

There are many variations of this theorem [14] covering broad spectrum of different requirements, like open covers or closed covers, or requirement of connectedness and *contractibility* and others. In essense nerve theorems stated in 2 and 3 claim, that for some *good covers* of the topological spaces, their *nerves* serve as an approximation to the topology of the underlying topological space.

Consider a compact set and a cover of that, given for example by a subset of points located inside the set and encircling balls around those with a given radius of ϵ. By *Borel-Lesbegue theorem* such cover admits reduction to a finite subcover. *Lesbegue number lemma* (see 1) further on assures that by reducing the radius of the balls constituting the cover, one can achieve any precision in approximating topological properties of the set, thus motivating striving for finer covers. Lastly with *Nerve theorem* for open covers (2) at hand, applying it to the very same open cover would allow to approximate the underlying space topology. In a similar fashion we could consider a paracompact space and use other base sets, for example *Voronoi cells*, in this case 3 would be used.

Based on the definition, *nerves* depend on the cover chosen for a given set. *Geometric simplicial complex*, which might be given as a *geometric realization* of an *abstract simplicial complex*, corresponding to a *nerve* of the cover is called a *nerve complex*. The most commonly used type of such complexes is called *Čech complex*. [16]

Definition 3.1.12 (Čech complex)**.** Let $X \in \mathbb{R}^n$ - a finite set of points, define closed balls $B_\epsilon(x)$ centered at point x with radius ϵ, then Čech complex is defined as $C_\epsilon(X) = \{\hat{X} \in X | \bigcap_{x \in \hat{X}} B_\epsilon(x) \neq \emptyset\}$

Definition 3.1.12 of *Čech complex* implies, that it is isomorphic to a nerve of the closed balls as defined higher and hence Čech complex is a nerve complex. While enjoying good theoretical properties backed by the nerve theorems like (2,3), computational side of *Čech complex* suffers when scaling dimensions or number of points. [17] This is closely related to the problem of finding closest neighbors, and checking if balls of a given radius intersect. A necessary and sufficient condition for a subset $\hat{X} \in X$ to be in the complex is that a smallest enclosing sphere of the points has radius smaller than parameter ϵ.

For the reason stated above computation of *Čech complex* might be restricting and since TDA considers the same data at different scales to produce insights about underlying topology, one needs to recompute complex for different values of parameter ϵ. For this purpose we will introduce another type of complexes, different from nerve complexes:

Definition 3.1.13 (Flag complex)**.** Let $\mathbb{A}$ - an abstract simplicial complex, such that $\forall S \in Vert(\mathbb{A})$ if $\forall \hat{S} \subseteq S, card(\hat{S}) = 2, \hat{S} \in X \rightarrow S \in S$.

This means that *flag complexes* are completely defined by simplexes of cardinality two, which is a more general definition of a commonly used term of a *clique complex*:

Definition 3.1.14 (Clique complex)**.** Let $\mathbb{A}$ - an abstract simplicial complex, such that $sk_1(\mathbb{A})$ is a clique of some undirected graph.

An important example of a clique complex is given by V*ietoris-Rips complex* [18]:

Definition 3.1.15 (Vietoris-Rips complex)**.** Let (X, ρ) - a finite metric space, then $Vr_t(X) = \{\hat{X} \subseteq X | diam(\hat{X}) \leq t\}$

Vietoris-Rips complex is a flag complex as defined in 3.1.13, as inclusion of simplex into an *abstract simplicial complex* is uniquely defined by pairwise distances. Hence computation of this type of complex boils down to calculation of a distance matrix over given set of points and inclusion/exclusion of a simplex based on thresholding. While not being a nerve complex *Vietoris-Rips complex* is isomorphic to *nerve*, although not being constructed from covers usually. [14]

Until now we have fixed a parameter defining scale of a chosen *geometric simplicial complex*, however as was mentioned earlier, TDA analyzes a given point cloud at different scales, which requires computation of complexes for a varying range of parameters. For this purpose we need to introduce a concept of *filtration*, which will play a pivotal role in the applications[14]:

Definition 3.1.16 (Filtration)**.** A parametric family of geometric simplicial complexes K_t forms a filtration if $t \leq s \to K_t \subseteq K_s$

Both *Vietoris-Rips* and *Čech complexes* form *filtrations*.

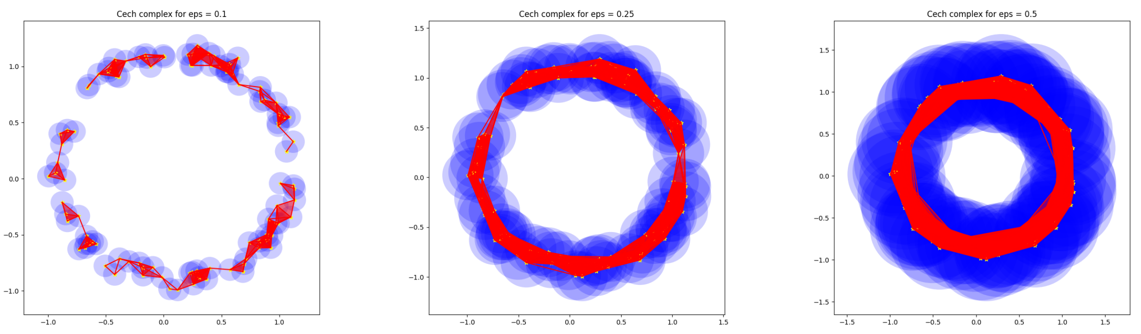
In 3.1 we can see an example of the filtration, designed to emphasize, how multi-scale nature of *Čech complex* can capture topology of the underlying space.

While not enjoying theoretically good properties based on the *Nerve theorem* (see 2), like for example *Čech complexes*, *Vietoris-Rips* complexes nevertheless play an important role by virtue of serving as a proxy between *Čech complexes*, thus helping to decrease computational overhead. Upon closer examination one can deduce that [17]:

$$\forall X \in C_\epsilon(X) \to X \in Vr_{2\epsilon}(X)$$

In order to make use of *Vietoris-Rips* complexes to approximate *Čech complexes*, one would need another side inclusion, achieved via the following result due to Jung:

Theorem 4 (Jung(1901))**.** Let $Q \in \mathbb{R}^n$ - a bounded set with $diam(Q) \leq t$ then Q is contained in a closed ball with radius $r \leq \theta t$, where $\theta = \sqrt{\frac{d}{2(d+1)}}$



(a) $\epsilon = 0.1$ (b) $\epsilon = 0.25$ (c) $\epsilon = 0.5$

Figure 3.1: **Čech complexes built on samples from a circle for epsilon value of** $\epsilon \in \{0.1, 0.25, 0.5\}$. Figures a-c show how topology of the underlying space is evolving. Starting from 0.25 circular shape is recognized by filtration.

This makes inclusion of the *Vietoris-Rips* complex into a higher radius *Čech complex* possible and thus helps to sandwich one filtration by another by utilizing theorem 4.

$$\hat{X} \in Vr_{eps} \to X \in C_{\theta\epsilon}(X)$$

For this reason *Vietoris-Rips filtrations* are usually applied in practice in order to approximate *Čech filtrations* and preserve later good properties allowing one to infer about the topology of the underlying topological space. [17]

3.1.2 Homologies

One of the most important question discussed in TDA is deciding if a pair of two topological space is *homotopic*. This is however a complex problem and might not be solved generaly. Homologies represent a way to address this issue. To introduce this concept we will need to define a notion of *homology*.[15]

Definition 3.1.17 (Chain)**.** Let $\mathbb{A}$ - a finite a*bstract simplicial complex*, then for a given d, chain is defined as $c = \sum_{\sigma \in \mathbb{A}, card(\sigma)=d} \lambda_\sigma \sigma$, where coefficients $\forall \sigma \in \mathbb{A}, \lambda_\sigma \in \mathbb{G}$, $\mathbb{G}$ - Abelian group (usually chosen as $\mathbb{Z}_2$)

Most common of the group $\mathbb{Z}_2$ corresponds to the choice of inclusion/exclusion of simplices, so that simplices which come across two times in the *chain* expansion cancel out. While other common choices feature groups $\mathbb{Z}_p$ for prime numbers p or other Abelian groups, universal coefficient theorem implies that integer coefficient based homology completely determine homology for different choice of coefficients.[15] Additivity of each component of sums in *chain* definition suggests that they themselves form a group structure, which will be called $\mathbb{C}_d(\mathbb{A})$ To draw a connection between chains of a different order (dimension of simplices making up a sum), we will need to define a boundary operator in the following way:

Definition 3.1.18 (Boundary operator)**.** Let $\delta_d : C_d(\mathbb{A}) \to C_{d-1}(\mathbb{A}) : \forall \sigma \in \mathbb{A}, card(\sigma) = d, B_d(\sigma) = \sum_{\tau \subset \sigma, card(\tau)=d-1} \tau$, boundary operator is then extended by linearity for *chains* to $C_d(\mathbb{A})$ and is a homomorphism.

Definition 3.1.19 (Cycle)**.** Let $c \in C_d : \delta_d(c) = 0$, then c - d cycle, or formally $Z_d = (\delta_d)$

Boundary operator maps simplices in a *chain* to their highest order faces and in this way creates a link between *chains* of different dimension. Consider a special type of *chains*, that represent a boundary of another one degree of order higher simplex:

$$Let \sigma = \{a,b,c\}, asimplex, then \delta_3(\sigma) = \{a,b\} + \{b,c\} + \{a,c\} \tag{3.1}$$

Let us now apply *the boundary operato*r of an order lower to observe the following important structure of the *chains* of different order and the connection between them:

$$\delta_2(\delta_3(\sigma)) = \{a\} + \{b\} + \{b\} + \{c\} + \{a\} + \{c\} = 0 \tag{3.2}$$

This means that $\mathbb{I}(\delta_3) \subset \mathbb{K}(\delta_2)$ or in general $\mathbb{I}(\delta_d) \subset \mathbb{K}(\delta_{d-1})$. We define *boundaries* as a space B_d, such that $\exists c \in C_{d+1} : \delta_d(c) = b$ or formally as $\mathbb{I}(B_d)$. the previous statements are now equivalent to $B_d \subset Z_d$. We say that two *cycles* $c_1, c_2 \in Z_d$ define the same hole if they $\exists b \in B_d : c_1 = c_2 + b$. Such two *cycles* are called homologous and define an equivalence relation on the space *cycles*, *homologies* are then defined as:

Definition 3.1.20. Let $H_d = Z_d$
B_d - a quotient space of *cycles* over *boundaries*, then H_d - d order *homology* and $dim(H_d)$ - d-th *Betti number*.

In this way *homologies* now describe topological information like connectivity (H_0) or number of holes of different order H_d for hole of dimension d.

3.1.3 Persistent Homology and persistent diagrams

We now generalize the notion of homology to *filtrations* of *simplicial complexes* in order to capture multiscale information of the discrete dataset. Consider a *filtration*, then when computing homologies for different values of the parameters we observe that some spaces are born and some are dying at a given value of parameter. This information might be understood in the way, that some topological features, like holes, voids or connected components are appearing and disappearing. Formally define a *filtration* $\{K\}_t$, and an inclusion map on those, which would then map by functor to $i_t : C(K_{t-1}) \to C(K_t)$ as an inclusion of *chain* spaces and which afterwards would map to an i*nclusion of homologies*. This sequence of *homologies* for different values of parameter t denoted as $H(K_t)$ forms a chain of *homologies*. We can now form an object called *persistent diagram* - a multiset of pairs of birth-death times accounting for each homology class. [3] This object will be in the core of the result of this master thesis.

3.2 Topological Machine Learning

To this end we have introduced all the theoretical foundations required to setup the framework of the topological machine learning. We start by determining its role and place in the classical machine learning and then describing how the concepts outlined in the previous subsections might be rendered as learning tasks.

In the most common scenario, machine learning problems can be seen as the task of extracting insights from data, either with the help of labels (supervised learning) or without them (unsupervised learning). While the first paradigm has been dominating for a long period of time, researchers have been constantly subjected to strain originating from necessity to provide labeled information to their data and rendering their problem as a task-specific one. Unsupervised learning on the contrary seeks to discover information hidden in the data itself without binding to any problem to be solved. This information might have a different nature, but in general it refers to the structure of the dataset.

Some common approaches of classic unsupervised learning include dimensionality reduction, which relies on the fundamental manifold hypothesis, stating that high dimensional datasets usually reside on the manifolds and have lower intrinsic dimensionality. Prominent examples, like *PCA* would assume, that this manifold is linear, while others like *MDS* or *UMAP* would try to pay attention to local structure of the data and may reconstruct non-linear structures. [19, 20]

In the realm of deep learning, *autoencoders* have been widely adopted to produce meaningful embeddings of data. Proper informative latent space might be achieved by embedding model with a suitable regularization or loss. For example *variational autoencoders* are not only capable of generating new unseen samples, but also provide more continuous latent space, than a simple *autoencoder* would do. Such latent spaces might be useful for compressing data and preserving only meaningful features for downstream tasks. This idea serves as a foundation for a more unified view on the learning problems, encompassing both supervised and unsupervised learning called representation learning. In general any output of the hidden layer might be viewed as a representation, and the task of the representation learning is to make model produce features, that might be used for similar tasks. This is where topological machine learning finds its place.

3.2.1 Learning on Persistent diagrams

An intrinsic assumption, any deep learning algorithm is making is a continuity of the learned function. Learning on *persistent diagrams* is therefore motivated by the following result:

Definition 3.2.1 (Function sublevel filtration)**.** Let $f : X \to \mathbb{R}$ - continuous function, and let $X_r = f^{-1}((-\infty, r])$ - r-sublevel of f, then $r \leq s \to X_r \subseteq X_s$, such filtration of topological spaces is called function sublevel filtration. Inclusion map $h_p^{r,s} : H_p(X_r) \to H_p(X_s)$ is a homomorphism, spanned by inclusion $img(h_p^{r,s}) = H_p^{r,s}$ - persisent homology group of order p.

Definition 3.2.2 (Homological Critical Point)**.** Let f be a function, defined as previous, then a - homological critical point, if $\exists \epsilon > 0 : h_p^{a-\epsilon,a+\epsilon} : H_p(X_{a-\epsilon}) \to H_p(X_{a+\epsilon})$ is not an isomorphism.

Definition 3.2.2 of the *homological critical point* means formally, that some *homologies* die at the point of a. For our next result we will also need to introduce an important family of functions:

Definition 3.2.3 (Tame functions)**.** Let f be a function, defined as previous, then f - is a tame function, if it has only finite number of homological critical points and homology groups and each sublevel have a finite dimension.

We will now introduce an important metric between functions, which finds its application in many other domains and lets one quantify the distance between persistent diagrams.

Definition 3.2.4 (Bottleneck distance)**.** Let $f, g : X \to \mathbb{R}$ - tame functions on the same topological dpace X, and define D_p as a mapping, which maps function to its persistence diagram of dimension p, then bottleneck distance between $D_p(f)$ and $D_p(g)$ is defined as $d_B(D_p(f), D_p(g)) = inf_\gamma(sup_{x \in D(f), y=\gamma(x)} ||x - y||_\infty)$, where $\gamma : D_p(f) \to D_p(g)$, a bijection.

Bottleneck distance ,also known as *Wasserstein distance*, is commonly used in other domains, like *Wasserstein GANs* for example. Since computation of this metric involves optimization it might sometimes be costly to calculate it, especially for large batches. Now we are ready to state the main result, which motivates learning on persistent diagrams:

Theorem 5 (Stability Theorem)**.** [21] Let f, g - tame functions on a triangulable space X, then $\forall p > 0 : d_B(D_p(f), D_p(g)) \leq ||f - g||_\infty$

This theorem might be seen as the way to explain why small perturbations in the functions, or, as might equally be considered, datasets, will not influence the distance between *persistence diagrams* drastically.[21]

While working directly with *persistent diagrams* does guarantee theoretically good properties allowing, it might not always be easy to calculate the bottleneck distance between those. In practice its value is itself a result of an optimization routine. That is why other representations of the persistence diagrams are of importance. At this point we are ready to introduce a concept of *Betti Curve*, which will be used actively throughout the work:

Definition 3.2.5 (Betti Curve)**.** Let $D_p(f)$ be a persistence diagram of a function f, an consider the following mapping, $B_{D_p(f)}(x) = \sum_{i=0}^{n} \mathbb{I}(a_i \leq x \leq b_i)$, where $\{(a_i, b_i)\}_{i=0}^{n}$ - points in persistence diagram $D_p(f)$.

Betti Curve is an integer valued function, which counts the number of active intervals in the persistence diagram, effectively measuring a number of topological features a given dimension active at a given scale. They are simple to calculate and form a linear space, which might be endowed with metric to quantify distance between those. The downside of the *Betti Curves* on the other hand is that there is no guarantee as *stability theorem provides*, nonetheless their application is still of interest. We will consider the following kernel to quantify the distance between two persistence diagrams based on their Betti Curve representation [22]:

$$k(D_p(f), D_p(g)) = -(\int_0^\infty |B_{D_p(f)}(x) - B_{D_p(g)}(x)|^p dx)^{\frac{1}{p}} \tag{3.3}$$

Apart from *Betti Curves kernel* and *Bottleneck distance* one might want to directly work with pairings encoded in the persistent diagram. Anytime a homology of a given dimension dies, a birth of new simplex, which causes the homology to die happens. For *Vietoris-Rips filtration*, which is based uniquely on distance matrix between samples one can find simplices, addition of which causes death of homology. For example, death of a connected component (homology of dimension 0) would be caused by adding an edge between two, while death of hole would be caused by forming a face covering this hole. The same logic is valid for creation of the topological features, In this manner there is a pairing of features, accounting for birth and death of the *homologies*.[23]

We will now define second important loss function, that will be used in this master thesis, namely *pairing loss*:

Definition 3.2.6 (Pairing Loss)**.** [12] Let A_X - a distance matrix of points in the X, Z is a latent representation of the dataset with a distance matrix A_Z, then pairing loss L_t is defined as a sum of two terms $L_{X \to Z} = \frac{1}{2}||A_X(\pi_X) - A_Z(\pi_X)||^2$ and $L_{Z \to X} = \frac{1}{2}||A_X(\pi_Z) - A_Z(\pi_Z)||^2$, where π_X, π_Z - triplets (accounting for face for dimension 1 of homology) or pairs (accounting for edge for dimension 0 of homology).

The primary goal of the *pairing loss* is to unify topological features of the original space and the latent representation, which has proven to be a good regularization term accompanied with reconstruction loss. The main effect is achieved by aligning distances between the points in the original space and the latent one. Since only topologically relevant distances are selected, it is expected, that topology of the latent space will be preserved under the transformation (for example, *autoencoder*).[12] It is also possible to show, that in the context of batch learning batchwise *persistent diagrams* are close to the *persisent diagram* of the original dataset. The last metric to be considered is called *total persistence* and is defined as follows:

Definition 3.2.7 (Total Persistence)**.** Let $D_p(f)$ - persistence diagram of an order p of a function f, then $Pers(D_p(f)) = \sum_{\tau \in D_p(f)} |\tau_1 - \tau_2|^p$, where τ - point in $D_p(f)$

Here *total persistence* is constituted by values defining scale at which topological feature is alive and thus by minimizing this metric one aims to decrease overall topological activity.[4] This metric might be exploited as a regularization term to penalize model for predicting noisy data, where high number of points with low living time.

3.3 Neural Radiance Fields

In the realm of 3D Machine Learning multiple representations of the scene might be utilized. Each of those enjoy their own advantages and disadvantages when applied in different settings. Voxel grid for instance operates on the basis of the 3D uniform grid and hence allows for application of methods common in the area of 2D computer vision like convolutions or vision transformers. On the other hand restrictions imposed by the computational and space complexity of either data and models handling those make space for other data structures.

Neural Radiance Fields models also known as *NeRFs* was a leap forward in 3D object reconstruction with a limited number of views from different angles at hand. In the crux, *NeRFs* operate on the basis of 2 components - rays and their directions and try to extract information from those comprised by intensity of the color at the given point and color represented as an RGB vector. Directions of the rays might be seen as an angle, those are shoot from and together with coordinates on the rays, model operates on the basis of 5D input. Unlike previous approaches to reconstruction of 3D geometry of the object, which were predominantly based on direct voxel in-painting, *NeRFs* are optimized to predict values in continuous space.[7]

3.3.1 Architecture

The architecture of the original *NeRF* is fairly simple, as it is built solely on application of dense layers and skip-connections for some of those. All the layers except for skip connections and input share the same number of input/output channels, specifically 256 and the same activation of *ReLU*. An important aspect of the *NeRF* design lies in the positional encoding, which reminds that of transformer, but in this case directly changes the input of the layer. This transformation might be given by the following formula and according to [7] plays a crucial role in enabling model to learn high frequency terms:

$$pos(x) = x \oplus \bigoplus_{i=0}^{n} (sin(2^i \pi x), cos(2^i \pi x)) \tag{3.4}$$

Here x represents three coordinates and transformations are applied coordinate-wise with subsequent concatenation. The reason for that is that neural networks are naturally biased towards low frequencies and in context of *NeRFs* produce oversmoothed image. Attaching information with Fourier features to the original coordinates maps them to a higher dimensional space, where the network can more easily handle higher frequencies present in data.[7]

Another important aspect of how *NeRFs* are defined is the decoupling of directions and positions along the ray. This is performed by restricting model to predict density values based only on positions to ensure consistency independent of the view, whereas color values are predicted based on both position and direction information. This is performed by means of providing only position information as an input to predict density at the point while additionally supplying the model with direction information when finally producing color vector. The architecture is summarized in the Figure 3.2.

3.3.2 Optimization

NeRFs are optimized directly to predict color of the resulting 2D view from a given view direction, to produce colored image from densities and colors along the rays, volumetric rendering by means of ray tracing [7] , given by the following formulas:

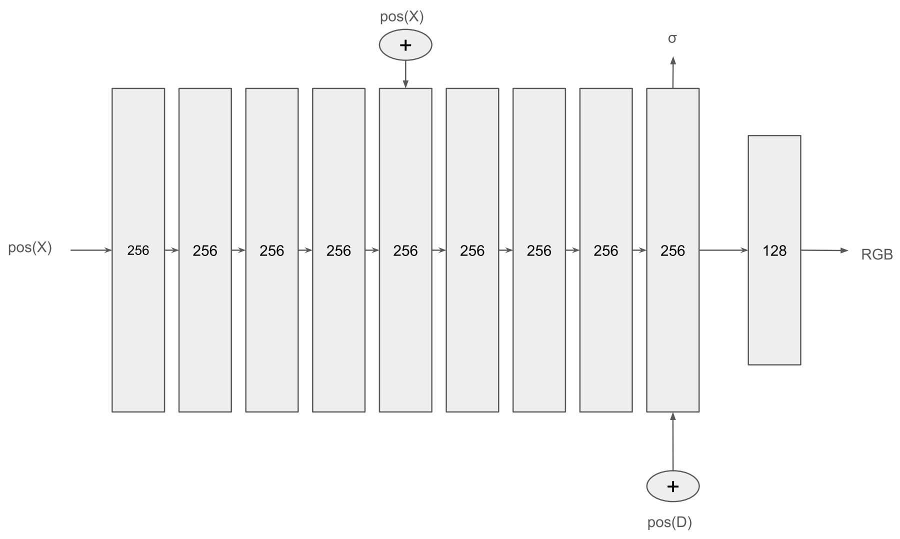


Figure 3.2: **Architecture of original NeRF model.** Here X - input rays positions, D - directions, pos(X), pos(D) - positional encodings of X and D respectively, σ - predicted intensity of the rays, RGB - predicted vector of colors

$$C(r) = \int_{t_{i-1}}^{t_i} T(t)\sigma(r(t))c(r(t), d)dt, T(t) = exp(-\int_{t_{i-1}}^{t} \sigma(r(\tau))d\tau), r(t) = o + d * t \tag{3.5}$$

Here o - origin of the rays, d - direction, t - step. $T(t)$ - represents an accumulated transmittance between two steps along the ray, which essentially gives a probability, that ray is stopped by an obstacle. $C(r)$ - represents a resulting color. During training values of t are sampled from bins of the following design:

$$t\tilde{}\mathbb{U}([t_n + \frac{i-1}{N}(t_{n+1} - t_n), t_n + \frac{i}{N}(t_{n+1} - t_n)]) \tag{3.6}$$

Here $\{t\}_{i=1}^{N}$ - form an initial evenly-placed partition and the actual positions t are sampled from the intervals to ensure continuity. Now, using quadratures integrals from formula 3.5 might be approximated by the following sums:

$$C(r) = \sum_{i=1}^{N} T_i(1 - exp(\sigma_i \delta_i))c_i, T_i = exp(-\sum_{j=0}^{i-1}(\sigma(r(t_i))\delta_i)), \delta_i = (t_{i+1} - t_i) \tag{3.7}$$

Here substitution of $\sigma_i \delta_i$ directly by $(1 - exp(\delta_i \sigma_i)$ is performed to reduce the problem to alpha-compositing by equivalence for small value of δ_i.

3.3.3 Hierarchical sampling

During training *NeRF* doesn't differentiate between regions of low density and high density, which may lead to paying attention to empty domains, contributing poorly to representation capacity. Another issue might be posed by occluded regions. For this purpose contributors of the original model came up with a concept of hierarchical sampling, which relies on using two separate *NeRF* models - coarse and fine. First raw positions are fed through the coarse model to produce information on the density in vicinity of the points. Then the initial rays are resampled based on the following weights:

$$w_i = T_i(1 - exp(\sigma_i \delta_i)), \hat{w}_i = \frac{w_i}{\sum_{i=0}^{N} w_i} \tag{3.8}$$

Weights $\hat{w}_i$ are used to produce importance sampling along the rays and concatenate raw samples with resampled positions. This new locations are used as an input for the model sharing the same architecture titled as a fine model. The overall loss term is then comprised of two summands:

$$L(\hat{c}, c) = MSE(\hat{c_f}_{fine}, c) + MSE(\hat{c}_{coarse}, c) \tag{3.9}$$

Ablation studies have shown, that both positional encoding and hierarchical sampling make a significant improvement over the original model architecture.

3.3.4 Quantifying NeRF performance

Results, that are produced by *NeRF* model was originally beanchmarked with respect to were *Peak Signal Noise Ratio* (*PSNR*) and *Structural Similarity Index Measure* (*SSIM*), which are also actively employed for other types of reconstruction tasks. In this section we will briefly review, how this metrics are formulated, intuition behind them and their application.

Definition 3.3.1 (Peak Signal Noise Ratio (PSNR)). $PSNR(I, J) = 10 * log_{10}(\frac{MAX(I^2)}{MSE(I,J)})$, where I, J - input image and noise image respectively.

This metric is originally derived from the field of reconstruction of original image from compressed image and effectively depicts, how large is the scale of original image with respect to the mean of corrupting signal.

Definition 3.3.2 (Structural Similarity Index Measure). $SSIM(x, y) = \frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(-\sigma_x^2+\sigma_y^2+c_2)}$, where μ_x, μ_y - empirical means of the pixel values of two images, σ_x, σ_y - empirical variances and σ_{xy} - empirical covariance

3.4 Normalizing Flows

An important part of this work revolves around the concept of *Normalizing Flows (NF)* is a probabilistic model used to approximate complex posterior distributions arising during variational inference as well as to perform sampling from distributions and likelihood estimation, where no explicit density is available. The basic concept of a normalizing flow relies on the density transformation formula, which governs how density of the distribution is changed upon application of a transformation. In general this formula might be extended to any piecewise differentiable transformation, although in context of the current approach, only bijective and differentiable transformations are of concern.[24] This restriction allows to track exactly how the point sampled from a target distribution is being transformed.

Definition 3.4.1 (density transformation formula). Let $Y = g(X)$, where g - monotone differentiable function, and let f - pdf of the random variable X, then density of the random variable Y is given by $f_Y(y) = f_X(g^{-1}(y))|\frac{d}{dy}(g^{-1}(y))|$

When multiple mappings are used, they form a flow, whereas term normalizing refers to aa base distribution, usually chosen to represent a latent variable with a structure tailored to a specific task. Most often, this distribution is chosen to be a multivariate normal, however other options, which could describe more complicated topology of the target distribution might be used. For example, to model distribution on the surface, having one void and two loops, *Von-Mises* distribution might be chosen as a base, since it models random variable located exactly on the torus, sharing the same topological characteristics as a target distribution.

Not all *normalizing flows* are explicitly invertible and there might not exists an expression to obtain inverse under a given transformation. In such cases, one has to sacrifice ability to sample from such probabilistic models and potentially invert a flow by changing a direction, it flows from. This will allow to estimate likelihood of observed data and eventually conduct training. By chaining multiple transformations constituting a flow resulting formula of the log likelihood has the following look:

$$log(p_Y(D|)) \tag{3.10}$$

When designing a *normalizing flow* architecture, several things are of concern, specifically computational efficiency and theoretical properties, like bijectivity and differentiability.[24] For this reason, arbitrary transformations, which, for example do not allow exact computation of a Jacobian. Moreover Jacobians of full matrices

3.4.1 Planar and Radial Flows

Originally *normalizing flows* were designed with no binding to neural networks in the context of variational inference and were employed in later on in the architecture of *variational autoencoders (VAE)*.[25] Originally two types of flows, namely planar and radial were introduced, which, despite their simplicity could not be inverted easily.[26] These flows laid foundation to other *normalizing flows*:

Definition 3.4.2 (Planar Flow). $g(x) = x + uh(w^T x + b)$, where $h : \mathbb{R} \to \mathbb{R}$ - smooth non-linear function. Here u, w, b - learnable parameters.

This flow stretches and contracts points along directions defined by w and is equivalent to a single layer perceptron with a skip connection. Not any choice of h is permissible in this setting, and inverse of this flow is not usually available in closed form, however the determinant of this flow might be computed effectively in $O(D)$.[26]

Definition 3.4.3 (Radial Flow). $g(x) = x + \frac{\beta}{\alpha + ||x - x_0||}(x - x0)$, where $\beta, \alpha > 0, x_0 \in \mathbb{R}^n$ - learnable parameters.

This flow sends point into a given direction defined by the vector x_0 and the distance it is send to is defined by parameters β, α. This flow also allows effective computation of the determinant and its inverse also may not always be computed explicitly.[26]

3.4.2 RealNVP

For this work we consider a popular type of *normalizing flows* with a name *RealNVP* (*Real Non-Volume Preserving*).[27] This type of a flow is primarily used for density estimation and enjoy a wide spectrum of advantages like tractable likelihood, stability and computational effectiveness. The transformation is given by the following formula:

$$\begin{cases} y_{1:k} = x_{1:k} \\ y_{k+1:d} = x_{k+1:d} \odot exp(s(x_{1:k})) + t(x_{1:k}) \end{cases} \tag{3.11}$$

Here s, t - are scale and translation functions, which might be given by any neural networks with trainable parameters. It is also common to choose this functions to map original input into a higher dimensional space. Regarding the structure of the Jacobian of this transformation, it has lower diagonal design of the following form:

$$\begin{pmatrix} \mathbb{I}_{kxk} & 0 \\ \frac{dy_{k+1:d}}{d_{x_{1:k}}} & exp(s(x_{1:k})) \end{pmatrix} \tag{3.12}$$

This reduces complexity of computation from $O(d^3)$ to $O(d)$, by virtue of this enabling scaling RealNVP to high dimensional datasets. This flow is moreover explicitly invertible leading to applications not only in estimating the density of the probability in a given region, but also enabling sampling from the resulting probabilistic model. Though coupling layers might seem to lack expressiveness as they only change only one portion of the input, leaving other untouched, they proved to be powerful when stacked one after another. To alleviate a problem authors decided to implement scaling and translation networks as residual connection networks and equipped them with batch normalization, which is applied at the level of coupling rather than inside the neural networks. In this manner normalization becomes yet another flow in the chain and showed to accelerate training for deep flows.

3.5 Point cloud Machine Learning

In this section we will briefly discuss, how machine learning on point cloud effectively differs from classic machine learning. In general point clouds have no meaning of ordering and therefore a natural invariance to be induced into network is permutation invariance. In the field of point cloud machine learning one of the most prominent architectures is *PointNet* model.[28]

In the setting of point cloud machine learning input data is given as an unordered set of points. Except for permutation invariance, other restrictions like invariance with respect to global rotation of the dataset or translation might be imposed. Handling such problems might be hard in context of classic machine learning ang there have been several attempts to approach this data representation. Point sets are also processed poorly by *MLPs*, since the later fail to capture local dependencies and combinatorial structures important for learning on point clouds.

3.5.1 PointNet

PointNet model was one of the first models, that shifted the focus of point cloud learning from convolutions to special architectures tailored to tackling point sets.[28] Before most of the researchers would render point clouds as 2D or 3D image and process them with standard convolutional networks. This however incurs high computational overhead for 3D datasets, where number of voxels needed to represent a dataset is a restricting factor.

The idea of *PointNet* architecture is simple yet expressive. In order to induce required symmetries, contributors designed their network to process input pointwise with subsequent symmetric aggregation at the end. They reformulate the problem in the following way:

$$f(\{x_1, \dots x_n\}) \approx g(h(x_1), \dots h(x_n)) \tag{3.13}$$

Here, $g : \mathbb{R}^{nK} \to \mathbb{R}$ - is a symmetric function, $h : \mathbb{R}^n \to \mathbb{R}^K$ and $f : 2^{\mathbb{R}^N} \to \mathbb{R}$. To allow network to align points together, authors came up with a *TNet* - a network, which architecture resembles the larger network's and which acts on point features attempting to align them. Result of *TNet* is a linear transformation which is then applied to points. Resulting linear transformation is regularized to be close to the orthogonal.

4 Related Work

In this chapter, some of the work done in the field to approach the problem of 3D reconstruction from a finite number of views using topological supervision is reviewed. Papers like *Topological autoencoders* [12] and [4] as well as recent advances in probabilistic generative modelling are also discussed. [27].

4.1 Neural Radiance Fields

As was mentioned in section 3.3, *NeRF* model is a novel architecture, which main purpose is to model radiance fields in order to reconstruct views from previously unseen angles.[7] Although the authors introduced several original ideas — most notably Fourier features, which enhance the convergence of fine details—they did not explicitly leverage the topological information embedded in the data.

Further improvements in the face of paper [29] attempted to make use of spatial conical frustums instead of one dimensional rays, which reduced aliasing effects of the 3D representation, when views were sampled from more distant viewpoints. In this way, authors exploited 3D structure of the space and made a step towards training *NeRF* with more detailed attention to spatial relationship.

Recent work by [30] pioneered usage of surface levels in context of neural radiance fields, where authors studied deformations of views produced by the original model. These studies suggest growing interest of community towards topological approaches in the field of implicit 3D machine learning. Despite this trend, novel topological methods employed in works like [12] or [4, 6], which proved to be essential in 3D reconstruction and representation learning, have been remaining out of scope so far. This might be explained by lack of methodology, needed to approach *neural radiance field*s, based uniquely on surfaces produced by neural networks. This motivated us to experiment with different tools from the realm of TDA in tackling this challenge.

4.2 Topological Autoencoders

As was mentioned in the section 3.2.1, equipping *autoencoders* with a suitable regularization might contribute to structured representations arising in the latent space during training. In paper [12] authors proposed to endow classic *autoencoder* with *pairing loss* (see 3.2.6). Topological supervision of deep learning algorithms is itself a challenging task due to discrete nature of the computations involved. [12]. *Pairing loss* however helps to circumvent this issue reducing computations to distance matrices and allowing for direct flow of the gradients through the network. Discrete part of the topology is now completely covered by persistence pairings, which represent topologically relevant structures (edges, faces, e.g) leading to birth/death of topological features, while distances are now used to directly supervise training process.

In their research authors found that using only calculations for dimension zero it was enough to capture topology of the original space, while higher dimensional persistence diagrams only slowed down convergence. [12] It was also described why the resulting *pairing loss* function is differentiable and how to compute its gradient, thus making it possible to use it in an end-to-end training fashion.

Based on this work it can be concluded that topological supervision in the latent space might be a reasonable strategy to address our problem with. For this purpose it was decided to experiment extensively with different topological regularization terms and loss functions applied to intermediate representations of our models.

4.3 SHAPR

SHAPR stands for SHApe PRediction autoencoder [4] was proposed to tackle a problem of 3D reconstruction for cellular data. While this model was initally designed with no application of topological machine learning, it was later found to be beneficial to include regularization term penalizing deviation of topology of predictions from that of ground truth.[4]

Authors contemplated that both segmentation and reconstruction tasks rely on the likelihood approaches, which cast the problem as predicting occupancy of a given pixel/voxel. As 3D reconstruction is extremelly hard inverse problem, common choice of loss function like *BCE* or *Dice* might not be the best choice, since they do not capture any structural information. For this reason contributors came up with a *total persistence loss* term (see 3.2.7) and *Wasserstein loss* term (see 3.2.4), that was used to regularize the training process. Combination of geometry-based and topology-based losses showed to improve reconstruction over multiple metrics.[4]

To benchmark the results, authors compared model with and without topological supervision with respect to *intersection over union* (*IoU*), *relative volume error* and *relative surface error*, which showed, that topological supervision increased overall performance of the model in terms of all of the metrics.

Together with Example 4.2, SHAPR showcases the importance of topological methods in the field of 3D reconstruction, further motivating the choice of tools for the problem addressed in this thesis. While *topological autoencoders* suggest that topological regularizers applied in the context of simple models help structure the latent space, the *SHARR* example demonstrates that this improvement goes beyond representation learning and may find applications in downstream tasks, such as reconstruction or segmentation.

4.4 RealNVP

RealNVP is a *normalizing flows* architecture, which was the first to introduce affine coupling layers. [27] Along with this authors explained how batch normalization and skip connections might be integrated into flow models allowing for deeper architectures to be trained. This was one of the first deep learning papers suggesting application of neural networks for *normalizing flows*, which previously relied on composition of planar and radial flows (see 3.4.1) [26] and were used in context of *variational autoencoders*.[26] As was stated in *RealNVP* paper, model can be used also in settings, where priors are not chosen to be multivariate distribution, which might help to capture topology of the target distribution more carefully.[27] While modelling distributions on complex surfaces or spatial structures might be complicated, we hypothesize, that *normalizing flows* models can benefit from intrinsic knowledge of topology in the underlying dataset, which was addressed in paper [24]. Based on these proceedings it was decided to embed topological supervision into *normalizing flows* in order to estimate its effect on training.

5 Methods

This chapter discusses our methodology, that the further research is built upon. It starts by explaining how different loss functions, models, and architectures are utilized, as well as our motivation behind them. Apart from this our choice of software is presented.

5.1 Tools

In this section, metrics implemented from scratch are outlined, which were not included in any known PyTorch packages. These metrics were designed in an end-to-end fashion, enabling their use during training and allowing gradients to flow through them directly.

5.1.1 Betti Curve Transformation

To the best of our knowledge, there is no publicly available implementation of the *Betti Curve transformation* (see 3.2.5) or the *Betti Curve loss* (see 3.3). Although several libraries, such as GiottoTDA [31], provide implementations, these cannot be directly integrated into an end-to-end deep learning pipeline. In this work, the code was adapted to be fully differentiable and readily usable as a layer in topological machine learning. Additionally, the *Betti Curve alignment algorithm* (see Algorithm 2), enabling the comparison of two curves, and a vectorized *Union-Find* data structure (see Algorithm 4) were developed. Another contribution to the field of topological deep learning included the vectorized computation of persistence diagrams from 1D functions (see Algorithm 3). These implementations can be utilized beyond the scope of this master's thesis and may assist researchers in the field of *TDA*.

Original implementation of the *Betti Curve transformation* relies on hash tables, arrays and *Union-Find* data structures. However, these structures are unsuitable for vectorized computations in most deep learning frameworks. Given that number of points in the diagrams is the same, curves are represented as an array of shape $n \times k \times 2$, where k - number of points in the curve. The first coordinate describes number of topologically active features, whereas the second accounts for filtration values, where those change. To build such curves values from persistence diagrams, those are first sorted by the first coordinate. Afterwards a series of masking operations describing how many intervals are active at the point of filtration are accumulated. The resulting array constitutes *Betti Curve*. Apart from this *symbolic perturbation* is applied beforehand to ensure that multiplicity of tuples constituting a diagram does not hinder processing. This algorithm is formally written as Algorithm 1

Algorithm 1 Vectorized Differentiable Betti Curve transformation

```
Require: $persistenceDiagram$ return $y = B$
  $persistenceDiagram \leftarrow symbolicPerturbation(D)$
  $n \leftarrow size(D)[0]$
  $persistenceDiagram \leftarrow sortByFirstColumn(D)$
  $intervals \leftarrow flatten(persistenceDiagram)$
  $B \leftarrow zeros(len(unique(persistenceDiagram)))$
  for $i \in 1, \ldots, k$ do
      $idxFrom \leftarrow findIndexStart(intervals, i)$
      $idxTo \leftarrow findIndexStop(intervals, idxFrom)$
      $X \leftarrow X \times X$
      $N \leftarrow \frac{N}{2}$
      if $len(idxTo) > 0$ then
          $maskEdges \leftarrow calculateMaskEdges(intervals)$
          $maskBetween \leftarrow calculateMaskBetween(idxFrom, idxTo, intervals)$
          $B_c \leftarrow calculateBettiConstituent(maskBetween, maskEdges)$
          $B \leftarrow B + B_c$
      end if
  end for
  return $B$
```

Complexity of the Algorithm 1 is $O(nk^2)$, where n - number of curves (or batch size), k - number of points in the curve. Since in most cases number of points is less then batch size, algorithm would be linear in n for large batch sizes or equivalently number of curves.

Another concern for deep learning models is differentiability of the resulting function. In case of our implementation *symbolic perturbation* is completely differentiable and all other operations performed throughout Algorithm 1 boil down to application of masks of different design. It might be then inferred that the resulting Jacobian (or its approximation) would be a constant matrix, or at least a matrix with singular values close by modulo to 1. This explains why algorithm 1 is stable and may not cause gradient explosion or dying gradients.

5.1.2 Betti Curve Loss

To utilize *Betti Curves* for training, a metric needs to be implemented on them (see, for example, 3.3). However, in most cases, the length of the *Betti Curves* is inconsistent across samples from different datasets. For instance, if the target dataset is sampled from a circle without perturbation, the number of tuples in its *persistence diagram* would be just one. On the other hand, a small perturbation would already cause the number of tuples to increase significantly. For this reason, in order to calculate the *Betti Curve loss* as defined in 3.3, the domains of the two *Betti Curves* need to be aligned.

Due to inconsistencies in the number of intervals in the *Betti Curve* representation, iterating over each pair of compared curves is required. This may potentially hinder the performance of the algorithm. First, the curves are aligned to ensure they have the same underlying domain (as there might be a different number of points in each curve). Next, the points of the curves are ordered and used to construct intervals. Interleaving consecutive intervals and forming a larger domain helps build the final representation by repeating values from both curves the required number of times. The sought alignment is therefore equivalent to extending the *Betti Curve definition* to the union of points from two or more curves. The algorithm is formalized as Algorithm 2.

Algorithm 2 Betti Curve Domain Alignment

```
Require: $Bs \in \mathbb{R}^{nxk}$ return $y = Bs$
  $n \leftarrow size(D)[0]$
  $Bs \leftarrow padToSameSize(Bs)$
  $minX \leftarrow findGlobalMinimum(B_s)$
  $maxX \leftarrow findGlobalMaximum(B_s)$
  for $i \in 1, \ldots, n$ do
      $curve1 \leftarrow Bs[i][0]$
      $curve2 \leftarrow B[i][1]$
      $curveMaskLess \leftarrow makeMaskLess(curve1, curve2)$
      $curveMaskGreater \leftarrow makeMaskGreater(curve1, curve2)$
      $numberInterleaves1 \leftarrow getNumberOfInterleaves(curve1)$
      $numberInterleaves2 \leftarrow getNumberOfInterleaves(curve2)$
      $curveInterleave1 \leftarrow interleave(curve1, curve2, numberInterleaves1)$
      $curve2 \leftarrow interleave(curve2, curve1, numberInterleaves2)$
      $curve1 \leftarrow curveInterleave1$
  end for
  return $Bs$
```

Alignment of Betti Curves in algorithm 2 also has complexity of $O(nk^2)$, where n - number of curves (or batch size), k - number of points in the curve. This is evident from the loop in the core of the algorithm and quadratic complexity of procedures inside. As in previous case this shows, that for large batch sizes algorithm works in near linear time. Resulting aligned Betti Curves can now be plugged in any metric to quantify difference between them based on the value of functions since they are defined at the same set of arguments.

5.1.3 Betti Curve Regularization

One of the applications of *Betti Curves* revolves around measuring how close is a given function to having only one maximum. While other approaches might be used, *Betti Curves* built on top of the level sets of the 1D function capture information about how rapidly the function changes and how the topology of the superlevel sets is evolving. This intuition motivated us to consider calculating AUC (Area Under the Curve) of a *Betti Curve representation* and apply this term as a regularization in different scenarios, which will be discussed in the experiments section 6. A broad prospective on the differentiable algorithm for constructing persistence diagram for 1D function based on higher sublevel or superlevel sets is provided below.

Complexity of the Algorithm 3 is $O(nklog(k))$, where n is batch size, k - number of points constituting a curve. Here the logarithmic complexity is caused by sorting operation taking place in the loop, given that our implementation of the vectorized union find is linear in both k and n we get the following performance. Algorithm for vectorized *Union-Find* is provided in 4

To understand the intuition behind minimizing the *Betti Curve AUC*, it is helpful to first consider the concept of function sublevel filtration, as defined in 3.2.1. According to this definition, a function with low topological activity—such as a unimodal function—will have fewer intervals or shorter intervals in its *Betti Curve* representation, leading to a smaller AUC. This provides insight into how minimizing the AUC is similar to minimizing the *total persistence* (see 3.2.7) of a *persistence diagram*. Both approaches would lead to a function with fewer topological features, such as a smoother function with suppressed local maxima.[4]

It is also worth mentioning, that all of the outlined algorithms in this form were implemented in PyTorch framework and can easily be run in CUDA backend resulting in significant speed-up for large scale problems.

Algorithm 3 Vectorized Calculation of Persistence Diagrams (1D)

```
Require: $function \in \mathbb{R}^{n \times k}$, $order \in \{sublevel, superlevel\}$
Ensure: $persistenceDiagram \in \mathbb{R}^{n \times m \times 2}$
  $function \leftarrow function + symbolicPerturbation(function)$
  $indices \leftarrow sortIndices(function, order)$
  $uf \leftarrow UnionFindVectorized(n, k)$
  $persistencePairs \leftarrow initializePairs(n, k)$
  for $i \in 1, \dots, k$ do
      $idx \leftarrow indices[:, i]$
      $value \leftarrow gather(function, index)$
      $u, v \leftarrow neighbors(value, idx)$
      $mergeLeft, mergeRight \leftarrow calculateMergeConditions(value, u, v)$
      if $isLocalMaximum(value, u, v)$ then
          $olderParent \leftarrow determineOlderComponent(uf, index)$
          $persistencePairs \leftarrow updatePairs(olderParent, idx)$
      else
          $uf \leftarrow mergeComponents(uf, idx, mergeLeft, mergeRight)$
      end if
  end for
  $persistenceDiagram \leftarrow gatherPairs(persistencePairs, function)$
  return $persistenceDiagram$
```

Algorithm 4 Vectorized UnionFind

```
procedure FIND($q$)
    Input: Query tensor $q$ of shape $(b)$
    $b, n, \_ \leftarrow$ parents.shape
    current $\leftarrow$ parents
    while True do
        Reshape $q$ to match batch dimensions:
            q_reshaped $\leftarrow$ $q.unsqueeze(0).repeat(n, 1).T$.flatten().view($b, n, 1$)
        Create a mask to find $q$'s position in parents:
            mask $\leftarrow$ parents[..., 0] $==$ $q$
            mask $\leftarrow$ concatenate(mask, mask)
        Update current by selecting parent nodes:
            current $\leftarrow$ parents[mask].view($b, 2$)
        Update $q$ with the new parent:
            $q$ $\leftarrow$ current[..., 1]
        if current[..., 0] $==$ current[..., 1] for all elements then
            break
        end if
    end while
    return current[..., 0]
end procedure
procedure MERGE(requests)
    Input: requests of shape $(b, m, 2)$ containing pairs to merge
    $b, n, \_ \leftarrow$ parents.shape
    for $j, q$ in enumerate(requests.unbind(1)) do
        Extract pairs $q_1, q_2$ from $q$
        Find their roots:
            $q_1 \leftarrow$ Find($q_1$), $q_2 \leftarrow$ Find($q_2$)
        Reshape $q_1$ for matching batch dimensions
        Create mask for $q_1$ in parents
        Update parents to merge the sets:
            Replace parent of $q_1$ with $q_2$
    end for
    return parents
end procedure
```

5.2 Models

The models employed in this study focused on the application of topological terms, utilized either as regularizers or additional loss terms, depending on the architecture. A comprehensive overview of how these approaches are implemented or combined to enhance the functionality of the classic *NeRF* is provided in Chapters 6 and 7.

5.2.1 Topological Autoencoders with Betti Curve supervision

Our first experiment studied quantification of the effect of *topological autoencoders* as defined in paper [12] with *Betti Curve loss* supervision. It was found that *Betti Curve loss* alone did not suffice to preserve the topology of the latent space under the transformation defined by the *autoencoder*. Unlike *pairing loss* (see 3.2.6), *Betti Curve loss* lacks the ability to capture which elements of the dataset are topologically relevant. On the other hand, when added as a supplementary regularization term, accelerated convergence of the model was observed. Indeed, while local structure is captured by *pairing loss*, global topological information is covered by *Betti Curve loss*. The proposed architecture is summarized in Figure 5.1.

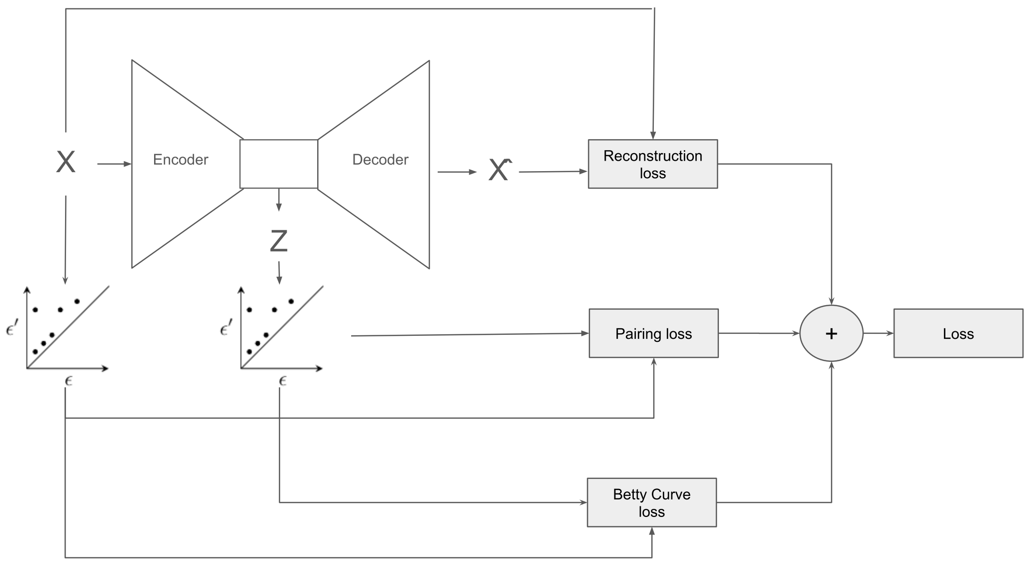


Figure 5.1: **Architecture of Topological Autoencoder with Betti Curve Supervision.** Here, X - input dataset, $\hat{X}$ - reconstructed dataset, Z - latent code. Reconstruction loss is chosen to be MSE Loss in this case.

5.2.2 PointNet with Betti Curve supervision

To further build the logic around how *Betti Curve transformation* affects overall training process it was decided to experiment with *PointNet*[28] architecture, which was used as a backbone model with an aim to train the model to classify between different shapes. The overall design of the model is provided in Figure 5.2.

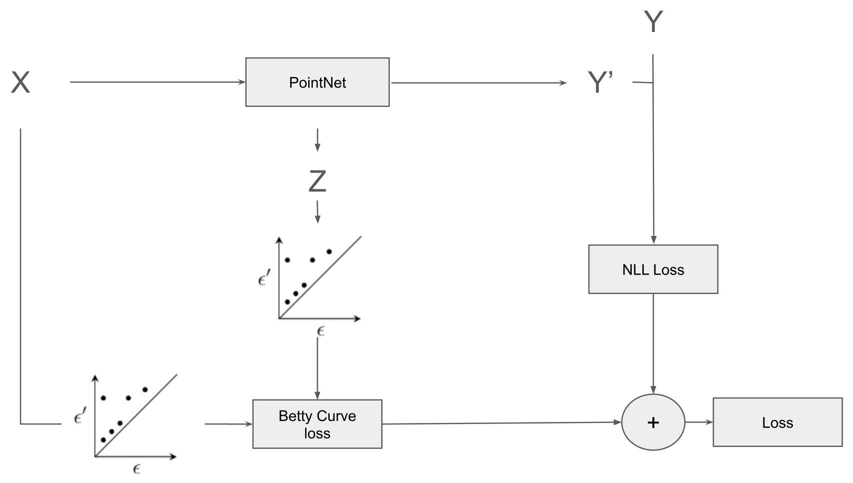


Figure 5.2: **Architecture of PointNet with Betti Curve supervision**. Here, X - input dataset, Y′ - output labels, Y - ground truth labels and Z - latent code. NLL Loss - Negative Log Likelihood Loss

5.2.3 Topology supervised Normalizing Flows

As described in section 3.4, *Normalizing Flows* (*NF*) are powerful generative models that can be used to model the probability density function (pdf) of a given distribution. In the context of *NeRF*, this is useful because the *normalizing flows* model can be trained to predict the probability of samples in 3D space that have not yet been visited by *NeRF* or suffer from insufficient density allocation. To embed topological awareness into the *normalizing flows* model, a *Wasserstein distance* (see 3.2.4) term, as described in [4], was added to the log-likelihood loss. This led to faster support convergence and less noise in the samples generated by the model. Several popular *normalizing flows* designs (see 3.4) were employed in our experiments.

For further modeling, the *RealNVC* architecture was chosen, as it can be used both to estimate the probability of observations and to generate new, unseen samples from the learned distribution [27]. The architecture of our approach is shown in Figure 5.3.

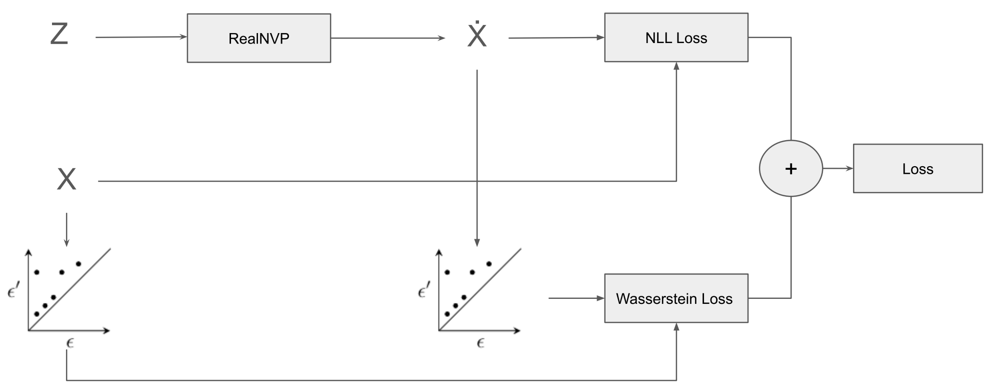


Figure 5.3: **Architecture of Topology supervised Normalizing Flows.** Here, Z - latent code, $\dot{X}$ - generated samples, X - original dataset. NLL Loss - Negative Log Likelihood Loss

5.2.4 Gaussian Mixture Normalizing Flows

Since some low-density regions of the objects are difficult to model with the *NF* model, it was decided to first determine the topologically relevant features of dimension zero of the original shape and then consider a Gaussian Mixture Model (GMM) with means initialized at the centers of the edges.

In this way, the 3D scene was segmented into regions, which were then used to fit the *NF* model separately for each region. During the training phase, sampling from the mixture components was used to ensure smooth stitching along the edges of the regions.

One of the limitations of *normalizing flows* is their inability to capture the topology of the manifold from which the dataset is sampled. This occurs because the model is highly dependent on the bias introduced by the choice of the base distribution. While the popular choice of a multivariate normal distribution works in most cases, it is practically impossible to eliminate the effect of discrepancies in the topologies of the latent space and the target distribution. For example, when attempting to model a distribution with support residing on disjoint subsets of the original space, one would observe long-lasting bridges between the two components. This happens because continuous functions cannot change the topology of the transformed set when defined on a convex space. For this purpose, initialization with a disjoint Gaussian distribution or adopting a strategy such as that in paper [24] might help. This limitation further motivated our choice of using a Gaussian Mixture Model to preemptively split the dataset into components, making it easier for the *NF* model to learn.

5.2.5 Topological NeRF

In this section we describe our way to build final model of this work - *NeRF Topological* and which strategies like regularization approaches were employed. Over the course of our study we moved from exploiting traditional *MLP* [7] architecture to attention based model. Eventually we stopped on *transformer* based architecture as implemented in paper [32], which was additionally been equipped with regularization over attention weights matrix using *Betti Curve AUC regularization*.

This approach encouraged the model to learn dependencies between positions on the rays. However, it was observed that the resulting attention masks were noisy, which led to suboptimal performance. To address this issue, topological regularization was introduced to smooth the weights. The regularization was applied row-wise, utilizing the vectorization scheme described in Algorithm 1. Thanks to the differentiability of our approach, model could be trained in an end-to-end fashion.

Inspired by BERT [33] only mid-level attention weights were subjected to regularization, as those are excelling the most at capturing deep semantic relationships. In our case it was expected to capture connection between rays and thus spare time for traversing the whole entirety of rays. Architecure of *NeRF Topological* is illustrated in 5.4

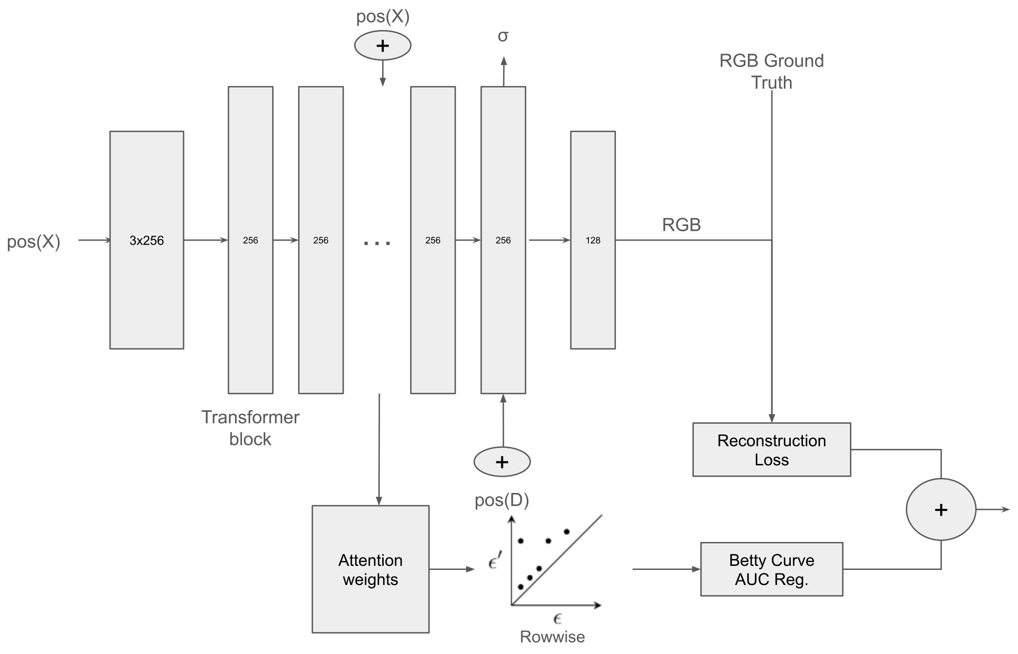


Figure 5.4: **Architecture of Topological NeRF (TopoNeRF)**. Here X - input rays positions, D - directions, pos(X), pos(D) - positional encodings of X and D respectively, σ - predicted intensity of the rays, RGB - predicted vector of colors

6 Basic Experiments

This section highlights the main results achieved in this work. We start with benchmarking the *Betti Curve regularization* term and loss in combination with various architectures, as described in Chapter 5. Several basic datasets, representing different topological properties, are examined. Specifically, algorithms are evaluated on datasets sampled from single circle and double circles. Additionally, examples such as the "Spheres" dataset from the *topological autoencoder* paper and the FashionMNIST dataset are included.[12]

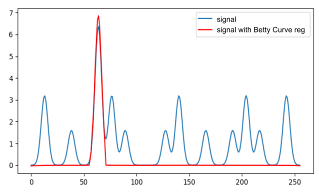
Beyond topological datasets, the effect of regularization via the *Betti Curves AUC*, as defined in Section 6.1, is also investigated.

6.1 Effect of Betti Curve Regularizer

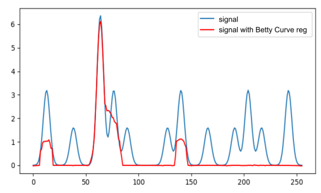
As was explained in Chapter 5, the effect of *Betti Curve AUC Regularization* is expected to manifest as smoothing, which relies on minimizing the number of topological features, similar to the effect of *total persistence* (see 3.2.7). [4] For instance, this might involve minimizing the number of local maxima. To validate this assumption, several experiments were conducted.

In the first test, a signal was modeled as a mixture of *Gaussians*. For the second experiment, a signal was composed of incoming *exponential distributions* with random offsets but the same scale. By doing so, incoming excitations were emulated. Such a signal is referred to as a *Hawkes signal*, as it mimics a point process with self-excitation as described in the paper. [34] In both experiments, an *MLP* was trained to minimize reconstruction loss with the regularization term of *Betti Curve AUC*. Training was performed for 200 epochs, with a learning rate of $\alpha = 10^{-2}$ and regularization values $\lambda \in \{1, 10^{-1}, 10^{-2}\}$. For the final test signal, the density function arising from *NeRF* during training was considered.

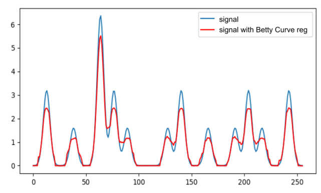
Results illustrated by figures 6.1, 6.2 and 6.3 suggest that the effect of *Betti Curve regularization* manifests itself as expected, with higher regularization aligning with a higher degree of smoothness. When compared to other schemes commonly employed to smooth a signal, it is most similar to *L1-regularization*, which is, in turn, connected to *soft-thresholding*.[35] It should, however, be noted that, unlike *LASSO*, it is exclusively based on minimizing the topological activity of the signal.



(a) $\lambda = 1$

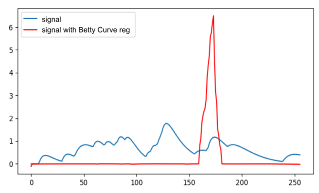


(b) $\lambda = 1 * 10^{-1}$

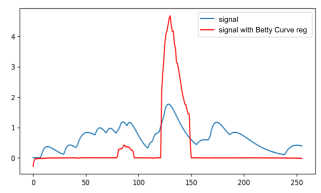


(c) $\lambda = 1 * 10^{-2}$

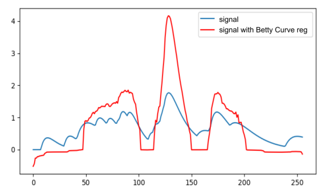
Figure 6.1: **Betti Curve Regularization effect for Gaussian signal.** Here λ denotes regularization power. Signal is smoothed and only global maximum persists when regularization strength is increased.



(a) $\lambda = 1$



(b) $\lambda = 1 * 10^{-1}$



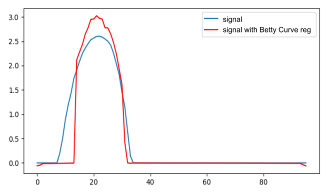
(c) $\lambda = 1 * 10^{-2}$

Figure 6.2: **Betti Curve Regularization effect for Hawkes signal.** Here λ denotes regularization power. Regularized signal spikes at the point proximal to argmaximum and allocates mass there

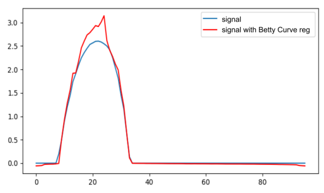
An important remark must be made regarding results depicted in Figure 6.3. In this scenario, we operated at the level of a typical intensity profile obtained from training a *NeRF* and extracted during a period where the original model had not yet converged but had already started exhibiting a tendency for allocating higher values to a specific region.[7] Despite being an expected profile, it might nevertheless be too broad, causing the resulting image to appear blurry. The effect of *Betti Curve regularization* is exactly what helps here, as it cuts the heavy tails of the signal and tightens the support. This provides further intuition on how to apply this approach in the context of *NeRF*.

6.2 Classification based on Betti Profile

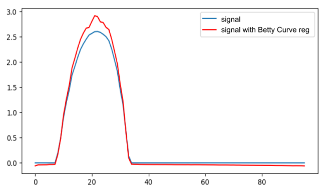
Another experiment that was conducted to estimate expressivity of *Betti Curve Transform* was testing if a simple classifier could differentiate between shapes with only *Betti Curve* provided as input. Due to inconsistency of number of points over *Betti Curves* in the batch, we applied alignment algorithm 2 to represent each curve on



(a) $\lambda = 1$



(b) $\lambda = 1 * 10^{-1}$



(c) $\lambda = 1 * 10^{-2}$

Figure 6.3: **Betti Curve Regularization effect for NeRF signal.** Here λ denotes regularization power. Support of the signal is narrowed down redistributing density over a tighter region

the same domain and thus enable their comparison. Unlike paper [12] we chose dimension one as a target feature, since objects in our dataset differed by the number of holes.

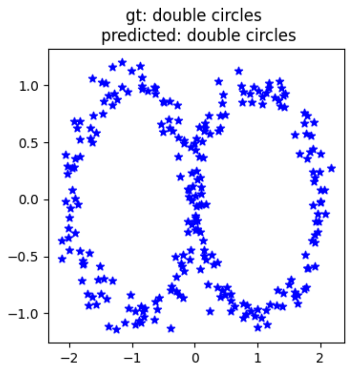
Our classifier was benchmarked against simple dataset comprised by perturbed single and double circles. Stability theorem [21] ensures that low magnitude noise does not change persistence diagrams drastically with high probability and thus *Betti Curves* could serve as a feasible descriptor for this task. This time even though all *Betti Curves* in batch were mapped to the same domain, dimension might vary over batches. Therefore we trained a 1D fully convolution neural network to mitigate input size restriction and handle *Betti Curves* of different number of points.

Experiment was repeated for several noise levels: $\sigma \in \{10^{-1}, 2.5 * 10^{-1}\}$. Figures 6.4, 6.5 highlight that a simple neural network could learn to discriminate between shapes based on *Betti Curves* as descriptors only, thus demonstrating their expressivity in context of topological machine learning.

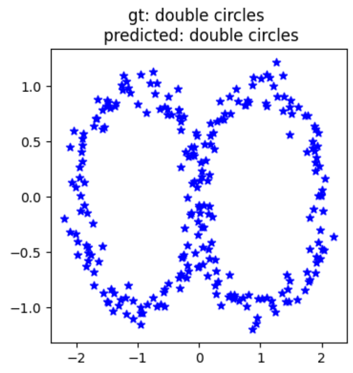
Together with Experiment 6.1 and 6.2 we can now deduce, that *Betti Curves* implemented in this work can indeed be used for various machine learning tasks, which are not only constrained to the scope of topological machine learning. This is showcased by the Experiment 6.1, which demonstrated aptitude of the model for smoothing functions for different purposes.

6.2.1 Accelerated Convergence of Topological Autoencoders

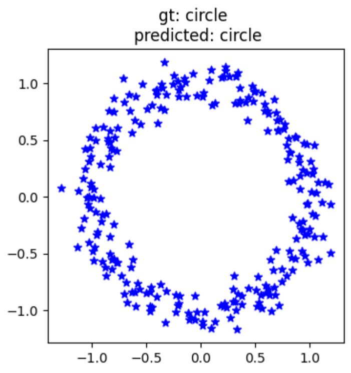
In the next experiment, a topological autoencoder, as defined in [12], was trained under the same settings as those used by the authors for the Spheres dataset. While utilizing the *pairing loss* defined in 3.2.6 as a regularization term in combination with reconstruction loss, it was additionally supplemented with the *Betti Curve loss* regularization (see 3.3) to further improve convergence speed. Similarly to the *pairing loss*, additional regularization was applied to compare the *Betti Curves* of the original



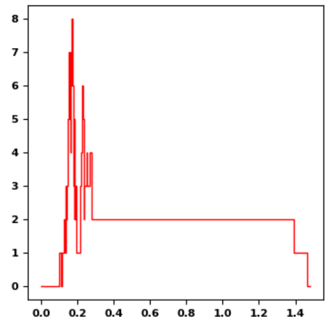
(a)



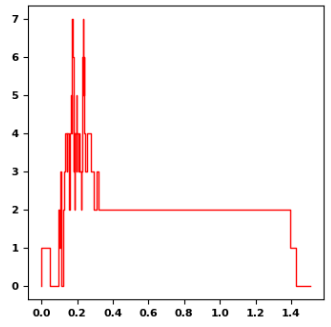
(b)



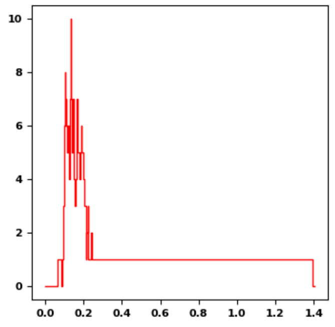
(c)



(d)

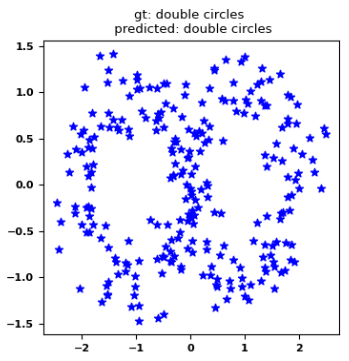


(e)

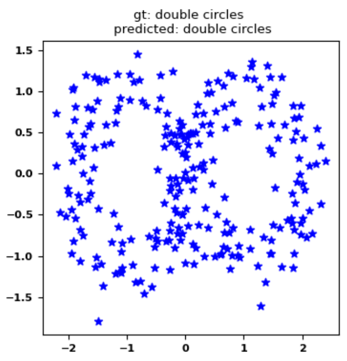


(f)

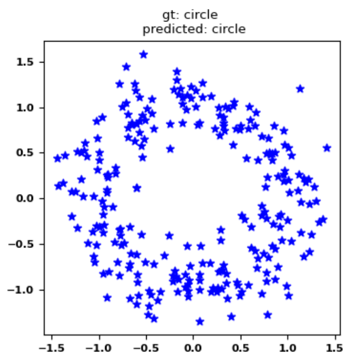
Figure 6.4: **Point sets and their *Betti Curves* subjected to classification for different shapes with noise** $\sigma = 1 \times 10^{-1}$ (a) - (c) show different shapes with noise pertubation while (d) - (f) illustrate respective *Betti Curves*. A clear definitive trait of a long living plateau with level 2 for double circles and level 1 for a single circle is visible, which is a defining feature for the model



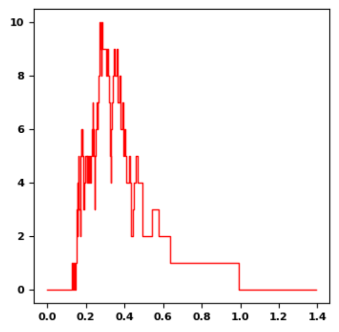
(a)



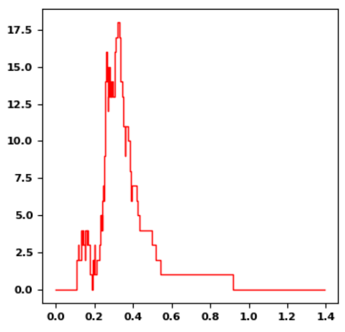
(b)



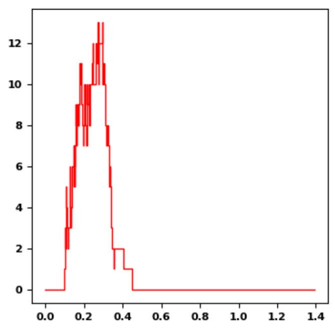
(c)



(d)



(e)



(f)

Figure 6.5: **Point sets and their *Betti Curves* subjected to classification for different shapes with noise** $\sigma = 2.5 \times 10^{-1}$**.**(a) - (c) show different shapes with noise pertubation while (d) - (f) illustrate respective *Betti Curves*. Plateau is less highlighted due to increased noise in the samples. Model however managed to classify shapes correctly

data with the *Betti Curves* of the latent code. The model was trained multiple times for only 25 epochs and demonstrated accelerated convergence to a state similar to what was described in the paper.[12] This was further validated by the metrics comparison after evaluating the training results, as illustrated in Figure 6.6.

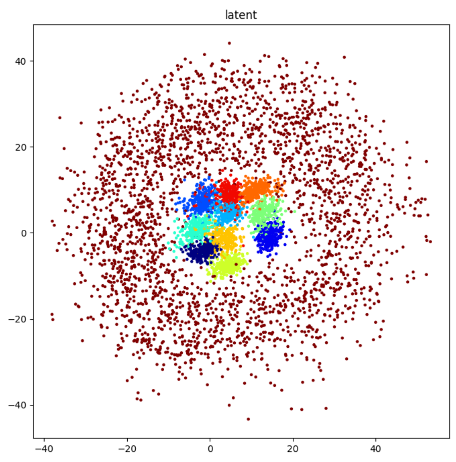
Another possible application of topological supervision involved imposing regularization via *Betti Curve AUC* on the *Betti Curves* of the latent representations. However, it was found that this regularization, acting as a restriction on topological activity, did not perform well and hindered the model's performance. This outcome was expected, as the neural network needed to explore the latent space, and the imposed loss caused it to perform poorly in an exploratory setting.

From the results provided in Tables 6.1 and 6.2, it may be deduced that using additional supervision of *Betti Curves*, based on comparing latent representations and reconstructed samples, contributed to the overall reconstruction quality by reducing the corresponding loss term. The *pairing loss* is higher than that of the model without the *Betti Curve* supplement. This could be viewed as scattered attention of the network during training, as two regularization terms are now included for optimization. Figure 6.6 suggest that latent representations remained undisturbed by the introduction of new information.

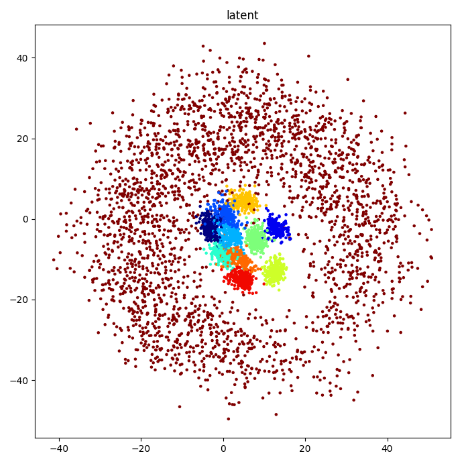
Unlike Experiment 6.2, information from the *Betti Curve* is not sufficient to ensure salient representations in the latent space. Indeed, such descriptors of persistence diagrams bear no notion of pairing. This resulted in linear-segmented behavior in the latent space observed after training. Spheres were squeezed and stretched along random axes, which preserved distances but lacked pairings. Consequently, the *Betti Curves* appeared the same, yet they failed to capture the topological profile of the input data.

From Figure 6.6, it may be inferred that when trained with additional *Betti Curve loss* regularization, clusters are better separated, and the encompassing sphere is less interfering with the interior spheres. On the other hand, without regularization, the *topological autoencoder* (*TopoAE*) exhibited tighter cluster locations, with the encircling sphere more intertwined with the inner spheres.

A supplementary experiment was conducted to investigate the model's performance on a dataset more related to the real world. For this purpose, the FashionMNIST dataset, described in 2.2 and also used to benchmark the original model, was chosen. [12] The model was then trained for 25 epochs on the dataset under the same two settings as were used for the Spheres dataset. Upon closer examination, Figure 6.7 reveals quite similar patterns. This similarity might be attributed to the fact that the manifold on which the dataset resides is likely more complex than that of the Spheres dataset, causing both models to struggle in capturing its topology in a visually interpretable manner.

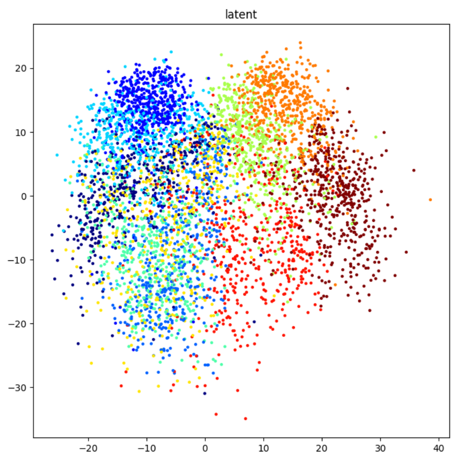


(a) $\lambda = 0$

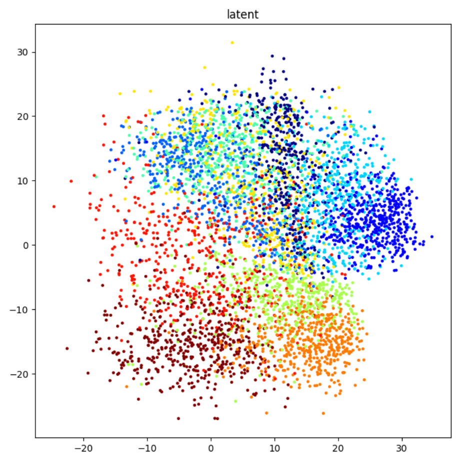


(b) $\lambda = 1 * 10^{-1}$

Figure 6.6: **Topological Autoencoder with Betti Curve Loss Regularization, trained for 25 epochs on Spheres Dataset.** Here λ denotes regularization power. Separation is more visible in our model and highlights how regularization improves learned representations



(a) $\lambda = 0$



(b) $\lambda = 1 \times 10^{-1}$

Figure 6.7: **Topological Autoencoder with Betti Curve Loss Regularization, trained for 25 epochs on FashionMNIST Dataset.** Here λ denotes regularization power. Representations share similar patterns in the latent space emphasizing complexity of the underlyin manifold

Model	Reconstruction Loss	Pairing Loss	Betti Curve Loss
TopoAE	3.45	**54.23**	0.33
TopoAE + Betti Curve Loss	**3.34**	54.62	**0.16**

Table 6.1: **Training results for different variations of Topological AE on Spheres dataset**. TopoAE + Betti Curve Loss captures topology better than simple TopoAE compared by metrics, pairing loss is lower probably due to sharing regularization weight with *Betti curve loss*

Model	Reconstruction Loss	Pairing Loss	Betti Curve Loss
TopoAE	0.042	12.89	0.17
TopoAE + Betti Curve loss	**0.04**	**11.82**	**0.16**

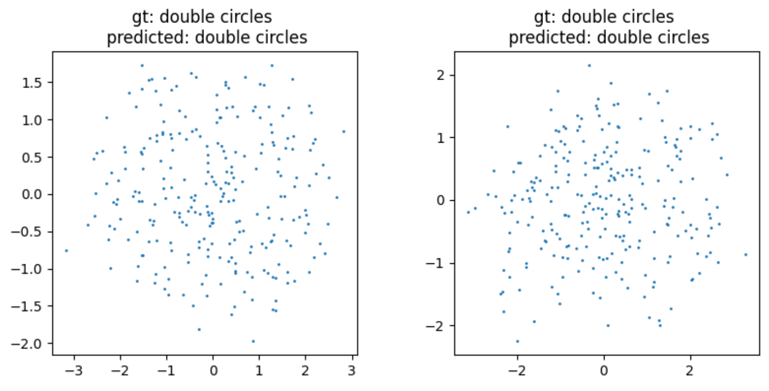
Table 6.2: **Training results for different variations of Topological AE on FashionMNIST dataset**. All metrics are lower for the toplogical autoencoder model with *Betti Curve loss* supervision

6.3 PointNet with Betti Curve Regularization

For the next experiment, a classification task similar to Experiment 6.2 was considered; however, this time, a *PointNet* [28] model was used as the backbone. Unlike ordinary neural networks, where interactions must be learned independently, this model is capable of capturing spatial features pertinent to a point cloud as an object. Inspired by paper [12], the model was provided with topological information by applying regularization to the last layer before the head of the model. In this way, it was encouraged to preserve the same topology of the latent code as that provided by the input.

Additionally, with regularization embedded by a term comparing the *Betti Curves* of the original dataset and those of the latent space, the model was trained for 200 epochs with a learning rate of $\alpha = 5 \times 10^{-3}$ and a regularization strength of $\lambda = 1 \times 10^{-2}$.

The *PointNet* model is primarily used for classification or segmentation tasks [28]. However, it is less effective for tasks involving warping or spatial transformations due to its architecture, which assumes rotation and translation invariance [28]. Regardless of whether topological supervision was applied, the model produced similar results in terms of both accuracy and loss. This suggests that *PointNet* is inherently a robust learner of the spatial relationships within the data.



(a) $\lambda = 0$ (b) $\lambda = 1 \times 10^{-2}$

Figure 6.8: **PointNet trained with and without topological regularization.** Here λ denotes regularization power. Both Figures show that tasks were handled correctly by both models. Presence of topological regularization did not improve convergence nor metrics

6.4 Topology supervised Normalizing Flows with simple shapes

In our last experiment our focus was set on normalizing flows model, specifically *RealNVP* architecture.[27]. A simple dataset was chosen to benchmark our models. It represented two disjoint non-convex point clouds reminiscent of crescent moons. Despite successful convergence discussed in paper [27], learning such distribution might be theoretically problematic for traditional normalizing flows architecture with multivariate Gaussian base distribution. It might be hypothesized that while not being able to exactly match the topologies the model can still converge faster to the setting, where the support is more topologically near to that of a target distribution. Nevertheless there is always a gap caused by the bias of choosing base distribution.

RealNVP based normalizing flow was trained for the following three settings: without any topological guidance, with *Wasserstein* supervision and finally using Gaussian mixture label assignment. The learning rate was chosen to be $\alpha = 10^{-3}$ and number of epochs was set to 1000. Performance was compared with respect to negative log-likelihood, *Wasserstein loss* (see 3.2.4), and *Betti Curve loss* (see 3.3).

Figure 6.11 shows that, for all models except the *Wasserstein*-regularized one, there is a slight gap at the point on the *Betti Curve* where the subsequent value equals 1. This point marks the stage in the filtration where all topological activity stabilizes. This observation indicates that the regularization indeed contributed to more accurately capturing the topology of the original dataset. Despite this, the

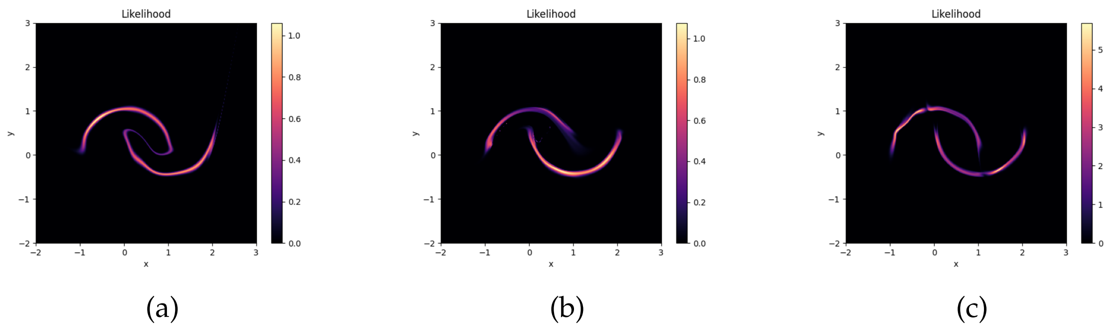


Figure 6.9: **Likelihood for various *NF-based* sampling methods**(a) There is a non-zero probability bridge-like region connection two components in ordinary *RealNVC NF*. (b) A bridge-like artifact is less visible in the case of *Wasserstein* regularization. (c) GMM Based RealNVC produces no artifact, however there is a slight problem at the borders of the mixture components.

Gaussian mixture-normalizing model achieved a comparable approximation of the topological structure, as evidenced by Figure 6.9. Both models outperformed the original, regularization-free normalizing flow model, which is evident from the log-likelihood values averaged across all observations given in Table 6.3.

The likelihood plots in 6.9 highlight the issue mentioned above regarding the inability of normalizing flows to capture topologies that deviate from the support of the base distribution. Upon closer inspection, a long-living, narrow region bridging two components is observed in both the original model and the *Wasserstein*-regularized model. The Gaussian mixture normalizing flow model, on the other hand, explains the choice of this architecture, as it effectively separates the two regions. However, a problem arises at the decision boundary of the mixture model, recognizable by lower densities near the center of mass of the crescents 6.10c. To address this, alternative component distributions could be considered. Samples generated from the models, shown in Figure 6.10, illustrate how these models behave in practice. In particular, they demonstrate how low-density regions can pose challenges when generating a large number of samples.

This experiment helped us understand that topological supervision is beneficial for generating samples from unknown distribution in order to facilitate training for *NeRF* and help it deal with insufficiently explored regions in 3D space.

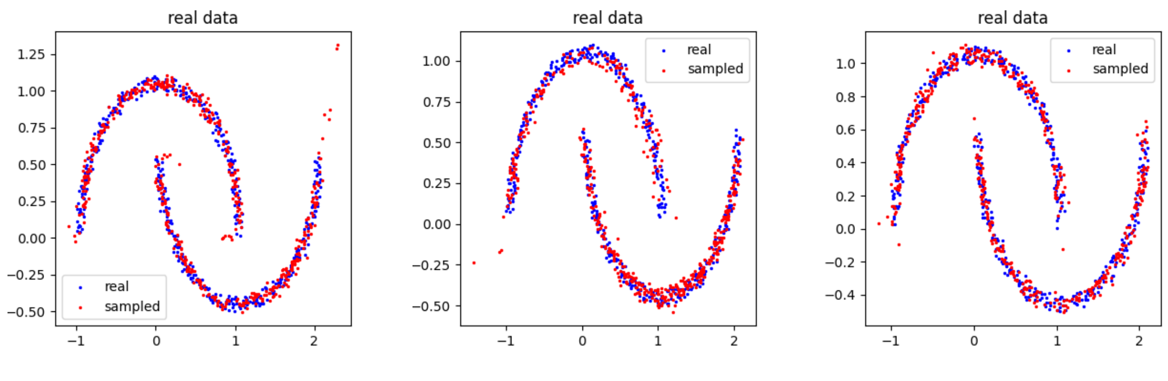
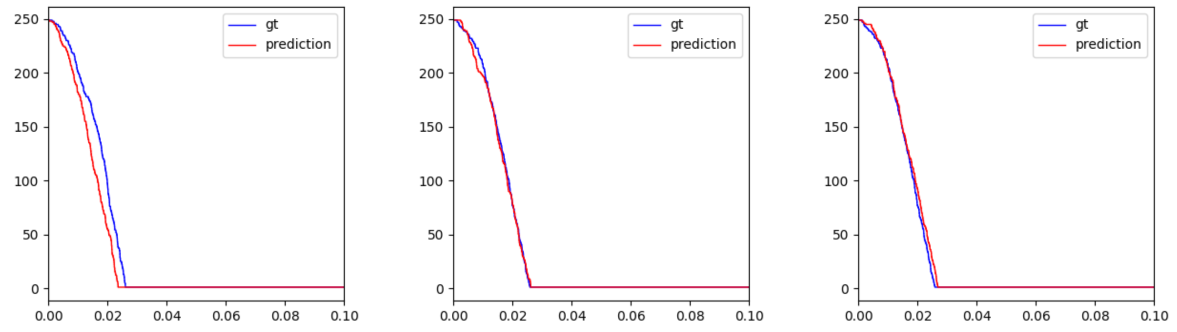


Figure 6.10: **Samples for various sampling models NF-based sampling methods.** Figure compares the samples generated by three different normalizing flow models. (a) A simple *NF* model without topological regularization, which produces a slight brige between two disjoint components. (b) *NF* with *Wasserstein regularization*, where the generated samples exhibit improved alignment with the underlying data distribution. (c) *NF* conditioned on a Gaussian Mixture, which demonstrates a better ability to capture complex data distributions.

Model	Log-likelihood	Bottleneck distance	Betti Curve Loss
Simple NF	0.13	2.71	0.99
Simple NF + Wasserstein Reg.	0.04	2.09	0.04
Simple NF + Gaussian Mixture	**-10.57**	**1.26**	**0.03**

Table 6.3: **Training results for different variations of Normalizing Flows**. Gaussian Mixture based *NF* is clearly dominating over others model, on the other hand it introduces additional computational overhead. Metrics suggest that addition of *Wasserstein* regularization improved on original *RealNVC* model.



(a) simple NF (b) NF with Wasserstein Reg. (c) NF with GMM

Figure 6.11: **Betti Curves for various NF-based sampling methods.** The figure compares samples produced by three methods: (a) *Betti Curves* by the original *RealNVC* model, (b) samples from *Wasserstein-regularized* model, (c) and *GMM*-conditioned *NF* model samples. The red curve represents the ground truth *Betti Curve*, while the blue curve corresponds to the *Betti Curve* computed from samples generated by the learned distribution. A slight deviation is observed at the points where the curves drop to one. Notably, the *Wasserstein-regularized* model yields the closest match to the original *Betti Curve*.

7 Results

In this section, the main results achieved during the course of this work are outlined. The original *NeRF* model was trained for various objects, and its shortcomings were analyzed. Subsequently, the toolset described in Chaper 5 and Chaper 6 was applied. As demonstrated in Section 6.1, the application of *Betti Curve regularization* can be beneficial when addressing noisy signals characterized by an abundance of local maxima.

To build upon these findings, a *NeRF* model was trained alongside a *normalizing flows* model, reinforced with topological regularization, to provide faster convergence. This approach was implemented side by side to produce a prior probability density for the neural radiance fields model. Finally, a transformer architecture from paper [32] was employed to enable information sharing between rays, and the model was enriched with topological knowledge by incorporating a *Betti Curve AUC regularization* term.

7.1 Original NeRF

In order to benchmark other models against a ground truth, original *NeRF* architecture was trained on several objects of the simulated data described in Section 2.3. Specifically three 3D objects were chosen - chair, lego and a ficus plant. The setting was completely similar to that of the original implementation [7] except for omitting *hierarchical sampling* for simplicity. Another reason was our initial struggle to demonstrate that topological approaches might help to narrow down the regions of interest and thus decrease training time drastically. The model was trained for 200 epochs with batch size of 4096 rays, number of samples 96 per direction and positional encoding as described in paper.[7] Rays were marched through a cube $[2,6]^3$. Training was conducted on images of reduced size, specifically 256 x 256 in order to iterate faster and be able to estimate the effect of our experiments. From now on this size was considered original for all images. To downsample the images nearest neighbor algorithm was applied to preserve the data type of images.

Upon the model's convergence, it was observed that there were no significant defects in the reconstruction of the original images. However, some images exhibited coarseness and a lack of fine details. In the original *NeRF*, fine details are typically learned later in training , while coarse features, such as object contours, are captured earlier, what explains such behavior as in Figure 7.1

7.2 NeRF with Betti Curve Regularization on Rays

Before proceeding with the analysis of the results from the *NeRF* model reinforced with topological regularization, it must be stated that the original model did not perform well on the down-sampled dataset. This may be explained by the lack of spatial information, which is present in neighboring rays. For instance, in the full-resolution dataset, marching rays through pixels in a 2D image inherently provides a prior for rays passing through nearby pixels or points in 3D space. This observation was highlighted in paper [29], where the authors adjusted the original *positional encoding* to act as a cone, capturing proximal information.

To this end, there are several directions for improvement. Attempting to accelerate the learning of fine structures might be complicated, as topological information contained in colors is usually not explicit. Global structure, on the other hand, presents a more promising direction. Endowing the original *NeRF* model with knowledge of topology might yield more rapid convergence to the real shape of the object. Having seen how different architectures might benefit from the addition of global topological data in Chapter 6, this approach was tested.

To address this, regularization was applied after each forward pass. However, this approach proved to be overly restrictive for the model, leading to longer training times. The reduced exploration rate, caused by oversmoothed intensity values along the rays, hindered effective learning. Both the train and validation losses were dropping, but at a slower rate than when regularization was not applied. For this reason, it was decided to restrict the model to only subject certain rays to the regularization procedure. The candidates were sampled according to two strategies. In the first setting, a biased coin was randomly flipped, and regularization was applied to rays that landed on heads. In the second scenario, the variance of intensities was used to decide if regularization would take place. Specifically, rays with variance higher than the 0.9 quantile within the batch were subjected to *Betti Curve regularization*.

While being less informative, the first approach offered less computational overhead and fewer potential stability issues, as the variance of intensities could vary drastically over training. Yet, in practice, the second strategy yielded better results. Validation curve plots depicted in Figure 7.7, which were measured 15 times for each setting — with and without *Betti Curve regularization* — showed that the behavior displayed by the training processes varied. To be more precise, the regularization term accelerated the convergence of neural radiance fields, a phenomenon that was not observed when larger image sizes were used.

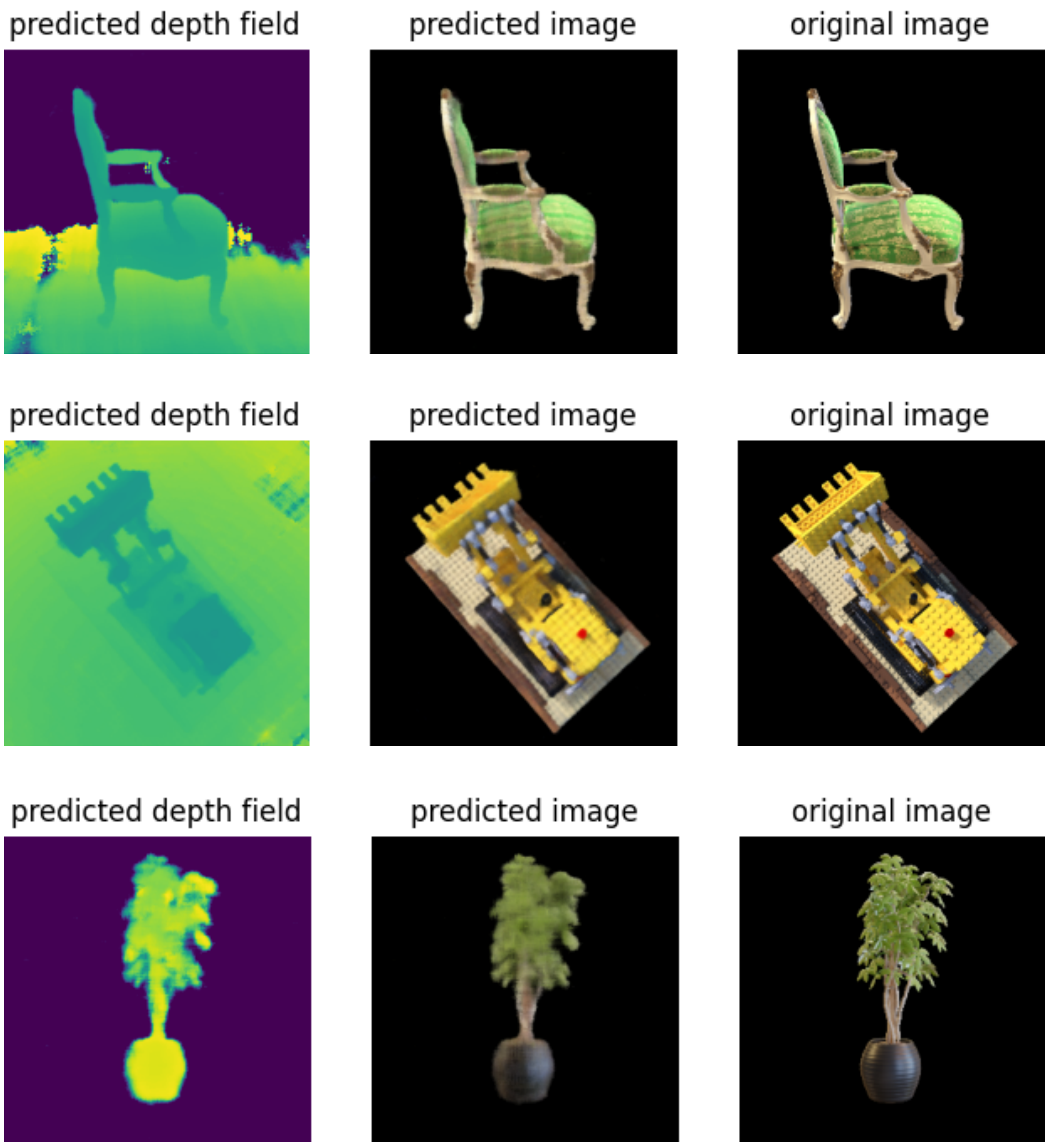
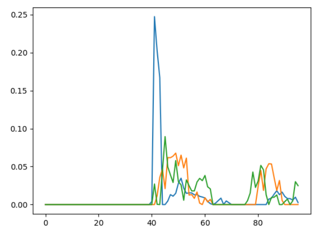
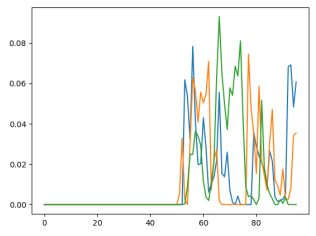


Figure 7.1: **Original *NeRF* Reconstruction after 200 epochs for chair, lego and ficus.** Despite overall structure captured well, resulting views are still blurry and depth maps display a certain degree of artifacts


(a)


(b)

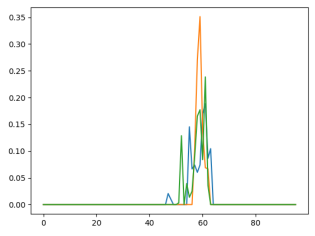

(c)

Figure 7.2: **Original *NeRF* Rays profiles for random 3 rays sampled from chair, lego and ficus objects.** Each ray is rendered with 96 points spaced evenly between 2 and 6. Local maximums help to determine expected location of the chair (a), lego (b) and ficus (c) in the space along given direction

In order to estimate regularization effect on multiple 3D objects and quantify their impact on the training process images were downsampled even further - to 64 x 64. In such setting it was possible to run training multiple times so as to statistically confirm our observations. It was found that all shapes either benefited or experienced no change after addition of *Betti Curve regularization*. The learning curves were steeper as opposed to a flatter analogue for model without topological regularization.

Judging on the observed results proposed topological regularization term might help improve convergence in low-resolution scenarios, while not exhausting computational resources of the program. A possible explanation of this effect might be that under low-resolution conditions information referring to vicinity of the points plays much lower role, because lower-dimensional pixels already compress this information. To illustrate accelerated convergence to the shape, each model was trained for 50 epochs and used to predict depth field and the view from a given angle. The results illustrated by Figure 7.6 asserted that our topological supplement helped model benefit for certain shapes and was not restrictive for others. Original model was also trained in this setting and compared to the regularized version, highlighting the effect of our approach. The results are illustrated by Figures 7.3, 7.4 and 7.5

7.3 NeRF + Normalizing Flows prior

In the previous section it was explored how information from *Betti Curves* can be utilized to achieve a certain degree of training acceleration when dealing with low-resolution images. In this experiment, the *NeRF* model was supplied with a prior probability of a point along the ray being occupied, produced by a *normalizing flows* model. This model is further referred as *Topological NF + NeRF*

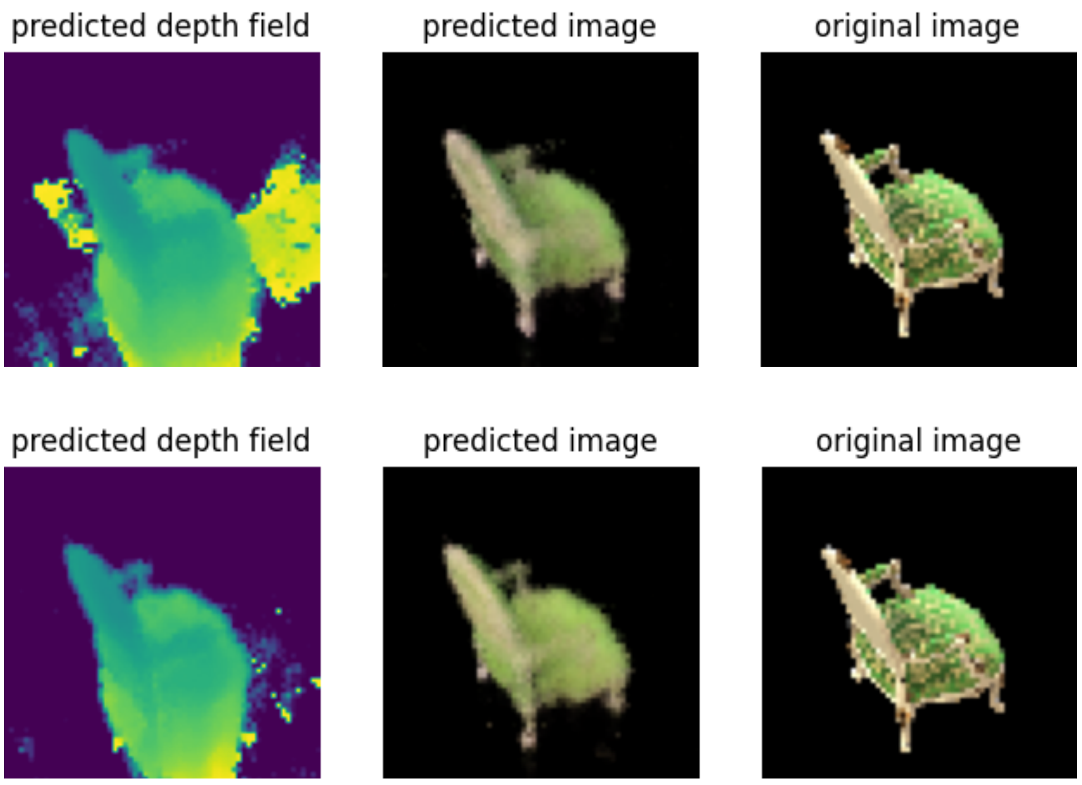


Figure 7.3: **NeRF with and without Betti Curve regularization on rays for chair object after 50 epochs.** (a) Predicted depth map and image of the *Betti Curve* regularized *NeRF* model shows less artifacts and handles topological features of the image e.g. hole between handle and seat of the chair better. (b) Original *NeRF* model predicts hole blurry and introduces more artifacts in the depth map

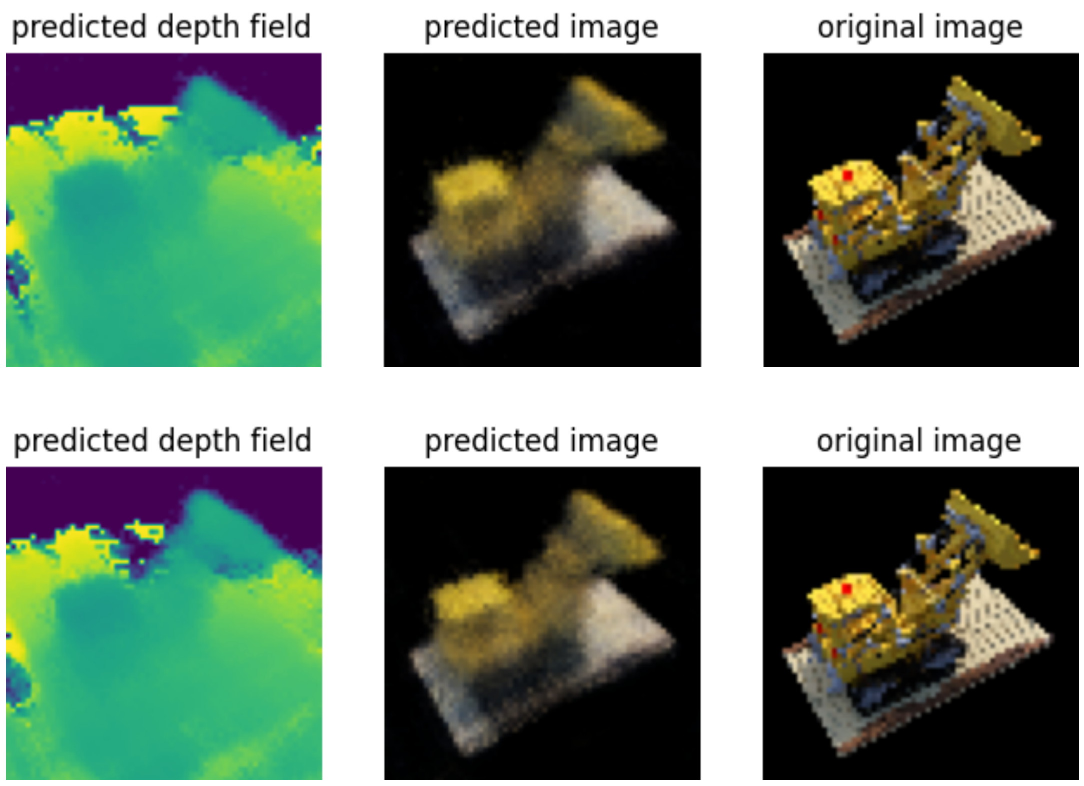


Figure 7.4: **Comparison of NeRF with and without Betti Curve regularization on rays for the chair object after 50 epochs.** (a) The predicted depth map and image of the *Betti Curve* regularized *NeRF* exhibit fewer artifacts compared to the original model. (b) The classic *NeRF* model shows poorer color capture.

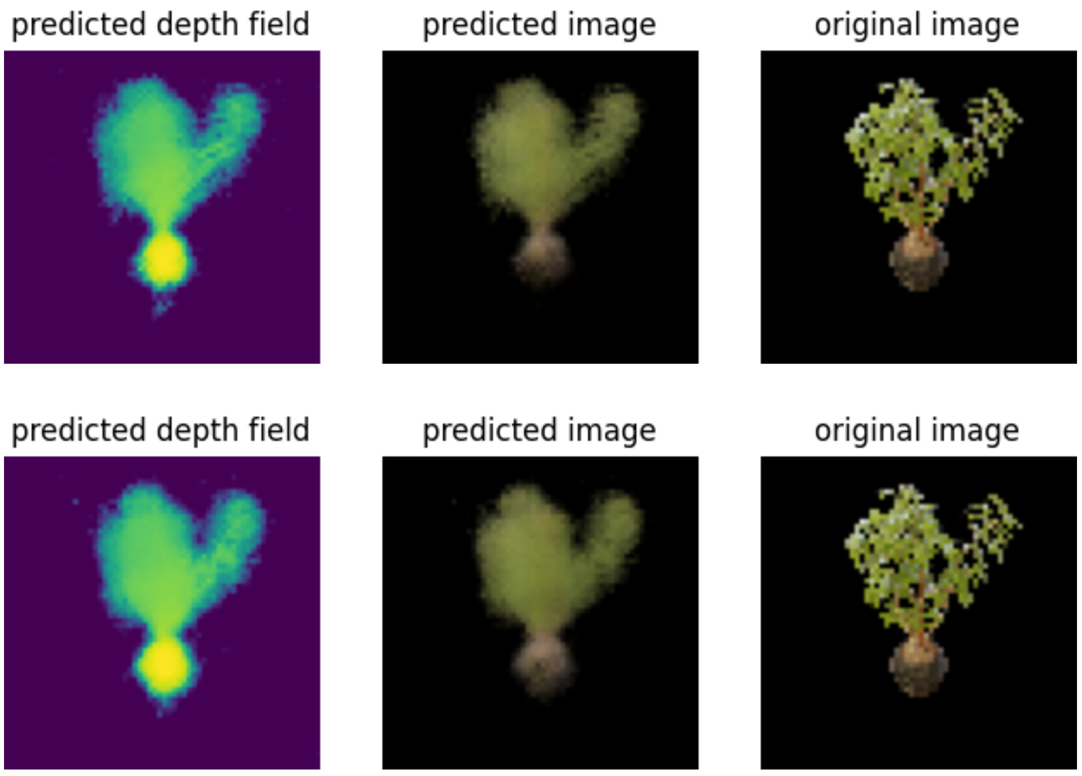
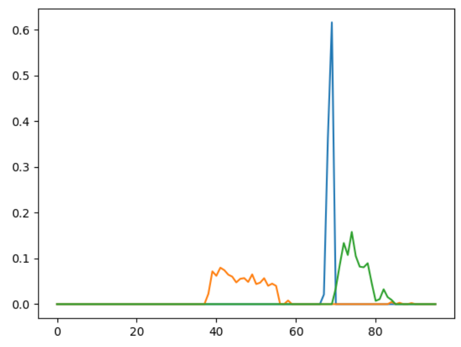
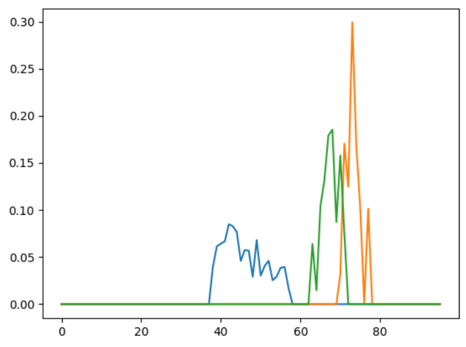


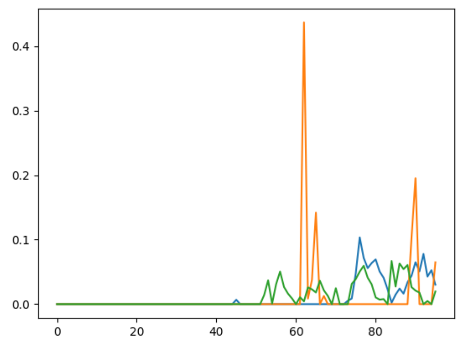
Figure 7.5: **NeRF with and without Betti Curve regularization on rays for ficus object after 50 epochs.** Both *Betti Curve* regularized *NeRF* (a) and original *NeRF* model (b) display similar pictures.



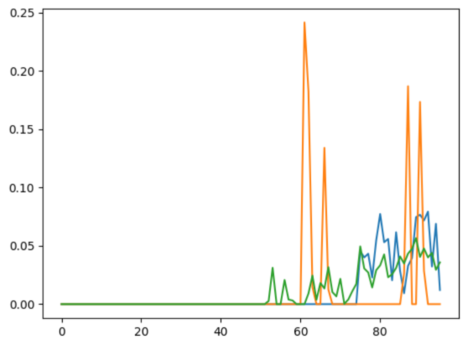
(a)



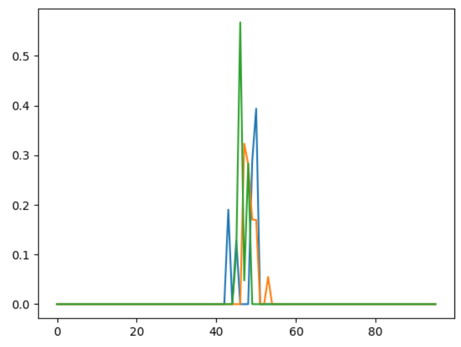
(b)



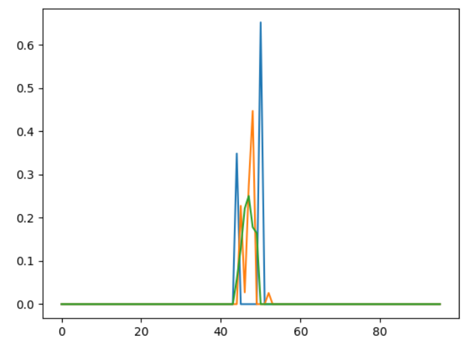
(c)



(d)



(e)



(f)

Figure 7.6: **Random rays profiles for chair, lego and ficus objects trained with and without topological regularization after 50 epochs**. Here (a) and (b) display chair object, (c) and (d) - lego and (e) and (f) - ficus. Rays from the topologically regularized model (a, c, e) exhibit lower variance and offer greater certainty regarding the true location of the object compared to the original *NeRF* model (b, d, f) for all objects except for ficus.

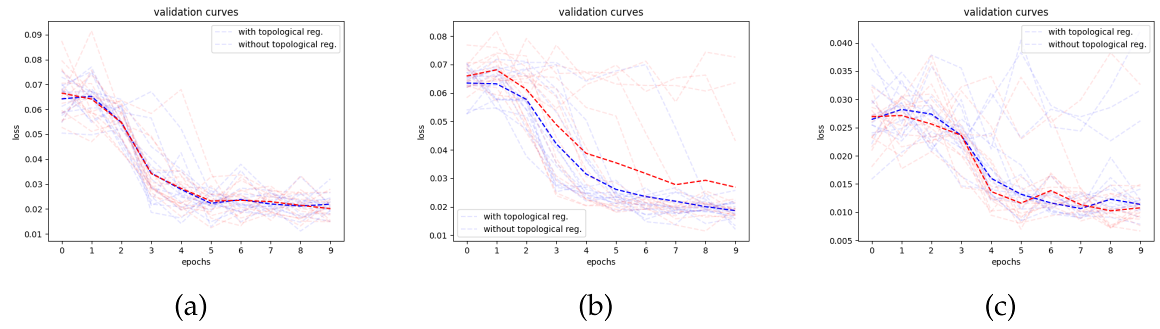


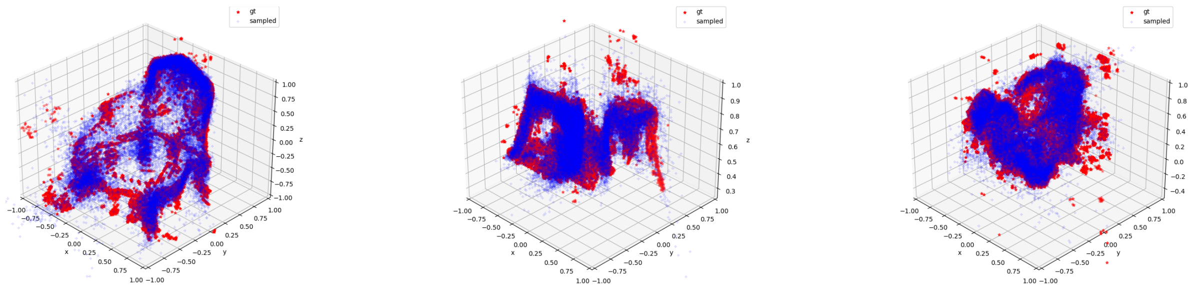
Figure 7.7: **Validation curves for 10 epochs for chair, lefo and ficus objects with median curve highlighted, number of model runs - 15.** For each of the objects - chair (a), lego (b) and ficus (c) it is clearly visible that addition of regularization is not harmful and sometimes beneficial in low-resolution setting

For this purpose, separate models were trained for the *chair*, *lego*, and *ficus* objects. During the training phase, the *NeRF* model visits multiple positions in 3D space to accurately capture the underlying structure of the object. The original *NeRF* model must iterate over the same positions multiple times to account for dependencies on the view angle. During these iterations, the model calculates the density profile of points in the space. *Normalizing flows* explicitly models the probability density function (PDF) of the dataset under certain assumptions about the model architecture.[26] For *RealNVP*, these assumptions are met, making it feasible to use the density values produced by this model.[27]

Although some studies [36] suggest training a *conditional normalizing flows* model in an end-to-end fashion with *NeRF*, it was chosen to simplify the process to avoid shifting from point estimation to probabilistic modeling. Instead, the two models were trained stepwise by iterating between the *NeRF* and *normalizing flows* models.

The *normalizing flows* model was further enhanced with *Wasserstein regularization*, as described in Section 6.4. This regularization demonstrated fast convergence to the true support of the distribution and did not require extended iterations, unlike the *GMM normalizing flow*. The neural radiance field (*NeRF*) model was initially trained for five epochs to warm up, after which training alternated between the two models, with five epochs allocated to *NeRF* and ten epochs to the *normalizing flows* model. This schedule allowed the probabilistic model to adapt more quickly to changes in the 3D space distribution.

At each iteration, training of the *NeRF* model started after the first epochs of the *normalizing flows* model had elapsed. The radiance field model then continued training using the density profile generated by the generative model as an additional



(a) (b) (c)

Figure 7.8: ***Nf* model samples and locations sampled from NeRF after 10 iterations of training.** Original Data (red) and Samples (blue) from upper hemisphere showing that for chair (a), lego (b) and ficus (c) NF model managed to learn distribution successfully

regularization term. KL divergence was applied as a regularization term to quantify the discrepancy between distributions. To collect samples from the trained *NeRF* model, initial certainty threshold was set to 0.6 and annealed to 0.8 over the course of training, iterating over ten directions randomly sampled from the unit sphere. Sample learned point clouds in Figure 7.8 illustrate that *normalizing flows* model was able to capture distribution in 3D space effectively.

This approach effectively accelerated training. As shown in Figure 7.9, the generative model converged after 200 epochs to the true support of the object, leaving the fine representation of the scene to the *NeRF* model while handling the coarse structure. However, maintaining two models concurrently proved resource-intensive and the approach was not studied further. Additionally, the need to perform sampling from the entire sphere along which the camera moves to construct complete training data for the *normalizing flows* model posed another significant challenge.

7.4 Topological NeRF with Transformer Architecture

Finally, our last model, described in Section 5.2.5, was trained and benchmarked against the three aforementioned objects at a resolution of 256×256. This time, the architecture was switched to an attention-based model [32], with *Betti Curve regularization* applied to the attention weight matrix. This modification aimed to encourage the model to focus on local information while exploiting information sharing between rays. This approach bears some similarity to [29], where the authors employed spatial encoding to encoder vicinity of each point. The model is further referred as *TopoNeRF*

Since the attention-based mechanism requires longer training, the regularization

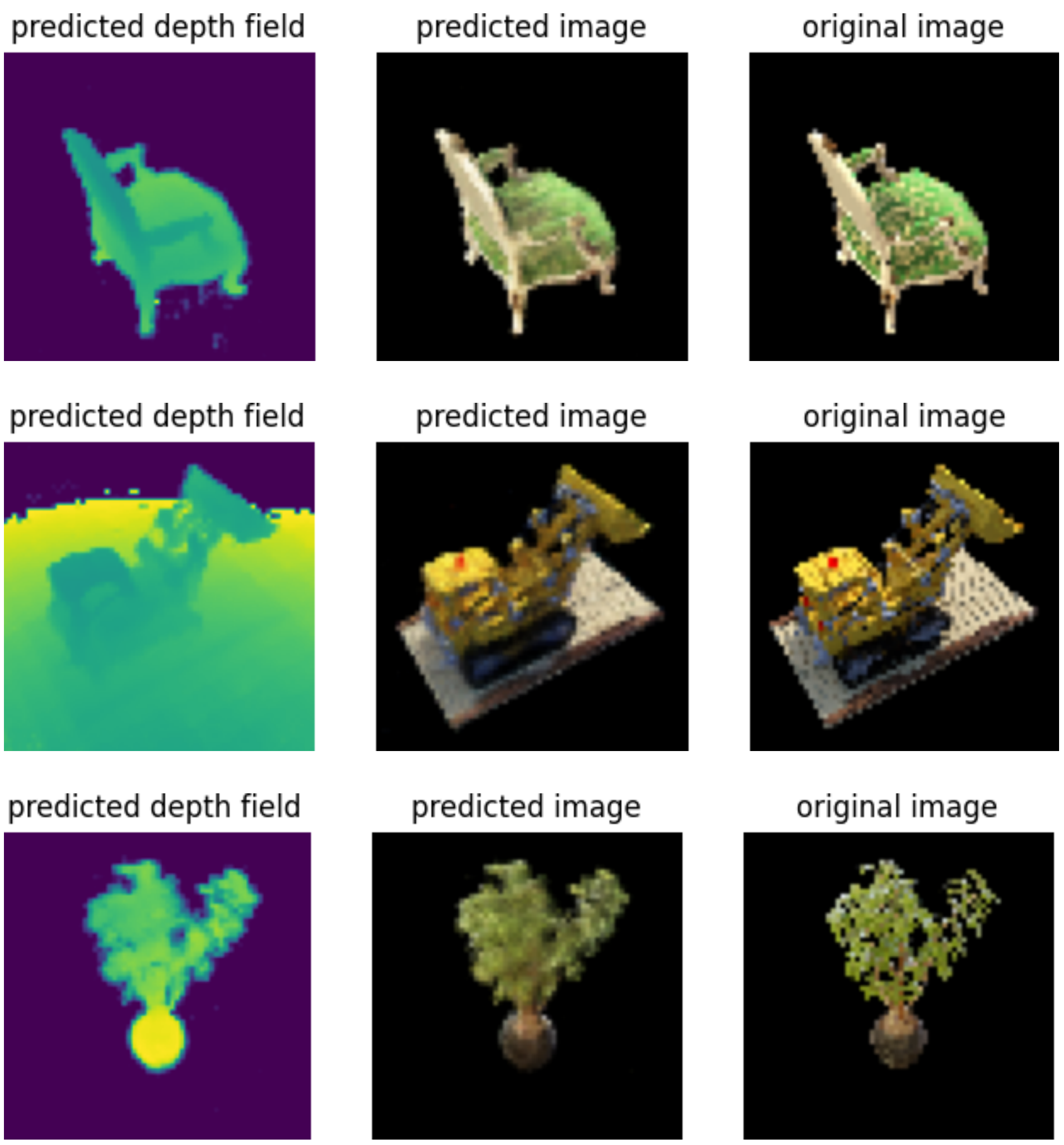


Figure 7.9: **Topological NF + NeRF results for chair, lego and ficus objects after 50 epochs.** Images show few artifacts at predicted depth map for chair, lego and ficus, reconstructions are still somewhat blurry, however at better quality than when supplied with *Betti Curve regularization*

strength of the *Betti Curve AUC regularization* term was gradually increased from 0 to 10^{-2}. Without this adjustment, the model would get stuck in a basin, consistently predicting zeros. Training was conducted for 200 epochs, revealing slower iterations and a flatter learning curve. Despite this convergence behavior - primarily attributed to finer changes in color faster shape convergence was observed, similar to the results seen in the previous experiment. These findings are supported by Figure 7.10.

Our approach demonstrated that changing the architecture to a more robust and expressive attention-based model, supplemented with topological supervision, outperformed our efforts to enhance the MLP-based architecture. Fine details related to color were better captured by the topological *NeRF* compared to its original counterpart. Supporting metrics, including PSNR (see 3.3.1) and SSIM (see 3.3.2), are listed in Table 7.1. Notably, the total running time of our model differed from that of the original by less than one hour, with both architectures struggling to learn finer details. This limitation is also inherent to the classic *NeRF* architecture.

7.5 Analysis and Bechmarking

In this section, the models described in this chapter are benchmarked, compared, and discussed. The classic *NeRF* model demonstrated good experimental convergence but was hindered by slow training and the need for a sufficient number of views to accurately capture fine details of a 3D object. Several studies have proposed extensions to the original architecture to address these limitations and enable training with fewer views.[37, 38]

In our experiments, the classic *NeRF* equipped with *Betti Curve regularization* applied to rays, as described in section 7.2, produced promising results when applied to low-resolution data. Validation curves were flatter for some objects, particularly those with complex topology, and the depth fields of the resulting reconstructions were less noisy. The ray profiles also exhibited improved behavior, with densities being more tightly packed, indicating that the model allocated less probability mass to empty regions.

However, the model struggled to replicate these results on higher-resolution samples. This limitation was likely due to lower-dimensional images containing a greater proportion of information described by individual pixels, making topological regularization more effective in such cases. Despite this fact our approach can still be applied to real world problems. Specifically, biomedical images are usually captured at low resolution, which might be a perfect setting for our model.

Our second model, described in Section 7.3, was trained in a stepwise manner involving two stages. In the first stage, *normalizing flows* equipped with topological regularization generated density profiles, which were subsequently used by the

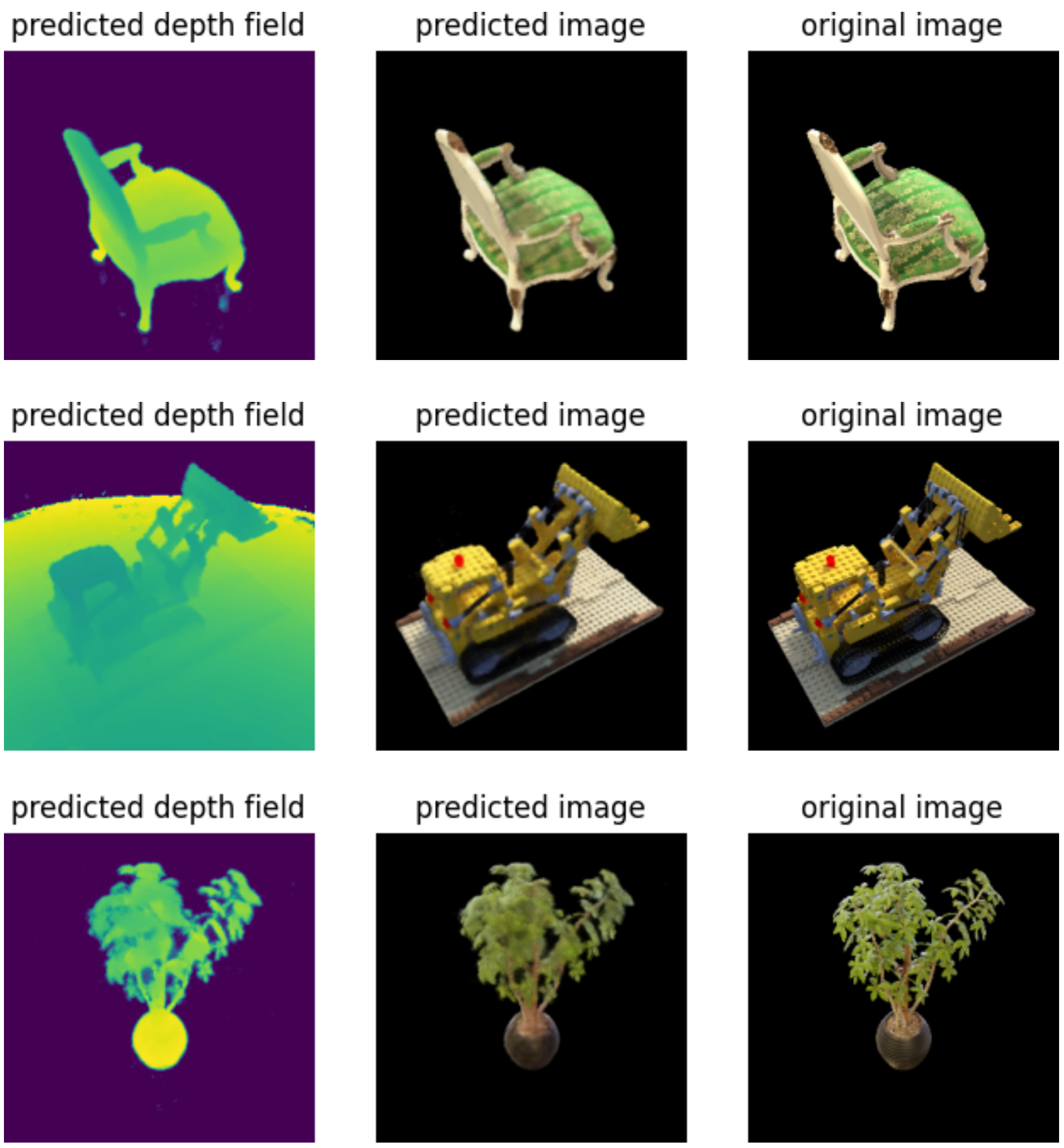


Figure 7.10: **Topological NeRF Reconstruction after 200 epochs.** It is visible, that fine details are captured better than in case of original model, trained the same number of epochs

neural radiance fields (*NeRF*) model in the second phase. This study was conducted on lower-dimensional images, where the model exhibited faster convergence to the true shape of the object.

However, when applied to full-dimensional images, the convergence time increased significantly and our model design was no more reasonable for this setting. Despite this limitation, the stepwise approach outperformed the previous method described in Section 7.2.

Our final attempt appeared to be comparable to the original model in terms of speed of convergence. Unlike classic *NeRF* however shape information was captured at much more earlier step. Therefore our novel architecture managed to successfully unify topological machine learning principles with novel neural radiance fields model. As might be observed from Table 7.1 our model outperformed original architecture by all metrics.

Model	MSE	PSNR	SSIM
NeRF	0.008	25.13	0.85
TopoNeRF	**0.002**	**27.18**	**0.87**

Table 7.1: **Training results for NeRF and TopoNeRF trained for 200 epochs and averaged on 3 different objects.** All of the provided metrics suggest improvement over the original *NeRF* architecture

8 Conclusion

In this work we developed a novel architecture called *Topological NeRF* based on the original *NeRF* model ideas and additionally supplied with topological supervision, which enjoyed accelerated shape convergence in comparison to the classic model.

For this purpose, we referred to a popular persistence diagram descriptor, namely *Betti Curves*, providing a simplistic yet expressive and intuitive way to directly embed knowledge of topology into different models. Due to absence of public implementation allowing for end-to-end gradient flow, we set off to write pytorch modules from scratch, resulting in vectorized and fully differentiable implementation, which might be used beyond this thesis. To build intuition behind this tool we run experiments outlining the effect of *Betti Curve AUC regularization*.

Further on we conducted studies covering landscapes of probabilistic generative modeling, specifically normalizing flows model, and notably *topological autoencoders*. Our research showed, how versatile architectures could be benefit from addition of topological supervision and how future researchers might enhance performance of this models by means of additional topological regularization.

In order to approach *NeRF* from topological prospective, we started with equipping original architecture with notion of topology by imposing regularization on rays profiles. This however introduced additional overhead and did not contribute to overall convergence of the model. Despite this fact, our model demonstrated aceceler-ated convergence in a low-resolution setting when supplemented with topological regularization.

On the basis of the results of the experiments, we outlined further strategy on approaching *NeRF* from topological point of view. For this purpose we trained a separate model merging neural radiance fields with normalizing flows and training them in conjunction in order to accelerate convergence of the *NeRF* model. Even though the model experienced increased convergence rate, it was too cumbersome to train two models causing longer iteration time. This motivated us to introduce our novel variant of *NeRF* based on *Transformer* architecture and additionally reinforced with *Betti Curve regularization* on top of attention weights. With the help of this model we finally experienced faster shape convergence with cost per iteration not too higher than that of original architecture.

We believe that *NeRFs* can still be improved by endowing them with even finer topological information. Our findings in this work may serve as a solid foundation for understanding where future studies might take off, while also contributing to publicly available codebase by implementing *Betti Curve transformation* and thereupon-based metrics and losses. Microscopy research for instance, often operating on low resolution images might utilize *NeRFs* with Betti Curve regularization, while complex scene capturing at high resolution will benefit from *NeRF Topological*. To conclude, *NeRFs* are capable of being supplied with topological information and we expect this trends to grow in the future studies.

List of Figures

List of Tables

Bibliography

[1] R. Forman. “Morse Theory for Cell Complexes”. In: *Advances in Mathematics* 134 (1998), pp. 90–145. URL: https://api.semanticscholar.org/CorpusID:59477712.

[2] S. Barannikov. “The framed Morse complex and its invariants”. In: *Advances in Soviet Mathematics* 21 (1994), pp. 93–116.

[3] H. Edelsbrunner, D. Letscher, and A. Zomorodian. “Topological Persistence and Simplification”. In: *Discrete & Computational Geometry* 28 (2000), pp. 511–533. URL: https://api.semanticscholar.org/CorpusID:9191014.

[4] D. J. E. Waibel, S. Atwell, M. Meier, C. Marr, and B. Rieck. “Capturing Shape Information with Multi-scale Topological Loss Terms for 3D Reconstruction”. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*. Springer Nature Switzerland, 2022, pp. 150–159. ISBN: 9783031164408. DOI: 10.1007/978-3-031-16440-8_15. URL: http://dx.doi.org/10.1007/978-3-031-16440-8_15.

[5] B. A. Rieck. “On the Expressivity of Persistent Homology in Graph Learning”. In: *ArXiv* abs/2302.09826 (2023). URL: https://api.semanticscholar.org/CorpusID:257038607.

[6] L. O’Bray, B. A. Rieck, and K. M. Borgwardt. “Filtration Curves for Graph Representation”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (2021). URL: https://api.semanticscholar.org/CorpusID:236980271.

[7] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. arXiv: 2003.08934 [cs.CV]. URL: https://arxiv.org/abs/2003.08934.

[8] J. J. Park, P. R. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove. “DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 165–174. URL: https://api.semanticscholar.org/CorpusID:58007025.

[9] M. Niemeyer, L. M. Mescheder, M. Oechsle, and A. Geiger. "Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 3501–3512. URL: `https://api.semanticscholar.org/CorpusID:209376368`.

[10] R. Martin-Brualla, N. Radwan, M. S. M. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth. "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 7206–7215. URL: `https://api.semanticscholar.org/CorpusID:220968781`.

[11] O. Gordon, O. Avrahami, and D. Lischinski. "Blended-NeRF: Zero-Shot Object Generation and Blending in Existing Neural Radiance Fields". In: *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)* (2023), pp. 2933–2943. URL: `https://api.semanticscholar.org/CorpusID:259224726`.

[12] M. Moor, M. Horn, B. A. Rieck, and K. M. Borgwardt. "Topological Autoencoders". In: *ArXiv* abs/1906.00722 (2019). URL: `https://api.semanticscholar.org/CorpusID:173990640`.

[13] H. Xiao, K. Rasul, and R. Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *ArXiv* abs/1708.07747 (2017). URL: `https://api.semanticscholar.org/CorpusID:702279`.

[14] U. Bauer, M. Kerber, F. Roll, and A. Rolle. "A unified view on the functorial nerve theorem and its variations". In: *Expositiones Mathematicae* (2022). URL: `https://api.semanticscholar.org/CorpusID:247291819`.

[15] A. Hatcher. *Algebraic topology*. Cambridge University Press, 2002., 2005.

[16] E. Čech. "Théorie générale de l'homologie dans un espace quelconque". In: *Fundamenta Mathematicae* 19 (1932), pp. 149–183.

[17] A. Choudhary, M. Kerber, and R. Sharathkumar. "Approximate Cech Complex in Low and High Dimensions". In: *International Symposium on Algorithms and Computation*. 2013. URL: `https://api.semanticscholar.org/CorpusID:5770506`.

[18] L. Vietoris. "Über den höheren Zusammenhang kompakter Räume und eine Klasse von zusammenhangstreuen Abbildungen". In: *Mathematische Annalen* 97 (1927), pp. 454–472. URL: `http://eudml.org/doc/182647`.

[19] J. B. Kruskal. "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis". In: *Psychometrika* 29 (1964), pp. 1–27. URL: `https://api.semanticscholar.org/CorpusID:48165675`.

[20] L. McInnes and J. Healy. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: *ArXiv* abs/1802.03426 (2018). URL: `https://api.semanticscholar.org/CorpusID:3641284`.

[21] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. "Stability of Persistence Diagrams". In: *Discrete & Computational Geometry* 37 (2005), pp. 103–120. URL: `https://api.semanticscholar.org/CorpusID:10864618`.

[22] B. A. Rieck, F. Sadlo, and H. Leitte. "Topological Machine Learning with Persistence Indicator Functions". In: *ArXiv* abs/1907.13496 (2019). URL: `https://api.semanticscholar.org/CorpusID:199000949`.

[23] A. Zomorodian. "Technical Section: Fast construction of the Vietoris-Rips complex". In: *Computers & Graphics* 34 (2010), pp. 263–271. URL: `https://api.semanticscholar.org/CorpusID:1941821`.

[24] P. Izmailov, P. Kirichenko, M. Finzi, and A. G. Wilson. "Semi-Supervised Learning with Normalizing Flows". In: *International Conference on Machine Learning*. 2019. URL: `https://api.semanticscholar.org/CorpusID:209516199`.

[25] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *CoRR* abs/1312.6114 (2013). URL: `https://api.semanticscholar.org/CorpusID:216078090`.

[26] D. J. Rezende and S. Mohamed. In: *ArXiv* abs/1505.05770 (2015). URL: `https://api.semanticscholar.org/CorpusID:12554042`.

[27] L. Dinh, J. Sohl-Dickstein, and S. Bengio. *Density estimation using Real NVP*. 2017. arXiv: `1605.08803 [cs.LG]`. URL: `https://arxiv.org/abs/1605.08803`.

[28] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: `1612.00593 [cs.CV]`. URL: `https://arxiv.org/abs/1612.00593`.

[29] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. "Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields". In: *ICCV* (2021).

[30] K. Park, U. Sinha, P. Hedman, J. T. Barron, S. Bouaziz, D. B. Goldman, R. Martin-Brualla, and S. M. Seitz. "HyperNeRF". In: *ACM Transactions on Graphics (TOG)* 40 (2021), pp. 1–12. URL: `https://api.semanticscholar.org/CorpusID:235624060`.

[31] G. Tauzin, U. Lupo, L. Tunstall, J. B. Pérez, M. Caorsi, A. Medina-Mardones, A. Dassatti, and K. Hess. *giotto-tda: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration*. 2020. arXiv: `2004.02551 [cs.LG]`.

[32] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is All you Need". In: *Neural Information Processing Systems*. 2017. URL: `https://api.semanticscholar.org/CorpusID:13756489`.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *North American Chapter of the Association for Computational Linguistics*. 2019. URL: `https://api.semanticscholar.org/CorpusID:52967399`.

[34] A. G. Hawkes. "Spectra of some self-exciting and mutually exciting point processes". In: *Biometrika* 58 (1971), pp. 83–90. URL: `https://api.semanticscholar.org/CorpusID:14122089`.

[35] R. Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the royal statistical society series b-methodological* 58 (1996), pp. 267–288. URL: `https://api.semanticscholar.org/CorpusID:16162039`.

[36] J. Shen, A. Agudo, F. Moreno-Noguer, and A. Ruiz. *Conditional-Flow NeRF: Accurate 3D Modelling with Reliable Uncertainty Quantification*. 2022. arXiv: `2203.10192 [cs.CV]`. URL: `https://arxiv.org/abs/2203.10192`.

[37] P. Wang, Y. Liu, Z. Chen, L. Liu, Z. Liu, T. Komura, C. Theobalt, and W. Wang. "F2-NeRF: Fast Neural Radiance Field Training with Free Camera Trajectories". In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2023), pp. 4150–4159. URL: `https://api.semanticscholar.org/CorpusID:257771805`.

[38] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su. "TensoRF: Tensorial Radiance Fields". In: *European Conference on Computer Vision (ECCV)*. 2022.