POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Advanced Deep Learning Methods for Anomaly Detection in Point Clouds

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Stefano Bruno Gusmeroli**

Student ID: 944781
Advisor: Prof. Giacomo Boracchi
Co-advisors: Luca Frittoli
Academic Year: 2021-22

A
Coralli

Abstract

Thanks to the increased diffusion of sensors such as LiDARs, in recent years we have witnessed the rise of a peculiar type of data: point clouds. The reasons behind such success are to be found in their nature; in fact, a point cloud is constituted by a set of points to which can be associated some features. From this, it follows that they represent a compact and very convenient way to depict the surface of a 3D object. More precisely, it is possible to sample a point cloud from a three dimensional surface, hence, in so doing, the spatial arrangement of the obtained points can be used to describe the original shape. In consideration of the above, there has been a flourishing literature of deep learning methods, with many models that have been proposed for the classification and the segmentation task. However, to our concern, we have noticed that the anomaly detection is a much more disregarded field. In this work, we aim therefore to address the unsupervised anomaly detection task on point clouds. To do so, we have developed two different methods. In particular, we propose an extension of the Deep Robust One Class Classification method [36] by adopting a suitable network and introducing an adapted version of the original loss. Besides it, we propose a model for point cloud anomaly detection inspired by NeuTraL [73] and composed of a pre-trained feature extractor, a set of learnable transformations, and an encoder. We have evaluated the performance of both of our methods by conducting several experiments on the ShapeNet [17] and ModelNet [95] datasets. Our DROCC method for point cloud anomaly detection proved to be on par with some other similar ones. Furthermore, our Neural Transformation Learning model achieved impressive performance on both datasets, outperforming every other considered method by a large margin. On some classes it even approaches the perfect classifier, making register values of AUC astoundingly close to 1.

Keywords: point cloud, unsupervised, anomaly detection, 3D anomaly detection, one class classification, novelty detection

Abstract in lingua italiana

Grazie allo sviluppo di sensori come i LiDAR, negli ultimi anni abbiamo assistito a una crescente diffusione di un particolare tipo di dati: le Point Cloud. Le ragioni dietro a tale successo sono da ricercare nella loro natura. Infatti, esse sono costituite da un insieme di punti ai quali possono essere associate delle features, come ad esempio i colori. Per questo motivo esse rappresentano un modo molto compatto ed efficace di rappresentare la superficie di un oggetto 3D. Più precisamente, è possibile campionare dei punti da una superficie tridimensionale e le coordinate dei punti ottenuti possono essere impiegate per descriverne la forma. In conseguenza di ciò, si è sviluppata una fiorente letteratura di metodi di deep learning, con innumerevoli modelli che sono stati proposti per trattare il tema della classificazione e della segmentazione. Tuttavia, abbiamo notato che l'ambito del rilevamento della anomalie è stato molto meno considerato. In questo lavoro ci siamo quindi posti l'obiettivo di affrontare il problema dell'Unsupervised Anomaly Detection per point clouds. A questo fine abbiamo sviluppato due diversi metodi. Il primo si tratta di una estensione dell'approccio Deep Robust One Class Classification [36] nel quale abbiamo adottato una rete apposita e abbiamo introdotto una versione modificata della loss originale. Oltre a ciò, abbiamo proposto un modello per l'anomaly detection per point cloud che trae ispirazione da NeuTraL [73] e che è composto da un feature extractor pre-addestrato, un insieme di trasformazioni apprendibili e da un encoder. Al fine di valutare le prestazioni dei nostri metodi abbiamo condotto svariati esperimenti usando come datasets ShapeNet [17] e ModelNet [95]. Il nostro metodo DROCC per point cloud si è dimostrato essere sullo stesso livello di altri metodi simili. D'altro canto, il nostro modello di Neural Transformation Learning ha raggiunto dei risultati impressionanti su entrambi i dataset, migliorando considerevolmente l'attuale stato dell'arte. Su alcune classi ha addirittura fatto registrare dei valori di AUC estremamente vicini a 1, rasentando così il classificatore perfetto.

Parole chiave: point cloud, dati 3D, rilevamento delle anomalie, rilevamento degli outlier

Contents

Introduction

Over the past few years, we have witnessed a great development in the technology behind sensors like LiDARs and depth cameras, thanks to which they have become cheaper, easier to be implemented, and available to a greater public. Just to give an idea, nowadays these kinds of sensors can be found in almost every assisted-driving car and even in the top-of-the-range smartphones. All of the above has led to a huge increment in the availability of a new type of data: point clouds.

As the name suggests, a point cloud is a set of points distributed in a certain space, in such a way that altogether usually depicts a surface. We can regard point clouds to be composed of two elements: a set of coordinates corresponding to the points in $\mathbb{R}^d$ and some features associated to each of them. For sake of generalisation, we can consider a function φ which associates to each point its features. These features can represent various characteristics, the most common ones are colours and the direction of the normal vector to the surface of the object at that point, but they can represent even temperatures and other information.

The case in which the points live in a three-dimensional space is probably the most important and researched one. This is because they can be used to describe a 3D object in a very convenient way, having to store in memory only the coordinates and the features of the points. It is worth noting that point clouds do not necessarily have to be 3D data, as they can live in any finite-dimensional real vector space. There are examples of two-dimensional point clouds, which can be imagined as some points scattered on a plane. The reasons behind why the latter case is less regarded have to do with the existence of images, for which there is already a variegated amount of methods available. Another motivation can be found in the unordered nature of point clouds, which makes them harder to be processed. Indeed, the main difference between images and point clouds is that the former are arranged on a regular grid and so the pixels can be identified simply by a pair of indices (i, j), whereas the latter have to be described by coordinates.

Thanks to their capability of depicting a 3D object in a very compact way, in recent years, point clouds have found application in a wide variety of fields, ranging from computer

vision [38], robotics [68] and autonomous driving [57], [20], [51]. A multitude of methods have been proposed to process them, with most of them focusing on the multi-class classification or on the segmentation tasks [69], [71], [56], [13], [60].

This being established, we have noticed that one research area on point clouds has been much less regarded: **anomaly detection**. In fact, up to this date, only a few works address this task on point clouds and, on top of that, many of them propose methods that tackle very specific issues. Therefore these latter are highly tailored to their scope, such as pole-like objects [75], scans of urban areas [4] or additive manufacturing [59]. To the best of our knowledge, only a handful of methods address the anomaly detection problem on point clouds in a more general setting [63], [31], [32].

In light of the above, the objective of our work is to design and explore methods that are suitable to the anomaly detection task when the input is presented in the form of a point cloud. We focus on deep learning models, since these techniques have been proven to be very successful in many areas, from image processing to speech recognition [54], [80].

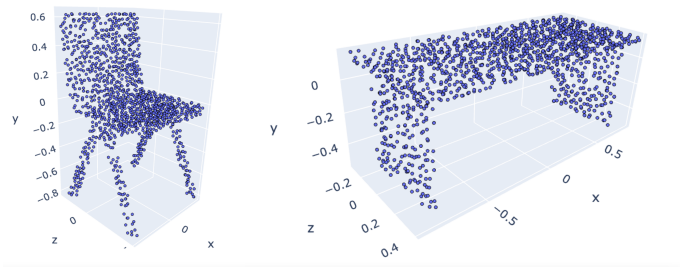
More precisely, in this work we present two different methods that address the anomaly detection problem on point clouds. The first of them is an adaptation of the Deep Robust One Class Classification approach proposed by [36], whereas the second is inspired by the Neural Transformation Learning method of [73]. It has to be noted that neither [36] nor [73] take into consideration anomaly detection on point clouds. Indeed, the former is originally applied to images and tabular data, whilst the latter mainly focuses on tabular data and time series.

In order to assess the effectiveness of our models, we perform several experiments and we evaluate their performance on two well established datasets in the point cloud deep learning community: ShapeNet [17] and ModelNet40 [95]. We adopt the unsupervised anomaly detection approach and the *1 Vs. All* setup, as in [78], [36], [34], [73], [63], [31], [32]. This means that we train our models using exclusively instances belonging to one class, which is considered the *"normal class"*. After that, at evaluation time, the model is presented with test data from the whole dataset and its task is to detect which samples are anomalous, i.e. do not belong to the class on which it was trained. In order to investigate the results that the method achieves on the whole dataset, we select one class at the time, we regard it as the normal class, we train the model using solely those data and finally we test its performance using the entire testset. The aforementioned process is reiterated for each class of the dataset. This way of proceeding is analogous to the one followed by [63], [31] and [32], thus it enables us to compare the results of our models with theirs.

Outline

For the sake of clarity, here we report a brief outline of this work:

- In **Chapter 1** we state more rigorously the problem that we aim to address, formally defining the input and introducing the notation that we intend to adopt. Moreover, we discuss the requirements that the methods we research have to fulfil.
- **Chapter 2** is devoted to conveying some background on the topics that are related to our work. More precisely, in section 2.1 we present some deep learning models that process point clouds, among which the composite layers [32]. Moreover, in section 2.2, we explain several approaches to the anomaly detection task in various fields. Finally, in section 2.3 we examine the current methods that perform the aforementioned task on point clouds.
- In **Chapter 3** we describe our proposed solutions by introducing, in section 3.1, our adapted version of the DROCC approach [36] to point clouds. Furthermore, in section 3.2 we present our Neural Transformation Learning model for anomaly detection on point clouds inspired by [73].
- **Chapter 4** is dedicated to the experiments that we have performed in order to assess the performance of our previously introduced models. We first illustrate the datasets and the evaluation metric that we employ. After a brief overview of the competing methods, in section 4.4 we explore the anomaly detection capabilities of our DROCC method for point cloud. In section 4.5 we investigate the performance of our Neural Transformation Learning model in various scenarios.
- **Chapter 5** concludes this work. In it, we make some final remarks and we discuss some potential future developments.



(a) Chair (b) Table

Figure 1: Examples of point clouds

1 | Problem Formulation

In this chapter, we provide a precise statement of the problem that we address and we characterise the input, namely 3D point clouds, by illustrating its salient properties and the main challenges that arise when working with it.

1.1. Definition of point cloud

In this work, we address the problem of unsupervised anomaly detection in 3D point clouds. We employ a point cloud to describe a three dimensional object because it allows a compact and convenient representation, as it requires storing in memory only a set of coordinates and of features. The aim is to design a model which, given a 3D point cloud representing an object, is able to distinguish whether the element is normal or anomalous.

More rigorously, we define a point cloud $\mathcal{P}$ as a pair (P, φ), where $P = \{x_i\}_{i=1}^n$ is a set of points in a d-dimensional Euclidean space $\mathbb{R}^d$. Since we focus on three dimensional objects, the space in which the points live corresponds to $\mathbb{R}^3$, namely we have that $d = 3$. All the metric spaces that we consider are endowed with the Euclidean distance $d_2 = \|\cdot\|_2$, which is therefore the one that we refer to when talking about "distances" between the elements.

The function φ is such that $\varphi : P \to \mathbb{R}^I$ and its purpose is to associate, to the coordinates of each point x, the features $\varphi(x)$, hence $\varphi : x \mapsto \varphi(x)$. The vector of the features has dimension I, namely it is the case that $\varphi(x) \in \mathbb{R}^I$. Each feature is a characteristic related to the point x, such as the intensity of the three RGB channels, the temperature, or the direction of the normal vector to the surface of the object at x.

Mathematically speaking, we regard a point cloud as a set in which it is not present a canonical order [1]. From this, it follows that one of the main challenges of processing a point cloud is the lack of a specific order among its elements.

[1] Although in principle it could be argued that every set can be well ordered [99], we do not intend to do so since every order would result to be meaningless to our purposes [80]. This is because small variations in the coordinates may significantly alter the order of the points and affect consequently the method as two almost identical point clouds might yield extremely different results

1.2. Anomaly detection on point clouds

In a rather general way, an anomaly can be defined as "an observation that significantly deviates from a certain concept of normality" [16]. Depending on the field of competence, it can also go under the names of "outlier" or "novelty". The task of anomaly detection therefore consists in identifying the data that do not conform to the undergoing definition of normality. Although this may seem at first glance a rather easy task, especially compared to multi-class classification, its nature is much more insidious and it poses several subtle challenges.

We consider the framework of unsupervised anomaly detection, which means that the training samples are not labelled. Therefore an anomaly detection algorithm must be trained using solely instances belonging to the normal class [16], [77], [32], [34], [63]. Exclusively at evaluation time the model is presented with anomalous elements, with the purpose of performance assessment.

The objective is to develop a method that, when it is presented with an element, is able to discriminate on whether it belongs to the normal class or not. The output of the model is therefore a score, which goes under the name of *anomaly score*, indicating the magnitude of anomalousness or, conversely, of normality. To this purpose, we formally introduce the *anomaly score function*, which is a function $\mathcal{A}_S : \mathbb{R}^{|P|\times(d+I)} \to \mathbb{R}$; $\mathcal{A}_S : \mathcal{P} \mapsto \mathcal{A}_S(\mathcal{P})$ that associates to each point cloud $\mathcal{P}$ one real value in $\mathbb{R}$ such that high values of $\mathcal{A}_S(\,\cdot\,)$ indicate that $\mathcal{P}$ is anomalous, whereas low values of $\mathcal{A}_S(\,\cdot\,)$ indicate that $\mathcal{P}$ belongs to the normal class. Based on the values of the anomaly score function, it can be then chosen a threshold in order to classify each element $\mathcal{P}$ as either normal or anomalous. Therefore to each point cloud $\mathcal{P}$ will be assigned an integer representing the designated class, for instance "1" if it is an anomaly and "0" otherwise. This can be seen in practice as constructing a binary classifier.

1.3. Properties required

All the methods that consume point clouds are called at fulfilling some requirements that arise from the nature of this type of input. To better understand the challenges that working on point clouds poses, we shall explore more in detail the main three characteristics that these methods must have, which are:

1) Invariance w.r.t. permutations:
As we have mentioned in Sec. 1.1, a point cloud is an unordered set. From this it follows that every method that works on point clouds needs to be invariant under $n!$ permutations,

where n is the cardinality of the set P. This property is of particular concern since storing a point cloud in memory implicitly assigns an order to the points, for instance when they are arranged in a Pytorch's tensor. Furthermore, the most important layers of a neural network, such as fully connected and convolutional ones, are influenced by the order of the input. Indeed, changing the order of the elements passed as input to these layers can give rise to very different outputs. From this observation it follows that the whole architecture needs to be carefully designed in order to result invariant with respect to the order in which the input points are presented to the network.

2) Invariance w.r.t. some rigid transformations:
Some rigid transformations do not alter the nature of a shape, therefore it is expected that the method is not affected by them. For instance, it is usually required that the model is invariant with respect to rigid translations and in many scenarios also to rigid rotations along the vertical axis. To give an example, in an object classification task, a table should always be classified as such, no matter how far from the origin it is positioned or its orientation.

3) Interaction among points:
The points live in a metric space $(\mathbb{R}^d, \|\cdot\|^2)$ in which there is a well defined notion of distance and of neighbourhood. The neighbouring points constitute a meaningful subset that carries important geometrical information for the learning process. Therefore the methods ought to leverage this aspect and to be able to capture and aggregate the information contained in the local structures of the shape.

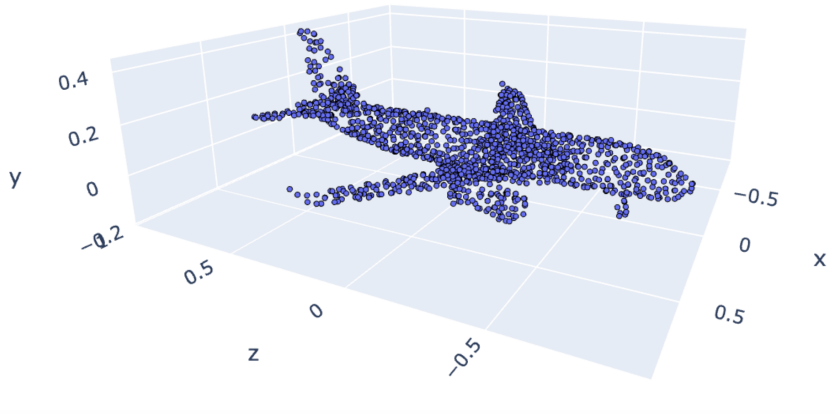


Figure 1.1: Point cloud representing an airplane

2 | Related Works

In this chapter we provide an overview of the history of point clouds processing in the deep learning era, exploring the methods that can be considered the milestones and highlighting their main contributions as well as their drawbacks. After that, we present the different approaches to the anomaly detection task, briefly explaining the directions that have been pursued in this field. Lastly, we connect these first two topics by analysing some methods that currently address the problem of anomaly detection on point cloud data.

2.1. Deep Learning on Point Clouds

As mentioned in section 1.3 and better explained in the Appendix A *The matter of the order*, mathematically speaking a point cloud has the structure of a set in which there is not a canonical order. This property is of particular importance for methods that process point clouds since they are required to be invariant with respect to the permutations of the points, fact that is not generally granted by algorithms in the Deep Learning field.

The first methods have tried to exploit the architecture of classical convolutional neural networks by extending it to other types of data that can be derived from point clouds. Among these works we recall Multi-view CNNs [89] and Volumetric CNNs [64] that use as input images and volumes, respectively. These earliest methods have approached the inconvenience given by the unordered nature of point clouds by circumventing it and trying to resort to what was already consolidated at that time, namely convolutions on a regularly structured domain. **Multi-view CNN** [89] relies on generating 2D views of an object and then it treats them as images by applying a more traditional 2D Convolutional Neural Network. On the other side, **Volumetric CNNs** [64], [80] work instead on a volume and they apply a three dimensional convolutional filter. Both these approaches, namely Multiview CNN and Volumetric CNN, require a pre-processing step, which consists in either projecting the input to 2D views or constructing a voxel grid. This necessary process has two main drawbacks: the first one is that it can cause loss of information, for instance due to occlusion and/or low resolution; whilst the second is represented by the high number of hyper-parameters to be chosen. As a matter of fact, the number and

position of viewpoints, the rendering parameters, the number and dimension of voxels are all hyper-parameters to be set, choices that may not be obvious in several scenarios. The above complications can be ideally avoided by developing methods that process directly point clouds.

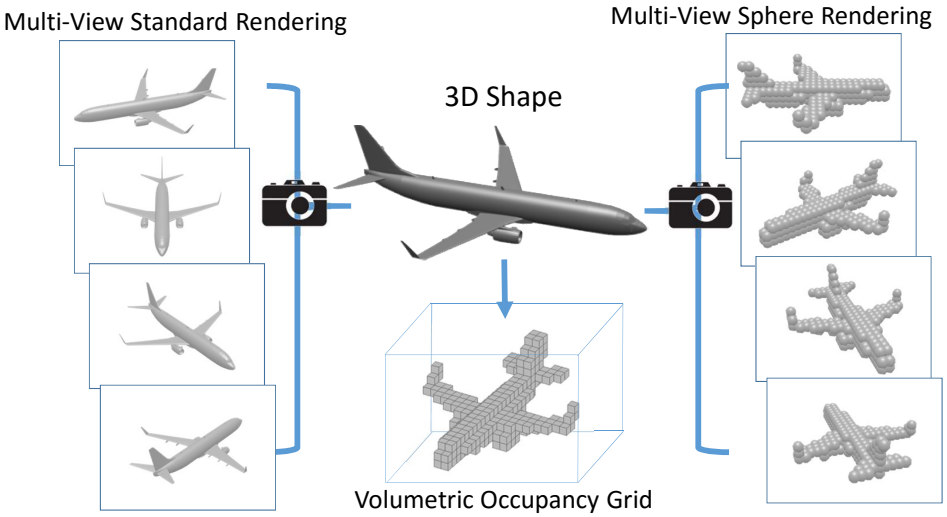


Figure 2.1: 3D shape representations, image from [70]

It is with PointNet [69] that it is introduced a method able to consume directly point clouds. Although it proved to be rather effective in the classification task, this method lacks in capturing local geometries. For this reason, the same authors developed a new method, called PointNet++ [71] which applies PointNet hierarchically.
Since then, there have been developed many other methods that work directly on point clouds and the main researched direction has been trying to emulate the functioning of a convolutional neural network. Among these methods we can cite PointCNN [56], ConvPoint [13], KPConv [93], PointMLP [60], and many more.

2.1.1. Background on methods that process point clouds

As we have explained in section 1.3, all the methods that work on point clouds are required to be *1. invariant w.r.t. permutations*, *2. invariant w.r.t. some rigid transformations* and to exploit the *3. interaction among points*.

Concerning the "invariance w.r.t. some rigid transformations" property, in order to make a model invariant w.r.t. rigid translations, it suffices to centre the shape in the origin by subtracting to each point the mean. In addition to this, it is also possible to rescale the point cloud, for instance by normalising the coordinates with respect to their standard

deviation or by making them fit in a unit ball. The rescale process is usually performed when we believe that the size of an object is not of crucial importance or worse, when it can raise confusion. In most of the methods that we consider they are applied both centring and rescaling [69], [71], [13], [32].

To address the "interaction among points" property, it is considered quite naturally a system of neighbourhoods. There are two main ways of defining these neighbourhoods: KNN and Ball query.

The "invariance w.r.t. permutations" is however probably the most insidious aspect. To deal with the fact that a point cloud is an unordered set there are three main strategies: sort the input in a canonical order; train a Recurrent Neural Network on the augmented data with all the possible permutations; using a symmetric function. The latter is by far the most common approach, as the former two can be very problematic [80]. In Appendix A *The matter of the order* we cover these aspects in deeper detail. All the methods that we consider in this work adopt the third strategy, although they resort to different symmetric functions.

KNN and Ball Query

Many methods that process point clouds rely on the creation of a collection of neighbourhoods from which to extract the relevant information. There are two main approaches to define these neighbourhoods: KNN and Ball Query.

KNN stands for K-Nearest-Neighbours (KNN) and, given a centre point x, it consists in considering as neighbourhood of x its K closest neighbouring points (in the sense of the d_2 distance). On the other side, Ball Query consists in setting a radius r and simply taking a collection of balls of radius r centred at each point. Both methods have their pros and cons. The KNN approach is the one that raises fewer problems, as it allows points to be processed in batches, an aspect that eases the training process of the network. This is due to the fact that each neighbourhood is guaranteed to have K elements. Furthermore, there is no need to choose a radius, contrary to what happens in Ball Query, where the radius is an additional hyper-parameter not always easy to be set.

Ball Query, on the other hand, generates neighbourhoods of a fixed scale and thus it may ease the task of the local feature extractor as regions have a certain coherence [80]. However, we have that a different number of points can fall into each ball. This makes it non-obvious how to aggregate the input in batches and so points usually need to be processed separately, which slows down the training.

2.1.2. Consuming directly point clouds

As already mentioned, point clouds allow for a simple and compact representation of 3D data since they are made only of a set of coordinates and features. Methods that work directly on point clouds are able to make use of this characteristic and in this way, they avoid the possible artefacts that can arise during the quantisation step [80].
On the other side, they are required to be invariant under permutations of the elements of the set and also they need to be not affected by specific rigid transformations, such as translations.

At the end of 2016, Charles R. Qi et al. proposed what has become a pioneering method in this direction: PointNet [69]. First of all, this is a network that consumes directly point clouds and therefore it is not subject to the occlusion and the artefacts that may arise during the pre-processing step required for Multi-View and Volumetric CNNs. Second, it uses a symmetric function to summarise the information from each point, in a way that makes it invariant with respect to the order of the points. Moreover, especially in its "vanilla" version, it's a very lightweight architecture that doesn't require as many resources and doesn't have as many parameters as MVCNN [89] or VoxNet [64].

Nevertheless, PointNet has several flaws, the most important of them is the lack of ability to capture context at different scales. Since the whole shape is used altogether to create the global descriptor, PointNet does not examine different scales of the object nor takes them into account. This also leads to a limited ability in recognising fine-grained patterns, which are overshadowed by the main structure of the object. Another consequence is the low generalisability to complex scenes, which might make it unsuitable for certain applications where precision is key.

To mend the shortcomings of PointNet, Charles R. Qi proposed an evolution of this previous work: PointNet++ [71]. The main idea behind it is to create a cover of the point set and then extract a spacial encoding from each subset by means of a local feature learner. In this way it is obtained a new set of points, one for each subset, to each of which is associated a new descriptor that acts as a feature vector. This process is reiterated gradually reducing the cardinality of the set, until only one single point is left.
To deal with point clouds of non-uniform point density, two different density adaptive layers have been proposed [71]. This allows PointNet++ to be able to aggregate information at various scales and to be robust with respect to variations in the points' density.

2.1.3. PointNet

The core of PointNet comprehends a Multi Layer Perceptron $h(\cdot)$ of output dimension 1024 that is applied independently to every single point of the shape. After this, all the resulting n vectors are arranged in a matrix of dimensions $n \times 1024$, to which a Max Pooling operator is applied. The Max Pooling operator simply takes, for each of the 1024 columns, the maximum value and it returns as global shape descriptor a vector of dimension 1024. This descriptor is then processed by another MLP named $\gamma(\cdot)$ whose task is to reduce the dimension from 1024 to k. In the case of a classification problem, k corresponds to the number of different classes.

As shown in the image 2.2, in the complete version there are two additional mini-networks, called T-Net, whose purpose is to predict an affine transformation matrix. The idea behind them is to try to align the shapes in few canonical poses so that all the objects of a certain class may have a common orientation. This allows to ease the feature extraction process for the last part of the structure, for example in a classification task.
A ReLU activation function and batch normalisation are used for all layers except for the output one and dropout is used before the last layer [69].

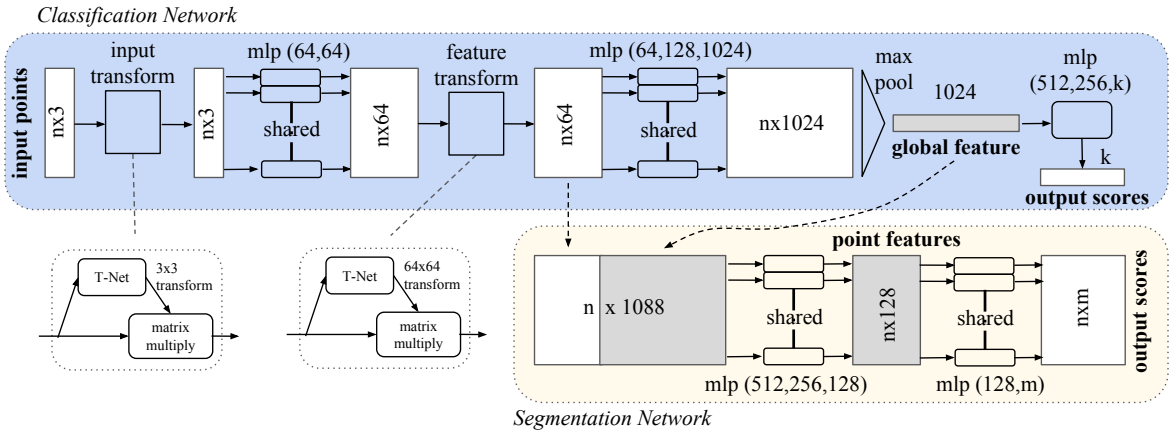


Figure 2.2: PointNet architecture; in the blue box the classification network, below the additional part of the segmentation network. Image from [69]

PointNet [69] acted as a watershed, being the first neural network to consume directly point clouds in an effective and efficient way. It achieved the state of the art without being so operationally demanding as a MVCNN [89] and without requiring so much memory as a Volumetric CNN [64]. Its architecture makes it scalable with respect to the number of points and robust to various types of input corruption.

2.1.4. PointNet++

To overcome the scarce ability of PointNet to capture local structures, C. R. Qi [71] proposed an evolution that applies PointNet hierarchically on a nested partitioning of the point cloud. As it is conceptually in continuation with the previous work, this new network was given the name of PointNet++.

PointNet++ has a hierarchical design in which each stage takes as input a set of points with the associated features and returns a new set of points with new features. Each of these stages has been called *"set abstraction level"* since it processes and performs an abstraction of the input set of points to produce a new one with fewer elements [71]. A set abstraction level is made of three elements: a *Sampling layer*, which samples a subset from the input point cloud using Furthest Point Sampling (FPS) algorithm; a *Grouping layer* that creates a collection of neighbourhoods via Ball Query and a *PointNet Layer* that applies a PointNet [69] to each one of the constructed neighbourhoods, extracting in this way a local descriptor vector.

On account of the above, PointNet++ [71] evolves the structure of PointNet [69] and it mends one of its most critical shortcomings: capturing context at different scales. Thanks to its hierarchical structure it is indeed able to take into account and to aggregate features at several scales. In this way it is capable of exploiting also fine-grained patterns, which can be beneficial in many tasks. This hierarchical architecture is probably the most important contribution introduced by PointNet++.

2.1.5. Convolution on point clouds

One of the most prominent directions in point cloud deep learning has been trying to extend and apply the operating principles of a convolutional neural network to this type of data. The key point is how to define the convolution for point clouds and the associated kernel.
In image Convolutional Neural Networks, the kernel is a relatively small matrix of parameters that is learned during the training process. For each channel, the kernel usually consists of a square matrix, for instance of dimensions 5×5. These parameters are then used to compute a weighted sum of the features, in what is the convolutional formula. The pixels are arranged on a regular grid and so it comes natural selecting a patch of the image with the same dimensions of the filter and performing the Hadamard product between the two. All the resulting values are then summed and in this way are obtained the new features that will be of the pixel at the patch's centre.

A general way of defining a convolution for a set of points X_y can be:

$$\psi_j(y) = (\varphi * g_j)(y) = \sum_{x \in X_y} \sum_{i=1}^{I} \varphi_i(x) g_{ij}(x - y) \tag{2.1}$$

where $\varphi_i(x)$ is the i^{th} component of the input feature vector $\varphi(x)$ associated to x and $\psi_j(y)$ is the j^{th} component of the output features $\psi(y)$ of y. X_y is a set of points and it constitutes the convolutional window.
In this equation, the vector function $g_j(x-y)$ acts as the filter and it is deputed to process the spatial information contained in X_y. It takes as input the relative positions of the points of X_y with respect to the centroid of the neighbourhood, namely y. This is done to guarantee the invariance of each component $g_{ij}(\cdot)$ with respect to rigid translations applied to the input points. In addition, $g_{ij}(\cdot)$ is applied independently to the coordinates of a single point, which makes it invariant also to the input's order.
Similarly to what happens in a classic convolutional layer, also in this case it is possible having several filters $g_j : \mathbb{R}^d \to \mathbb{R}^I$ such that their number is equal to J. To each filter g_j it corresponds one output feature ψ_j and as a result the output feature vector $\psi(y)$ has dimension J. Each filter is a function defined over the whole space $\mathbb{R}^d$. Furthermore, every filter g_j has I components, one for each input feature φ_i.

Following this intuition, there have been several proposals of point convolutional formulas, among which we recall ConvPoint [13], PA-Conv [96], KP-Conv [93], CompositeNet [32], PointMLP [60], just to mention a few.
In the forthcoming section we explain more in detail the functioning of ConvPoint [13], as it represents an excellent example of convolution for point clouds. After that, we present the work of Floris at al: CompositeNet [32]. We devote particular attention to the composite layers [32] since they are functional to our purposes.

2.1.6. ConvPoint

ConvPoint [13] introduces a convolutional layer for set of points that resembles in basically every aspect the ones used in image processing. As we have remarked, the kernel of a convolutional layer that processes images has the same format of an image. Inspired by this principle, the author [13] decided to make the kernel $\mathcal{K}$ of ConvPoint resemble the input. It is therefore chosen to be explicitly a set of points with some weights associated to them.

According to the notation that we have adopted so far, the convolutional formula defined

by ConvPoint [13] can be written in the form:

$$\psi_j(y) = \sum_{x \in X_y} \sum_{i=1}^{I} \varphi_i(x) \sum_{m=1}^{M} \tilde{w}_{ijm} \gamma_m(x - y) \tag{2.2}$$

in which we have that:

$$g_{ij}(x - y) = \sum_{m=1}^{M} \tilde{w}_{ijm} \gamma_m(x - y)$$

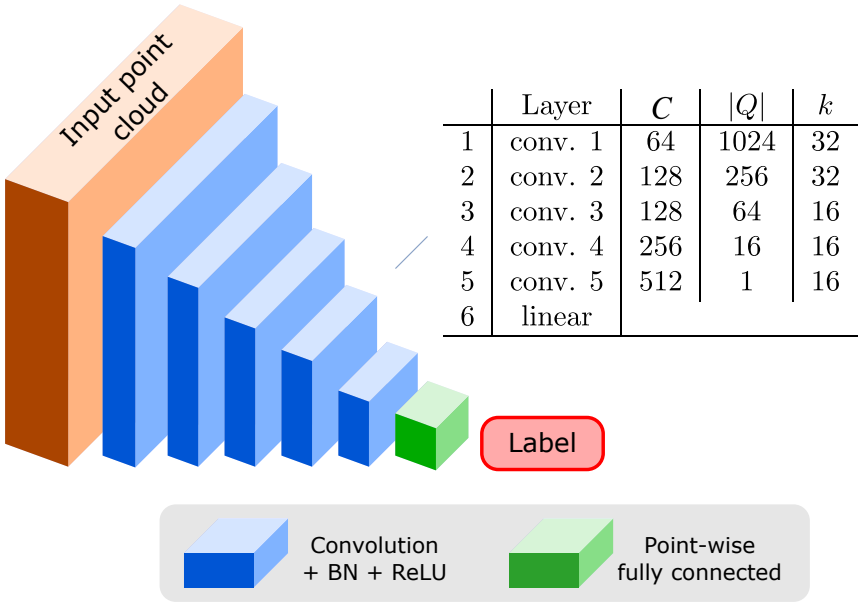
where $\tilde{w}_{ijm}$ are the learnable weights and γ_m are the weighting functions. These latter actually depend on $(x - y) - c_m$, which are the differences between the relative position of the centred input points and the elements c_m of the kernel $\mathcal{K} = \big(\{c_m\}_{m=1}^{M},\ \xi \big)$.

Obviously $\gamma_m(\,\cdot\,)$ must be invariant with respect to permutations of the input points, in a similar way that the whole network has to be. To fulfil this requirement, the author has decided to apply independently $\gamma_m(\,\cdot\,)$ to each point x [13].
In addition, $\gamma_m(\,\cdot\,)$ needs to be invariant also to rigid translations applied to both the kernel and the input point cloud. In order to achieve this, A. Boulch opted to use only the relative coordinates of the input points with respect to the kernel's elements [13]. In other words, $\gamma_m(\,\cdot\,)$ is applied to the difference $\{x - c_m;\ for\ m = 1, ..., M\}$ which is the set of relative positions of x with respect to every element of the kernel.

One thing that has to be noted is that the spatial coordinates are not considered as input features, unlike what happens for PointNet [69] and PointNet++ [71]. In this formulation, the coordinates are solely processed by the function $\gamma_m(\,\cdot\,)$ which is in charge of establishing a relation between the ones of the input and the ones of the kernel. The features, on the other side, are multiplied together and weighted using the values returned by $\gamma_m(\,\cdot\,)$. In ConvPoint [13] we observe a clear separation between the role of the coordinates and the role of the features as they are not processed altogether. The resulting expression (2.2) is much more similar to a discrete convolution than the methods that we have previously analysed.
Similarly to what happens in a discrete convolutional layer, also a ConvPoint's convolutional layer can be made of several kernels. Instead of having just one kernel, it is indeed possible choosing many of them; each one will be made of a set of points with its own coordinates and features. In this way we can have several channels, one for each kernel, and the output will consist of a vector [13].

In the role of weighting function $\gamma(\,\cdot\,)$, the author [13] has chosen to directly learn a function by means of a Multi-Later Perceptron. In such a way, no specific assumptions on the behaviour of $\gamma(\,\cdot\,)$ need to be made. As far as it regards the kernel, the author

has decided to initialise the coordinates $\{c_m\}_m$ of its points by randomly sampling them in a unit sphere [13]. During training they are then treated as parameters and learned accordingly. All the parameters, including the coordinates $\{c_m\}_m$ and the features ξ of the kernels, are optimised using Stochastic Gradient Descent [103].



	Layer	C	$\lvert Q\rvert$	k
1	conv. 1	64	1024	32
2	conv. 2	128	256	32
3	conv. 3	128	64	16
4	conv. 4	256	16	16
5	conv. 5	512	1	16
6	linear			

Figure 2.3: Architecture of the ConvPoint network used for classification; from [13]

To compute the set of output points of a layer, it is adopted a non-uniform sampling of the input points, which is carried out by penalising the elements that have already been drawn once. To each point it is associated a score, which is inversely proportional to the probability for said point to be selected. At the beginning, each point has a score of 1 and every time a point y is drawn its score is incremented by 100. This makes it less likely that the same point will be chosen again, diminishing the chances of a sub-optimal result [13]. In addition, the scores of the neighbouring points of y are increased by 1 as well, which is beneficial to ensure better coverage. The points are iteratively drawn according to this protocol until the desired number is reached. As grouping method it is employed KNN, since it yields neighbourhoods of the same cardinality that can be processed more efficiently in mini-batches.

An example of ConvPoint [13] architecture is reported in Fig. 2.3, it is composed of five convolutional layers and of a fully connected one. The convolutional layers take as input the shape and progressively reduce the number of points, while increasing the number of channels. The last convolutional layer reduces the point cloud to a single element, which

can then be fed to the fully connected layer. The architecture resembles the ones that are employed in image processing, such as LeNet [55]. Batch normalisation and ReLU activation function have been used after each layer but the last one.

One of the main remarks to be made about the so defined convolutional layer is that it is agnostic to the object scale. All the point clouds are normalised in a unit ball and therefore the information about the scale of different objects is lost. This can either be problematic or not so much, depending on the application. In some cases, such as photogrammetric point clouds, the scales may not be even available. However, in metric scans, the scale can be valuable information for easing the task, as some shapes (for instance humans) tend to have almost always similar dimensions. Nevertheless, as reported by A. Boulch, removing the normalisation step would cause the kernel and the input points to have different volumes [13].

This formulation is flexible and computationally efficient [13] and can be used to create various network architectures for different tasks. In addition, it is robust with respect to both the input and the neighbourhood size [13]. In essence, ConvPoint is an excellent example of how the concept of convolution can be extended also to point clouds.

2.1.7. Other convolutional methods

Since the publication of ConvPoint [13], there have been many other proposals of convolutional layers that are able to process point clouds. The basic principles tend to be the same and to resemble the ones that we have presented, although each network retains its own peculiarities.

Similarly to ConvPoint, also the authors of KP-Conv [93] have used as kernel a set of points with some associated weights, however they consider them to be regularly arranged and their position to be rigid. They adopt Ball Query to construct the convolutional windows instead of KNN and they employ a linear correlation as weighting function. In addition, they introduce a deformable version of their kernels, in which some shifts are learnt during training.

With CompositeNet [32], the authors have introduced two variants of *composite layers*: the first, that is the *"convolutional composite layer"*, defines a new convolutional formula for point clouds, whereas the second one, called *"aggregate composite layer"*, relies on a nonlinear aggregation approach for combining the features and the spatial information.

In PA-Conv [96] the core idea is to "construct the convolution kernel by dynamically assembling basic weight matrices stored in a Weight Bank, where the coefficients of these

weight matrices are self-adaptively learned from point positions through ScoreNet".
PointMLP [60] introduces a "lightweight local geometric affine module that adaptively transforms the point features in a region" and it employs a residual MLP framework.
This said, there are countless more methods that process point clouds in a convolutional manner, however, due to practical reasons, we cannot consider nor mention them all. Therefore we have decided to focus our attention on ConvPoint [13], which we have just presented, and on CompositeNet [32], which we illustrate next.

2.1.8. CompositeNet

In CompositeNet [32] it is introduced a new convolutional layer composed of two elements: a spatial function s and a semantic function f. The former is in charge of extracting the information from the spatial arrangement of the input points by processing their coordinates. After that, the latter is responsible for combining the spatial information learned by s with the features associated with the points.
Thanks to this design the two functions can be customised independently and the complexity of each one can be adjusted without severe repercussions on the other. This allows a notable flexibility in terms of form and number of parameters. Moreover the function s acts like an additional regularisation term by extracting and compressing the spatial information.

To illustrate the design flexibility of the composite layer, the authors have provided two distinct variants that combine the geometrical information and the features in two different ways. A first one is called *"convolutional composite layer"* and it is an innovative approach to formulating point convolution. In addition to this, it is also presented the *"aggregate composite layer"*, which implements a nonlinear method to combine the spatial information and the features. Thus it represents a novelty compared to most of the existing works, which only combine them in a linear fashion [32].

Composite layers

The input of a composite layer is a point cloud (P, φ) and the output returned by it is another point cloud (Q, ψ). Following the same logic, we have that Q is a set of points in $\mathbb{R}^d$, whereas $\psi(\cdot)$ is the function that associates to them the new features. The vector of the coordinates of a point in Q is named y and the dimension of an output feature vector is J, so we have that $\psi(y) \in \mathbb{R}^J$ and $\psi : Q \to \mathbb{R}^J$. As a matter of fact, y goes also under the name of centroid, while its relative neighbourhood is called X_y. The neighbourhood X_y represents the convolutional window.

Point-convolutional operator

Recalling (2.1), the convolution for a set of points X_y can be defined as:

$$\psi_j(y) = (\varphi * g_j)(y) = \sum_{x \in X_y} \sum_{i=1}^{I} \varphi_i(x) g_{ij}(x - y)$$

where $g_j : \mathbb{R}^d \to \mathbb{R}^I \quad j = 1, ..., J$ are the filter functions and to each g_j it corresponds one output feature ψ_j. The vector of the output features $\psi(y)$ has therefore dimension J. Also in this case each filter g_j is a function defined over the whole space $\mathbb{R}^d$ and it has I components, one for each input feature φ_i.

In the light of the above, the formula of the point convolution performed by a composite layer [32] can be written, at a very general level, as:

$$\psi_j(y) = f_j(\varphi, s)(y) \tag{2.3}$$

from this expression we can see that, as previously mentioned, the layer is the composition of a semantic function $f(\,\cdot, \cdot\,)$ which combines the geometrical information extracted by $s(\,\cdot\,)$ with the input features. The semantic function $s(\,\cdot\,)$ takes as input the coordinates of the points in a set X_y, and thus it is a function $s : \mathbb{R}^d \to \mathbb{R}^K$ for some $K \in \mathbb{N}$. Subsequently the output of $s(\,\cdot\,)$ is processed by the semantic function $f(\,\cdot, \cdot\,)$, alongside the set of input features $\{\varphi(x) : x \in X_y\}$, to generate the vector of output features $\psi(y)$.

In the composite layers, both the spatial function and the semantic one can be customised independently, without the other being severely affected by this. This flexibility in the design, especially in terms of the number of parameters, can be of crucial importance when the datasets are small, where the risk of overfitting is high [32].

Sampling method

Each composite layer receives as input the set of points P and it returns as output another set of points Q, which has therefore to be computed. The authors [32] have decided to obtain the coordinates of these output points by following a similar approach to the one adopted in ConvPoint [13]. In other words, the coordinates of the output points are obtained by randomly sampling the set P using a penalisation strategy, until the desired number of points is drawn. This method is computationally lighter than FPS, albeit still being able to offer a good coverage of the input point cloud.

Grouping method

Analogously to ConvPoint [13], also a composite layer relies on a collection of neighbourhoods $\{X_y\}_{y \in Q}$ to define the convolutional windows. Of the two mainstream approaches, the authors have decided to opt for KNN [32]. In this way, all the subsets are constructed to contain the same number of points, fact that allows the input to be processed in mini-batches. Thanks to this, the optimisation process is more efficient.

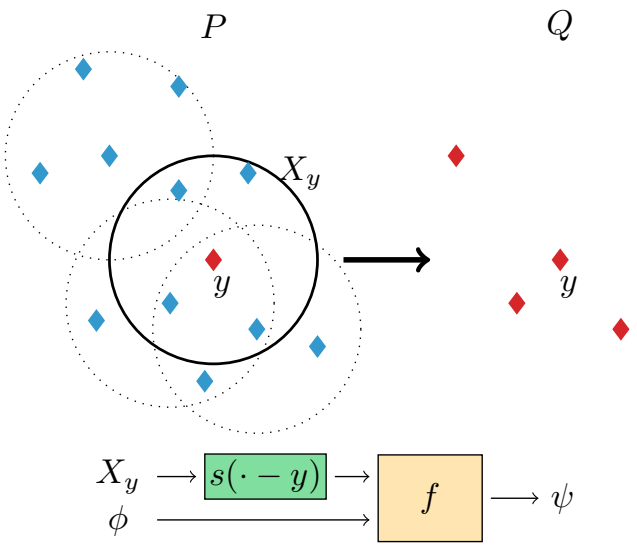


Figure 2.4: Scheme of the functioning of a Composite layer; on the left there are the input points P grouped in subsets, on the right the output points Q. Below, it is presented the depiction of the operations performed by the spatial function s, which takes as input X_y, and by the semantic function f that receives as input the output of s and the features φ. Image from [32]

2.1.9. Convolutional Composite Layer

The two main components of a convolutional composite layer [32] are the spatial function s and the semantic function f. In this section, we investigate further in detail how these elements are implemented.

Spatial Function

The authors, inspired by previous works [93], [5], opted to use a Radial Basis Function Network (RBFN) with M centers as spatial function s. This choice comes from the fact that a RBFN has the same approximation capability of a Multi-Layer Perceptron [65],

but still, it relies on fewer hyper-parameters.

The function s is vector-valued and has output dimension K, which controls the expressiveness of the spatial information. On the same line of thought of ConvPoint [13], the points in a neighbourhood X_y are firstly centred by subtracting to them the coordinates of the centroid y. This way of proceeding, we remind, allows the function to be invariant to rigid translations of the input points. After that, it is taken their relative positions with respect to the elements of the kernel, as to make the functions dependent only on this difference. To each kernel, there are associated M centers c_m which are points in $\mathbb{R}^d$ as well. The result is that each component s_k receives as input the vector $(x - y) - c_m$ and it is defined as:

$$s_k(x - y) = \sum_{m=1}^{M} v_{km} h(\|(x - y) - c_m\|) \tag{2.4}$$

where v_{km} are learnable weights $\forall k \in \{1, ..., K\}$, $\forall m \in \{1, ..., M\}$. As far as it regards the choice of the Radial Basis Function $h(\cdot)$, it is adopted a Gaussian function of the form:

$$h(\|(x - y) - c_m\|) = exp\left\{ - \frac{\|(x - y) - c_m\|^2}{2\sigma^2} \right\} \tag{2.5}$$

in which σ is a hyper-parameter common to all the radial basis functions [32].
Similarly to what happens with ConvPoint [13], the positions of the centers $\{c_m\}$ are parameters that are learned during training.

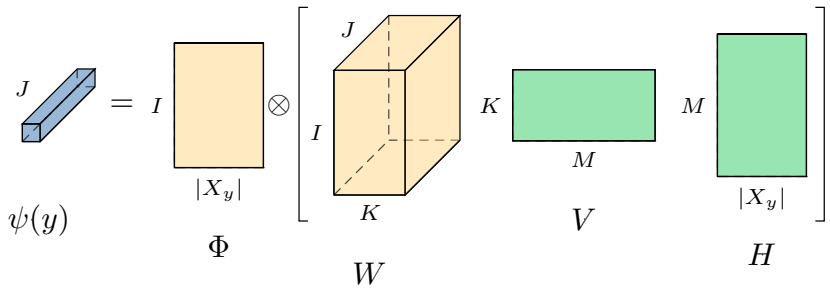
To better convey their reasoning, the authors have depicted these operations also in a matricial form [Img. 2.5 (a)]. In this scenario, the spatial function can be represented as the multiplication between a matrix $V = (v_{km})$ and a matrix H, which stores the output of the functions $h(\cdot)$. V is the matrix of the weights v_{km} and it has dimensions $K \times M$. On the other side, H contains the values $h(\|(x - y) - c_m\|)$ and it has M rows, each one corresponding to a center c_m, and $|X_y|$ columns, one for each point $x \in X_y$. Therefore H is of size $M \times |X_y|$ and so the multiplication is well defined, with the resulting product matrix VH having dimensions $K \times |X_y|$. Since it is the case that $K < M$, we have that the spatial function performs a compression of the information contained in the columns of H to a K-dimensional space [32].

Semantic function

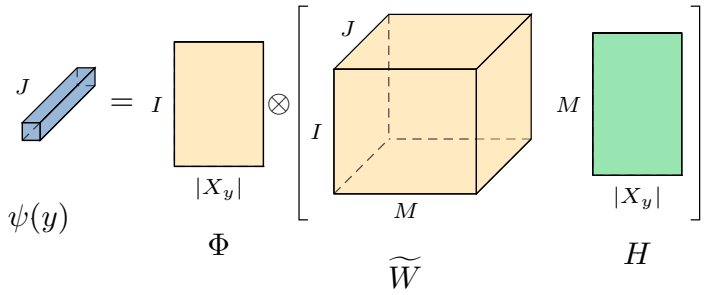
In this case, the authors [32] wanted the resulting operation to be in the form of a point convolution, therefore the semantic function f has been defined as:

$$\psi_j(y) = f_j(\varphi,\, s)(y) = \sum_{x \in X_y} \sum_{i=1}^{I} \varphi_i(x) \sum_{k=1}^{K} w_{ijk} s_k(x - y) \tag{2.6}$$

In this formula it can be recognised the product between the features of the points $x \in X_y$ within the convolutional window and the filter functions $g_{ij}(x - y)$. Indeed, by setting $g_{ij}(x - y) = \sum_{k=1}^{K} w_{ijk} s_k(x - y)$ we have that the above equation resembles the general formula (2.1) that we have introduced at the beginning of this chapter.
The weights w_{ijk} are learnable parameters $\forall i \in \{1, ..., I\}$, $\forall j \in \{1, ..., J\}$ and $\forall k \in \{1, ..., K\}$ corresponding to each input feature, output feature and component of the spatial function, respectively.



(a) Convolutional composite layer



(b) ConvPoint / KPConv

Figure 2.5: Illustration in a matricial form of the operations performed by point-convolutional layers; from [32]

From a matricial point of view, this operation can be expressed as the multiplication of three entities [Img. 2.5 (a)]. The matrix representing the filter functions $g_{ij}(x - y)$ is given by the product between the matrix VH and the tensor $W = (w_{ijk})$, which contains all the various weights w_{ijk}. We remember that VH stores the values of the spatial

function and it is itself the result of the multiplication between the weights $V = (v_{km})$ and H, as discussed in the previous subsection. The tensor W has size $I \times K \times J$ and the multiplication is performed along the dimension K, therefore the result is a tensor $W\ VH$ of size $I \times |X_y| \times J$. Ultimately, each component $\psi_j(y)$ of the output feature vector can be represented as the Frobenius inner product between the matrix Φ and the j-th slice of $W\ VH$. Φ contains the input features $\varphi_i(x)$, the vectors $\varphi(x)$ are arranged on the columns and so Φ is of dimensions $I \times |X_y|$. The final outcome is the vector $\psi(y)$ which has length J.

Comparison with ConvPoint

As explained above, the computations performed by a composite convolutional layer can be expressed as a product between Φ and $W(VH)$. The latter tensor comes from the multiplication of W, V and H, which have respectively sizes $I \times K \times J$, $K \times M$ and $M \times |X_y|$.
Following the same way of reasoning, we now investigate how the operations carried out by ConvPoint [13] would be represented. In ConvPoint it is implemented another example of point convolution, which can be taken as prototypical since the majority of point convolutional layers have a very similar structure. Studying it from a matricial point of view therefore allows us to better understand how a composite layer relates to the rest of the literature.
The equation that describes the point convolution of ConvPoint we recall to be:

$$\psi_j(y) = \sum_{x \in X_y} \sum_{i=1}^{I} \varphi_i(x) \sum_{m=1}^{M} \tilde{w}_{ijm} \gamma_m(x - y)$$

The values of the weighting functions $\gamma_m(x - y)$ give rise to a matrix H of size $M \times |X_y|$, since the kernel has M elements and the cardinality of each neighbourhood is $|X_y|$. In a similar way, the weights $\tilde{w}_{ijm}$ can be stored in a tensor $\tilde{W}$ which has dimensions $I \times M \times J$. The matrix H is of the same size of the one that is found in the formulation of the composite layer, so they are similar under this aspect. However, in ConvPoint [Img. 2.5 (b)] we have that H is multiplied directly by $\tilde{W}$, whereas in CompositeNet [Img. 2.5 (a)] V first multiplies H and then their product is multiplied by W. The matrix of the input features Φ is multiplied in the same way in both cases to the tensors $\tilde{W}$ and W.

Inspecting more carefully the differences displayed by the image 2.5, we have that $\tilde{W}$ from ConvPoint comprehends $I \cdot M \cdot J$ parameters, whilst in W and V are involved a total of $I \cdot K \cdot J + K \cdot M$ parameters. Since K is set to be lower than M $(K < M)$

[32], we have that in the latter case the total number of parameters can be significantly lower than the former. Moreover, in a convolutional composite layer it is possible to boost the complexity of the spacial function without causing an enormous increase in the total number of parameters. This is due to the fact that increasing the complexity of the spatial function translates into increasing the number of centers, namely M. In a composite layer this is done without having necessarily to change K and, consequently, the dimension of W [32]. On the contrary, doing such a thing for ConvPoint implies changing the size of the three dimensional tensor $\tilde{W}$, which results in many more parameters. On account of the above, in a composite layer the multiplication between W and H allows us to tune the expressiveness of the spatial and of the semantic function separately, as the two concepts are decoupled.

The authors suggest that it is possible improving the classification accuracy of CompositeNet [32] without having to considerably increase the number of parameters. Furthermore, they demonstrate that the number of learnable weights can be effectively reduced without jeopardising the performance [32].

2.1.10. Aggregate composite layer

In addition to the convolutional composite layers, the authors have implemented also a variant that combines the spatial information and the features in a non-linear manner [32]. The spatial function s has the same design of the one that we have presented when talking about the convolutional version in section 2.1.9, namely it is implemented by means of a Radial Basis Function Network. However, in this layer the semantic function f aggregates the geometrical information extracted by s with the input features φ by resorting to a non-linear operation. More into the details, given the output $s_k(x-y)$ of the semantic function for every $x \in X_y$, firstly they are defined the component-wise mean $\mathcal{M}_s(y)$ and the component-wise standard deviation $\mathcal{S}_s(y)$, namely:

$$\left[\mathcal{M}_s(y)\right]_k = \frac{1}{|X_y|} \sum_{x \in X_y} s_k(x-y)$$

$$\left[\mathcal{S}_s(y)\right]_k = \sqrt{\frac{1}{|X_y|-1} \sum_{x \in X_y} \left\{ s_k(x-y) - \left[\mathcal{M}_s(y)\right]_k \right\}^2}$$

We observe that $s(x-y)$ are vectors in $\mathbb{R}^K$, from which it follows that also $\mathcal{M}_s(y) \in \mathbb{R}^K$ and $\mathcal{S}_s(y) \in \mathbb{R}^K$.

In an analogous manner, they are computed the point-wise mean and the point-wise

standard deviation of the input feature vector $\varphi(x)$:

$$\big[\mathcal{M}_\varphi(y)\big]_k = \frac{1}{|X_y|}\sum_{x\in X_y}\varphi_k(x)$$
$$\big[\mathcal{S}_\varphi(y)\big]_k = \sqrt{\frac{1}{|X_y|-1}\sum_{x\in X_y}\big\{\varphi_k(x)-\big[\mathcal{M}_\varphi(y)\big]_k\big\}^2}$$

The input feature vectors have dimension I, as $\varphi(x) \in \mathbb{R}^I$, therefrom we have that $\mathcal{M}_\varphi(y) \in \mathbb{R}^I$ and $\mathcal{S}_\varphi(y) \in \mathbb{R}^I$ as well. The point-wise mean and standard deviation act like pooling operators that receive $|X_y|$ elements each and condense the contained information in only one vector of their same dimension.

Once in possession of the above components, the first two vectors are concatenated and so do the last two; this gives rise to:

$$\theta(y) = \big[\mathcal{M}_s(y),\, \mathcal{S}_s(y)\big] \qquad\qquad \eta(y) = \big[\mathcal{M}_\varphi(y),\, \mathcal{S}_\varphi(y)\big]$$

As a consequence we have that $\theta(y) \in \mathbb{R}^{2K}$ and $\eta(y) \in \mathbb{R}^{2I}$.

Finally, the semantic function f is defined as:

$$\psi_j(y) = f_j(\varphi,\, s)(y) = \theta(y)^T\, W_j\, \eta(y) = \sum_{i=1}^{2I}\sum_{k=1}^{2K}\theta_k(y)^T\, w_{ijk}\, \eta_i(y) \tag{2.7}$$

where W_j is a matrix of size $2K \times 2I$ and it is the j-th slice of the 3D tensor W that in this case has dimensions $2K \times J \times 2I$.
The concatenation of the two vectors guarantees that the semantic function is nonlinear. Moreover, by making the network learn from the mean and standard deviation, it is introduced an additional form of regularisation [32].

2.2. Anomaly Detection

In recent years, thanks to the development of deep learning approaches, it has been witnessed a renewed interest in the anomaly detection task.
Anomaly detection is a well established research area in statistics, with the first documented methods that date back to the 19^{th} century (such as Edgeworth 1887 [24]), although there is every reason to believe that it was already studied prior to that period. Since then several methods have been introduced to deal with this problem and the proposed solutions vary from density based models, reconstruction principles, generative models, decision boundaries, transformation based methods, and many more.

Anomaly detection methods have found application in a miscellaneous assortment of sectors, that varies from cybersecurity [62, 67], fraud detection, insurance, medicine [18, 87, 90], telecommunication [12, 102], quality control, fault and damage detection [74, 101], infrastructures monitoring [101] and many more [77].

2.2.1. The type of supervision

Broadly speaking, anomaly detection techniques can be divided into three categories, depending on the use of labelled data.
A first group is the one of Supervised methods, which require an extensive dataset of labeled data, both for the normal class and, more critically, for the anomalies. This approach is not much taken into consideration because of one major obstacle: the lack of such datasets. A second category is called Semi-supervised anomaly detection, in which it is assumed that only one portion of the data is labelled. The third class, and by far the most relevant and commonly studied, is the one of Unsupervised approaches, in which the data are unlabelled.

In most of real-world scenarios, anomalies are rare and can occur in variegated forms, therefore making it hard to construct an extensive dataset. It is the very nature of anomalies, namely the high heterogeneity and the unpredictability, that makes it difficult and expensive to characterise all notions of anomalousness. The latter are therefore failed to be captured in the training data, fact that often leads a supervised (or semi-supervised) method to perform poorly. It is even the case that supervised methods perform worse than a their supervised counterpart when trained on the same dataset [77]. The causes of this behaviour are once again to be found in the exposure of the supervised method only to a restricted variety of anomalies, which is not fully representative and thus makes the method ineffective. All of the above is the reason why the first two categories are

of less interest than the third one, which is consequently also the most investigated: **unsupervised anomaly detection**.

2.2.2. Classical methods

In the years numerous statistical approaches have been proposed. The traditional methods comprehend:

- density based methods, in which an observation is deemed anomalous if it is not generated by the stochastic model assumed [3], [86], [6], [9];
- spectral techniques, that map the data to a lower dimensional space where anomalies appear to be different from normal observations, such as Principal Component Analysis (PCA) [45], Kernel PCA [42], [83] and Robust PCA[43], [88]
- clustering based, which rely on the principle that normal instances belong to a cluster in the data, while anomalies do not, like k-Means [52], Kernel Density Estimator (KDE) [47], Gaussian Mixture Models (GMM) [53], DBSCAN [28], ROCK [37], SNN clustering [26], WaveCluster algorithm [84]
- Nearest Neighbour based, that check the distances between the k^{th} nearest neighbors [14], [27], [2], [100]

Other types of methods rely on a one-class classification approach, for instance, One-Class Support Vector Machine (OCSVM) [82] and Support Vector Data Description (SVDD) [92], which both learn a decision boundary in the data. In the last decade have been proposed also *ensemble methods*, that aim to explicitly isolate anomalies from the rest of normal data, such as Isolation Forest [58].

OC-SVM, which stands for One Class Support Vector Machine, is a well-established approach also outside the realm of anomaly detection. In essence, its objective is to find a separating hyperplane in some feature space. The feature space can in theory be the original input space $\mathcal{X}$, although, since the drawn decision boundary is linear, this is usually deemed to be not sufficiently expressive. A more suitable feature space can be represented by a RKHS, i.e. Reproducing Kernel Hilbert Space; in this case we talk about Kernel OC-SVM. Despite being still a shallow method, the fact that the hyperplane is sought after in a more complex feature space (often of infinite dimension) has as consequence that the decision boundary looks nonlinear when projected back into $\mathcal{X}$. In this way, it is possible obtaining a nonlinear decision boundary that more easily divides the normal data from the anomalies.

Another family of models is made of the ones inspired by the Support Vector Data Descriptor [92]. The objective of SVDD can be shown to be conceptually related to the one of OC-SVM [16], however in this case the boundary consists in a hyper-sphere that encompasses the normal class. The task they solve translates into finding the minimum enclosing volume of the normal class.
Likewise to OC-SVM, there exists also kernelised versions of SVDD [91], which consider a more elaborated feature space induced by a kernel and draw the decision boundary in it. Once again, when transferred back into the original input space, the boundary will no more have the shape of a perfect hyper-sphere and this allows a higher expressiveness.

2.2.3. The Deep Learning Era

With the advent of Deep Learning, many have tried to extend these techniques also to the field of Anomaly Detection. The great advantage of a deep learning based method is that it does not require the manual feature engineering process, which, especially for complex problems, can represent a burden. Conversely, Deep Learning allows an automated feature learning process, which in many areas has brought drastic improvements in performance, such as in image processing. The proposed approaches rely on different principles, among which we recount the density based ones [49], deep one class classifiers, reconstruction models [35], [81] and transformation learning [34].

2.2.4. Generative models

A first direction consists in modelling the distribution of the normal class, fact that in the deep learning scenario is done by using a deep neural network. This approach is in the same line of thought of classical methods such as Gaussian Mixture Models [39, 66], histogram estimators [23] and kernel density estimators (KDE) [23, 47]. These shallow methods however require an enormous amount of data in order to retain a good accuracy when the dimension of the feature space increases. This effect is known as the *curse of dimensionality* and, by employing a deep model, the underlying idea is to avoid it.

Examples of these methods are Variational Autoencoders (VAE) [49] and Generative Adversarial Networks (GAN) [35]. The aim of these neural generative models is to learn a function that maps the elements sampled from a simple input distribution Q to the distribution P^+ of the normal class. The function is modelled by means of a neural network and as input distribution, for simplicity, it is usually considered a uniform Gaussian. In Variational Autoencoders, as anomaly score it can be used either an estimate of the likelihood of an observation of belonging to the normal class, either the reconstruction

probability [1].

As far as it regards GANs [35], they are composed of two networks: a generator G and a discriminator D. The task of the generator, as the name suggests, is to generate an element of the input space $\mathcal{X}$ starting from a point in a latent space $\mathcal{Z}$, on which it can be cast a simple input distribution. On the other hand, the discriminator D is trained to distinguish between the elements generated by G and the samples from the normal class. As anomaly detection score, the authors of AnoGAN [81] recommend for instance of using a convex combination between the reconstruction loss and the discrimination loss.

2.2.5. Reconstruction models

A second family of methods is based on the idea of embedding the input in a lower dimensional space and subsequently trying to reconstruct it. These are among the earliest deep learning approaches applied to the field of anomaly detection [44], [40] and are among the most common.
The goal of a deep reconstruction method such as [44], [40] is to learn two functions ϕ_E and ϕ_D. The first function ϕ_E performs a compression of the input data u to the latent space, based upon which ϕ_D has to reconstruct back the original element. If we do not constrain the dimension of the latent space to be smaller than the input space, a trivial optimal solution would consist in taking the identity function $\phi_D \circ \phi_E(\,\cdot\,) = I(\,\cdot\,)$. However, by requiring that the latent space has a lower dimension, the model has to learn a lower dimensional representation of the input, in what can be considered to be a bottleneck.

The model is trained using solely the data from the normal class, therefore it is optimised to reconstruct exclusively the normal instances. As a consequence, when an anomaly is fed to the model, the latter will very likely fail to meticulously reconstruct the element. On this account, as anomaly score it is usually employed some kind of measure of the reconstruction error, for instance the square norm of the difference between the original input and the reconstructed version:

$$S(u) = \|u - \phi_D \circ \phi_E(u)\|^2$$

In the years these methods have proven to be rather effective at detecting anomalies, nevertheless, it has to be pointed out that this strategy has a quite remarkable downsize. Strictly speaking, these networks are trained to perform another task rather than anomaly detection and they are then adapted to the latter use. Indeed, in order to be able to discern between normal and anomalous instances, the model is required to learn how to

reconstruct the whole input, which is a very demanding task. Therefore, these models are called to solve a significantly more difficult problem than just a binary classification, aspect that requires a lot of resources and it makes these approaches rather inefficient.

2.2.6. Decision boundary based

The task of anomaly detection can also be seen as finding a decision boundary that allows the model to discriminate between normal elements and anomalies with a low error. Some traditional methods fall within this category as well, and, among the most famous types, it is worth mentioning OC-SVM [82] and SVDD [92].

The common source of concern for this family of methods is that the engineering of the feature space may be non-trivial. In principle it has to be noted that it is also possible creating handcrafted features (for instance by taking the product of some dimensions), although this usually requires a lot of intelligence about the problem.
If working in the input space $\mathcal{X}$ yields scarce performance, creating handcrafted features is complicated, and using the features induced by a kernel still requires a thorough exploration of the possible choices, with the advent of Deep Learning it has been tried to develop deep versions of these approaches to circumvent all the aforementioned issues. The main benefit of the deep learning counterparts consists in removing the burden of feature engineering via learning the features directly during the optimisation process [54]. To this end, many works have been proposed, among which we recall Deep SVDD [78], [79], [33] and deep OC-SVM variants [25], [15]. For sake of clarity, in the following section we give an example of a deep single class classifier by illustrating the principles behind Deep SVDD [78].

Deep SVDD

In the work of Ruff et al [78] it is introduced a new model for anomaly detection called Deep Support Vector Data Descriptor, which takes inspiration from the Kernel SVDD and other traditional non-deep methods. The main idea is to train a neural network to learn a representation of the inputs, while at the same time minimising the volume of an hyper-sphere enclosing the learnt representations. By minimising the volume, the network is encouraged to extract the common factors of variation since it has to closely map the elements to the center of the sphere [78].

More precisely, given an input space $\mathcal{X} \subseteq \mathbb{R}^N$, a set of training data $\mathcal{D} = \{u_i\}_{i=1}^n$ such that $\mathcal{D} \subset, \mathcal{X}$ and an output space $\mathcal{Z} \subseteq \mathbb{R}^p$; it is considered a neural network $\phi(\,\cdot\,; \mathbb{W}) : \mathcal{X} \to \mathcal{Z}$ with L layers of weights W^l and $\mathbb{W} = \{W^l\}_{l=1}^L$. For each element $u \in \mathcal{X}$ of the input

space, $\phi(u\,;\mathbb{W}) \in \mathcal{Z}$ is its representation in the feature space $\mathcal{Z}$ produced by ϕ. The goal of Deep SVDD is therefore to learn the parameters $\mathbb{W}$ of the network while simultaneously minimising the volume of an hypersphere in $\mathcal{Z}$ that encloses the representations $\phi(u\,;\mathbb{W})$. The hypersphere is centred at a point $c \in \mathcal{Z}$ and it has radius $R > 0$.

A first version of this method is called *soft-margin Deep SVDD* and it takes its name from the fact that it allows a small portion of the input elements of being outside of the hyper-sphere. Its objective can be formulated as:

$$\min_{R,\,\mathbb{W}} \left[R^2 + \frac{1}{\nu\, n} \sum_{i=1}^{n} \max\left\{ \|\phi(u;\,\mathbb{W}) - c\|^2 - R^2,\; 0 \right\} + \frac{\lambda}{2} \sum_{l=1}^{L} \|W^l\|_F^2 \right] \tag{2.8}$$

From this expression we can see that the first term is the squared radius R^2 and hence minimising it implies shrinking the hyper-sphere. As far as it regards the second term, this is zero if the vector $\phi(u_i;\,\mathbb{W})$ is inside the hyper-sphere and it is equal to

$$\|\,\phi(u_i;\,\mathbb{W}) - c\,\|^2 - R^2 > 0$$

if it falls outside of it. Hence, this allows elements to be mapped outside of the bounded ball, although they are discouraged by the minimisation mechanism. For this reason the method is called *soft-margin*. $\nu \in (0,\,1]$ is a hyper-parameter that controls the tradeoff between the volume of the sphere and the violations of the boundary [78]; in a certain sense we can say that it controls the *softness* of the margin. The third term consists in the sum of the Frobenius norm of the weights of each layer and it has a regularisation purpose; $\lambda > 0$ decides the magnitude of this effect.

As a result of the optimisation process, the normal instances are mapped near the center c and within the hypersphere, whereas anomalies will very likely fall outside of this ball.

The objective function of the *soft-margin Deep SVDD* is constructed in such a way that it can also deal with anomalies during the training phase. However, in the framework of anomaly detection, it can be assumed that most of the data are normal. Actually, it is very often the case that all the available elements belong to the normal class, as it happens in unsupervised anomaly detection. Taking this remark into account, the authors have proposed also a simplified version, which has been called *One-Class Deep SVDD*. The optimisation problem that has been developed is:

$$\min_{\mathbb{W}} \left[\frac{1}{n} \sum_{i=1}^{n} \|\,\phi(u_i;\,\mathbb{W}) - c\,\|^2 + \frac{\lambda}{2} \sum_{l=1}^{L} \|W^l\|_F^2 \right] \tag{2.9}$$

Once again the last term serves a regularisation purpose and is equal to the one of the previous formulation (2.8). However, in this expression the radius and the penalties in case of violation of the boundary have been substituted by a term that accounts for the distance of the elements from the center of the hyper-sphere. This term is still able to incentive the mapped elements to be close to the center c in the feature space $\mathcal{Z}$ without explicitly requiring the shrinkage of the radius. Indeed, instead of minimising the radius directly, it minimises the average distance of the representations from the center, process that can be done in presence of solely normal elements.

After having trained any of the two models, quite naturally, it can be taken as anomaly score of an element $u \in \mathcal{X}$ the distance from the center c of its representation $\phi(u; \mathbb{W}^*)$:

$$s(u) = \| \phi(u; \mathbb{W}^*) - c \|^2 \tag{2.10}$$

where $\mathbb{W}^*$ refers to the parameters of the trained model, which ideally correspond to the optimal ones.

The main drawback of the Deep SVDD model is that it can suffer from a rather insidious aspect, that goes under the name of *representation collapse*. Up to now it has not been mentioned how to set the hyper-sphere's center c, which, as it turns out, is a very delicate matter. Since we are in the deep learning field, one might think that a good solution would be of considering it as a parameter and let the algorithm learn it during the training process. Although this may sound reasonable, letting the model learn the center c, brings to the collapse of the hyper-sphere. More precisely, if we consider all the weights of the network to be zero, the latter maps all the inputs to a same point $c_0 = 0$. Hence, if it is possible that $c = c_0$, we have all the elements are mapped to the origin $c_0 = 0$ and that the radius results to be equal to zero $R^* = 0$, hence an optimal solution. For this reason the authors recommend fixing the center c a-priori and they suggest empirically setting c as the average of the network representations when performing a first initial forward pass of some training data [78]. In addition, it goes without saying that setting $c = 0$ has to be avoided as well, since it would lead to the same effect. Another problem is that, if the layers of the network have a bias term, this will once more give rise to a trivial optimal solution in which all the weights but the biases are zero. This is because it is possible that the network learns a constant mapping to a vector c. Moreover, the authors reported that also the use of bounded activation functions, such as a sigmoid, could act like a bias term when they are saturated. On this account, in a Deep SVDD model all the activation functions should be unbounded, for instance like a ReLU function [78].

In outline, when implementing a Deep SVDD model, it has to be kept in mind that the

center c has to be fixed and to be different from zero, all the bias terms from the layers need to be removed and the activation functions employed have to be unbounded, such as ReLU. Using these precautions, the authors have successfully trained Depp SVDD models for the task of anomaly detection on several datasets, such as MNIST [55] and CIFAR-10 [50], obtaining good results. From this we can conclude that, although the training can be rather delicate, a Deep SVDD model is capable of reaching remarkable performance.

Deep Robust One Class Classification (DROCC)

In 2020, Goyal et al proposed a work [36] that shares many core aspects with Deep SVDD [78], although it is designed to be more robust to the representation collapse. From this comes its name: DROCC, which stands for Deep Robust One Class Classification.

DROCC is based on the assumption that the normal data lies on a low-dimensional manifold that can be locally approximated with a Euclidean space and that is well sampled in the training data [36]. The elements are deemed to be anomalous if they lie outside the union of a collection of hyperspheres centred at some typical training points [36]. In addition, this allows the generation of synthetic anomalous data by a gradient ascent phase during training, in what resembles an adversarial search. In this way it is found the most adversarial element, which is then added to the set of anomalies. The model is optimised to discriminate between the normal data and the progressively generated anomalies, fact that encourages it to learn an appropriate representation. This is similar to what happens in the case of Deep SVDD [78].

Let $\mathbb{R}^N$ be the input space and denote by $\mathcal{D} = \{u_i\}_{i=1}^n$ the set of training data with $u_i \in \mathbb{R}^N \ \forall i = 1, ..., n$. It is considered a function $\phi_\theta(\,\cdot\,) : \mathbb{R}^N \to \{0, 1\}$ such that $\phi_\theta(u) = 0$ if u belongs to the normal class and $\phi_\theta(u) = 1$ if u is an anomaly. In this case $\phi_\theta(\,\cdot\,)$ is modelled by a neural network and θ corresponds to the set of learnable weights.

Given a neural network $\phi_\theta(\,\cdot\,)$ and a radius $r > 0$, the objective of DROCC is to estimate the parameters $\hat{\theta}$. In order to achieve this, it is minimised the loss $\mathcal{L}_{\text{DROCC}}$ defined as:

$$\mathcal{L}_{\text{DROCC}} = \sum_{i=1}^{n} \left[\ell(\phi_\theta(u_i),\, 0) + \mu \max_{\tilde{u}_i \in N_i(r)} \ell(\phi_\theta(\tilde{u}_i),\, 1) \right] + \lambda \| \, \theta \, \|^2 \tag{2.11}$$

where:

$$N_i(r) := \left\{ \| \, \tilde{u}_i - u_i \, \|_2 \leq \gamma \cdot r; \quad r \leq \| \, \tilde{u}_i - u_j \, \|_2 \ \ \forall j = 1, ..., n \right\} \tag{2.12}$$

Moreover $\lambda > 0$ and $\mu > 0$ are hyper-parameters of the loss $\mathcal{L}_{\text{DROCC}}$ and they are mixing coefficients. The term $\lambda \| \, \theta \, \|^2$ serves a regularisation purpose.

$N_i(r)$ defines the set of the elements $\tilde{u}_i$ off the manifold, which are the ones at distance greater than r from every training point (i.e. $\| \tilde{u}_i - u_j \|_2 \geq r, \ \ \forall j = 1, ..., n$). In addition, it is considered an upper bound $\gamma \cdot r$, which is of use for regularising the optimisation problem, as it limits the adversarial search of anomalies inside the ball $\| \tilde{u}_i - u_i \|_2 \leq \gamma \cdot r$ centered at u_i and of radius $\gamma \cdot r$. In this expression γ is another hyper-parameter and it is such that $\gamma \geq 1$.
In (2.11), ℓ is a classification loss function, with $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, and its aim is to classify the normal elements u_i as label "0" and the generated anomalous examples $\tilde{u}_i$ as label "1" [36]. In the role of ℓ it can be chosen any classification loss, for instance, the authors [36] opted for a classic cross-entropy loss.

One of the key points of DROCC is the generation of negative examples by means of a gradient ascent phase. More precisely, given an element $z \in \mathbb{R}^N$, the objective is to find:

$$\tilde{u}_i = \arg \min_{u \in \mathbb{R}^N} \| u - z \|_2 \quad s.t. \quad u \in N_i(r) \tag{2.13}$$

Ideally, $N_i(r)$ contains elements that are anomalous since they are at least at distance r from every element of the normal class. However, the authors report that taking into consideration all of the elements in the normal class, as in (2.12), is too computationally challenging [36], therefore they redefine the set $N_i(r)$ to be:

$$N_i(r) := \left\{ r \leq \| \tilde{u}_i - u_i \|_2 \leq \gamma \cdot r \right\} \tag{2.14}$$

At this regard, the authors state that, since the normal class is assumed to lie on a low dimensional manifold of $\mathbb{R}^N$, the adversarial search in $N_i(r)$ as defined in (2.14) yields an element which empirically does not belong to the normal class [36]. In addition they suggest giving less weight to the classification loss of the generated negative example in order to guard against possible non-anomalous points in $N_i(r)$.
The projection onto the set $N_i(r)$ is given by the scalar multiplication:

$$\tilde{u}_i = u_i + \alpha \cdot (z - u_i) \tag{2.15}$$

with:

$$\alpha = \begin{cases} \gamma \cdot r / \beta & if \ \beta \geq \gamma \cdot r \\ r / \beta & if \ \beta \leq r \\ 1 & otherwise \end{cases} \tag{2.16}$$

where $\beta = \| z - u_i \|$. Starting from u, the anomalous example is generated in a gradient ascent phase by expressing $\tilde{u}$ as $\tilde{u} = u + h$ and then computing the gradient of the loss ℓ with respect to it.

The authors [36] have successfully applied DROCC to the anomaly detection problem on tabular data, time series data and image data. In the latter case they have used as datasets MNIST [55] and CIFAR-10 [50].

2.2.7. Geometric transformations

A rather different approach to anomaly detection is based on learning salient structures of the normal data by means of predicting geometric transformations. This is done by training a model to identify which transformation was applied to a certain element.

To this end it is considered a predefined set of geometric transformations and a neural network in the role of discriminator. During the training phase, all these transformations are applied to each member of the normal class, resulting therefore in several versions of the same element. After that, the obtained copies are fed to the neural network that is called to identify, for each version, which transformation was used. This process is actually a classification task since the discriminator in practice returns the classification scores over the set of considered transformations. The idea behind this type of approach is that the discriminator, in order to distinguish between the transformed elements, is required to capture salient geometrical features, some of which are characteristic of the normal class [34].

These methods fall in the category of Self-Supervised strategies, as in the process it is created a set of labels corresponding to the different transformations involved. Nevertheless, it can still be considered as a type of unsupervised anomaly detection since there is no information involved regarding the anomalies.

Recently there have been introduced other models that, moving from the same principles, have tried to find the proper transformations in an automated way. This allows to remove the burden of having to define manually the set $\mathcal{T}$ and therefore they can be more easily adopted on various types of input. Among the most prominent works, we mention Neural Transformation Learning by C. Qiu et al [73].

Deep Anomaly Detection Using Geometric Transformations

In their work from 2018, Golan and El-Yanin [34] present a novel approach to anomaly detection for image data. Their idea is to train a multi-class model to discriminate

between several geometric transformations applied to the normal input images [34]. This encourages the network to learn features that are useful in the task of detecting anomalies.

The first step consists in creating a set of K transformations $\mathcal{T} = \{T_0, T_1, ..., T_{K-1}\}$ with $T_k : \mathcal{X} \to \mathcal{X} \quad \forall k = 1, ..., K-1$ and where $T_0(u) = u$ is the identity. After that, each element of the training set is transformed using all of the defined transformations and it is kept track of which one was applied.
Therefore, starting from a dataset $\mathcal{D}$ of n images and proceeding this way, it is obtained a new dataset of $n \cdot |\mathcal{T}|$ elements, each one of them being a pair:

$$\mathcal{D}_{\mathcal{T}} := \big\{(T_k(u),\, k) :\ u \in \mathcal{D},\ T_k \in \mathcal{T}\big\}$$

where k is the label that allows to identify the transformation $T_k(u)$.
Once in possession of $\mathcal{D}_{\mathcal{T}}$, a neural network f_θ is optimised to learn to identify which geometric transformation was used given the element $T_k(u)$. The training is performed by following a classical K-class classification protocol and using a standard cross entropy loss.
For each input, the classifier f_θ returns a vector of dimension K to which has been applied a softmax operator and can be therefore interpreted as the probabilities assigned to each transformation.

These response vectors are then combined at evaluation time to create a measure of anomalousness of a given image. More precisely, let $\mathbf{y}(u) := softmax\big\{f_\theta(u)\big\}$ be the vector of the softmax responses for an element u. The normality score is defined as the combined log-likelihood of a transformed image conditioned on each applied transformation, namely:

$$\mathcal{N}_S(u) := \sum_{k=0}^{K-1} \log\, p[\,\mathbf{y}(T_k(u)) \,|\, T_k\,] \tag{2.17}$$

The underlying assumption is that all the conditional distributions are independent, even though this is not usually the case. Each conditional distribution is modelled with a Dirichlet distribution:

$$\mathbf{y}(T_k(u)) \,|\, T_k \sim Dir(\alpha_k)$$

of parameter vector $\alpha_k \in \mathbb{R}_+^k$ and each transformation is considered to be uniformly probable, namely $k \sim U(0,\, K)$. During the training phase they are estimated the parameters

of the Dirichlet distributions and they are denoted by $\hat{\alpha}_k$.

As normality score it is then employed:

$$\mathcal{N}_S(u) = \sum_{k=0}^{K-1} (\hat{\alpha}_k - 1) \log [\, \mathbf{y}(T_k(u)) \,] \tag{2.18}$$

Given two elements u_1 and u_2, $\mathcal{N}_S(u_1) > \mathcal{N}_S(u_2)$ signifies that u_1 is more normal than u_2 and therefore a high value of $\mathcal{N}_S(\cdot)$ indicates normality [34].

As we have already mentioned, the model is trained utilising solely data from the normal class, which are then used in a self-supervised manner to create a dataset of augmented images. This dataset consists of the normal elements and of their transformed versions. It has to be remarked that the set of predefined transformations is a hyper-parameter of the method. In fact, the transformations employed have been manually selected by the authors, who have used, for instance: rotations, flips and translations. This aspect is one of the main weaknesses of this approach, since it requires precise information on the type of data involved. In some scenarios, usually rather than images, it is non-trivial how to define these transformations, which makes this method difficult to be applied to certain types of inputs. They are examples tabular data and time series.

Neural Transformation Learning for Anomaly Detection

One of the main drawbacks of the method proposed in [34] is having to manually predefine the set of geometric transformations, which in some contexts might be non-trivial. For instance, in the case of tabular data or time series it is often unclear how to define these transformations. To alleviate this issue, Qiu et al proposed to directly learn the transformations such that the transformed samples share semantic information with their original version, while at the same time being easily distinguishable [73].

The authors point out two requirements that the transformations are supposed to meet:
1) **Semantics**: *the transformations should produce views that share relevant semantic information with the original data [73]*
2) **Diversity**: *the transformations should produce diverse views of each sample [73]*

The designed pipeline of NeuTraL AD is made of two components: the first one is a set of learnable transformations, which are modelled by neural networks, whilst the second part is constituted by an encoder that embeds the transformed elements in a latent space.

In the specific, let $\mathcal{X}$ be the input space and denote by $\mathcal{D} = \{u_i\}_{i=1}^n$ the set of training data, with $u_i \in \mathcal{X} \quad \forall i = 1, ..., n$. It is considered a set of transformations $\mathcal{T} = \{T_1, T_1, ..., T_K\}$

such that $T_k : \mathcal{X} \to \mathcal{X} \quad \forall k = 1, ..., K$. Moreover, it is assumed that each transformation T_k is learnable; in other words, T_k is modelled by a parameterised function whose parameters ϑ_k can be optimised during the training process. In the work of Qiu et al [73], each T_k is modelled by a feed forward neural network.
Therefore, given an element u, they are computed its transformed versions by applying each T_k to it, obtaining in this way the set $\{ u_k = T_k(u) \ \forall k = 1, ..., K \}$. Subsequently the encoder $\phi_\theta : \mathcal{X} \to \mathcal{Z}$ embeds the transformed versions u_k in the latent space $\mathcal{Z}$. Once again the encoder is modelled by a neural network whose parameters are denoted by θ.

The key element of this method is the loss function that has been implemented, which goes under the name of *Deterministic Contrastive Loss (DCL)* [73]. This loss incentives the transformed versions $u_k = T_k(u)$ to retain some similarity with their original element u, while at the same time it encourages the transformed versions to be dissimilar from each other. More precisely, given an element $u \in \mathcal{D}$, let's consider two transformed versions of u, denoted by $u_k = T_k(u)$ and $u_l = T_l(u)$ with $k \neq l$. The score function between two transformed versions is defined as:

$$h(u_k, u_l) = \exp\left\{ \frac{1}{\tau} sim\big[\phi_\theta(T_k(u)),\ \phi_\theta(T_l(u))\big] \right\} \tag{2.19}$$

where τ is the *temperature* hyper-parameter and as similarity function $sim[\,\cdot\,,\,\cdot\,]$ it is adopted the cosine similarity, namely $sim[a, b] = a^T b / \|a\| \|b\|$. In this scenario the encoder network ϕ_θ acts as a feature extractor.
The Deterministic Contrastive Loss is then defined as:

$$\mathcal{L}_{\text{DCL}} := \mathbb{E}_{u \sim \mathcal{D}} \left[-\sum_{k=1}^{K} \log \frac{h(u_k, u)}{h(u_k, u) + \sum_{l \neq k} h(u_k, u_l)} \right] \tag{2.20}$$

When $\mathcal{L}_{\text{DCL}}$ is minimised during the training process, the numerator in the eq. (2.20) encourages each embedded transformed version u_k to be close to the embedding of the original element u. On the other side, the denominator pushes all the embeddings of the transformed versions away from each other. Thus, the former aspect induces the transformed versions to be similar to the original element and hence to preserve relevant semantic information, whereas the latter incentives the learnt transformations to be dissimilar from each other and henceforth diverse [73].

From (2.20) the authors derived also the anomaly score to be used at evaluation time, which is:

$$\mathcal{A}_S(u) := -\sum_{k=1}^{K} \log \frac{h(u_k,\, u)}{h(u_k,\, u) + \sum_{l \neq k} h(u_k,\, u_l)} \tag{2.21}$$

Since during the optimisation process the $\mathcal{L}_{\text{DCL}}$ (2.20) is minimised for the elements $u \in \mathcal{D}$ belonging to the normal class, it follows that (2.21) shall assume low values for normal instances and high values for anomalies [73].

All the parameters ϑ_k of the transformations and the parameters θ of the encoder are jointly optimised during the training phase by minimising the loss function $\mathcal{L}_{\text{DCL}}$ (2.20).

2.3. Anomaly detection on point clouds

The problem of anomaly detection has been extensively analysed for a wide range of data, nevertheless, the case of point cloud data has been much less regarded. The majority of the methods tackle very specific problems, either related to industrial manufacturing, such as detecting defective products in additive manufacturing [59], or to objects with a very characteristic signature, such as the detection of pole-like objects in urban environments [75]. Other works address the one class classification of airborne LiDAR scans of urban areas [4] and the real-time object detection in the context of autonomous driving [97]. However, all these methods are tailored to the specific issue they face and therefore they do not generalise well to every anomaly detection task on point clouds. Differently from them, we would like to address the unsupervised anomaly detection task on point clouds in a more general way.

Another line of work is represented by "Anomaly detection in 3d point clouds using deep geometric descriptors" [11], which focuses on anomaly localisation in high resolution point clouds. The dataset used, namely MVTec-3D [10], is specifically designed for this task and the developed method is an adaptation of the student-teacher framework. However, this problem differs in many aspects from the one we are addressing. More precisely, the dataset used (MVTec-3D [10]) is characterised by a very low variability within the class and the objects contained in it are all disposed in roughly the same position. In addition, the point clouds are composed of thousands of points and thus have a much higher resolution.

A first attempt of performing point cloud anomaly detection in a more general way is presented by Masuda et al in [63], where it is used a Variational Autoencoder. In [31] it is proposed an extension of the DeepSVDD [78] to the case of point clouds, whilst in [32]

it is developed an adaptation of the Self-Supervised method [34].

2.3.1. 3D Variational Autoencoders for Anomaly Detection

In the work by Masuda et al. [63] it is introduced a deep Variational Autoencoder adapted to the task of anomaly detection in point clouds. The authors decided to adopt a reconstruction-based approach, similar to what we have explained in section 2.2.5.
The model learns the distribution of the normal class and during training it minimises a metric $\mathcal{A}_S(\cdot)$; at evaluation time $\mathcal{A}_S(\cdot)$ is used an anomaly score. Since this function is minimised during the optimisation process when exclusively normal elements are involved, it follows that $\mathcal{A}_S(\cdot)$ assumes low values for elements belonging to the normal class and high values for anomalies.

More in the specific, the model is made of an encoder and of a decoder. In the architecture of the encoder are introduced skip connections and graph max-pooling layers that estimate local features [85]. As decoder it is employed a FoldingNet architecture [98]. Given an input point cloud $\mathcal{P}$, the encoder returns two vectors, μ and σ, which are used to parametrise a Gaussian $\mathcal{N}(\mu, \sigma^2)$. The decoder then tries to reconstruct the original element starting from m vectors sampled from $\mathcal{N}(\mu, \sigma^2)$.

Since the data consist of point clouds, in order to account for the reconstruction error, it has been adopted as reconstruction loss the Chamfer Distance (CD) [7]. It has to be noted that the Mean Square Error between the original and the reconstructed object cannot be employed because of the lack of an effective bijection between the points before and after the compression step.

The authors decided to follow a traditional VAE approach [29], therefore they considered the KL divergence between a standard Gaussian $\mathcal{N}(0, 1)$ and a distribution $\mathcal{N}(\mu, \sigma^2)$ computed from the original point cloud $\mathcal{P}$:

$$\mathcal{L}_{KL} = D_{KL}\big(\mathcal{N}(\mu, \sigma^2) \,||\, \mathcal{N}(0, 1)\big) \tag{2.22}$$

Furthermore, it is considered the KL divergence between $\mathcal{N}(0, 1)$ and the distribution $\mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2)$, which is retrieved in an analogous way by providing as input the reconstructed version $\tilde{\mathcal{P}}$ to the encoder.
The model is trained by minimising the total loss that is given by the sum of the three components introduced above, namely:

$$\mathcal{L} = D_{KL}\big(\mathcal{N}(\mu, \sigma^2) \,||\, \mathcal{N}(0, 1)\big) + D_{KL}\big(\mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2) \,||\, \mathcal{N}(0, 1)\big) + d_{CD}\big(\mathcal{P}, \tilde{\mathcal{P}}\big) \tag{2.23}$$

As anomaly score it is employed the Chamfer Distance between the original point cloud and the reconstructed one:

$$\mathcal{A}_S(\mathcal{P}) = d_{CD}\big(\mathcal{P}, \tilde{\mathcal{P}}\big) \tag{2.24}$$

As an alternative, it is considered the Earth Mover's Distance [76] in the role of reconstruction loss and anomaly score $\mathcal{A}_S(\,\cdot\,)$, although the authors [63] report that it performs worse than the Chamfer Distance.

A final remark about this approach is that it solves a very demanding task, as the network has to learn how to reconstruct entirely the input. Doing this for point clouds poses even more challenges than the case of images, since a proper reconstruction loss has to be defined. For this reason, it has to be adopted a more complex reconstruction loss, such as the Chamfer Distance [7] or the Earth Mover's one [76], choice that is of crucial importance to the success of the optmisation process.

3 | Proposed solutions

In this chapter, we explain the models that we have implemented in order to address the anomaly detection task on point clouds. First, we develop an extension of the Deep Robust One Class Classification approach [36] to the case of point cloud data. In particular, we employ a specific architecture and an adapted loss function. Then, we take inspiration from the approach proposed in Neural Transformation Learning [73] and we apply it to the anomaly detection problem on point clouds.
We remark that none of these methods had ever been applied to point cloud data before, hence this is one of the motivations for our work. More specifically, DROCC [36] is originally implemented for images and tabular data, whilst [73] focuses on tabular data and time series. Other than that, in our work we devote our attention to the case of point clouds by employing networks based on point convolutional operators.

3.1. DROCC for point cloud anomaly detection

As we have discussed in section 2.2.6, the Deep Robust One Class Classification approach relies on the discrimination between elements belonging to the normal class and adversarially generated anomalies. The underlying assumption is that the normal data lie on a well sampled low-dimensional manifold in the high dimensional input space, which can be locally approximated by a Euclidean space. This is known as the *manifold assumption* and it is motivated by the fact that the normal instances tend to share common characteristics among them and so to be close in the space they live in. Therefore, their variability can be well captured also by a lower dimensional representation of them.
We retain the manifold assumption to hold also in the case of point cloud data for every normal class that we consider. Furthermore, to guarantee the quality of the sampling, in our experiments we consider normal classes constituted by thousands of elements.

The input of the model is a point cloud $\mathcal{P} = (P, \varphi)$ and it is processed by a function $\phi_\theta(\,\cdot\,)$, which is in charge of associating to $\mathcal{P}$ a value to be then used as normality score. In our case, we define $\phi_\theta(\,\cdot\,)$ by means of a neural network based on the Composite layers [32]. We have experimented with different architectures employing these layers in order to

evaluate their effectiveness in this context. Furthermore, we have considered other types of neural networks in the role of $\phi_\theta(\cdot)$, such as a classical PointNet [69], a ConvPoint [13] and a more advanced PointMLP [60].

3.1.1. The loss function

The loss function that we have utilised has the same form of the one presented in section 2.2.6, although it has been adequately adapted to the case of point cloud data. More precisely, given a set of input point clouds $\{\mathcal{P}_m\}_{m=1}^{M}$, the loss function that we employed can be written as:

$$\mathcal{L}_{\text{DROCC}} = \sum_{m=1}^{M} \left[\ell(\phi_\theta(\mathcal{P}_m),\, 0) + \mu \max_{\tilde{\mathcal{P}}_m \in N_m(r)} \ell(\phi_\theta(\tilde{\mathcal{P}}_m),\, 1) \right] + \lambda \| \theta \|^2 \tag{3.1}$$

where $\tilde{\mathcal{P}}_m$ indicates the generated anomaly.

Moreover, in accordance with Goyal et al [36], we have adopted a standard cross entropy loss in the role of the binary classification loss ℓ. In our case, the normal class is identified by the label "0", whereas anomalies are indicated by the label "1". The aim of $\phi_\theta(\cdot)$ is to classify the normal instances $\mathcal{P}$ as belonging to the normal class and the adversarially generated examples $\tilde{\mathcal{P}}$ as anomalies.

In addition, we have added the L^2 - regularisation term $\lambda \| \theta \|^2$ in the loss, as shown in (3.1), which is missing in the implementation of [36].

3.1.2. The adversarial search

One of the core aspects of DROCC is the synthetic generation of anomalous examples $\tilde{\mathcal{P}}$. This is done during a gradient ascent phase that resembles an adversarial search. More precisely, it is sought after the element that the network deems to be the most anomalous one in terms of the function ℓ. In this section we cover more in details what this search consists of and how it can be carried out, in particular for the case of point cloud data.

During training, DROCC generates an anomalous example $\tilde{\mathcal{P}}$ for each normal element $\mathcal{P}$ of the training set. These artificially generated anomalies are then of help in teaching the model how to discriminate between normal instances and anomalous ones. This is indeed the reason why we employ a binary classification loss ℓ.

The need for artificial anomalies comes from the assumption of unsupervised anomaly detection, which means that all training data belong to the normal class. From this it

follows that it is not possible using real anomalies to this end, as we do not have them at our disposal.

Let us consider a point cloud $\mathcal{P} = (P, \varphi)$. It is easy to note that a point cloud has two components: a set $P = \{x_i\}_i$ of points in $\mathbb{R}^d$ and the associated features $\varphi(x_i)$. On this account, an element $\mathcal{P}$ might be deemed to be anomalous based on the set of coordinates P, based on the features φ, or based on both of them. This gives rise to three different kinds of anomalies and so there are as many that can be produced.
However, we observe that many datasets include only constant features (i.e. $\varphi \equiv 1$) and this condition is actually the case for ModelNet40 [95] and ShapeNet7C [17], which we use in our experiments. When this is the scenario, we can only generate elements that can be deemed anomalous just with respect to the coordinates.
Nevertheless, for sake of completeness, we have implemented the adversarial search such that it could be carried out only in the space of the coordinates, only in the space of the features, or in both. This said, since in our experiments we perform the adversarial search solely in the space of the coordinates, in what follows we illustrate only this latter case.

The objective

Given a normal instance $\mathcal{P}$, the aim of the adversarial search is to return an anomalous element $\tilde{\mathcal{P}}$. We now focus our attention on what this exactly means and how it could be performed. First of all, the adversarial search that we perform consists of a gradient ascent phase during training. More precisely, the function that is utilised to quantify the concept "most adversarial element" is the binary classification loss $\ell(\,\cdot\,)$. In addition, this element $\tilde{\mathcal{P}}$ is regarded as an anomaly, from which it follows that its classification label must be "1". For this reason we want the function $\phi_\theta(\,\cdot\,)$ to predict $\tilde{\mathcal{P}}$ as anomalous, too. This is done by maximising the agreement between the prediction $\phi_\theta(\tilde{\mathcal{P}})$ of the network and the label "1" according to the loss function $\ell(\,\cdot\,)$. Putting all together, we have that finding the most adversarial element can be formally expressed as solving the optimisation problem:

$$\max_{\tilde{\mathcal{P}} \in N(r)} \ell\left(\phi_\theta(\tilde{\mathcal{P}}),\, 1\right) \tag{3.2}$$

Initialisation

The initialisation stage consists in adding to each point $x_i \in P$ a zero-mean Gaussian noise $h_{i;0} \sim \mathcal{N}(0,\, \sigma^2 \cdot I)$ of standard deviation σ. To ease the implementation, instead of extracting $|P|$ Gaussian vectors of dimension d, we extract a unique Gaussian vector $h_0 \sim \mathcal{N}(0,\, \sigma^2 \cdot I)$ of dimension $d \cdot |P|$, namely $h_0 \in \mathbb{R}^{d \cdot |P|}$, and we rearrange it. We are

entitled to do so since we consider a diagonal covariance matrix $\sigma^2 \cdot I$, which implies that all the components of the Gaussian vector h_0 are independent and thus the two ways of proceeding are equivalent. For this reason we indicate the resulting set of coordinates as $P + h_0$. We consider the same value of σ for every dimension since we do not have any information that would lead us to do otherwise.
In principle, instead of using as starting point $P + h_0$, it is also possible to use directly h_0, which is a set of points randomly scattered in the space. However, we believe that starting the adversarial search from h_0 would yield worse results than starting from a normal element and it would jeopardise the motivations behind this search. Some experiments that we have conducted corroborate this hypothesis, as the model showed to be less effective.

The domain

On the same line of thought of [36], the anomalous example $\tilde{P}$ is sought after in the set:

$$N(r) := \left\{ r \leq \| \tilde{P} - P \|_2 \leq \gamma \cdot r \right\} \tag{3.3}$$

In accordance with what stated by Goyal et al [36], instead of the more rigorous definition of $N(r)$ reported in (2.12), we adopt the simplified version. More precisely, rather than always considering the elements that are at least at distance r from all the training samples, we define $N(r)$ to be the set of the elements at least at distance r from the single P under consideration. Even with the simpler set $N(r)$ as in (3.3), the adversarial search empirically yields generated samples that still do not belong to the normal class, as we have asserted upon visual inspection.
The upper bound $\gamma \cdot r$ is functional to the success of the optimisation procedure, as we have had the opportunity to verify. Indeed, by setting a too large value of γ we have experienced that the training process failed, allegedly due to the gradient explosion.

Optimisation process

We maximise (3.2) by the traditional optimisation method of gradient ascent [8]. The idea is to find the steepest ascent direction of $\ell(\cdot)$ w.r.t. h_{s-1}, which corresponds to the one given by the gradient of $\ell(\phi_\theta(P + h_{s-1}), 1)$ with respect to h_{s-1}. After that, we make a step in the direction of the gradient of length η, namely:

$$h_s = h_{s-1} + \eta \frac{\nabla_{h,\, s-1} \ell(h_{s-1})}{\| \nabla_{h,\, s-1} \ell(h_{s-1}) \|} \tag{3.4}$$

where $\ell(h_{s-1}) := \ell\left(\phi_\theta(P + h_{s-1}),\, 1\right)$, s refers to the number of the iteration and the *step size* η is a hyper-parameter.
Analogously to Goyal et al [36] we project h_s of (3.4) into the set $N(r)$ centered at P (3.3) by computing:

$$h_s = \alpha \cdot h_s \qquad \text{where} \qquad \alpha = \begin{cases} \gamma \cdot r / \| h_s \| & if \ \| h_s \| \geq \gamma \cdot r \\ r / \| h_s \| & if \ \| h_s \| \leq r \\ 1 & otherwise \end{cases} \tag{3.5}$$

in which the operator $\cdot$ indicates the scalar multiplication between $\alpha \in \mathbb{R}$ and the vector $h_s \in \mathbb{R}^{d \cdot |P|}$. The procedure described above is repeated for each step $s = 1, ..., S$.

Remarks

As disclaimed, we usually perform the adversarial search taking into account only the coordinates of a point cloud. This is made out of necessity since in the datasets that we consider the features are equal to the constant 1 (i.e. $\varphi \equiv 1$) and therefore they do not carry any useful information. On this ground, when we adversarially generate solely the set of points $\tilde{P}$, we associate to it the original features of P, namely we set $\tilde{\mathcal{P}} = (\tilde{P}, \varphi)$.

Overall, the hyper-parameters of the adversarial search are:

- S: the number of gradient ascent steps to be performed
- η: the step-size
- r: the radius of the set $N(r)$ defined in eq. (3.3)
- γ: the value used for the upper bound of the set $N(r)$ as in eq. (3.3)

3.1.3. Modelling the function ϕ_θ

In the role of the function $\phi_\theta(\,\cdot\,)$ we have employed a neural network designed to process point clouds. We have primarily devoted our attention to networks based on the composite layers proposed by Floris et al [32]. The motivations behind such a choice are twofold: first, we want to investigate their effectiveness in this context. Secondly, this allows us to directly compare our results with those obtained by the anomaly detection solutions in [32]. In this way we believe to provide a more comprehensive picture of the anomaly detection problem on point clouds. In addition, for sake of completeness, we have experimented also with other types of neural networks that process point clouds, with some of them being significantly more complex in terms of the number of parameters

and architecture.

Architectures based on the composite layers

The network which we have primarily utilised is made of three convolutional composite layers followed by one fully connected layer and it has been named *ADCompositeNet3*. In it, the task of the composite layers is to capture the information contained in the point cloud $\mathcal{P}$ and to condense it in a shape descriptor, which is then fed to the fully connected layer. The output dimension of the last layer is always 1, as it corresponds to the anomaly score assigned to the input $\mathcal{P}$.
More details about the architecture of ADCompositeNet3 are reported in Table 3.1. The same architecture has been used for the *ADCompositeNet3 (Aggr.)* network, which uses aggregate composite layers instead of the convolutional ones.
We also have considered a slight variation of this network, which has an additional fully connected layer and has been called *ADCompositeNet3b*. In this case, the convolutional layers are identical to the ones in Table 3.1, although they are followed by two fully connected layers. The input and output dimensions of the latter are $(6 \cdot J_0 \to 2 \cdot J_0)$ for the first and $(2 \cdot J_0 \to 1)$ for the second.

Table 3.1: Architecture of ADCompositeNet3; J is the number of output features, $|X_y|$ the cardinality of neighbourhoods and $|Q|$ the number of output points. BN stands for Batch Normalisation

ADCompositeNet3

Layer type	J	$\lvert X_y \rvert$	$\lvert Q \rvert$
Composite + BN + LeakyReLU	J_0	32	256
Composite + BN + LeakyReLU	$3 \cdot J_0$	16	64
Composite + BN + LeakyReLU	$6 \cdot J_0$	16	1
Fully Connected	1	-	-

In Table 3.1 J represents the number of output features, whose values are defined in terms of J_0, that is the number of output features of the first layer. This is mainly done out of convenience, as in this way J_0 is the only hyper-parameter on which the dimensions of the output features depend. Moreover this ensures a greater flexibility than just fixing the dimensions of each layer and at the same time it eases the hyper-parameters tuning. In the case of fully connected layers, J refers to the output dimension. The cardinality of the neighbourhoods used in each layer is indicated by $|X_y|$ and the number of output points by $|Q|$. Although not expressively mentioned, the number of input points of $\mathcal{P}$

that we consider is always equal to 1024. After each layer, it is used batch normalisation and as activation function it is employed a LeakyReLU [61] of slope 0.02. Similarly to what happens in CNNs for image processing, where the coordinates of the pixels are not changed, nonlinearities and batch normalisation are applied only to the features and not to the spatial coordinates of the points, as also done by [32].

Table 3.2: Architecture of ADCompositeNet5

ADCompositeNet5

Layer type	J	$\lvert X_y \rvert$	$\lvert Q \rvert$
Composite + BN + LeakyReLU	J_0	32	1024
Composite + BN + LeakyReLU	$2 \cdot J_0$	32	256
Composite + BN + LeakyReLU	$4 \cdot J_0$	16	64
Composite + BN + LeakyReLU	$4 \cdot J_0$	16	16
Composite + BN + LeakyReLU	$8 \cdot J_0$	16	1
Fully Connected	1	-	-

In addition to ADCompositeNet3, we have also analysed a more complex network, called *ADCompositeNet5*. As the name may suggest, it is composed of five convolutional layers. Its architecture can be found in Table 3.2 and it resembles the one of ADCompositeNet3. The notation is the same that we have previously introduced.

Other architectures considered

In addition to the networks constructed using the composite layers [32], we have considered also a ConvPoint [13] network, whose architecture is reported in Table 3.3.

Table 3.3: Architecture of ADConvPoint3; J is the number of output features, $\lvert X_y \rvert$ the cardinality of neighbourhoods and $\lvert Q \rvert$ the number of output points. BN stands for Batch Normalisation.

ADConvPoint3

Layer type	J	$\lvert X_y \rvert$	$\lvert Q \rvert$
Point convolutional + BN + LeakyReLU	16	32	256
Point convolutional + BN + LeakyReLU	48	16	64
Point convolutional + BN + LeakyReLU	96	16	1
Fully Connected	1	-	-

To make the comparison between it and ADCompositeNet3 (Tab. 3.1) as fair as possible, we have designed the former network such that its architecture resembles as closely as we could the one of the latter. Therefore we have used three point convolutional layers, followed by a fully connected one. Following our consolidated naming scheme, we called this network *ADConvPoint3*. In addition, also the dimensions of the output features are the same among the two networks, as well as other hyper-parameters such as the cardinality of the neighbourhoods and of the output sets of points.

Another network that we have taken into account for our experiments is a more traditional PointNet [69], which we have presented in section 2.1.3. The architecture that we designed is reported in Table 3.4. The first three fully connected layers constitute an MLP which is used to process each single point, subsequently, a max pooling operator aggregates the information from all the points. After that, a second MLP made of three fully connected layers is in charge of returning the output, which in our case corresponds to the anomaly score. Overall this architecture is rather simple, nonetheless, because of the massive presence of fully connected layers, this network counts 736641 learnable parameters.

Table 3.4: Architecture of PointNet

PointNet

Layer type	Input dim	Output dim
h FC1	3	64
h FC2	64	128
h FC3	128	1024
MAX pooling (1024)		
γ FC4	1024	512
γ FC5	512	128
γ FC6	128	1

Outline of the algorithm

Once established the aspects that we have described so far, let us put together all the elements in order to convey a unifying view of the method.

Input

The training data consist of a set of point clouds $\{\mathcal{P}_j\}_j$. During the optimisation process, the latter is divided and processed in mini-batches, so as to take full advantage of the GPU's computational power. Each mini-batch is composed of a $batch\_size$ number

of point clouds, which is a hyper-parameter of the method. In our case we opted for $batch\_size = 32$.

Hyper-parameters

Most of the hyper-parameters have already been introduced in the previous sections, however, for sake of clarity, we believe that it can be of use reporting them all together:

- n_points: number of points of each point cloud
- $batch\_size$: dimension of each mini-batch
- lr: initial learning rate of the optimiser
- μ: mixing coefficient in $\mathcal{L}_{\text{DROCC}}$, eq. (3.1)
- λ: regularisation coefficient in $\mathcal{L}_{\text{DROCC}}$, eq. (3.1)
- Hyper-parameters of the adversarial search:
  - r: radius that characterises the set $N(r)$ defined in eq. (3.3)
  - γ: value used for the upper bound of the set $N(r)$ as in eq. (3.3)
  - σ: standard deviation of the additive noise h_0 in the initialisation stage
  - S: number of gradient ascent steps performed for each sample $\mathcal{P}_j$
  - η: step-size of each gradient ascent step

Objective

The objective is to minimise the loss function:

$$\mathcal{L}_{\text{DROCC}} = \sum_{m=1}^{M} \left[\ell(\phi_\theta(\mathcal{P}_m),\, 0) + \mu \max_{\tilde{\mathcal{P}}_m \in N_m(r)} \ell(\phi_\theta(\tilde{\mathcal{P}}_m),\, 1) \right] + \lambda \| \theta \|^2$$

The optimisation process is carried out by means of a gradient-descent method such as Stochastic Gradient Descent [103] or Adam [48].

3.1.4. Scheme of the algorithm

The complete scheme of the training procedure is presented in Alg. 3.1. Instruction 7 consists in the projection in the set $N_m(r)$ and we report it here below:

$$h_s = \alpha \cdot h_s \qquad \text{where} \qquad \alpha = \begin{cases} \gamma \cdot r / \| h_s \| & if \ \| h_s \| \geq \gamma \cdot r \\ r / \| h_s \| & if \ \| h_s \| \leq r \\ 1 & otherwise \end{cases}$$

The set $N_m(r)$ we recall to be:

$$N(r) := \left\{ r \leq \| \tilde{P} - P \|_2 \leq \gamma \cdot r \right\}$$

Algorithm 3.1 DROCC for Point Cloud Anomaly Detection *(one epoch)*

Input: $\{\mathcal{P}_j\}_j$

1: **for** each batch $\mathcal{B}_b$, $b = 1, ..., B$ **do**
2: **for** each point cloud $\mathcal{P}_m \in \mathcal{B}_b$ **do**
3: Sample $h_0 \sim \mathcal{N}(0, \sigma^2 \cdot I)$
4: **for** each step $s = 1, ..., S$ **do**
5: $\ell(h_{s-1}) := \ell\left[\phi_\theta(P + h_{s-1}, \varphi), 1\right]$
6: $h_s = h_{s-1} + \eta \frac{\nabla_{h,\, s-1} \ell(h_{s-1})}{\| \nabla_{h,\, s-1} \ell(h_{s-1}) \|}$
7: $h_s = \alpha \cdot h_s$
8: **end for**
9: $\tilde{\mathcal{P}}_m := (P + h_S, \varphi)$
10: **end for**
11: $\mathcal{L} = \sum_{\mathcal{P}_m \in \mathcal{B}_b} \left[\ell(\phi_\theta(\mathcal{P}_m), 0) + \mu \max_{\tilde{\mathcal{P}}_m \in N_m(r)} \ell(\phi_\theta(\tilde{\mathcal{P}}_m), 1) \right] + \lambda \| \theta \|^2$
12: $\theta = \theta - Gradient\_step\{\mathcal{L}\}$
13: **end for**

3.2. Neural Transformation Learning for point cloud anomaly detection

A promising direction in anomaly detection is based on transforming the input data to produce surrogate labels and then predicting which transformation had been applied. In [32], the original method from [34] has been extended to point clouds, using as transformations 8 rotations around a fixed horizontal axis. However, this set of transformations

has proven to be too limited for guaranteeing the success of the method on certain classes of point clouds such as "lamp" [32] of ShapeNetCore [17]. In general, the main drawback of [34] is that the transformations have to be defined a-priori. Motivated at mending the shortcomings of [34], we have developed a new method for anomaly detection on point clouds that takes inspiration from the one proposed in [73], in which the transformations are modelled by neural networks and are learnt during training.

We have attempted to learn the transformations directly on point clouds, however this resulted in very poor performance, in some cases. For this reason, inspired by [73] and the rest of the literature [94], [19], [89], [11], [72], we decided to employ a pre-trained feature extractor to process the point clouds. The feature extractor returns a vector for each input point cloud. Proceeding similarly to [73], the transformations are learnt on these feature vectors. Moreover, following the suggestions provided by [73], we use an encoder to map the transformed elements to a lower dimensional latent space before computing the values of the loss function.

3.2.1. The structure of the model

As we have anticipated, the model is composed of three components: a feature extractor, a set of learnable transformations, and an encoder. Here we illustrate more in detail their roles and how we have implemented them.

Feature extractor

The task of the feature extractor is to process a point cloud $\mathcal{P}$ and to return a vector that ideally condenses the useful information contained in the input. This vector can be thought as a global descriptor of the shape.
Formally, the feature extractor is a function $\phi_{PC} : \mathcal{P} \mapsto \phi_{PC}(\mathcal{P})$ that associates to each point cloud $\mathcal{P}$ a vector $\phi_{PC}(\mathcal{P}) \in \mathbb{R}^{\bar{J}}$, where $\bar{J}$ is the dimension of the global descriptor.
For this purpose, we consider a neural network based on the composite layers [32]. More precisely, our feature extractor is made of five convolutional composite layers that gradually reduce the cardinality of the point cloud until only one point is left. The output feature vector of this remaining point is then employed as the global descriptor. These vectors describe the point cloud since the network is trained to perform the classification task, a fact that requires the convolutional layers to extract meaningful features in order for the network to succeed in the aforementioned task.

Learnable transformations

On the same line of thought of [73], we have implemented the transformations by means of deep neural networks. In such a way, we can regard these transformations to be learnable and we are relieved from having to manually define them.

The set of learnable transformations $\mathcal{T} = \{T_1, T_2, ..., T_K\}$ is constituted by K functions $T_k : \mathbb{R}^{\bar{J}} \to \mathbb{R}^{\bar{J}} \quad for\ k = 1, ..., K$, whose parameters are denoted by $\vartheta_k \quad for\ k = 1, ..., K$. Each transformation takes as input a global descriptor vector $\phi_{PC}(\mathcal{P})$ and it returns a vector in $\mathbb{R}^{\bar{J}}$ of the same dimension. To model each transformation we have used a neural network made of three fully connected layers and we have employed a ReLU as activation function.

As in [73], the output of each neural network can be either directly considered as the transformed version $T_k(\phi_{PC}(\mathcal{P}))$ or it can be combined with the input $\phi_{PC}(\mathcal{P})$. In the latter case, we regard the output of the neural network as a mask M_k and we combine it with the global descriptor $\phi_{PC}(\mathcal{P})$ in two different manners. More precisely, given an input $\phi_{PC}(\mathcal{P})$ and the output $M_k(\phi_{PC}(\mathcal{P}))$ of a neural network, they are defined three types of parametrisation of the learnable transformations:

- *Feed forward:* $T_k(\phi_{PC}(\mathcal{P})) = M_k(\phi_{PC}(\mathcal{P}))$
- *Residual:* $T_k(\phi_{PC}(\mathcal{P})) = M_k(\phi_{PC}(\mathcal{P})) + \phi_{PC}(\mathcal{P})$
- *Multiplicative:* $T_k(\phi_{PC}(\mathcal{P})) = M_k(\phi_{PC}(\mathcal{P})) \odot \phi_{PC}(\mathcal{P})$

In the *multiplicative* case, $\odot$ denotes the Hadamard (i.e. point-wise) product. The residual version introduces what can be defined a skip connection, a strategy that proved to be effective in other deep learning fields [41] as the network has to learn only the difference with respect to the input. The multiplicative version is somehow similar to a skip connection, although in this case the input and the output are multiplied together instead of being summed.

Encoder

The encoder maps the transformed versions of a same point cloud to a lower dimensional space where ideally they are more easily distinguishable [73]. The representations in this latent space are then used to compute the loss. The encoder is therefore a function $\phi_{enc} : \mathbb{R}^{\bar{J}} \to \mathbb{R}^{L}$ and the same encoder is used to map all the transformed versions. We have employed as encoder a simple Multi-Layer Perceptron made of two fully connected layers.

3.2.2. The loss function

The key aspect of this method is the loss function, namely the *Deterministic Contrastive Loss (DCL)*. Let us first define the score function between two transformed versions as:

$$h(\mathcal{P}_k, \mathcal{P}_l) := \exp\left\{ \frac{1}{\tau} sim\big[\phi_{enc}(T_k(\phi_{PC}(\mathcal{P}))),\ \phi_{enc}(T_l(\phi_{PC}(\mathcal{P})))\big] \right\} \tag{3.6}$$

where τ is an hyper-parameter and $sim[\,\cdot\,,\,\cdot\,]$ is a similarity function. In line with [73], we have adopted the cosine similarity, that is $sim[a,\, b] = a^T b/\|a\|\|b\|$.

The Deterministic Contrastive Loss function can be therefore written as:

$$\mathcal{L}_{\text{DCL}} := \mathbb{E}_{\mathcal{P}\sim\mathcal{D}}\left[-\sum_{k=1}^{K} \log \frac{h(\mathcal{P}_k,\, \mathcal{P})}{h(\mathcal{P}_k,\, \mathcal{P}) + \sum_{l\neq k} h(\mathcal{P}_k,\, \mathcal{P}_l)} \right] \tag{3.7}$$

When $\mathcal{L}_{\text{DCL}}$ is minimised, the numerator pushes the embedding of each transformed version $T_k(\phi_{PC}(\mathcal{P}))$ close to the one of the original element $\phi_{PC}(\mathcal{P})$, fact that incentives the transformations to retain relevant semantic information. On the other side, the denominator encourages the transformed versions to be dissimilar from each other by pulling away the relative embedding.

3.2.3. Anomaly score

The anomaly score which is used as evaluation time resembles the Deterministic Contrastive Loss (3.7) and, for an input point cloud $\mathcal{P}$ it is defined as:

$$\mathcal{A}_S(\mathcal{P}) = -\sum_{k=1}^{K} \log \frac{h(\mathcal{P}_k,\, \mathcal{P})}{h(\mathcal{P}_k,\, \mathcal{P}) + \sum_{l\neq k} h(\mathcal{P}_k,\, \mathcal{P}_l)} \tag{3.8}$$

3.2.4. Outline of the algorithm

The training procedure can be summarised as in Alg. 3.2. The input of the algorithm are the set of training data $\mathcal{D} = \{\mathcal{P}_j\}_j$ and the (pre-trained) feature extractor $\phi_{PC}(\,\cdot\,)$. The point clouds are processed in batches due to computational reasons. The learnable weights θ include the parameters ϑ_k of each neural transformation T_k and the parameters of the encoder ϕ_{enc}. The parameters of the feature extractor $\phi_{PC}(\,\cdot\,)$ are assumed to be fixed since they have already been optimised to capture the information contained in the input point clouds.

Algorithm 3.2 Neural Transformation Learning for Point Cloud Anomaly Detection *(one epoch)*

Input: $\{\mathcal{P}_j\}_j$, $\phi_{PC}(\cdot)$ pre-trained

1: **for** each batch $\mathcal{B}_b$, $b = 1, ..., B$ **do**
2: **for** each point cloud $\mathcal{P} \in \mathcal{B}_b$ **do**
3: Compute the global descriptor: $\phi_{PC}(\mathcal{P})$
4: **for** each $k = 1, ..., K$ **do**
5: Compute the transformed version: $T_k\big(\phi_{PC}(\mathcal{P})\big)$
6: Encode each $T_k\big(\phi_{PC}(\mathcal{P})\big)$ in the latent space $\mathbb{R}^L$: $\phi_{enc}\big(T_k\big(\phi_{PC}(\mathcal{P})\big)\big)$
7: **end for**
8: **end for**
9: Compute the loss:
$\mathcal{L} = \mathbb{E}_{\mathcal{P}\in\mathcal{B}_b}\left[-\sum_{k=1}^{K} \log \frac{h(\mathcal{P}_k, \mathcal{P})}{h(\mathcal{P}_k, \mathcal{P}) + \sum_{l\neq k} h(\mathcal{P}_k, \mathcal{P}_l)} \right]$
where: $h(\mathcal{P}_k, \mathcal{P}_l) := \exp\left\{ \frac{1}{\tau}\, sim\big[\phi_{enc}(T_k(\phi_{PC}(\mathcal{P}))),\ \phi_{enc}(T_l(\phi_{PC}(\mathcal{P})))\big] \right\}$
10: Update the weights: $\theta = \theta - Gradient\_step\{\mathcal{L}\}$
11: **end for**

3.2.5. Implementation details

Pre-training procedure

As we have mentioned, we employ a pre-trained feature extractor. This feature extractor is trained on a dataset different from the one used for the anomaly detection task. In addition to this, if the former shares some classes with the latter, these common classes are removed from the dataset used for the pre-training. This is done in order to avoid the possibility that the feature extractor during the pre-training process is exposed to the same types of objects that are being used for the anomaly detection task. In fact, we have reasons to believe that performing the pre-training on objects belonging to the classes used for anomaly detection would provide an unfair advantage to the overall model. This "unfair advantage" is intended with respect to methods that do not make use of a pre-training strategy.

To make an example, when we pre-train $\phi_{PC}(\cdot)$ on ModelNet40 with in mind the objective of performing the anomaly detection task on ShapeNet7C, we remove from ModelNet40 the classes that these two datasets have in common. The feature extractor is then optimised using solely the data of the remaining classes.

The pre-training protocol that we have developed consists of training a model on a multi-

class classification task. In order to succeed in this task, the model is required to learn how to extract the discriminative features of the different objects. Ideally, the features learnt by the model, especially at low levels, are simple and therefore they generalise well. A pre-training strategy is also exploited by [73] when performing anomaly detection on image data.
In the role of the feature extractor we have employed a CompositeNet, whose architecture resembles the one presented in [32]. In particular, the network that we have used is composed of five composite layers, followed by one fully connected layer. Once the pre-training procedure is over, we discard the fully connected layer and we retain the composite layers as feature extractor.

3.2.6. Networks' architecture

In this section, we present the architecture of the networks that we have employed in the roles of feature extractor, transformations and encoder, respectively.

Feature extractor

As we recall, the input of the feature extractor $\phi_{PC}(\cdot)$ is a point cloud $\mathcal{P}$ and its purpose is to return a vector $\phi_{PC}(\mathcal{P}) \in \mathbb{R}^{\bar{J}}$ that acts like a global descriptor. In light of this, since the feature extractor has to process point clouds, we have decided to model it by means of a neural network based on the composite layers. The architecture that we have designed slightly reminds us of the one that is used for the classification task in [32], since it is declared by the authors to be the best performing one in said scenario. More specifically, the feature extractor that we have constructed is composed of five convolutional composite layers which gradually reduce the cardinality of the point cloud. In Table 3.5 it is reported the architecture that we have implemented.

Table 3.5: Architecture of the feature extractor; J is the number of output features, $|X_y|$ the cardinality of neighbourhoods and $|Q|$ the number of output points

Feature extractor

Layer type	J	$\lvert X_y \rvert$	$\lvert Q \rvert$
Composite + BN + LeakyReLU	64	32	1024
Composite + BN + LeakyReLU	128	32	256
Composite + BN + LeakyReLU	256	24	64
Composite + BN + LeakyReLU	512	16	16
Composite	1024	16	1

We have decided to employ the convolutional version of the composite layer since it has fewer parameters than its aggregate counterpart and the difference in performance is not so significant. Indeed, the gap in the classification accuracy found by [32] is of around 1 % and furthermore it has to be reminded that our primary objective is not the classification task. On the other hand, the lower number of parameters is important to us since we already use a higher dimension for the output features, fact that substantially contributes to increasing the total number of parameters to be learnt. The reason behind the latter choice is that we want the global descriptor vector to be expressive, hence we adopt a high dimension for it. In the role of this descriptor vector we employ the output features returned by the last convolutional layer. The whole architecture is designed such that the information contained in the input point cloud is condensed in the features of a single output point.

The number of output points of each layer is reported in Table 3.5 and it is denoted by J. Moreover, in said table, $|X_y|$ indicates the cardinality of the neighbourhoods and Q the cardinality of the output set of points. As we can see from it, the feature extractor gradually increases the dimensions of the output features, while simultaneously reducing the cardinality of the set of points to 1.
As far as it regards the dimension of the spatial function s of each composite layer and the number of centers, we have decided to set the former to 16 and the latter to 64. This is done in accordance with [32]. The choice of the output dimension of s is further motivated by the will of keeping the total number of learnable parameters under control, since this value is another factor that highly affects the number of parameters, as we have explained in 2.1.8. We have used batch normalisation and a leaky ReLU of negative slope 0.02 activation function after each composite layer but the output one.

Learnable transformations

The transformations that we consider are modelled by neural networks, in such a way they result to be learnable. To this end we have employed a classic Multi-Layer Perceptron made of three fully connected layers. More specifically, the output dimension of each layer is 1024, which also corresponds to the input dimension. Thus the MLP that we implemented results to be:

$$FC1(1024,\, 1024) \rightarrow FC2(1024,\, 1024) \rightarrow FC3(1024,\, 1024)$$

where the first number in the parenthesis indicates the input dimension whilst the second is the output one.

An illustration of the architecture of each transformation is reported in Figure 3.1.
We have used a ReLU nonlinearity as activation function after each layer but the last one.

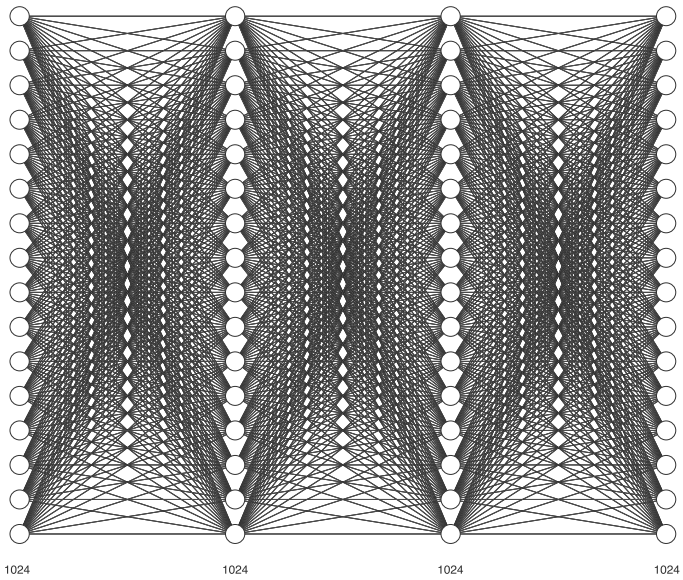


Figure 3.1: Depiction of the MLP employed as learnable transformation

To each transformation it corresponds one neural network as the one here represented and they do not share parameters. In total, for our experiments, we have used 15 learnable transformations.

Encoder

The aim of the encoder is to map the transformed versions $T_k(\phi_{PC}(\mathcal{P}))$ in a latent space where the representations of the latter are more easily distinguishable. In line with [73], we have decided to model our encoder $\phi_{enc}(\,\cdot\,)$ with a Multi-Layer Perceptron, since it is probably the simplest network that could have been employed in such a role. More in the specific, in this case we have decided to use only two fully connected layers, as we did not find beneficial increasing their number. According to the same notation we have adopted earlier, this network can be written as:

$$FC1(1024,\, 640) \rightarrow FC2(640,\, 256)$$

The input dimension of the first layer is 1024, that is the dimension of $T_k(\phi_{PC}(\mathcal{P}))$, whereas its output dimension is 640. The output dimension of the last layer is 256, which corresponds to the dimension of the latent space. We chose these values because it is

the best configuration that emerged from our experiments. The resulting architecture is illustrated in Figure 3.2.
As activation function after the first layer we have employed once again a ReLU.

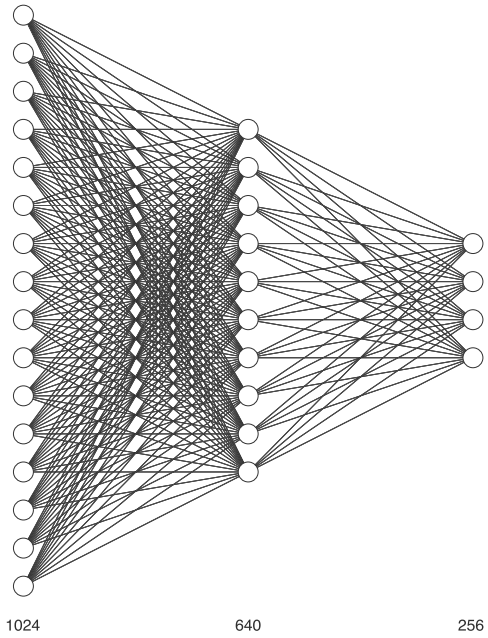


Figure 3.2: Illustration of the encoder

As we can see, the network that we have implemented as encoder is very simple. This is motivated by the fact that we believe it is not necessary to design an over-complicated architecture for such a role.

4 | Experiments

In this chapter, we illustrate the experiments we have carried out to assess the performance of our anomaly detection methods. We first present the datasets that we have used and the figures of merit we have employed for evaluation. After that, we investigate the results obtained by our extension to point clouds of DROCC [36] (section 3.1), both when using different architectures based on the composite layers [32] and other types of networks [13], [69]. Moreover, we compare them to the ones of other methods such as [63], [31] and [32]. Ultimately, we explore our adapted version of the Neural Transformation Learning approach [73], which we have developed in section 3.2, when applied in different scenarios.

4.1. Datasets

In our experiments we have utilised the ShapeNet7C and the ModelNet40 datasets. The former is a subset of the ShapeNet dataset [17], which is widely used in point cloud deep learning [71], [13], [32]. We have employed ShapeNet7C as our principal benchmarking dataset since it allows us to compare our results with the ones obtained in [32], [31], and in [63]. ModelNet40 [95] is mainly used in the experiments related to Neural Transformation Learning, either for the anomaly detection task or for the pre-train of the feature extractor.

4.1.1. ShapeNet7C

ShapeNet7C is composed of objects from 7 different classes and it is a subset of the wider ShapeNet [17], which contains 55 classes. This latter dataset however suffers from severe class imbalance, with some classes that contain less than one-hundred training samples and others that contain more than five-thousands of them. Since each anomaly detection model is trained using solely data belonging to one class (that is regarded as the normal class) this can lead to potential issues. In particular, since the availability of data is crucial for a deep learning method [54], the models trained on the less populous classes are not expected to reach the same level of performance as the ones trained using thousands of samples. To avoid this problem we therefore consider only the 7 most represented classes of ShapeNet and we build a dataset with them, which we refer to as *"ShapeNet7C"*.

Table 4.1: ShapeNet7C: number of training samples and test samples for each class

	Class	Train	Test
0	Airplane	3232	807
1	Car	2812	702
2	Chair	5388	1347
3	Lamp	1848	461
4	Table	6716	1678
5	Sofa	2524	831
6	Rifle	1897	474
	tot.	24417	6100

The classes and the number of both training and test samples from ShapeNet7C are reported in Table 4.1.

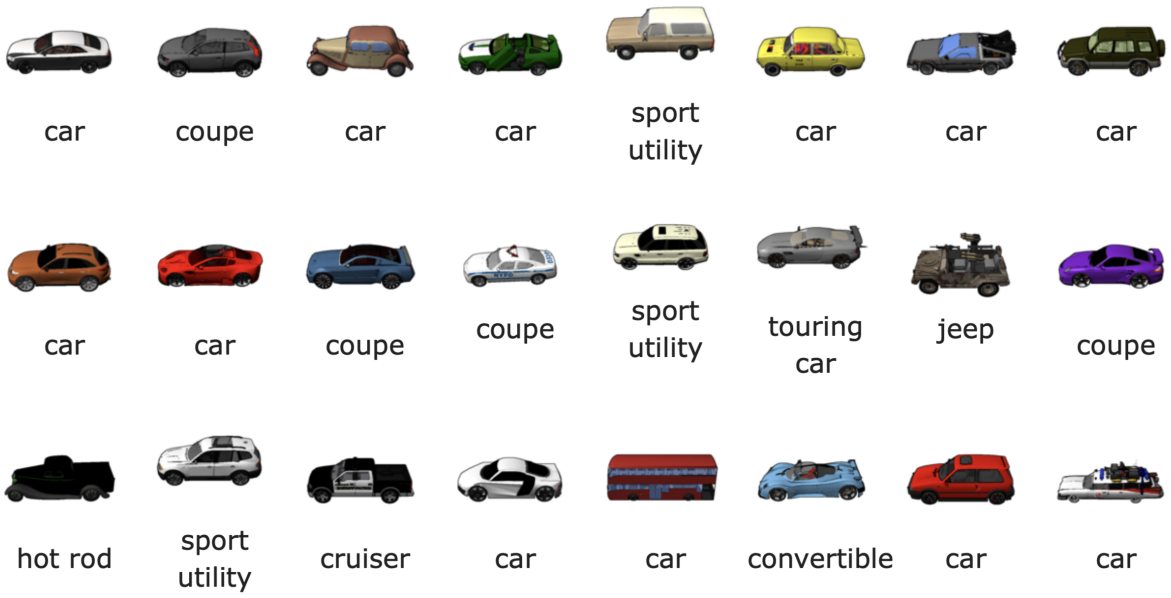


Figure 4.1: Examples of objects from the category "cars" of ShapeNetCore [17]

In addition, we have to mention that in this dataset there is a rather high variability inside each class. This is because one of the aims of ShapeNet is actually to comprehend all the objects that are characterised by a specific condition that makes them belong to that certain category [17].

More precisely, the concept that defines each class, as reported in the official website[1], is:

0. Airplane: *"an aircraft that has a fixed wing and is powered by propellers or jets";*
1. Car: *"a motor vehicle with four wheels; usually propelled by an internal combustion engine";*
2. Chair: *"a seat for one person, with a support for the back";*
3. Lamp: *"a piece of furniture holding one or more electric light bulbs";*
4. Sofa:*"an upholstered seat for more than one person";*
5. Table: *"a piece of furniture having a smooth flat top that is usually supported by one or more vertical legs";*
6. Rifle: *"a shoulder firearm with a long barrel and a rifled bore"*

Figure 4.2: Subdivision of the objects in the class "chair" of ShapeNet [17]

For this reason, inside the class "lamp" are present table lamps, chandeliers, floor lamps,

[1]https://shapenet.org/taxonomy-viewer

wall lamps, and spotlights. Inside the category "car", besides the Porsche 911, are included also ambulances, emergency vehicles, military vehicles, and even double-deckers. In the class "chair" are present all sorts of chairs, from dining chairs, armchairs, lawn chairs, and chaise longues, as we can see from Figure 4.2.

The point clouds contained in ShapeNet7C are synthetically generated, meaning that they do not come from scans of real objects, but rather from CAD models. The ShapeNet dataset is born as a collection of 3D meshes in the form of CAD files. A point cloud has been obtained from each mesh by sampling points from the surface of the object with a probability proportional to the area of each face, after that, the position of the points is slightly jittered. The point clouds are rescaled in a unit ball as done in [69], [13] and [31]. For our experiments we have used 1024 points for each shape. The most notable consequence of this procedure is that the point clouds have a uniform density, therefore they are not present regions of lower density of points that would make more challenging the learning process. Moreover, the objects are not occluded nor have parts missing, differently for instance from what happens with real-life datasets such as ScanNet [21]. Therefore, if from one side the objects are characterised by a high variability within the class, from the other their synthetic nature eases the learning process.

Another fact that has to be mentioned about ShapeNet7C is that the feature function of each point cloud is identically equal to the constant "1", namely $\varphi \equiv 1$. From this it follows that when dealing with this dataset, they are the coordinates of the points that play the most important role.

4.1.2. ModelNet40

Another dataset that we have used for our experiments is ModelNet40 [95]. It comprehends a vast collection of objects of everyday use divided into 40 categories. More precisely, it contains 12311 elements, of which 9843 constitute the training set and 2468 of them are used for testing purposes. In our experiments we always adopt the official training-testing split. A detailed list of the classes of ModelNet40 can be found in Table 4.2, where there are reported also the cardinality of the training data and of the test data for each class.
Similarly to ShapeNet, the objects are synthetically generated, therefore the shapes are complete, they are not occluded and the point clouds have a uniform sampling density. These characteristics make it very suitable for tasks such as multi-class classification. ModelNet40 [95] is probably the most widely used dataset in the point cloud deep learning scenario [69], [71], [13], [4], [60]. The reasons of its success are to be found in the

nature of its elements and in the fact that it represented one of the first comprehensive datasets of such 3D objects.

Also in this dataset the features of the point clouds are constant, that is $\varphi \equiv 1$. Within each class there is a rather high variability, although the latter is not as high as the one present in ShapeNet7C. Nevertheless, many classes have a very low cardinality, with some of them being made of less than eighty training samples. Because of this, most of the categories are not really suited to be regarded as the "normal class" in a deep anomaly detection task. In addition, the most numerous classes are almost identical to the ones of ShapeNet7C. Nevertheless, we employ this dataset in the experiments related to our Neural Transformation Learning method for point clouds, especially for the pre-train of the feature extractor.

Table 4.2: ModelNet40: number of training samples and test samples for each class

Class	Train	Test	Class	Train	Test	Class	Train	Test
Airplane	626	100	Dresser	200	86	Range hood	115	100
Bathtub	106	50	Flower pot	149	20	Sink	128	20
Bed	515	100	Glass box	171	100	Sofa	680	100
Bench	173	20	Guitar	155	100	Stairs	124	20
Booksheld	572	100	Keyboard	145	20	Stool	90	20
Bottle	335	100	Lamp	124	20	Table	392	100
Bowl	64	20	Laptop	149	20	Tent	163	20
Car	197	100	Mantel	284	100	Toilet	344	100
Chair	889	100	Monitor	465	100	TV stand	267	100
Cone	167	20	Night stand	200	86	Vase	475	100
Cup	79	20	Person	88	20	Wardrobe	87	20
Curtain	138	20	Piano	231	100	Xbox	103	20
Desk	200	86	Plant	240	100	*tot.*	9843	2468
Door	109	20	Radio	104	20			

4.2. Figures of merit

Anomaly detection can be seen as a binary classification problem, where the aim is to assign the label "1" to anomalies and the label "0" to elements belonging to the normal class. The output of all the models that we train is an anomaly score $\mathcal{A}_S$ which indicates the magnitude of anomalousness of each input point cloud $\mathcal{P}$. Then, we set a threshold

ν and regard all the elements whose anomaly score is less than ν as normal and all the others as anomalous. Therefore, we can use traditional metrics for binary classification such as True Positive Rate and False Positive Rate. In particular, instead of considering a fixed threshold, we compute the trend of the true (TPR) and of the false positive rate (FPR) when varying the threshold ν. These values are then used to construct the Receiver Operating Characteristics curve, better known as ROC curve. The Area Under the Curve (AUC) is then employed as a metric to assess the performance of the methods. This standard framework allows us to make our models comparable to the rest of the literature such as [32] and [63].

More rigorously, given a set of elements containing both normal and anomalous instances, and a classifier that assigns to them the labels $\{0, 1\}$, we can introduce the following quantities:

- **True Positives (TP)**: the number of anomalies correctly classified as such, hence the ones to which the classifier assigns the label "1"
- **False Positives (FP)**: the number of normal elements which are classified as anomalous, i.e. with "1", while in reality they belong to the normal class
- **False Negatives (FN)**: the number of anomalies to which the classifier wrongly assigns the label "0"
- **True Negatives (TN)**: the number of normal elements which are correctly classified as such, namely the ones to which the classifier assigns the label "0"

Table 4.3: Confusion Matrix

		Real class: **Positive**	Real class: **Negative**
Predicted class	**1**	**TP** True Positives	**FP** False Positives
	0	**FN** False Negatives	**TN** True Negatives

The **TP** and the **TN** correspond to the elements that are correctly classified, whereas the **FP** and the **FN** are the ones that are wrongly classified. For better understanding, all

these quantities can be effectively arranged in a matrix, which goes under the name of Confusion Matrix, as in Table 4.3.

After that, we can define the *True Positive Rate (TPR)* as:

$$TPR = \frac{TP}{TP + FN}$$

namely the ratio between the True Positives and the totality of the real positive elements, including hence the ones that are misclassified. The **TPR** is also known as *recall*, *sensitivity* or, in some cases, as *hit rate*.

Another quantity that can be defined is the *specificity*:

$$specificity = \frac{TN}{TN + FP}$$

Strictly related to the *specificity* is the *False Positive Rate (FPR)*:

$$FPR = 1 - specificity = \frac{FP}{TN + FP}$$

Once the above is established, given a binary classifier, they are computed the values of **FPR** and of the **TPR** when varying the threshold ν. Subsequently, the pairs obtained are plotted in a graph. The resulting curve takes the name of Receiver Operating Characteristics (**ROC**) curve. From the definition of True Positive Rate and of False Positive Rate, it follows that both of them are numbers in the interval [0, 1], hence the ROC is contained in the square $[0, 1] \times [0, 1]$. The integral on [0, 1] of the **ROC** goes under the name of **AUC** (or, alternatively, of **AUROC**) and it can be employed as an evaluation metric [30].

An important remark that has to be made about the AUC is that this metric is invariant with respect to the class skew of the data, namely the proportion between the anomalies and the normal instances. In addition, it does not depend on the range of the values assumed by the anomaly score function $\mathcal{A}_S$, which makes it possible to compare very different models. For instance, it enables the confront between models whose output is a probability, hence in [0, 1], and those whose anomaly score function assumes values in the whole $\mathbb{R}$.

The AUC is a measure of how well a model is able to separate the data into two classes. High values indicate excellent separation capabilities, whereas low values signify that the model is not very effective.

Another nice property of this metric is that a random guesser, namely a model which assigns the labels to the elements completely at random, makes register a value of AUC equal to 0.5. From this, we can deduce that a model whose AUC is less than 0.5 performs worse than one without any knowledge of the data, which is rather alarming. The maximum value of AUC that can be reached is 1 and it corresponds to the optimal classifier.

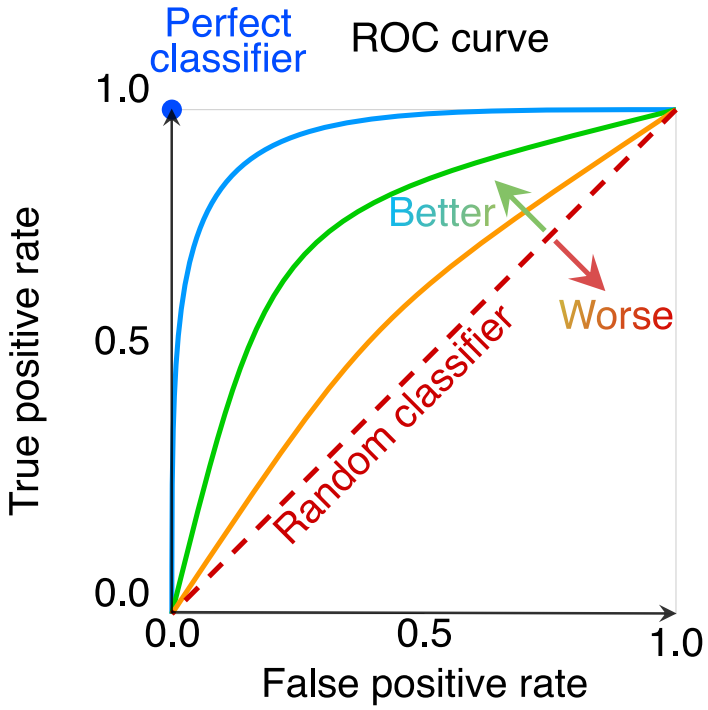


Figure 4.3: Depiction of some examples of ROC curves. A higher value of AUC indicates a better classifier. The random classifier corresponds to the red dashed line

4.3. Competing methods

After explaining the metric that we are using for evaluation, we now illustrate the methods to which we compare to. The field of anomaly detection on point clouds is a rather unexplored world, hence there is not a plethora of works that address this task. To the best of our knowledge, the followings are the sole methods that address this problem in a similar way to how we intend to do it:

- **IFOR** [32]: it is an Isolation Forest [58] on the Global Orthographic Object descriptors (GOOD) [46]. GOOD are handcrafted features extracted from each point cloud. This is a shallow method and hence it is employed as a baseline, following what has been done in [32];
- **VAE** [63]: this method consists of a Variational Autoencoder for 3D point cloud

anomaly detection and relies on the comparison between the original item and a reconstructed version. In fact, this model is trained to compress and subsequently reconstruct the point clouds belonging to the normal class;

- **DeepSVDD for point clouds** [31]: it is an extended version of the original DeppSVDD by Ruff et al [78]. Its aim is to map all the normal instances in an enclosing hyper-sphere in a certain latent space;
- **Self-supervised using geometric transformations** [32]: this method has been developed in [32] and it is an application to point clouds of the original self-supervised approach proposed in [34]. It is based on training a classifier to distinguish several rigid transformations applied to the input. In [32] as set of geometric transformations have been employed 8 rotations along a fixed horizontal axis.

Furthermore, it has to be noted that all the methods reported above adopt the AUC as evaluation metric, therefore we are able to compare the results in a natural way. This constitutes also one of the main reasons why we have decided to employ said metric.

4.4. Deep Robust Once Class Classification for point clouds

In this section, we illustrate the experiments that we have performed using the Deep Robust One Class Classification method for point cloud anomaly detection that we have introduced in section 3.1. We explore different architectures based on the composite layers [32] in order to evaluate how the latter perform in this scenario. In addition, we experiment with other networks, more importantly with ConvPoint [13]. A similar comparison between CompositeNet and ConvPoint networks can be found in [31] for the case of the DeepSVDD method. Finally, we compare the results achieved by our extension of DROCC to point clouds with the ones of the rest of the literature.

4.4.1. Experimental setup

In order to make the experiments more comparable, we have adopted a common experimental setup for every one of them. We decided to set the cardinality of each point cloud to 1024, similarly to [32]. In these experiments we have employed the ShapeNet7C dataset that we have presented in section 4.1.1. Since the classes that we consider contain a high number of samples, every model is trained for 15 epochs. Empirically we have found that after such time the value of the AUC stabilises and training them for more time does not

yield better performance. The batch size has been fixed to 64. All the models have been trained using Adam [48] optimiser with an initial learning rate $lr = 0.0005$.

DROCC Hyperparameters

We have extensively tuned the hyper-parameters of our DROCC method by conducting several experiments and trying different combinations of them. The network that we have employed for such purpose is the ADCompositeNet3, whose architecture is reported in Table 3.1. After that, we have taken the best performing hyper-parameters and we have adopted them in every model. Furthermore, since we are conscious that the concept of "best performing" hyper-parameters may depend on the specific model, we have also explored whether there were better performing hyper-parameters for each one of them. Nevertheless, we found that there were no significant differences in terms of performance between the latter and the former, hence we have decided to adopt the hyper-parameters tuned with the ADCompositeNet3 network for every model.

Therefore, the hyper-parameters that we have used are:

- Mixing coefficient: $\mu = 0.4$:
- Regularisation coefficient: $\lambda = 0.15$
- Standard deviation of h_0: $\sigma = 0.3$
- Radius: $r = 28$
- Upper bound: $\gamma = 2$:
- Ascent step number: $S = 30$
- Step size: $\eta = 0.001$

As far as it regards the radius, its value is in accordance with what found in [36], where it is suggested that the best performing radius corresponds to $\bar{r} = \sqrt{dn}/2$. In this expression, dn represents the dimension of the input space, which in our case is $dn = 3 \cdot 1024$. The upper bound $\gamma = 2$ is also analogous to the one that has been adopted in [36] and we found that increasing it, for instance to $\gamma = 4$, was not beneficial to the performance.

4.4.2. DROCC using Composite Layers

As we have declared, one of the aims of this work is to explore the performance of the composite layers when they are exploited by other anomaly detection methods on point clouds. Therefore we have used them to build several networks that we have then employed

in the role of the function $\phi_\theta(\cdot)$ of the method presented in section 3.1.

Of the networks presented in 3.1.3 we have tuned the various hyper-parameters and we found that the configuration which yields the best performance overall corresponds to:

- $J_0 = 16$
- $M = 64$
- $K = 25$

Where J_0 decides the number of output features of each composite layer, M is the number of centers of each spatial function s and K is the output dimension of the latter. Hence, from now on, we assign the values here reported to these hyper-parameters for each CompositeNet that we create, unless explicitly specified otherwise.

Results

Class		ADComposite5	ADComposite3	ADComposite3b	ADComposite3 (Aggr.)
0	Airplane	0,7925	0,7909	**0,8182**	0,7709
1	Car	**0,6936**	0,6370	0,6479	0,6311
2	Chair	0,5871	**0,7336**	0,7315	0,7266
3	Lamp	0,6426	**0,6821**	0,5942	0,5895
4	Table	0,5864	0,7816	0,7524	**0,8198**
5	Sofa	**0,7564**	0,6935	0,6999	0,6420
6	Rifle	0,8537	0,8515	0,8847	**0,9015**
Average AUC		0,7018	**0,7386**	0,7327	0,7259
Average rank		2,57	**2,29**	**2,29**	2,86

Table 4.4: Results of the anomaly detection task on ShapeNet7C for different architectures based on the composite layers. All the values are in terms of AUC

The results of the anomaly detection task that we obtained on ShapeNet7C when varying the networks' architecture are reported in Table 4.4. As we can see, there is not a clearly predominant architecture above the others. This does not represent a huge surprise since all the networks are similar as they are based on the composite layers. The most notable difference is that on average the ADCompositeNet5 tends to perform worse than the others. This can be explained by the fact that it has a more complex architecture constituted by five layers instead of the three layers of the other networks. On this basis, we can speculate that the DROCC method for point cloud anomaly detection reaches better results when simple networks are employed in the role of the function $\phi_\theta(\cdot)$.

Moreover, it seems that the convolutional composite layers yield slightly better results than their aggregate counterpart. Once again this behaviour is probably due to the more

complex structure of the latter. On the other hand, we did not find a great difference when the ADCompositeNet3 has one unique fully connected layer or when it has two of them. It has to be taken into account that the number of parameters in these cases is not dramatically different.

For sake of completeness, we have experimented also with a network constituted by 4 convolutional composite layers, called without much fantasy *ADCompositeNet4*. Its architecture resembles the one of ADCompositeNet5 in Table 3.2, with the sole difference that the fourth layer is missing. However, its results are very similar to the ones achieved by ADCompositeNet5, hence we decided to omit them as they would not have been much informative.

In essence, what we can deduce from the results that we obtained is that the method we developed basing ourselves on the DROCC approach tends to yield better results when simple architectures are involved. In fact, the best performing configuration corresponds to the simplest one that we have tried, namely the one with only three composite layers. Furthermore, the convolutional version is to be privileged in such models. This conclusion is for instance in contrast with the results that have been found in [32] regarding the classification task. In that scenario it was a five layers CompositeNet composed of aggregate composite layers the best performing network.

4.4.3. Comparison among different types of network

A second group of experiments is aimed at exploring how the performance of the Deep Robust Once Class Classification method for point cloud AD that we have developed is affected by the type of network that is employed. To this end we have considered, besides the ADCompositeNet3 that we have earlier introduced, a ConvPoint [13] network and a network based on the architecture of PointNet [69]. The functioning principles of ConvPoint [13] are similar to the ones of CompositeNet [32], as both networks implement a point convolutional operator. Furthermore, ConvPoint represents one of the sources of inspiration for CompositeNet [32] and the two networks are direct competitors. All of this adds more value to the comparison between them, as we aim to discover which architecture is more suitable to our method.

For the comparison, we have considered an ADCompositeNet3 since, as shown in section 4.4.2, it proved to be the best performing architecture based on the composite layers. In addition to ADCompositeNet3, we consider the ADConvPoint in Tab.3.3. The number of centers is chosen to be 64 for both the networks, whilst we decided to set the output dimension of the semantic function of ADCompositeNet3 to $K = 25$.

These two networks are designed to be as similar as possible, although obviously there are some differences between the two, the most relevant of which is the number of learnable weights, as the implementation of the convolutional layers is different. More precisely, the ADConvPoint3 counts 457601 learnable parameters, whereas the ADCompositeNet3 just 140423, which is around three times less.

Furthermore, we consider the PointNet network that we have represented in Tab.3.4, which counts 736641 parameters because of the fully connected layers. The number of parameters for each network is reported in 4.5.

Table 4.5: Comparison between the number of parameters

Network	Number of parameters
PointNet	736641
ADConvPoint3	457601
ADCompositeNet3	140423

Results

In Table 4.6 are reported the results that we obtained.

Table 4.6: Results in terms of AUC of our DROCC method for point cloud anomaly detection when different networks are employed

Class		ADConvPoint3	PointNet	ADCompositeNet3
0	Airplane	0,6738	**0,8067**	0,7909
1	Car	0,5218	**0,7635**	0,6370
2	Chair	0,6374	0,6024	**0,7336**
3	Lamp	0,4829	0,6734	**0,6821**
4	Table	0,6973	0,6522	**0,7816**
5	Sofa	0,5465	0,6711	**0,6935**
6	Rifle	0,7440	**0,8612**	0,8515
Average AUC		0,6148	0,7186	**0,7386**
Average rank		2,71	1,86	**1,43**

From that we can see that, in this experiment, ADCompositeNet3 is the best performing network for the Deep Robust One Class Classification on point clouds. PointNet, despite

the significantly higher number of parameters, achieves similar performance, making registering a difference of 0.02 points of AUC. On the other hand, ADConvPoint3 represents the worst performing network among the ones that we have considered.

Most remarkably, in spite of the similar structure, ADConvPoint3 is outperformed by the ADCompositeNet3 by more than 10 % of AUC on average. As we have alluded to in the previous section 4.4.2, this is probably due to the higher number of parameters, which makes the optimisation process more challenging.
Nevertheless, when we take into account that PointNet is the network with the highest number of parameters, we deduce that the latter is not the only aspect that influences the success of the method. Indeed, our PointNet network has roughly twice as many parameters as ADConvPoint3 and it is still able to reach significantly better results than the latter. We can explain this behaviour with the fact that the PointNet architecture is the simplest of the three, which then makes it simpler to be trained. On the contrary, ADConvPoint3 is characterised by a more complex design made of convolutional layers. Thus, the cause of its scarce success probably resides in these more sophisticated layers, which also comprehend most of the parameters of the network.

In addition, we have also carried out some experiments employing an advanced PointMLP network [60], which adopts a residual architecture. This network is currently one of the top performing models in the classification task on the ModelNet40 dataset [95], hence the choice. However, the results we obtained were below our expectations, as they were slightly worse than the ones of PointNet.

Collecting all the observations from the above experiments together, we can advocate that the optimisation process of the method we developed can be quite insidious. In order to avoid such complications we thus recommend employing a neural network with a simple design in the role of $\phi_\theta(\cdot)$ and eventually prefer a low number of parameters. Sophisticated architectures that count millions of learnable parameters, despite they may perform excellently in other tasks, are not well suited to our DROCC method for anomaly detection on point clouds.

4.4.4. Comparison with other methods

Once that we have identified the best configuration in terms of both type and architecture of the network, this last section is devoted to verifying how our anomaly detection method for point clouds compares to the rest of the literature. To this purpose, analogously to what has been done in [32], we consider as baseline an Isolation Forest [58] that relies on the Global Orthographic Object Descriptor (GOOD) [46] of the point clouds. This

method is regarded as "IFOR" in Table 4.7 and the related values of AUC have been directly reported from [32].
In addition, another method that is taken into account is the Variational Autoencoder presented in [63]. This is a deep reconstruction-based model and in Table 4.7 are represented two versions of it; the number next to the name "VAE" indicates the number of points used for the reconstruction of the original point cloud. Since the code is not publicly available, we report the official results from the original paper [63].

A third method that we have considered is the extension developed in [31] of the DeepSVDD model [78] to the scenario of unsupervised anomaly detection on point clouds. This comparison is of particular relevance since the moving principles of DROCC [36] are similar to the ones of the DeepSVDD [78]. However, one of the aims of DROCC is to develop a method more robust to the representation collapse than the DeepSVDD, as declared by its authors [36]. In this case, we have taken the code used by [31] in his work and we have run our experiments adopting the experimental setup that we have introduced in section 4.4.1. Moreover, we have employed as neural network the same ADCompositeNet3 that we have used with our DROCC per point clouds method. On this ground, we believe that we have made the comparison between these two methods as fair as possible. The results that we obtained are then reported in Table 4.7, alongside the ones of our method based on DROCC.

Table 4.7: Comparison between anomaly detection methods for point clouds on the ShapeNet7C dataset. All the values are in terms of AUC

Class		IFOR [58]	VAE 2048 [63]	VAE 4096 [63]	DeepSVDD [31]	DROCC (Ours)
0	Airplane	**0,912**	0,716	0,747	0,6898	0,7909
1	Car	0,712	0,752	**0,757**	0,6217	0,6370
2	Chair	0,571	0,918	**0,931**	0,6758	0,7336
3	Lamp	**0,962**	0,903	0,907	0,641	0,6821
4	Table	**0,883**	0,834	0,839	0,6585	0,7816
5	Sofa	**0,986**	0,778	0,777	0,5834	0,6935
6	Rifle	0,475	0,286	0,382	0,7422	**0,8515**
Average AUC		**0,7859**	0,741	0,7629	0,6589	0,7386
Average rank		**2,14**	3	2,29	4,43	3,14

As we can see from Table 4.7, our DROCC method for point cloud anomaly detection performs on average better than the DeepSVDD method. Nonetheless, it is surpassed in performance by the IFOR, VAE 4096 and it is on par with the VAE 2048. The better results of the Variational Autoencoder [63] allegedly have to do with the fact that it uses

point clouds of cardinality 2048, whereas in our experiments we have always considered only 1024 points. In addition, this method totally fails on the class "rifle", probably due to reconstruction issues. Also the IFOR performs very poorly on this class, making record a value of AUC below 0.5. Quite surprisingly, on said class, both the DeepSVDD and our DROCC methods achieve their best results.

IFOR, which we have used as baseline, still shows very good results on several classes and it is the best performing method in this experiment. However, all considered, it does not emerge one method that consistently performs better than the others, as they all have their criticalities on certain classes. Indeed, only the DeepSVDD and the DROCC methods reach values of AUC above the 0.5 threshold on every class. This threshold is relevant since it is the value that would be achieved by a random guesser, hence without having any sort of knowledge about the data.

In addition, it can be noted that our DROCC method performs better than the DeepSVDD on all the classes of this dataset, achieving an average AUC of 8 % higher than the latter. This suggests that the former approach is more effective. Furthermore, the DeepSVDD method is the one that performs the worst in terms of average AUC, proving that it does not work very well in the context of point cloud anomaly detection. This observation is in accordance with what found by [31].

This said, also the DROCC method that we have proposed does not prove to be extremely effective overall, since it is outperformed on several classes by the shallow baseline IFOR. If from one side it is true that it does not make register values of AUC as low as the ones of IFOR on classes such as "rifle" or "chair", from the other it never achieves any value of AUC greater than 0.90. Taking into account that we have widely experimented with various networks and explored different architectures, we speculate that the reasons for its non-outstanding performance lie in the approach itself. In particular, we suspect that the adversarial search plays a major role in this since we advocate that the context of point cloud data is much more challenging than the ones of [36] and hence it does not result to be as effective.

4.5. Neural Transformation Learning for point cloud anomaly detection

The second method that we have developed for the anomaly detection task on point clouds is based on the approach of [73]. Our model is composed of three ingredients: a feature extractor, a set of learnable transformations, and an encoder. This section is devoted

to illustrating the various experiments that we have conducted and to investigating the results achieved by this method.

4.5.1. Hyper-parameters' tuning

In order to tune the hyper-parameters we have conducted several experiments on the ShapeNet7C dataset by and trying different values. In particular, we have investigated the following combinations:

- Batch size: {**128**, 256, 512}
- Number of transformations: {7, **15**}
- Layers of each transformation: {2, **3**}
- Type of transformations: {$feed\_forward$, **multiplicative**, $residual$}
- Encoder output dimension: {128, **256**, 512}

More specifically, we adopted a semi-grid-search approach, examining only the most prominent combinations and branching out values that were deemed to yield sub-par results. For instance, when we noted that the *feed forward* transformations performed worse than the other types, we did not investigate them further. This observation is also in accordance with what reported in [73].
In bold are highlighted the hyper-parameters of the combination that yielded the best result. From now on, unless otherwise explicitly specified, we adopt these hyper-parameters in all our experiments.

We also attempted to change the architecture of each transformation in:

$$FC1(1024,\, 512) \rightarrow FC2(512,\, 512) \rightarrow FC3(512,\, 1024)$$

However, we got slightly worse results than with the one in Fig 3.1.

4.5.2. Anomaly detection on ShapeNet7C

We have performed several experiments in order to assess the performance of the method that we have developed for the anomaly detection task on point clouds. In this section, we utilise as a benchmark the ShapeNet7C dataset that we have presented in 4.1.1.

Pre-training of the feature extractor

Our method makes use of a pre-trained feature extractor $\phi_{PC}(\cdot)$ which is used to process the input point clouds. The pre-training protocol that we have applied is the one we presented in 3.2.5. More specifically, since our aim in this case is to investigate the anomaly detection problem on ShapeNet7C, we pre-train our feature extractor on a subset of ModelNet40 [95].

To do so we have removed from the original ModelNet40 datasets the classes that this latter has in common with ShapeNet7C, that are:

"Airplane", "Car", "Chair", "Lamp", "Table" and *"Sofa"*.

For sake of thoroughness, we have removed from ModelNet40 also the classes *"Stool"* and *"Desk"*, since they might be very similar to *"Chair"* and *"Table"*, respectively.

The resulting dataset contains 32 classes and 8487 samples, of which 6645 of them constitute the training set and the remaining 1842 are used for the test set. Because it is made of 32 classes, we can refer to this dataset as "ModelNet32". When creating it, we maintained the original train-test split of ModelNet40 [95].

Table 4.8: **ModelNet32**: the 32 classes of ModelNet40 used in our experiments. We report the number of train and test samples. The number beside the name of each class refers to the label of the original class in ModelNet40 [95]

	Class	Train	Test		Class	Train	Test		Class	Train	Test
1	bathtub	106	50	**15**	flower pot	149	20	**27**	radio	104	20
2	bed	515	100	**16**	glass box	171	100	**28**	range hood	115	100
3	bench	173	20	**17**	guitar	155	100	**29**	sink	128	20
4	bookshelf	572	100	**18**	keyboard	145	20	**31**	stairs	124	20
5	bottle	335	100	**20**	laptop	149	20	**34**	tent	163	20
6	bowl	64	20	**21**	mantel	284	100	**35**	toilet	344	100
9	cone	167	20	**22**	monitor	465	100	**36**	tv stand	267	100
10	cup	79	20	**23**	night stand	200	86	**37**	vase	475	100
11	curtain	138	20	**24**	person	88	20	**38**	wardrobe	87	20
13	door	109	20	**25**	piano	231	100	**39**	xbox	103	20
14	dresser	200	86	**26**	plant	240	100		*tot.*	6645	1842

Using as a basis the architecture of the feature extractor reported in Table 3.5, we have constructed a network suited for the classification task. In particular, we have completed the last composite layer in Table 3.5 by adding a batch normalisation layer and a Leaky ReLU activation function after it. In such a way the former is coherent with all the other

composite layers. Furthermore, we have added a final fully connected layer at the end of the network, which serves as a classification head. This fully connected layer has input dimension 1024 and output dimension 32, where the latter corresponds to the number of classes of ModelNet32.

We have trained the created network on ModelNet32 for 50 epochs using Adam optimiser [48] of initial learning rate 0.001. After that, the classification head is discarded in order to obtain the architecture described in Table 3.5.

Results

In order to assess the effectiveness of our model we trained it to perform the anomaly detection task on the ShapeNet7C dataset. The architectures that we have employed for the transformations and the encoder are the ones that we presented in sections 3.2.6 (Img. 3.1 and Img. 3.2, respectively). To make our results less subject to variability, we train the model five times and in Table 4.9 we report the average AUC over the five runs, as well as the standard deviation.

Table 4.9: Results over five runs in terms of AUC of our NeuTraL method for point cloud anomaly detection. The values of AUC are averaged over the 5 runs and Std. dev refers to their standard deviation

Class		AUC	Std. dev.
0	Airplane	0,9988	0,000108
1	Car	0,9984	0,000040
2	Chair	0,9573	0,002195
3	Lamp	0,9627	0,000904
4	Table	0,9916	0,000259
5	Sofa	0,9882	0,000628
6	Rifle	0,9984	0,000094
Mean		0,9851	0,000310

As it can be seen from Table 4.9, our Neural Transformation Learning method achieves excellent performance. In particular, on four of the seven classes of the ShapeNet7C dataset, our model reaches values of AUC greater than 0.99. It has to be remembered that the maximum value of AUC that can be achieved is 1.0. Our model, especially on classes such as "airplane", "car" and "rifle", gets extremely close to it. In addition, all the standard deviations are very low, hence our model reaches these results with consistency and the variability of its performance is small.

Since we have adopted the AUC as evaluation metric, we can easily compare the performance of our model with the ones of the rest of the literature. The results are reported in Table 4.10.
More into the specific, **"IFOR"** indicates an Isolation Forest [58] on the Global Orthographic Object Descriptors [46] of the point clouds as in [32]. **"VAE"** stands for the Variational Autoencoder proposed in [63], whose values of the AUC are taken directly from the official paper. **"Self-Sup"** denotes the extension of the Self-Supervised method [34] to point cloud anomaly detection presented in [32]. As far as it regards the latter method, we have decided to take into consideration the results that the authors obtained using a CompositeNet based on the aggregate composite layers, as this is one of the best performing architectures they tested [32]. For sake of completeness, we report also the results that we obtained with the best configuration of our DROCC for point cloud method that we have discussed in the previous section 4.4.2. The former are denoted by **"DROCC"** in the table. In addition, we present also the results from the **"DeepSVDD"** model that we have trained on this dataset using the original code from [31], as we have done in section 4.4.4.

Table 4.10: Comparison between point cloud anomaly detection methods on ShapeNet7C. All the values are in terms of AUC

Class		IFOR [32]	VAE [63]	DeepSVDD [31]	DROCC (Ours)	Self-Sup. [32]	NeuTraL (Ours)
0	Airplane	0,912	0,747	0,6898	0,7909	0,9700	**0,9988**
1	Car	0,712	0,757	0,6217	0,6370	0,9720	**0,9984**
2	Chair	0,571	0,931	0,6758	0,7336	0,9410	**0,9573**
3	Lamp	0,962	0,907	0,6410	0,6821	0,4210	**0,9627**
4	Table	0,883	0,839	0,6585	0,7816	0,8540	**0,9916**
5	Sofa	0,986	0,777	0,5834	0,6935	0,9440	**0,9882**
6	Rifle	0,475	0,382	0,7422	0,8515	0,9770	**0,9984**
Average AUC		0,7859	0,7629	0,6589	0,7386	0,8684	**0,9851**
Average rank		3,43	4,00	5,43	4,29	2,86	**1,00**

From this comparison, it emerges that our Neural transformation Learning for point cloud anomaly detection method is the best performing one on the ShapeNet7C dataset. In fact, it reaches the highest AUC on every class of said dataset, setting the new state of the art. More precisely, it achieves a mean AUC of 0.9851, which improves the previous mean AUC of more than 10 % and it approaches the perfect value of 1.
In addition, it can be noted that our method outperforms the shallow baseline IFOR [32], VAE [63], DeepSVDD [31] and DROCC (Section 3.1) by a very large margin. Moreover, differently from the other methods, it is much more consistent on every class, as its lowest value of AUC is equal to a solid 0.9573, which still represents an impressive performance.

The method that reaches the best results, besides ours, is the extension proposed in [32] of the self-supervised approach originally proposed in [34]. This method makes use of a set of geometric transformations which, differently from ours, has to be decided a-priori. In the original paper [34], this approach is applied to image data and the considered transformations include flips, rotations, and translations. Of these transformations however only rotations can be adopted in the case of point cloud data, therefore in [32] the authors opted for 8 rotations along a fixed horizontal axis. Indeed, as we have explained in section 1.3, the methods that process point clouds are required to be invariant to rigid translations, hence translations cannot be employed. Furthermore, many objects of the ShapeNet7C dataset are symmetric (for instance tables), which implies that flips do not yield transformed versions distinguishable from the original one. As far as it regards rotations, since the objects are not oriented in a canonical pose, usually only rotations w.r.t a horizontal axis can be employed. Taking these observations into account, we deduce that the set of transformations that can be employed for the case of point clouds is much more restricted than the one used on images.

From Table 4.10 we can see that the Self Supervised method of [32] fails on the class *"lamp"*, making register an AUC of less than 0.5, namely worse than a random guesser. We speculate that the motivation behind such behaviour is that the category *"lamp"* of ShapeNet7C comprehends very different types of such objects, including table lamps and chandeliers, as in [32]. Since the latter are very similar to the upside-down version of the former, when rotations of 180° degrees around a horizontal axis are used, they become indistinguishable by the classifier, thus jeopardising the learning process. On the contrary, since our method does not rely on pre-defined transformations, it is able to achieve a remarkable value of 0.9627 of AUC on that same class.

Discussion and limitations

The method that we have introduced for the anomaly detection task by extending the approach of [73] proved to be extremely successful. We hypothesise that these outstanding results are also thanks to the pre-trained feature extractor $\phi_{PC}(\cdot)$ and we believe that the pre-training procedure plays a major role in it.
From this, we can conclude that the pre-training strategy can be very useful in the context of anomaly detection, even when the classes used for the pre-training are different from the ones used for the latter purpose. This observation holds as long as the features extracted by $\phi_{PC}(\cdot)$ generalise well.

The pre-training procedure is probably one the of main criticalities of our method, as it requires additional data compared to alternatives. If it is true that these data do not

necessarily have to be exactly like the ones used during the anomaly detection task, they still have to yield general features. This fact might be not always granted, hence it has to be chosen a suitable pre-training dataset.

Another criticism that may arise concerns the amount of data necessary for the pre-training, as they could be problematic to be collected. To this regard, we can try to answer by taking into account the case that we have analysed. In fact, we have performed the pre-training on a dataset constituted by 32 classes of ModelNet40 [95], for a total of 6645 training samples. As we can see from Table 4.8, many categories count very few elements, with some classes such as "bowl", "cup" and "wardrobe" that count less than 100 instances. Furthermore, each class contains on average just 200 training samples. When compared to other datasets in the deep learning field, such as ImageNet [22], that contain millions of samples, these numbers can be argued to be low. Nonetheless, the features extracted seem to generalise very well, as we have proved. In the light of the above, we can sustain that it is not necessary a gargantuan amount of data for pre-training purposes.

4.5.3. Anomaly detection with few samples on ModelNet

In the previous section, we have assessed the anomaly detection capabilities of the method we developed using as benchmark the ShapeNet7C dataset. In that case, our model takes advantage of pre-trained feature extractor on ModelNet32. Here we devote our attention to exploring the consequences that arise when we invert the roles of the aforementioned datasets.

We believe this setup to be more challenging for two main reasons. The first one is that the feature extractor is pre-trained on a dataset that contains objects from only seven categories, versus the 32 of before. This means that $\phi_{PC}(\cdot)$ is exposed to a more limited selection of objects and consequently of features. Therefore, it is interesting to enquiring whether the extracted features still generalise well or not.

The second reason resides in the fact that the anomaly detection model is trained on a much smaller dataset. Indeed, as we have also discussed in section 4.5.2, the latter contains only 6645 training samples. In addition, some of its classes are made of less than one-hundred elements (Table 4.8). We recall that all the models under consideration in this work that address the anomaly detection task are trained solely using data from the normal class. This means that our model, on classes such as "bowl", "cup" and "wardrobe", is trained using less than ninety samples.

Pre-training of the feature extractor

The protocol that we have adopted for the pre-training of the feature extractor is analogous to the one we followed in section 4.5.2. Furthermore, we have employed the same architecture for $\phi_{PC}(\cdot)$, namely the one depicted in Table 3.5. In this case, we train the feature extractor on the ShapeNet7C dataset, which is constituted by 7 classes [2].

Results

The architectures that we have employed for the learnable transformations and for the encoder are the ones that we have illustrated in Img. 3.1 and Img. 3.2, respectively. We have trained our models for 100 epochs using Adam optimiser [48] of initial learning rate equal to 0.0002.

Similarly to what we have done for the case of ShapeNet7C in section 4.5.2, we have trained all the models five times and in Table 4.11 we report the average AUC over the five runs, as well as the standard deviation. In addition, we indicate also the cardinality of each class, which corresponds to the number of samples on which each associated anomaly detection model is trained.

From what we can see in Table 4.11, our Neural Transformation Learning model for anomaly detection on point clouds achieves excellent results on basically every class considered. The mean of the AUC over the whole 32 classes is of 0.9456, which is an impressive result. Furthermore, all the standard deviations of the AUC over the five runs are very small, which signifies that these results can be regarded as reliable.

In particular, our model is able to reach values of AUC above 0.99 on several classes such as: "bed", "bottle", "guitar", "keyboard", "laptop", "monitor" and "toilet". What is more remarkable is that on "keyboard" it makes register an AUC of 0.999 and on "laptop" of even 0.9997, values astoundingly close to the perfect score of 1. In addition, these latter two classes count a little less than 150 training samples, which is a rather low number, especially when compared to the cardinality of the classes of ShapeNet7C (Tab. 4.1) and to other deep learning datasets [22], [50], [55].

The only category on which our method seems to perform not as well as on the others is "radio", on which it obtains a value of AUC of “only” 0.7835. We hypothesise that such behaviour is related to the low cardinality of said class, since it counts just 104 elements.

[2] We have decided to proceed in this way and not by removing from ShapeNet7C the intersection of the classes between ModelNet40 and ShapeNet7C since otherwise there would have been only one class left: "rifle"

Table 4.11: Results in terms of AUC of anomaly detection on ModelNet32 obtained by our model. The values of AUC are averaged over the 5 runs and Std. dev refers to their standard deviation. Next to them we report the number of training samples of each class.

Class		AUC	Std. dev.	#Train
1	bathtub	**0,9672**	0,004525	106
2	bed	**0,9939**	0,000570	515
3	bench	**0,9259**	0,003129	173
4	bookshelf	**0,9817**	0,001611	572
5	bottle	**0,9949**	0,000367	335
6	bowl	**0,9753**	0,003333	64
9	cone	**0,9787**	0,000870	167
10	cup	**0,9255**	0,000609	79
11	curtain	**0,9788**	0,001358	138
13	door	**0,9659**	0,003784	109
14	dresser	**0,9680**	0,001279	200
15	flower pot	**0,8536**	0,007559	149
16	glass box	**0,9690**	0,000302	171
17	guitar	**0,9940**	0,001005	155
18	keyboard	**0,9990**	0,000512	145
20	laptop	**0,9997**	0,000068	149
21	mantel	**0,9481**	0,003124	284
22	monitor	**0,9951**	0,000453	465
23	night stand	**0,9503**	0,000966	200
24	person	**0,9704**	0,001575	88
25	piano	**0,9073**	0,000879	231
26	plant	**0,9556**	0,002248	240
27	radio	**0,7835**	0,006306	104
28	range hood	**0,9641**	0,004160	115
29	sink	**0,8296**	0,008381	128
31	stairs	**0,8106**	0,020754	124
34	tent	**0,9713**	0,001122	163
35	toilet	**0,9947**	0,000683	344
36	tv stand	**0,9575**	0,000800	267
37	vase	**0,9397**	0,001043	475
38	wardrobe	**0,9188**	0,009396	87
39	xbox	**0,8923**	0,008521	103
Average AUC		**0,9456**	0,001429	6645

Moreover, also the variability within the class might play a role in it, since it contains quite different types of radios. Additionally, the features extracted could be similar to the ones of some other categories and therefore confound our method.

Another observation that can be made concerns the number of training samples. In fact, from Table 4.11 we can see that our model achieves notable performance on the categories "bowl", "person", "cup" and "wardrobe". This happens in spite of the fact that they contain less than one hundred elements. Furthermore, it is very interesting to note that on the two former classes, namely "bowl" and "person", our model makes register values of AUC above 0.97 despite being trained on less than ninety samples. On the two former categories it obtains values around 0.92, which are still noteworthy.

Overall, we can notice that on the most numerous categories the performance seems to be better than on the ones with fewer samples. Indeed, all the values of AUC less than 0.90 are recorded on classes that contain less than 150 elements. This was to be expected since deep learning methods strongly rely on a huge amount of data for their success [54]. On the contrary, it is more surprising that our model achieves outstanding results even when few training samples are being used.

Discussion and limitations

In this section, we have discussed the application of our model to the anomaly detection problem on ModelNet32. We regard this case to be more challenging than the previous one since the feature extractor is pre-trained on fewer classes and, more importantly, since the model used to perform the anomaly detection is trained using only a few samples. Nonetheless, from the obtained results we have shown that our method proved to be very effective also in this scenario, consistently reaching remarkable values of AUC.

To the best of our knowledge, we are the first to address the anomaly detection problem on classes from ModelNet40 in a deep manner, as other works have probably deemed the number of training samples to be insufficient for such a purpose. Nevertheless, our model, by leveraging a pre-trained feature extractor, demonstrated to be able to achieve remarkable performance even in this context.

At this point, we believe that the main criticism that could be moved regards the pre-training strategy, as it is an operation that requires additional data. To this end, we advocate that such procedure proved to be very effective in both the scenarios that we have examined. In particular, in the first one we have performed it on a relatively small amount of data and the results we obtained ShapeNet7C outperformed every other considered method by a large margin. On the other hand, in the second scenario we have pre-

trained the feature extractor on just 7 categories and then we have performed the anomaly detection task using normal classes of even less than ninety samples. In spite of this, our model still confirmed to be substantially effective on every class of ModelNet32. On this ground, we speculate that the amount of pre-training data and the number of different categories used are not extremely fundamental to the good success of our anomaly detection model, as long as the extracted features can generalise well.

5 | Conclusions and future developments

In this work, we have addressed the anomaly detection task on point clouds by designing two different methods. The first of them, namely our Deep Robust One Class Classification method, extends the approach proposed by [36] to the case of point clouds. This has been done by developing a suitable network architecture, as well as a specific adversarial search, as we have explained in section 3.1. Furthermore, we have performed several experiments in order to evaluate its performance. We have discussed the best performing architecture and we have compared the results with the ones of other competing methods, which adopt different approaches to anomaly detection. Overall our DROCC model for point cloud shown better performance than the DeepSVDD model [31], which is the most similar method among the ones we have considered. This said, the results that it achieves are not very outstanding, but rather in line with the ones of most of the present methods, as showed in 4.4.4.

The second method that we have developed is inspired by [73], in particular, we adopt the loss function which has been proposed in said paper. The model that we have developed, taking advantage of a pre-trained feature extractor and of a set of learnable transformations, proved to be able to achieve astounding results. In fact, as shown in section 4.5, our model outperforms all the methods of which we are currently aware of. More precisely, on the ShapeNet7C dataset it reaches an average AUC of 0.9851, value that is astoundingly close to the maximum achievable, namely 1. In so doing it improves by a large margin the previous state of the art and, on some classes of this dataset, it even makes register values of AUC greater than 0.99.

Furthermore, we have employed our Neural Transformations Learning method also for the point cloud anomaly detection task on 32 classes from ModelNet40 [95]. Because of the very low cardinality of many of these classes, we have reasons to believe that this scenario is very challenging for a deep learning method. In spite of the odds, our model achieved noteworthy results on every class, even the ones with less than one-hundred training

samples. From this we deduced that, by leveraging a pre-trained feature extractor, our method is able to reach remarkable performance in the anomaly detection task even when very small training sets are used.

All things considered, we reckon that in the next years many more methods will be proposed to address the task of anomaly detection and some of them will eventually be better than ours. For instance, we believe that it would be interesting to develop methods that effectively learn the transformations directly on point clouds without the support of a feature extractor. We have attempted to make a step in this direction, however, the results that we obtained were mixed. More precisely, on some classes we have been able to achieve values of AUC greater than 0.95, however on some others the method made register values below the 0.5 threshold. For this reason, we have decided to omit the details about the implementation of this other method in our work. Irregardless, we think that this is a promising approach to be explored, probably by introducing an improved loss function.

This being said, we hope with our work to have made a small, yet tangible, contribution in the field of anomaly detection on point cloud.

Bibliography

[1] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.

[2] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *European conference on principles of data mining and knowledge discovery*, pages 15–27. Springer, 2002.

[3] Frank J Anscombe. Rejection of outliers. *Technometrics*, 2(2):123–146, 1960.

[4] Zurui Ao, Yanjun Su, Wenkai Li, Qinghua Guo, and Jing Zhang. One-class classification of airborne lidar data in urban areas using a presence and background learning algorithm. *Remote Sensing*, 9(10):1001, 2017.

[5] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018.

[6] Vic Barnett and Toby Lewis. Outliers in statistical data. *Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics*, 1984.

[7] Harry G Barrow, Jay M Tenenbaum, Robert C Bolles, and Helen C Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. Technical report, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1977.

[8] Mokhtar S Bazaraa, Hanif D Sherali, and Chitharanjan M Shetty. *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013.

[9] Richard J Beckman and R Dennis Cook. Outlier..... s. *Technometrics*, 25(2):119–149, 1983.

[10] Paul Bergmann, Xin Jin, David Sattlegger, and Carsten Steger. The mvtec 3d-ad dataset for unsupervised 3d anomaly detection and localization. *arXiv preprint arXiv:2112.09045*, 2021.

[11] Paul Bergmann and David Sattlegger. Anomaly detection in 3d point clouds using deep geometric descriptors. *arXiv preprint arXiv:2202.11660*, 2022.

[12] Richard J Bolton and David J Hand. Statistical fraud detection: A review. *Statistical science*, 17(3):235–255, 2002.

[13] Alexandre Boulch. Convpoint: Continuous convolutions for point cloud processing. *Computers & Graphics*, 88:24–34, 2020.

[14] Simon Byers and Adrian E Raftery. Nearest-neighbor clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association*, 93(442):577–584, 1998.

[15] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Anomaly detection using one-class neural networks. *arXiv preprint arXiv:1802.06360*, 2018.

[16] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[17] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[18] Sucheta Chauhan and Lovekesh Vig. Anomaly detection in ecg time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–7. IEEE, 2015.

[19] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[20] Yaodong Cui, Ren Chen, Wenbo Chu, Long Chen, Daxin Tian, Ying Li, and Dongpu Cao. Deep learning for image and point cloud fusion in autonomous driving: A review. *IEEE Transactions on Intelligent Transportation Systems*, 23(2):722–739, 2021.

[21] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet:

A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[23] Luc Devroye. Nonparametric density estimation. *The L_ 1 View*, 1985.

[24] Francis Ysidro Edgeworth. Xli. on discordant observations. *The london, edinburgh, and dublin philosophical magazine and journal of science*, 23(143):364–375, 1887.

[25] Sarah M Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58:121–134, 2016.

[26] Levent Ertoz, Michael Steinbach, and Vipin Kumar. A new shared nearest neighbor clustering algorithm and its applications. In *Workshop on clustering high dimensional data and its applications at 2nd SIAM international conference on data mining*, volume 8, 2002.

[27] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002.

[28] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[29] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[30] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[31] Alberto Floris. Composite convolution for 3d point clouds, 2021. M.Sc. Thesis, Politecnico di Milano.

[32] Alberto Floris, Luca Frittoli, Diego Carrera, and Giacomo Boracchi. Composite layers for deep anomaly detection on 3d point clouds. *arXiv preprint arXiv:2209.11796*, 2022.

[33] Zahra Ghafoori and Christopher Leckie. Deep multi-sphere support vector data description. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pages 109–117. SIAM, 2020.

[34] Izhak Golan and Ran El-Yaniv. Deep anomaly detection using geometric transformations. *Advances in neural information processing systems*, 31, 2018.

[35] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[36] Sachin Goyal, Aditi Raghunathan, Moksh Jain, Harsha Vardhan Simhadri, and Prateek Jain. Drocc: Deep robust one-class classification. In *International Conference on Machine Learning*, pages 3711–3721. PMLR, 2020.

[37] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Information systems*, 25(5):345–366, 2000.

[38] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(12):4338–4364, 2020.

[39] Wolfgang Härdle. *Applied nonparametric regression*. Number 19. Cambridge university press, 1990.

[40] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 170–180. Springer, 2002.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[42] Heiko Hoffmann. Kernel pca for novelty detection. *Pattern recognition*, 40(3):863–874, 2007.

[43] Peter J Huber. Robust statistics. In *International encyclopedia of statistical science*, pages 1248–1251. Springer, 2011.

[44] Nathalie Japkowicz, Catherine Myers, Mark Gluck, et al. A novelty detection approach to classification. In *IJCAI*, volume 1, pages 518–523. Citeseer, 1995.

[45] Ian T Jolliffe. *Principal component analysis for special types of data*. Springer, 2002.

[46] S Hamidreza Kasaei, Ana Maria Tomé, Luís Seabra Lopes, and Miguel Oliveira. Good: A global orthographic object descriptor for 3d object recognition and manipulation. *Pattern Recognition Letters*, 83:312–320, 2016.

[47] JooSeuk Kim and Clayton D Scott. Robust kernel density estimation. *The Journal of Machine Learning Research*, 13(1):2529–2565, 2012.

[48] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[49] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[50] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[51] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019.

[52] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. Outlier detection with kernel density functions. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 61–75. Springer, 2007.

[53] Rikard Laxhammar, Goran Falkman, and Egils Sviestins. Anomaly detection in sea traffic-a comparison of the gaussian mixture model and the kernel density estimator. In *2009 12th International Conference on Information Fusion*, pages 756–763. IEEE, 2009.

[54] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[55] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[56] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018.

[57] Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A Chapman, Dongpu Cao, and Jonathan Li. Deep learning for lidar point clouds in autonomous driving: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8):3412–3432, 2020.

[58] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[59] Jiaqi Lyu and Souran Manoochehri. Online convolutional neural network-based anomaly detection and quality control for fused filament fabrication process. *Virtual and Physical Prototyping*, 16(2):160–177, 2021.

[60] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. *arXiv preprint arXiv:2202.07123*, 2022.

[61] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[62] Ritesh K Malaiya, Donghwoon Kwon, Jinoh Kim, Sang C Suh, Hyunjoo Kim, and Ikkyun Kim. An empirical evaluation of deep learning for network anomaly detection. In *2018 International Conference on Computing, Networking and Communications (ICNC)*, pages 893–898. IEEE, 2018.

[63] Mana Masuda, Ryo Hachiuma, Ryo Fujii, Hideo Saito, and Yusuke Sekikawa. Toward unsupervised 3d point cloud anomaly detection using variational autoencoder. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3118–3122. IEEE, 2021.

[64] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 922–928. IEEE, 2015.

[65] Jooyoung Park and Irwin W Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

[66] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[67] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470, 2007.

[68] François Pomerleau, Francis Colas, Roland Siegwart, et al. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics*, 4(1):1–104, 2015.

[69] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[70] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.

[71] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[72] Jianjian Qin, Chunzhi Gu, Jun Yu, and Chao Zhang. Teacher-student network for 3d point cloud anomaly detection with few normal samples. *arXiv preprint arXiv:2210.17258*, 2022.

[73] Chen Qiu, Timo Pfrommer, Marius Kloft, Stephan Mandt, and Maja Rudolph. Neural transformation learning for deep anomaly detection beyond images. In *International Conference on Machine Learning*, pages 8703–8714. PMLR, 2021.

[74] Julien Rabatel, Sandra Bringay, and Pascal Poncelet. Anomaly detection in monitoring sensor data for preventive maintenance. *Expert Systems with Applications*, 38(6):7003–7015, 2011.

[75] Borja Rodríguez-Cuenca, Silverio García-Cortés, Celestino Ordóñez, and Maria C Alonso. Automatic detection and classification of pole-like objects in urban point cloud data using an anomaly detection algorithm. *Remote Sensing*, 7(10):12680–12703, 2015.

[76] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[77] Lukas Ruff, Jacob R Kauffmann, Robert A Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G Dietterich, and Klaus-Robert Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795, 2021.

[78] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class

classification. In *International conference on machine learning*, pages 4393–4402. PMLR, 2018.

[79] Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694*, 2019.

[80] Charles Ruizhongtai Qi. Deep learning on 3d data. In *3D Imaging, Analysis and Applications*, pages 513–566. Springer, 2020.

[81] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*, pages 146–157. Springer, 2017.

[82] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[83] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[84] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB*, volume 98, pages 428–439, 1998.

[85] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4548–4557, 2018.

[86] Walter Andrew Shewhart. Economic quality control of manufactured product 1. *Bell System Technical Journal*, 9(2):364–389, 1930.

[87] Nina Shvetsova, Bart Bakker, Irina Fedulova, Heinrich Schulz, and Dmitry V Dylov. Anomaly detection in medical imaging with deep perceptual autoencoders. *IEEE Access*, 9:118571–118583, 2021.

[88] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. Technical report, Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering, 2003.

[89] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[90] Lionel Tarassenko, Paul Hayton, Nicholas Cerneaz, and Michael Brady. Novelty detection for the identification of masses in mammograms. 1995.

[91] David Martinus Johannes Tax. One-class classification: Concept learning in the absence of counter-examples. 2002.

[92] David MJ Tax and Robert PW Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004.

[93] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019.

[94] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[95] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[96] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3173–3182, 2021.

[97] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[98] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 206–215, 2018.

[99] Ernst Zermelo. Beweis, daß jede menge wohlgeordnet werden kann. *Mathematische Annalen*, 59(4):514–516, 1904.

[100] Ji Zhang and Hai Wang. Detecting outlying subspaces for high-dimensional data:

the new task, algorithms, and performance. *Knowledge and information systems*, 10(3):333–355, 2006.

[101] Rui Zhao, Ruqiang Yan, Zhenghua Chen, Kezhi Mao, Peng Wang, and Robert X Gao. Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, 115:213–237, 2019.

[102] Yu-Jun Zheng, Xiao-Han Zhou, Wei-Guo Sheng, Yu Xue, and Sheng-Yong Chen. Generative adversarial network based telecom fraud detection at the receiving bank. *Neural Networks*, 102:78–86, 2018.

[103] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. *Advances in neural information processing systems*, 23, 2010.

A | Appendix A: The matter of the order

As we have mentioned in this work, we regard a point cloud as an unordered set of points. In this appendix we give some insights on why this choice and on the approaches that have been developed to deal with their unordered structure.

A first idea may be trying to sort the points in a canonical order. Although very reasonable, this solution is non-trivial and complex to be put into practice since already in $\mathbb{R}^2$ there is not a unique way to define a total order. One quite natural way to go is opting for the lexicographical order, but also in this case the sorted input may not be very meaningful. This is because small variations in the first component considered can results in drastic changes of the order, whereas high variations in the last component will very likely go unnoticed.
In general, fully connected networks applied on a sorted input tend to perform poorly, even though they still yield slightly better results than when the input is unsorted [80]. On account of the above, this strategy, as simple and intuitive as it may seem, in practice does not work well, hence it is rarely implemented.

A second possible approach consists in training a RNN and exploit the fact that, by its nature, it takes as input a sequence. The set of points is then treated as a sequence of vectors, which is fed to the network in a random order. To make the RNN invariant with respect to the order, many versions of the same point cloud are considered, each of them consisting of a randomly permuted sequence. The practice of feeding to the network slightly different versions of the same input is called data augmentation and it relies on the idea that in such a way the network increases its generalisation capabilities and becomes more robust. The process described previously falls under this umbrella.
Nonetheless, although augmenting the input is beneficial, it becomes quickly infeasible taking into account all the possible permutations for points clouds made of thousands of elements. This leads to an underrepresentation of all the possible permutations and so, during training, the network is exposed only to a very limited portion of them, making

the method suboptimal [80].

Another approach, by far the most common one, relies on the use of a symmetric function to aggregate intelligence from each point. Here by "symmetric function" we mean a multivariable function that takes as input n elements x_1, ..., x_n and returns a vector $f(\, h(x_1),\, ...,\, h(x_n)\,)$ so that the latter does not depend on the order of the elements. Several basic operations, such as addition and multiplication, have this property, as well as the the composition of them. What is usually done in this case is to process each point with the same function $h(\,\cdot\,)$ and then use a symmetric function $f(\,\cdot\,)$ to aggregate their information into $f(\, h(x_1),\, ...,\, h(x_n)\,)$. Since we're in the field of deep learning, it comes natural that $h(\,\cdot\,)$ is implemented by a neural network [80].

List of Figures

List of Tables

Acknowledgements

Vorrei innanzitutto ringraziare il Professor Giacomo Boracchi per la disponibilità e l'interesse dimostrato. Ringrazio specialmente anche il Dott. Luca Frittoli, per avermi assistito nella stesura di questo lavoro e per i suoi preziosi consigli.

Un ringraziamento particolare va alla mia famiglia, che mi ha sempre supportato in tutti questi anni e mi ha permesso di giungere a questo traguardo.
Sono inoltre grato a tutte le mie insegnati di matematica, che durante tutto il mio percorso scolastico sono riuscite, con il loro entusiasmo, a farmi appassionare a questa meravigliosa materia. Se ho intrapreso questo cammino è anche merito loro. A partire dalle elementari: Elisabetta Pipino, la prima che ha visto un qualcosa in me; Cristina Discacciati alle medie e infine Barbara Coppo al liceo.

Vorrei inoltre ringraziare tutte le persone che ho incontrato lungo questo percorso universitario, per tutti i pomeriggi passati a studiare assieme e per tutti i momenti di svago che abbiamo condiviso.
A special thanks goes to all the wonderful people I've had the luck to meet during my exchange year in Helsinki, who have contributed to making that period the most beautiful one of my studies. I'm particularly grateful for all the memories and the experiences we got to share, which will never be forgotten.
Vorrei infine ringraziare i miei amici del liceo, in particolare modo Cinzia, Alessia e Mauro, per tutti i momenti passati insieme. Un ringraziamento speciale va a Luca Coralli, per l'ineguagliabile senso dato alla parola amicizia, per la sua spensieratezza, per la sua lealtà e per il suo veloce volo, indimenticabile.