POLITECNICO DI TORINO

Master's Degree in Data Science And Engineering

Master's Degree Thesis

Self-Supervised Deep Learning via Colorization on 3D Point Clouds for Object Part Segmentation

Supervisors
Prof. Enrico Magli
Prof. Tatiana Tommasi

Candidate
Eduard Ciprian Ilas

Academic Year 2020-2021

Abstract

3D computer vision has grown in popularity in the last few years due to its centrality in many innovative industries such as robotics and autonomous driving, and the application of Deep Learning techniques to the 3D medium has given rise to new challenges. Common applications found in literature that target 3D point clouds range from scene semantic segmentation, to object classification, to object part segmentation. This thesis focuses on the latter - a task which requires the classification of different clusters of the 3D object point cloud into object parts - and aims at exploring the type of impact self-supervised learning techniques can have on such task. Self-supervision has been successfully applied to many Deep Learning applications in computer vision, and it requires the model to perform an additional task (known as pretext or auxiliary task) on unlabeled data. As object colours are often tied to their parts, a self-supervised colorization task, in which the model is tasked with predicting the color for each data point in the point cloud, is a promising candidate for driving the model into learning good representations of the data, in the hopes it can perform better on the real task (referred to as downstream task).
This thesis takes into consideration two point clouds datasets, containing synthetic object and real scenes point clouds, and explores the impact that a colorization self-supervised task can have at improving the performance of 3D object part segmentation tasks.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep Learning technologies have achieve remarkable results in 2D computer vision tasks revolving around classification, segmentation, detection and scene understanding. The 3D data can have multiple different representation, with different properties about the structure and geometry. The representation this thesis is focused on is called *3D Point Cloud*, a set of unstructured points distributed in 3D space that are meant to approximate the shape and surface geometry of 3D structures. The advantages of this representation over the others lies in the availability of acquisition technologies such as Kinect and LIDAR scanners that are able to reliably capture 3D point clouds and that have seen an increased usage in fields such as robotics and autonomous vehicle systems.

Typical deep convolutional neural networks that have proven to be very successful at computer vision task on 2D images are not suited to handle point clouds directly, due to their unstructured and unordered properties. Initial attempts at circumventing this issues have seen the introduction of pre-processing steps in the pipeline targeted at transforming point clouds into more structured 3D voxel data in order to retrieve a representation that is more suitable to be handled by 3D CNN architectures. However, this processes are often computationally expensive and are known to introduce quantization artifacts that can affect performance in classification and segmentation tasks. More recent development in research has seen the proposal of architectures that are able to directly process point cloud data and perform well in many supervised tasks such as 3D shape classification, part segmentation and scene semantic segmentation.

Self-supervised learning methods have been largely employed in 2D computer computer vision as a technique for learning useful representation for unlabeled data, and have recently been shown to be beneficial for learning 3D shape features as well. Such paradigm relies on auxiliary (*pretext*) task that leverages intrinsic property of the data to automatically source the supervisory signal to be used during the training phase, with the objective of driving the network into learning good representation features form the data that can be then transferred on the real (*downstream*) task. In this thesis, a self-supervised colorization pretext task is formulated in order to train models to colorize both 3D object and scene point clouds, in an attempt at exploring whether or not it can

improve performances on object part segmentation downstream tasks.

1.1 Purpose statement

Purpose of this thesis is to explore how self-supervised colorization on 3D point clouds can help neural network models perform better in 3D object part segmentation downstreams.

The main questions this thesis aims to find an answer to are:

Q1 *Is a self-supervised colorization task able to improve performances of models when the downstream task is 3D object part segmentation?*

Q2 *In scenarios in which training data for part segmentation task is reduced, does the colorization task provide enough robustness for the model to maintain an advantage in 3D object part segmentation tasks?*

1.2 Chapters outline

Contents of the chapters are as follows:

- *Chapter 1* contains a brief introduction as well as the main research questions this thesis aims at finding an answer to.
- *Chapter 2* contains a both theoretical and practical introduction to 3D point clouds, as well as a review about the main tasks Deep Learning methods are known to be applied to.
- A brief theoretical review about Artificial Neural Networks and common Deep Learning architectures that have been applied to point clouds are collected in *Chapter 3*, as well as an introduction to the self-supervised learning paradigm.
- *Chapter 4* reviews the main methods and datasets employed for this thesis.
- Finally, *Chapter 5* contains details about the experiments and results obtained.

Chapter 2

3D Point Clouds

2D imaging has a very long history of applications, being digital images the main way to have computers extract information about the real world. Two-dimensional images however fail when depth and positional information about the represented objects are required as well. Examples of such scenarios can be found in robotics, autonomous driving systems, and many other fields. When an array of 2D sensors is not able to capture the required information, there comes the need for 3D imaging, and various technologies have been developed to address such needs. The purpose of this chapter is to introduce 3D point clouds as well as other types of 3D data representation commonly found in literature and referenced in this thesis. Technologies employed in 3D scanning and their usage scenario will also be mentioned. Finally, a review of the main deep learning applications on 3D point clouds commonly explored in literature will be carried out.

2.1 3D Data representations

There is more than one way to represent 3D data, depending on the methods of acquisition or the structural proprieties required in order to process the shape [3]. 3D point clouds are often the direct unprocessed output of 3D sensors and being able to preserve the original geometrical and scaling information are often the format of reference for most applications. While the focus of the thesis regarding data representation is 3D point clouds, there are other formats that are sometimes found in literature and they each have their use, such as 3D meshes, 3D voxel grids, and RGBD images.

2.1.1 3D Point Clouds

3D Point Clouds are collections of unordered data points distributed in a three-dimensional space to form a 3D shape or structure. While of course point clouds, in general, don't have to be constrained to three dimensions alone, they are most commonly employed in the 3D Euclidean space as it offers a fairly good representation of real, three-dimensional shapes and scenes. As point clouds do not possess an underlying grid structure, however, [3] classifies them as a Non-Euclidean data representation.

Each point can be defined by its (x, y, z) coordinates alone, meaning that data points are self-contained and do not need additional information to be inferred from auxiliary data. A set of points encoding the spatial coordinates alone can suffice to describe shapes, however they can be enriched to include other useful attributes as well, in order to offer a more descriptive image. Colored point clouds, for example, can encode color information by defining each point with an $(x, y, z, r, g, b, \alpha)$ tuple, where (r, g, b, α) encodes red, blue, green and alpha (opacity) values as well for each point. Figure 2.1 showcases the descriptive powers additional attributes can offer.
As we'll see in Section 4.2, point cloud datasets usually store each dimensional value as floating-point numbers and RGB values as three-dimensional tuples of integers in $[0, 255]$.

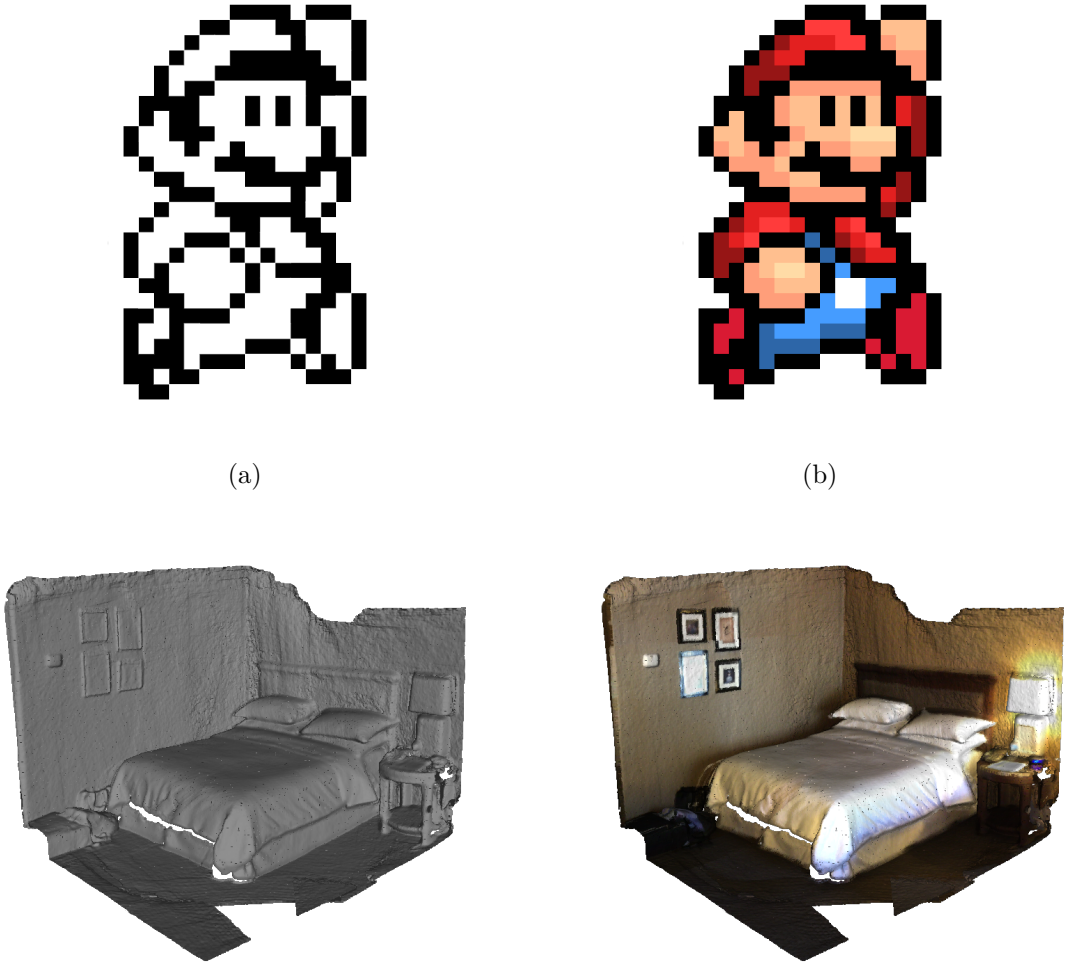
As pointed out in [35; 6], when dealing with 3D point clouds there are a few properties that need to be kept in mind:

a) **Unordered set.** When processing 2D images with deep learning models, there is a need to exploit the local spatial relations of the data structure, therefore the input data has to be structured as an ordered array of pixel data. Point clouds do not share this property, as each data point is autonomously defined within the space of reference, and so point arrays need not be ordered in order to be processed. Rather than an array or list, a point cloud is a set and as such the order in which the points are presented does not change the resulting shape represented by the point cloud. In other words, point clouds are *invariant to permutations*, which means that the method used to process them must take that into account and be invariant to permutations with respect to the order the input data is fed [35].

b) **Unstructured data.** As mentioned, point cloud data is not distributed over a fixed grid [29] and does not have to respect any rule concerning the spatial arrangement of the data points. This property has both strengths and weaknesses: dealing with unstructured data can pose a challenge, as assumptions about data structure are usually made when designing processing methods, but at the same time it allows for spatial density adjustments at will depending on the amount of information needed, without altering the structural properties of the point cloud.

c) **Irregularities or missing data.** Since point clouds are the result of 3D sensors scanning process by sampling surfaces, there are multiple obstructions that can interfere with the acquisition process. The resulting point clouds can have unevenly sampled zones [37] (where some areas are oversampled and some undersampled), or missing zones altogether. A curated point cloud dataset can be prepared in order to reduce irregularities and eliminate sampling errors, but in scenarios where points clouds are acquired and consumed in real-time - for instance by a processing pipeline on board of a vehicle that needs to perform object detection in real-time - a lot of work has gone into developing techniques that are able to process point clouds for noise and irregularities reduction [18].

d) **Local interaction among points.** When data points are spatially arranged, close points, according to a distance metric, can be clustered to form neighborhoods and



(a) (b)

(c) (d)

Figure 2.1: As it is the case with 2D images (a), binary information can offer sufficient information about the shape of the subject for 3D point clouds (c) as well. Enriching data with color information offers a more descriptive view of the subject for both images (b) and 3D point clouds (d). Sources: (a, b) [2], (c, d) [12]

combined into semantically meaningful local structures [35]. These spatial relations hold even though point clouds are not organized on a grid. A processing method should be able to pick up on such structures in order to be successful at different tasks involving point clouds.

e) **Invariance to transformations.** Just as points clouds need to be invariant to permutations, they also need to preserve their geometric structure when certain

transformations are applied [35]. This is essential as applying certain transformations and alignments is often done as a preprocessing step in points clouds processing pipelines.

Figure 2.2: 3D Point Cloud of a bunny.

2.1.2 Other common representations

3D Voxels

While volumetric representations were not employed by the methods of this thesis, they warrant a dedicated section as numerous references will be made given the attention this format received in literature regarding Deep Learning applications to three-dimensional data. A 3D voxel representation is the result of an attempt at bringing the two-dimensional, grid-like pixel structure of 2D images into the three-dimensional word [31; 49]. Just as pixels are the atomic element of a digital image and encode the perceived color information projected two-dimensionally, a *voxel* is a three-dimensional version of a pixel and the information it encodes can be either simply whether the volumetric area it represents is visible, occluded, or self-occluded [31], or include more information such as color if needed. Differently from data points in point clouds, voxels don't encode their own

spatial position, instead it is inferred during processing depending on different distance-based information relative to other voxels [43]. Given the underlying grid structure the representation relies on, [3] classifies it as Euclidean data, as it preserves the properties of grid-structured data such as "*having a global parametrization and a common system of coordinates*", differently from 3D point clouds.

It's not uncommon that a volumetric representation is obtained as a result of 3D point clouds processing [31; 43; 41], as the latter is often the direct output of many scanning technologies. In fact, as we'll see in Section 3.3, the grid structure makes it easy to extend 2D Deep Learning paradigms to handle volumetric data, making this representation fairly popular in literature as the first choice when dealing with 3D shapes [49; 31; 34].

Despite the simplicity of the representation and the advantage of having methods available that could handle it, it suffers from some intrinsic limitations, [3]:

- **Low efficiency.** Differently from point clouds, where data points themselves encapsulates information about the shape, volumetric data has to encode also the lack of information as it needs to include both occupied and non-occupied parts of the scene, which means even if the shape amounts to a small percentage of the volume, data has to be accounted for the empty volumes as well. This is known as the sparsity problem of volumetric data, and it grows with the resolution of the occupancy grid: "*as the voxel resolution grows, the grids occupied by shape surfaces get sparser and sparser*" [28], which implies that the memory requirements to handle such data grow cubically with the resolution. This behavior is shown in figure 2.3, which displays the sparsity of the occupancy grid as the fidelity of the representation increases [28]. Even though special methods have been proposed in literature that aim at dealing with the sparsity problem [28; 46], voxel-based representation is still generally considered unsuitable for representing high-resolution data.

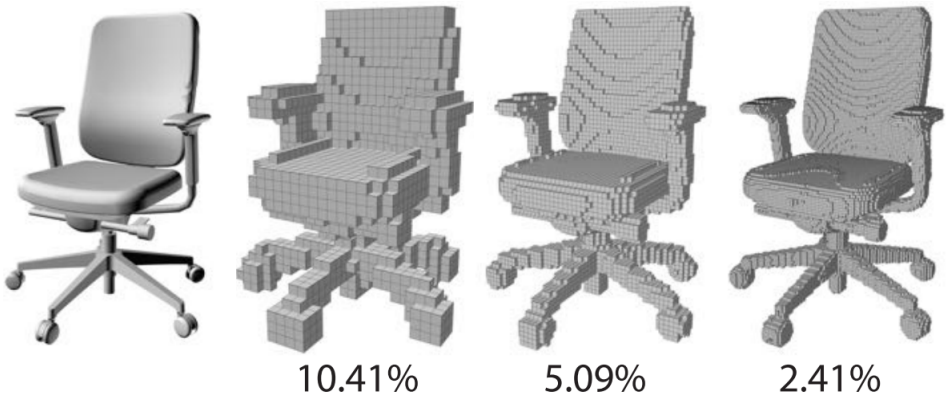
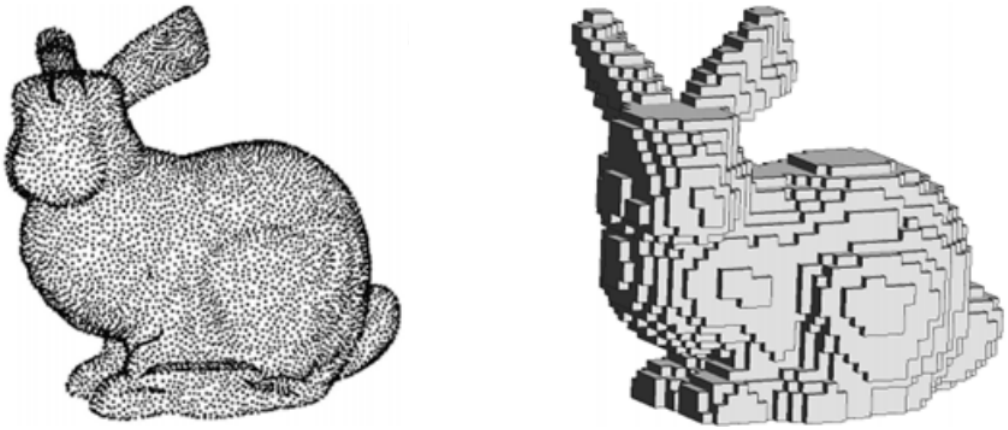


Figure 2.3: As the occupancy grid resolution of volumetric shape representations grows, the sparsity does as well. The percentage indicated is the density (defined as the ratio between the portion of grid that is occupied and the total grid) of the grid. Source: [28]

- **Intrinsic quantization artifacts.** As surface information is constrained to the fixed geometric grid, it's hard for voxel representations to preserve the complex geometry of 3D objects or the smoothness of the surfaces to a high enough fidelity [3]. Voxel-based data introduces quantization artifacts (Figure 2.4) that are intrinsic to the representation. Such errors can be reduced at the expense of higher sparsity occupancy grids, which means that deciding the resolution of the voxel grid comes off as a trade-off.

Other than the described intrinsic limitations of volumetric representations, there are a few other problems that emerge when traditional Deep Learning methods are adjusted to handle voxel-based data, which will be addressed in Section 3.3.



(a) (b)

Figure 2.4: A comparison between the two 3D data representations covered in Section 2.1.2: 3D point cloud (a) and volumetric voxel-based (b) representation. This figure is able to showcase how the resolution chosen for the voxelization process introduces quantization artifacts in the resulting shape (b), whereas in the original point cloud the geometry of the shape and the smooth surface are both preserved. Source: [20]

2.2 Practical Usage Scenarios

While the scope of this thesis does not cover acquisition methods or real-time processing scenarios, they are still relevant to the subject matter as some of the datasets employed are produced with the methods that will be described in this section, and some real-time usage scenarios might help the reader to better understand some of the constraints when dealing with point clouds processing methods.

2.2.1 Acquisition technologies

As the popularity of autonomous driving platforms and robotics applications grew in the last few years, more and more technologies have been developed and refined [15] in order to capture 3D information about the world around us to assist the mentioned systems.

RGB-D Cameras

A quite accessible technology for capturing 3D data is represented by depth cameras: devices that employ RGB-D sensors capable of capturing not only color, but also depth information at the same resolution and fidelity as a regular RGB sensor captures color details. The output of such sensors encode therefore both color and depth information on a per-pixel basis, and often delivered at real-time rates [54].
Depth information can be obtained through various methods based on stereoscopic cameras, structured light or time-of-flight. The first method makes use of two distanced sensors that are employed in order to obtain a stereo flow of visual data which can be processed in order to infer depth-information by comparing the disparities between the two visual flows obtained at different positions in space, similar to how humans perceive depth by relying on the binocular cues based on the flows of information from both eyes. The second method adopts a more active approach and usually works by projecting an infrared light pattern (or sequence of patterns) onto the target scene and then analyzing the visual distortions the pattern presents when observed from a different perspective, from which depth information can be inferred (Figure 2.5). Finally, time-of-flight (ToF) technologies work by measuring the time it takes for signals emitted from an infrared light emitter to travel to the target object and back to an infrared sensor. Depth information can then be easily obtained by accurately computing the distance from the sensor.

Depth cameras technologies have recently become more affordable and packaged into consumer-grade products, such as Microsoft Kinect (which employed both structured infrared light and ToF technologies in two different iterations of the product, making it a fairly popular device) or Google Tango. The diffusion of such products made it so that computer graphics and vision communities could employ them for research purposes, which made it so that "*the state of the art has been greatly advanced in computer graphics and computer vision research developing new methods to reconstruct models of the static and dynamic world around us*" [54]. Today, RGB-D cameras can be safely considered the method of choice for indoor applications in robotics [11] when RGB information alone is not enough.
Many proposed methods can be found in literature that attempt at reconstructing 3D geometry from RGB-D sensors data. Often, depth information (namely the distance of a certain point from the sensor) alone can be processed and directly transformed into a point cloud in real-time, which can be useful for many in-door robotics applications. Figure 2.6 shows an example of how Kinect is able to output a point cloud of the target object.

For more complex scene reconstruction from multiple RGB-D images, two methods can be mentioned: Visual Odometry (VO) and Visual Simultaneous Localization and Mapping

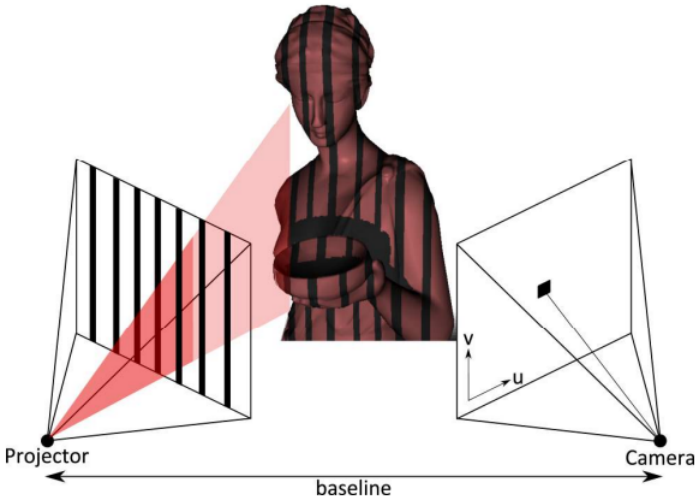
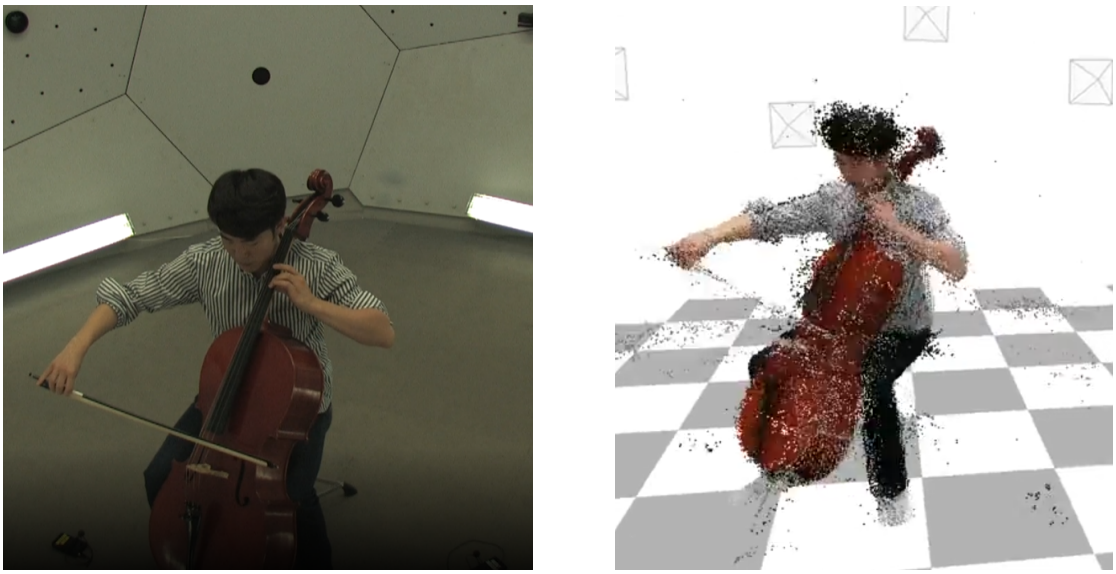


Figure 2.5: The core principle of structured light based RGB-D systems. Depth information is extracted by analyzing the visual distortions of the projected pattern. Source: [39]

.



(a) (b)

Figure 2.6: RGB (a) and color-enriched point cloud (b) processed from RGB-D data produced by Kinect devices. In this particular setting, an array of 10 Kinect devices distributed all around the subject was employed in order to cover all its surfaces. Source: [22].

(SLAM). The first one works by estimating the translation and rotation motions of the device in order to figure out the camera motions in a reference three-dimensional space,

while the second one tries to estimate a global map of the whole scene and reconstruct the camera trajectory within the map [11]. The resulting information can be processed in order to reconstruct a 3D scene, which can be represented as voxel-based data, surface meshes, point clouds, etc.
The described technologies and methods based on RGB-D sensors, while affordable and very accessible, are mostly intended for indoor scenarios, as they have limited range (within 10m) and precision, and often suffer from disruption from sunlight (in the case of infrared systems).

Laser Scanners

While RGB-D sensors are highly affordable, in outdoor scenarios and for high-range applications such as self-driving cars, laser scanners are employed. The most common technology is represented by LIDAR (*Laser Imaging Detection and Ranging*) scanners which, similarly to the ToF method in RGB-D sensors, employ ultraviolet and visible light signals to estimate depth by measuring the time light signals take to travel to an obstacle and be reflected back to the sensor. This technology is usually more expensive and therefore more common in scenarios where more accurate performances are required. There are different ways to declinate the core principle, depending on the desired field of view, range and orientation. LIDAR scanners commonly mounted on vehicles have mirrors that rotate and scatter the burst of laser in order to obtain a 360° horizontal field of view coverage. LIDAR scanners with no moving parts have also been developed (*Solid State LIDAR*) and although they possess a more limited field of view, are cheaper and more reliable as they have nonmoving parts. In the context of self-driving vehicles, they can be employed as additional short-range fixed LIDARs in addition to the long-range 360° one in order to enhance object detection in blind spot zones. Data from LIDAR scanners comes directly in the point cloud representation and can be fed through the processing pipeline for object detection or semantic segmentation tasks performed in real-time.

Synthesized Point Clouds

Finally, synthetic point clouds are computer-generated 3D point clouds by sampling 3D CAD models or using more advanced techniques in order to derive them from simple images or videos. ModelNet [48] and ShapeNet [10] are common examples of synthetic 3D CAD models of various objects datasets that are often used to extract high-quality 3D point clouds from. The resulting point cloud datasets are often used as training sets for Deep Learning models tasked with object detection and classification, object part segmentation or scene semantic segmentation (see Section 2.3).

2.3 Deep Learning applications

After introducing point clouds and their properties, as well as some real examples of scenarios where point clouds acquisition and processing is essential, this section aims at showcasing some of the most commonly explored 3D computer vision tasks in literature:

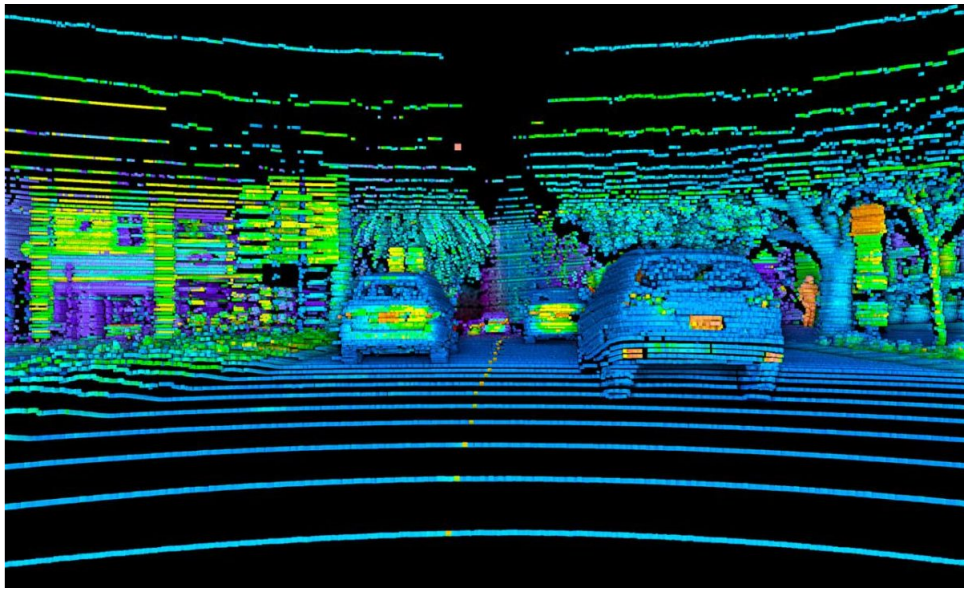


Figure 2.7: A visualization of the point cloud resulting from a LIDAR sensor. Source: [1]
.

object classification. A more in-depth technical review of Deep Learning techniques and models employed for the described tasks will be done in Chapter 3 instead.

2.3.1 Object classification and recognition

It appears pretty clear from Section 2.2.1 that, for example, a self-driving car on the streets that is acquiring 3D information through the dedicated sensors as point clouds, needs to process them in order to, for instance, recognize objects on the streets such as people or other vehicles. Simply detecting generic objects as obstacles is not enough, as the behavior of the car is of course heavily dependent on the type of object is being detected: hence the need for such systems to be able to recognize and classify the objects detected within 3D point clouds.
With classification, a model that is fed an object shape as point cloud for input needs to assign to the whole point cloud a label from the list of object categories available. The point cloud is therefore treated as a single entity, and a good model needs to be able to learn global point cloud features in order to be reliably used for object classification [35].

While good quality synthetic point clouds representing objects alone can be used for training and benchmark purposes, in a real scenario acquired point clouds represent objects that are surrounded by background information or occluded. In these cases, object classification tasks are often coupled with *object detection and localization*. In this case, given a real scene point cloud, the model would need to filter out background information and retain only data that belongs to the target object (be it a vehicle, human, etc). More formally, each point needs to be firstly labeled as either *background* or *relevant*, and

the relevant points need to be returned together with information about the geometric location, orientation [36] and semantic label of the object they represent.

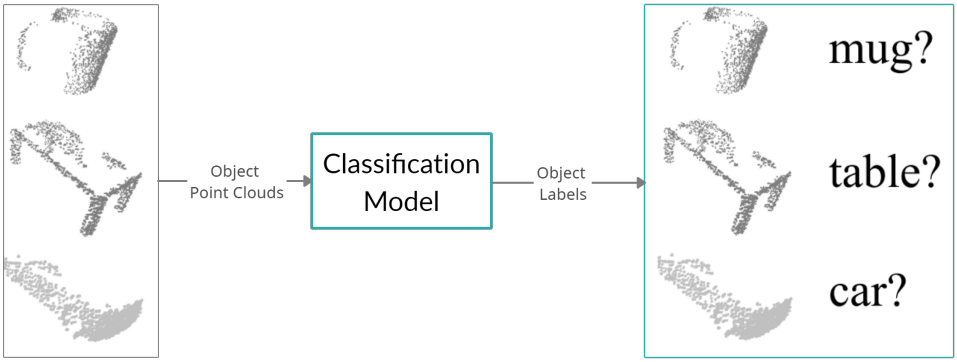


Figure 2.8: Object classifications model assign object labels to input point clouds.

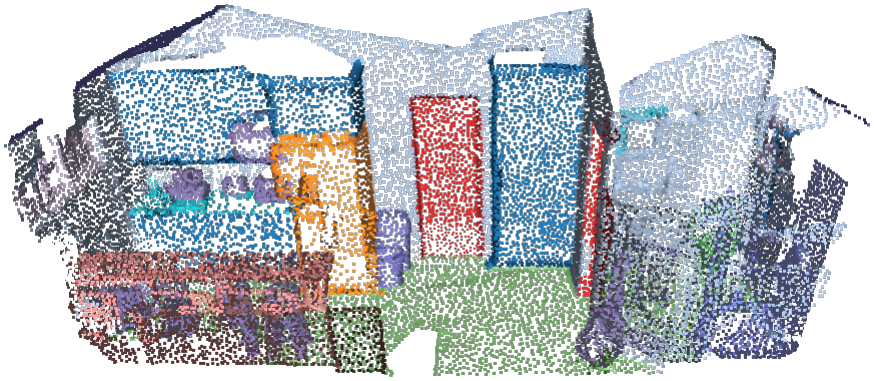
2.3.2 Scene semantic segmentation

Scene semantic segmentation tasks are similar to the object recognition task described in Section 2.3.1, except it is more ambitious as it requires the model to be able to derive a semantically meaningful understanding of the scene represented by the point cloud, in order to predict semantically meaningful labels for each point in the scene. This is a fairly common task in 2D computer vision and faces similar challenges in the 3D version: a model needs to capture both local and global features of the scene, as "*global information resolves what while local information resolves where*" [30], meaning that features need to encode both location and semantics information in a hierarchical relationship. Figure 2.9 shows the result of a semantic segmentation process applied to an indoor point cloud scene.

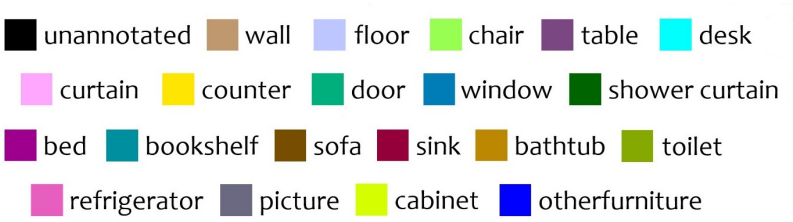
2.3.3 Object part segmentation

Object part segmentation tasks require that single object point cloud shapes be segmented into semantic parts, meaning that a model that is fed a point cloud shape as input is tasked to assign to each individual point within the cloud a semantic part label. A bag, for instance, would be segmented in simply *body* and *handle* labels, whereas a more complex object like a motorbike would have as part labels *wheel*, *seat*, *handle*, *light* and *gas tank*. The model therefore needs to figure out the neighborhoods of data points that need to share the same part label within the object shape. Despite not having to deal with the semantics of an entire scene, object segmentation is not a trivial task, as the point clouds are not structured and objects in the same category can have very diverse points distribution.
Accurate annotations for segmented object point clouds are useful in many scenarios, such as 3D shape synthesis [23], or 3D shape completion of partial point clouds [44]. Manual annotations are tedious and lengthy, therefore automating the process is highly advisable.

(a)



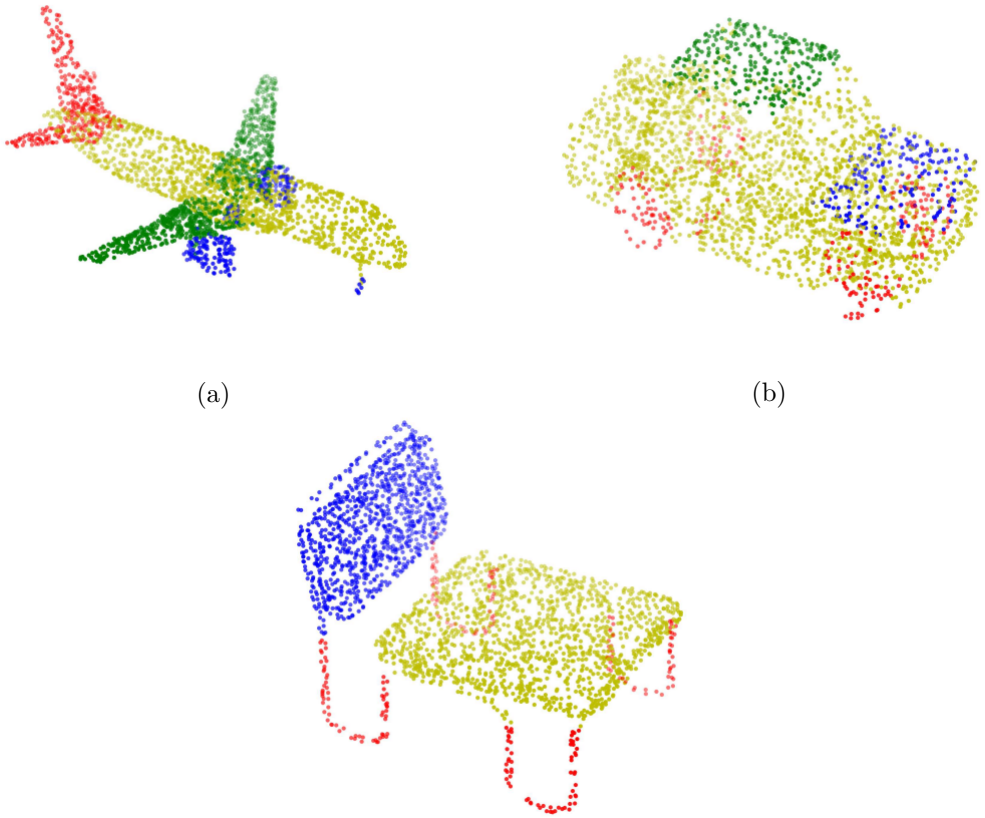
(b)



(c)

Figure 2.9: Scene semantic segmentation visualization. The (color enriched) point cloud representing an in-door scene (a) is semantically segmented, where each point is assigned to a semantic label (in this case door, floor, fridge, etc). The label is visualized through color in (b) according to the color legend (c). Point cloud sourced from ScanNet [12].

Figure 2.10 shows a few examples of object part segmentation applied to point clouds.



(a) (b)

(c)

Figure 2.10: Object point clouds segmented into parts. Color indicates part label. Source: [24].

Chapter 3

Technological Background

While the idea of Artificial Neural Network sparked as a reverse engineering attempt at recreating the functioning of neural cells within the brain, the modern approaches aim at developing a consistent and coherent learning theory, rather than emulating the mechanisms present within the brain that were since discovered to be much more complex. One such result of this line of work was the development of Deep Learning techniques, which have been made possible by the increasing computational power of recent years. This made it possible for such methods to gain a significant advantage in terms of speed and efficiency, and have since been made their way up to state of the art in many different applications. This chapter aims at providing a theoretical background about modern technologies that are being employed in point cloud processing tasks. A general picture about the nature of the Artificial Neural Network systems, as well as a review about different kinds of such architectures approaches and their application to point cloud processing are meant to offer to the reader the knowledge necessary in order to be able to interpret the experiments in the subsequent chapters.

3.1 Artificial Neural Networks

A *biological neural network*, also commonly referred to as *neural circuit*, is composed by a group of *neurons* - the fundamental unit cells within the brain - that come together to form connections between them. Such connections are possible through *synapses*, biological structures that allow for electrical and chemical signals to be passed exchanged the neurons. Even though the biological mechanisms involved are much more complex and were not completely understood at the time, the late 40s saw the first efforts at modeling how neurons work within the brain, in an attempt to simulate intelligent behavior.

One of the most influential descriptions of neural network activity is to be found in the 1949 book *The Organization of Behavior* by Donald Hebb. In his book, Hebb describes what will later be known as *Hebb's postulate*:

> *When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change*

takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. [19]

Hebb's principle, describing how neural pathways strengthen each time they are used ended up being crucial in translating the neural mechanisms into artificial computing systems. The first concrete model was proposed by the psychologist Frank Rosemblatt, who called it *Perceptron*. Core of the system was a model of an artificial neuron which had input and output connections (Figure 3.1). A weighted sum of the input data would be passed through a linear threshold function that would output either a '0' or a '1' depending on whether the weighted sum was below or above the threshold. An objective function would be adopted in the learning phase, whose minimization would adjust the weights of each input connection as input data would be successively passed to the network. The model was intended for image recognition and the algorithm employed could be considered as a binary classifier, but it was only able to handle linearly separable problems.

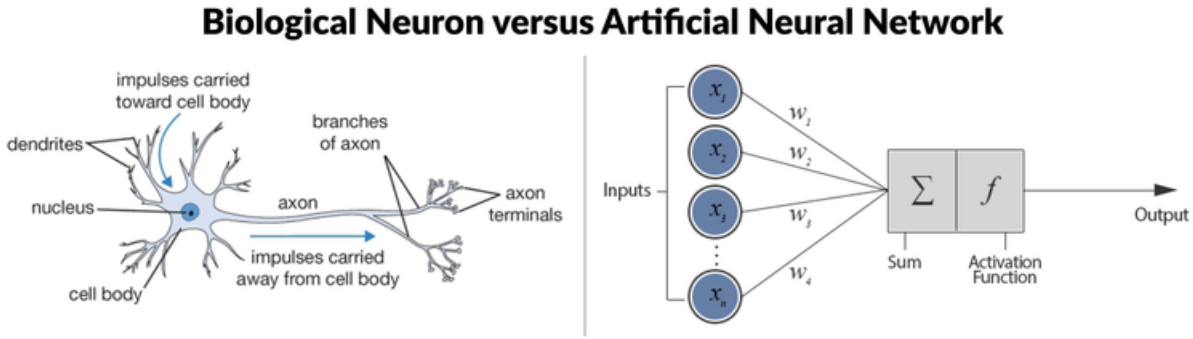


Figure 3.1: Artificial neurons are loosely inspired from biological neurons. Source: [47]

The 1958 Perceptron was the cornerstone to what today are called *feedforward neural networks*: artificial neural networks where neurons come under the *node* abstraction, which are distributed into multiple layers such that each neuron is connected to all the neurons from both the previous and successive layer. Each node takes as input the output values from the nodes in the previous layer, which can represent any kind of data, and is able to send its output to each of the other nodes in the next layer.
Figure 3.2 shows the architecture of a simple multilayer feedforward neural network with one hidden layer. As hinted by the arrows, the information flows in one direction: forward. Nodes outputs are calculated by computing the weighted sum of the inputs, to which an eventual bias value is added, and the result is passed through a - usually nonlinear - activation function, whose result is being forwarded to the nodes in the next layer. The purpose of the activation function is to introduce non-linearity in the process, and it allows for the network to be able to approximate any arbitrary non-linear function [1].

[1]It is possible to prove that by employing non-linear activation functions, a multiple-layer neural network can become a universal function approximator (Universal Approximation Theorem).

Common activation functions are sigmoid, softmax, ReLU, etc. and they all come with their features.

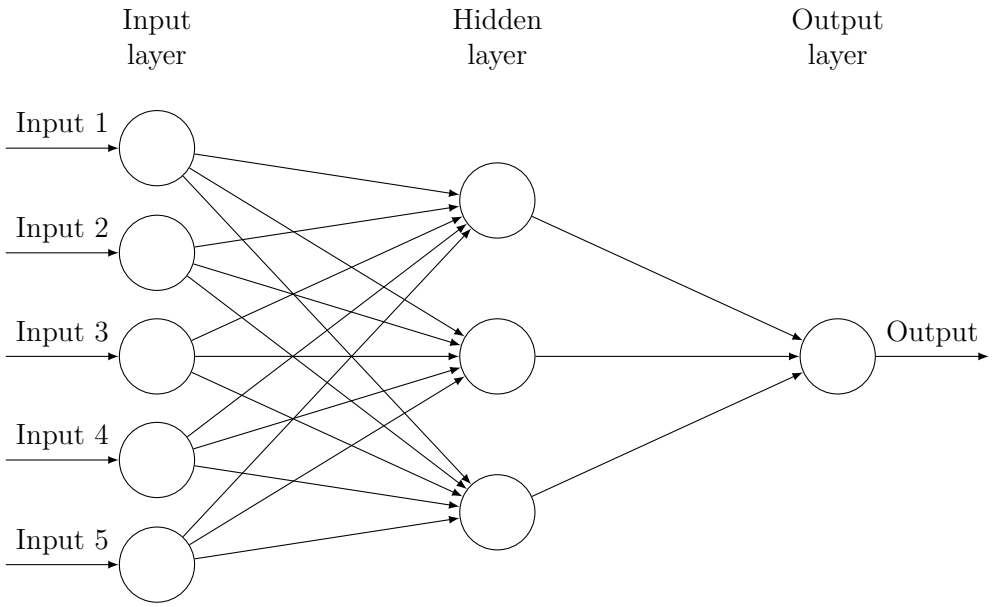


Figure 3.2: A multi-layer feedforward neural network with one hidden layer. Nodes in the input layer receive input values and their outputs are feed to the nodes in the next (hidden) layer.

During the training phase, weights are adjusted as the network learns to address a specific task - be it classification, regression or other - by being exposed to sample data. The task is formalized as an optimization problem, where an *objective function* is being defined and used to periodically evaluate the performance of the network. The objective function (also commonly referred to as *loss function*) is tailored to the type of task - and usually measures the discrepancy between the real output of the network and the required one - such that during training the weights of the network are adjusted in order to minimize the loss function. The algorithm able to perform the adjustments is known as *backpropagation*, and it works by calculating the gradients of the objective function versus the weights instances at every iteration, then propagating them backward (from the output layer, through the hidden layers, to the input layer) in order to update the weights with the ones that can better minimize the error given the input data. The gradients are efficiently computed for each layer by making use of the chain rule, and since the negative gradient vector indicates the directions towards the minimum of the objective function, many optimization algorithms performing gradient descent can be employed.
There are multiple architectural variations that have been proposed and shown to be successful at many different tasks. Some variations can involve the number of hidden layers and neurons, which is heavily dependent on the complexity of the task the network is

trained to solve, the possibility of skipping some connections during training or connecting neurons within the same layer.

3.1.1 Artificial Learning Methods

The assumption that has been made so far is that there is only one way for artificial neural networks to learn. However, computational learning theory identifies more approaches, which differ in the way data is presented to the network model and the way the learning task is designed.

Supervised Learning

As mentioned, the most common approach to machine learning is through supervision. When a training dataset is prepared, data is labeled with respect to the target output we want the algorithm to return. For example, an image classifier that is trained to classify images as containing, for instance, a cat, a dog or a giraffe, would be needing a dataset containing a large number of such images, each of them preventively labeled as either cat, dog or giraffe. During training, the network model would be fed the images (or a representation of them), and would be asked to classify them as one of either three classes, which is commonly done by returning, for each data instance, a score for each class, one for each category. Since we possess the actual labels for each image, we can design an objective function that would compute an error between the desired output (higher score for the right label) and the actual label. We then use that error to adjust the network parameters in order to improve its prediction: the model *learns* to do better.
Classification is not the only problem supervised learning can address. Another very common problem machine learning algorithms are tasked with is represented by *regression*. In this case, the valued property of deep ANNs of behaving like universal function approximators is more central, as we're interested in real continuous values predictions rather than classification of image features. A typical example could be simple temperature forecasting given certain conditional features as input.

Unsupervised Learning

As the name suggests, differently from the supervised approach where data labels are employed during training to adjust the model parameters, in the unsupervised one there are no labels at all. The network model is tasked with finding ways to separate the data into groups that express common patterns and similar features with no supervision, meaning that no information other than the data itself is used. This makes it possible for a bigger amount of data to be used, as the process of preparing labeled datasets is time-consuming and usually requires human labor.
There are different approaches to unsupervised learning. *Clustering analysis* aims at finding commonalities between data in order to cluster similar objects together, separating each group from the other into clusters of data. *Anomaly detection* works by classifying unlabeled observations either as regular or as an anomaly, under the assumptions that most of them are regular and data objects that are too dissimilar are most likely outliers.

Autoencoders are neural network models tasked with learning in an unsupervised manner how to efficiently encode unlabeled data into meaningful, compressed representations, and are commonly used as generative models (can generate new data similar to the data they have been trained on), or as feature extractor for improving classification results in a semi-supervised scenario [5] (where only part of the data have labels).

3.1.2 Close-Up: Self-Supervised Learning

Among the standard approaches to artificial learning discussed in Section 3.1.1, there is a method called *self-supervised learning* (SSL) that can be placed midway between supervised and unsupervised learning. The method in fact tries to take advantage of the strengths from both approaches: it works on unlabeled data and therefore avoids the cost of needing labeled large datasets, while at the same time is able to make use of *pseudo labels* [27] in order to train the model in a supervised manner towards learning a certain task.
Unsupervised methods, due to the lack of an objective function that would drive the network model to learn useful data representations, have been shown to have a hard time at extracting the right feature information from large unlabeled datasets [13]. Self-supervision addresses this by sourcing the supervisory signal from the data itself, without the need for additional annotations. This requires the definition of a so-called *pretext* task that revolves around an intrinsic feature of the data, so that by solving this task the model is forced to learn good representation features from the data that can be then transferred to the real task (known as the *downstream task*) in the hopes it can increase its performance. While the pretext task is not the original objective, by training the model on it, it is forced to learn relevant features and achieve a high level understanding of the data in order to solve it, which are then exploited on the real task. As the method is able to fabricate labels by automatically extracting them from the data itself, there is no need for manual labeling, which means the amount of feasible data is much larger. This technique has been proven to be effective at improving performances to state-of-the-art levels in many computer vision benchmarks and on different models [9; 13].

There are many types of pretext tasks that have been proposed in literature. As far as computer vision goes, they usually revolve around completing some visual task on incomplete data images:

- *Context prediction* pretexts tasks generally require the model to predict some spatial relation between randomly sampled portions of image. In [13], pairs of squared patches from eight possible spatial configurations are sampled, and the model is tasked with predicting the position with respect to each other (see Figure 3.8). The model will have to become familiar with the objects and scenes and recognize them within the images in order to be able to learn the spatial relations needed to solve the task.

- *Geometric and visual transformations* such as rotation, translation, scaling and color shifts can be applied input data fed to the model. [16] proposes 4-class classification task, where the network needs to predict the angle of rotation applied to the image. The assumption is that the model would be forced to become aware of the *right*

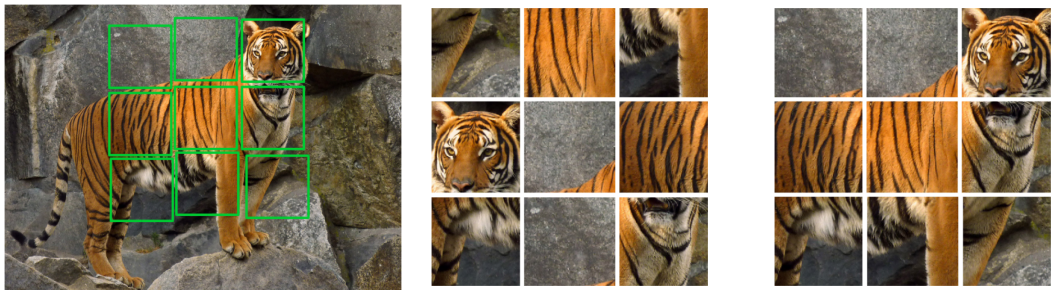
orientation of the objects, and therefore develop a spatial understanding of them when found in natural scenes. In [14] instead, every image from the dataset is its own class as different data augmentation techniques such as translation, scaling and color shifts are applied to them, and the model is required to predict the original.

- *Jigsaw puzzles* are a classic pretext task applied to images. They are proposed in [32], where multiple tiles are extracted from images, shuffled and fed to the model, which is required to solve them by predicting the correct permutation (see Figure 3.4). Context awareness and spatial relationship about objects need to be developed in order to solve the task.

- *Inpainting* works by removing region from the image, and tasking a generative model to fill in the gap (Figure 4.4). The model needs to be able to capture the context of the image in order to produce the missing part region. A "*deep semantic understanding of the scene [or object], and the ability to synthesize high-level features over large spatial extents*" [33] needs to be developed in order to solve the task.

- Finally, *image colorization* is another generative task that can be performed with self-supervision. The assumption is that useful semantic relations are deeply encoded into the color of the objects. A model that is presented with gray-scaled images and tasked to generate color requires awareness of the "*dependencies between the semantics and the textures of gray-scale images and their color in*" [53] to e achieved in order to produce visually plausible colors. Since colorization tasks cover the whole image rather than focusing on the single object or a portion of the scene, both local features (colors about single items) and global features (context awareness about the entire scene) need to be merged in order for the model to be able to solve the task [21]. As with most generative tasks, solving doesn't necessarily mean to come up with the exact colors found in the original image, but rather plausible ones (see Figure 3.6) that could pass a human review.

Figure 3.3: In context prediction tasks a patch randomly sampled in one of nine possible position together with the central patch is fed to the model which is tasked with predicting the sampling position. Source [13].



(a) (b) (c)

Figure 3.4: Jigsaw puzzle tasks require extracting image tiles from a portion of image (a), shuffling them (b) and then tasking a model with determining the correct permutation of the original tiles (c). Source: [32]

(a) (b)

Figure 3.5: Inpainting tasks consist of removing portions from images (a) and tasking generative models with filling in the gap (b) in a self-supervised generative manner. Source: [33]

Self-supervision within the training pipeline

While each of the mentioned self-supervised tasks have in common the ability to drive the model into learning deep data representation features, there is still one aspect about the training phase that needs to be mentioned, and it revolves around how the pretext task plays in during the training phase. Pre-training the model uniquely on the supervised task and then transferring over the weights to the downstream one has all the advantages that transfer learning brings to many neural network architectures, but also the disadvantage of needing to perform the training in two stages: one first training for the pretext, and the second one on the downstream task. This method is referred to as *multi-stage* training. An alternative to this approach is training for both tasks at the same time, eliminating the need to develop multi-stage pipelines. This second approach is called *multi-task*, and requires for a dedicated network branch (and a dedicated object function) to be reserved

(a) (b) (c)

Figure 3.6: Image colorization employs generative models in order to add color to gray-scaled (g) images. It's not important that the results exactly match the original colors (i), as long as the colors generated are plausible enough (h) to human evaluation. Source: [53]

on the model for the auxiliary task. In the forward pass, after the network branches off, the main branch will handle the main task, and the second one will be completely dedicated to the pretext task. During the backward pass instead, the objective functions for both the downstream and the pretext task will be contributing to the propagation of the gradients. This way, the main loss function will assure the backbone model is learning to solve the main task, while the pretext task loss function will contribute by making sure that data representation features will also be learned in a jointly fashion.

Self-supervision on 3D point clouds

While the major center of attention for self-supervision in computer vision has been around 2D images, there is nothing that prevents it to be applied on 3D data as well. In fact, as mentioned in Section 3.1.2, given the improvement this method has been proven to bring to so many downstream tasks, it has been naturally applied to 3D computer vision tasks as well. Since even more so than with 2D images, manually labeling large 3D point clouds datasets for supervised learning tasks is a substantial process, the advantage unsupervised pretext can bring is not only in terms of performance, but also in terms of reducing the amount of annotated data samples needed for downstream tasks.

As with 2D images, many self-supervised pretext tasks can be designed for point clouds that can take advantage of their flexibility in representation. Inspired from jigsaw puzzles and context prediction self-supervised tasks performed on 2D images, [40] proposes a self-supervised task in which 3D single object point clouds are divided into $3 \times 3 \times 3$ equally sized volumetric portions (as seen in Figure 3.7), each containing a considerable amount of point clouds and each of them being assigned an ID label. All the portions are then randomly swapped between them and fed as input to a neural network, which is then trained to predict the original ID label for each point, corresponding with the volumetric portion it originally belongs to. This method can work directly on raw point cloud, and by training on this task, models are forced to learn data embeddings that are proven to be beneficial to tasks like object classification, and object part segmentation and semantic

segmentation.
In [45], the property that points have of arranging in meaningful point neighborhoods is addressed by designing a task that is meant to learn local point-wise features by predicting the next point in a sequence of points distributed into space within the point cloud. The method has been shown to improve performances in part segmentation tasks. Point cloud classification accuracy has been boosted in [52] by designing a self-supervised task in which all the point clouds are cut in two parts, and the model is tasked with predicting whether two randomly sampled half point clouds originate from the same object point cloud, in an attempt to drive the network towards learning semantic features for raw point cloud data.
Another usage of 3D point cloud puzzle as self-supervised task can be found in [4]. In this case, the model is tasked with solving both the self-supervised 3D puzzle and a supervised object point cloud classification task in a multi-task manner. This method has been shown not only to improve results in object classification downstream tasks, but also to be effective at boosting the performance in scenarios where the training data for the downstream task has been reduced, suggesting that self-supervision for 3D point cloud can make up for the need for large labeled point cloud datasets.

3.2 Deep Learning architectures

The promising multy-layer aspect ANNs, together with the increased availability of training data and processing power, has led to a major breakthrough in the Machine Learning domain, so much that it sparked the birth of a new dedicated field: Deep Learning. For a long time, machine learning meant coming up with ways to design data features extractors that would make it possible to transform raw data into a more appropriate representation for learning algorithms to pick patterns from [26]. The *deep* adjective naturally refers to the architectural characteristic of employing many layers that subsequently process the input data. As input data is being passed through the layers, many non-linear transformations are applied to it in an attempt to progressively extract high level features which are then usually fed through a final classifier. The hierarchical distribution of the layers means that the first ones, as they are able to interact with a less transformed version of the data, are able to extract simple, low level features. The deeper layers are capable of further processing the low level features previously extracted into high level ones, more meaningful towards the specific task. For example, the first layers of a simple convolutional neural network (see Section 3.3.1) can extract simple visual features like edges, color gradients or basic shapes, which can then help deeper layers to learn more complicated, high-level features like faces, writings or complex geometric patterns.

> *Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human crafted features.* [7]

While the first instance of deep learning architecture is the feed-forward neural network

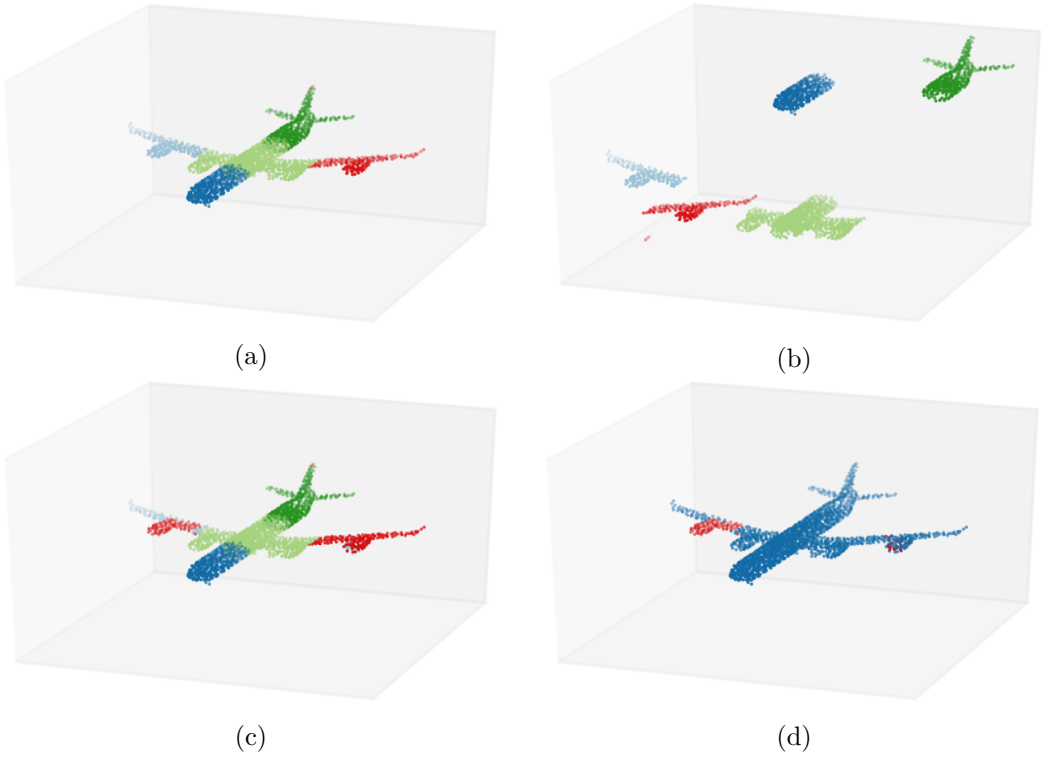


Figure 3.7: Visual representation of a self-supervised 3D point cloud puzzle. The original object point cloud is divided into regular 3D portions as seen in (a), where different color identifies points belonging to different portions, and spatially rearranged (b) before being fed to the model. (c) represents the predicted labels, and in (d) we can better observe the misclassified points. Source [40]. Source: [40]

previously mentioned, current models come in all shapes and forms, each of them proven to be successful in specific domains such as computer vision, speech recognition, natural language processing, etc. Among other common architectures we can mention Recurrent Neural Networks (specialized in handling long sequential data where the temporal information plays an essential role), Convolutional Neural Networks (which specialize in handling bi-dimensional data from images and are known to achieve excellent results in anything computer vision related), and recently Transformers (who exploit attention mechanisms and have been shown very good results when applied to many types of tasks).

3.3 Overview on Deep Learning with 3D data

Deep neural networks have been continuously achieving multiple breakthroughs in many fields. This chapter serves as a review of the main deep learning model architectures that have been proposed to address 3D data, as well as a few learning paradigms that while not necessarily related to geometric learning, they have been fundamental to the

development of this thesis work.

In computer vision, one particular type of deep feedforward network has shown to be able to b Computer vision has, especially in computer

3.3.1 3D Convolutional Neural Networks

In computer vision, one particular type of deep feedforward network has been shown to achieve many practical successes in applications such as image and video classification [25; 50], image semantic segmentation [30], object recognition [38] and many other types of image analysis tasks, all contributing towards them being generally considered as the main tool whenever image processing was required. Convolutional neural networks (CNNs) are deep, feedforward neural networks designed to process structured information in the form of multi-dimensional arrays that can represent many data modalities such as 1D signals and sequences, 2D images and 3D volumetric data or sequence of video frames [26]. The *convolutional* term in the name comes from the presence of so-called *convolutional layers*: a structure that is able to perform convolution operations between multiple *kernels* and the input data in order to obtain feature maps. A kernel is fundamentally a (usually 2D) weight matrix that is able to extract different features through the convolution operation. Feature maps obtained from convolutional layers are treated as input data for the next layers. As the data is passed through the first layers, simple low-level features are extracted such as edges, corners and simple color gradients while deeper layers are able to leverage on previously extracted features in order to extract more complex and higher-level features.

Given the success CNNs have achieved on many image processing tasks, soon enough they have been shown to be able to handle 3D data as well. Most of the first attempts saw usage of CNNs for 3D object classification and semantic segmentation on RGB-D data where color and depth information were being processed separately [42]. Further proposal saw usage of full 3D geometric shapes as 3D voxel grids (Section 2.1.2) rather than RGB-D data by employing convolutions with 3D filters for 3D shape classification [48; 31]. As CNNs were able to process structured data only, volumetric 3D representation was the only choice and since point clouds have no structure regularity, they would typically need to be voxelized before being feed to CNN-based architectures.

3.3.2 Pointnet: Pointwise Multilayer Perceptron

A big milestone in 3D computer vision was represented by the introduction of a new kind of architecture that does not need to rely on voxelization processes in order to handle 3D geometric data, but is instead able to take as input raw 3D point cloud data directly in order to perform multiple tasks such as 3D object classification, object part segmentation, and scene semantic segmentation. Such network architecture was called PointNet and was proposed in 2017 [35]. The network does not contain any convolution layers, but instead it relies on simpler multi-layer perceptron networks (MLP), which are feedforward neural networks consisting of fully connected layers. As seen in Figure 3.9, MLPs are the core of the model. They are used to map 3D input data to higher dimensional embedding spaces,

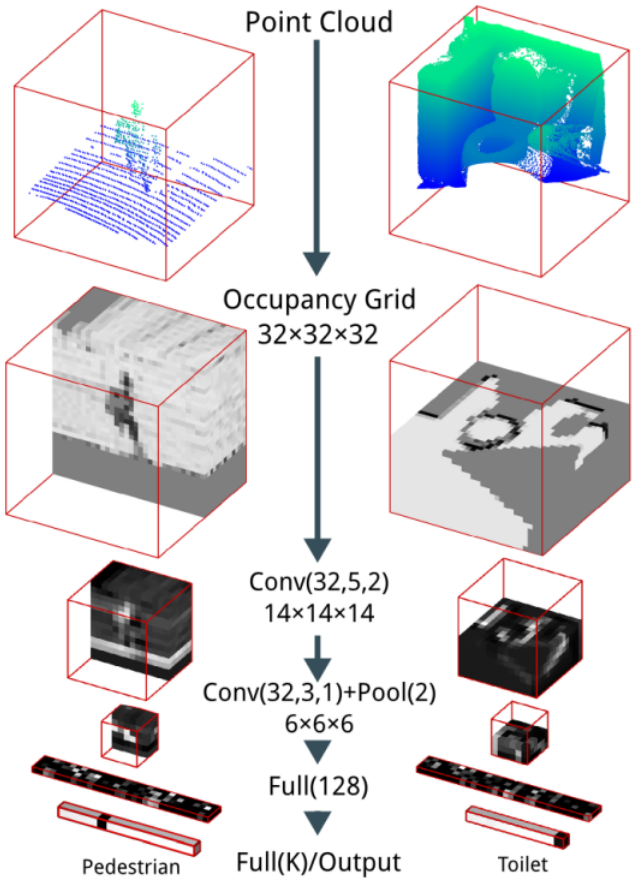


Figure 3.8: An example of CNN architecture (VoxNet) designed to perform 3D object classification. Point cloud data is transformed into 3D voxel occupancy grid (which are cross-sectioned for visualization purposes) in order to be handled by 3D convolutions. Source: [31].

and together with the max pooling as symmetric function they are meant to address the main challenge regarding point cloud processing (Section 2.1.1): the requirement for permutation invariance. As segmentation tasks require both global and local features to be learned, the max pooling layer also serves the purpose of aggregating both local and global information. Before feature extraction, the network employs a T-Net module in order to align input to a canonical space. The same alignment modules are used between the two main MLPs modules in order to align the extracted features as well.

3.3.3 Pointnet++: Hierarchical MLPs

PointNet has placed itself at the center of attention and is generally considered to be the turning point for models that are able to directly process point clouds. Despite this

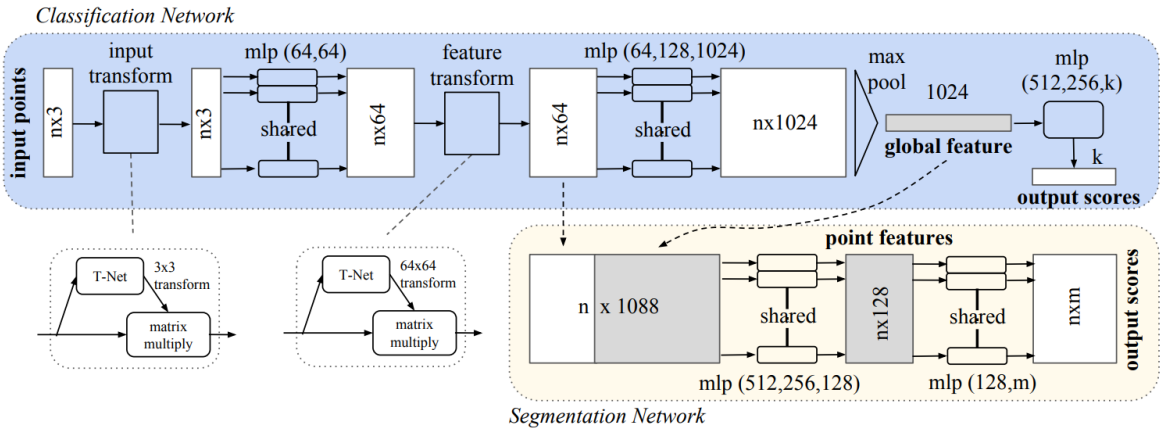


Figure 3.9: PointNet model architecture. Both the classification and the part segmentation is visualized. Source: [35].

achievement, the architecture does also has its shortcomings, namely not being able to leverage local spatial relationships of the input data. To address this, the same authors published PointNet++ [37], a new architecture build around the idea that local features need to be extracted at each step in a hierarchical manner, similarly to the way CNNs are able to exploit local structures in order to extract features that get increasingly more complex along the layer hierarchy. This is performed by partitioning the input point clouds into local regions from which extracting local features is beneficial, exploiting the useful geometric structures from local neighborhoods. Then, the local features extracted are grouped together, and then processed in order to hierarchically extract higher level features. Figure 3.10 is able to offer a visual representation of the feature extracting process. In the architecture we can identify set abstraction levels that are responsible for the sampling, grouping and encoding of local region patterns into feature vectors.

For the part segmentation version of the network, features are hierarchically propagated such that distance based interpolation can be performed, and the resulting interpolated features are concatenated with point features produced in the previous set abstraction levels, repeating the process in order to reconstruct the original set of point features used for per-point segment labeling.

3.3.4 Generative Adversarial Networks

As previously mentioned (Section 3.1.2), generative models are often employed in self-supervised manners to address many generative tasks for which training is challenging if done in a supervised way.

Introduced in 2014 [17] generative adversarial networks (GANs) are generative models that employ two neural networks, jointly trained one against the other in order to simulate supervised training. Generative and discriminative paradigms are combined in order to formulate a training process where, while the first model is trained on a generative task,

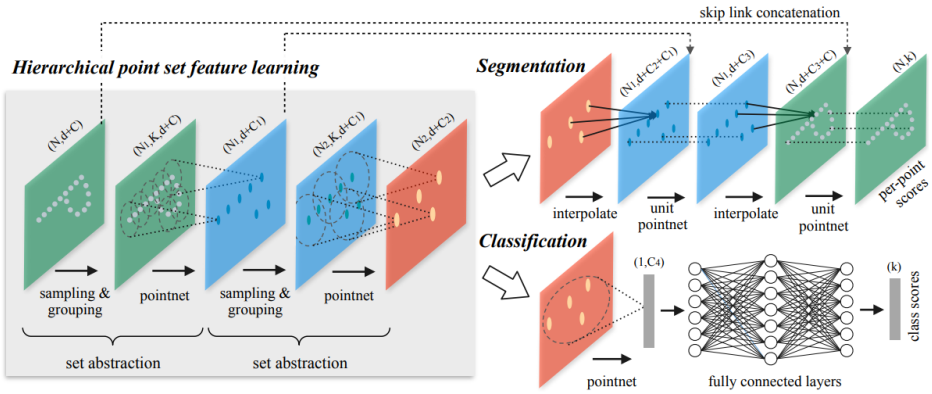


Figure 3.10: PointNet2 model architecture. Source: [37].

the second one is trained to act as supervisor towards the first. During training, the output data generated by the first model (*generator*) is used together with actual real data in order to formulate a discriminative self-supervised task for the second network (*discmininator*), which has to evaluate wether the generated data comes from the original dataset or not, essentially checking the generated output for authenticity. Whatever the type of the generated data, the goal of the discriminator is to figure out whether or not it is good enough to be mistaken for real.

The detailed steps are:

1. The generator network takes in its input data (which can be either random noise or meaningful data) and returns an instance of generated data. No supervision is involved in the process.

2. The discriminator network is fed both the generated image and an instance of real data from the target dataset. The correct labels therefore are going to be *fake* and *real* for the respective data items, which are automatically retrieved, making the discriminative process self-supervised.

3. The discriminator network takes the two instances of data and decides which of them is the one coming from the dataset (*real*) and which of them is the one generated by the generator network (*fake*).

4. The feedback from the discriminator is used to close the loop with the generator, which will learn to generate new samples that are closer to the domain by updating its parameters.

The interaction between the two models during the training process can be described as a zero-sum game, where both players compete to minimize their loss, which is one the opposite of the other, and whose Nash equilibrium is reached when the two players cannot improve their objective, implying that the generator is able to generate samples so convincing that the discriminator cannot but return equal scores for both.

Chapter 4

Methods

The purpose of this chapter is to offer a more practical formulation of the colorization problem on 3D point clouds. In doing so, the chapter will contain a review of the methodologies, datasets and models employed in this work thesis. A description on how the colorization task has been implemented, as well as how the models have been adjusted to be able to learn to colorize point clouds will be contained. The point cloud datasets and the steps required for them to be used for this task, as well as how the neural network models have been adapted to perform the colorization tasks. Furthermore, a brief analysis of the few scientific publications that layed the groundwork for the development of this thesis will be reserved.

4.1 Self-Supervised Point Cloud Colorization

As stated in Section 3.1.2, self-supervision tasks are meant to drive the model into learning high-level representational features of the data that, when carried over to downstream tasks, can improve the results both in terms of increased performance as well as in terms of the amount of labeled data needed during the training of the supervised task in order to reach the same level of performance.

4.1.1 Self-Supervision as auxiliary task (Multitask)

Given the advantage of the multi-task paradigm and the encouraging results previously achieved in literature with this approach applied to point clouds [4], this path was the first one to be explored. However, this method has its own disadvantages, and it is not always the best option. Reviewed below are some of the challenges that the multi-task approach has posed.

Architectural challenges. While both the colorization task and the supervised downstream one require the model to receive as input the raw point cloud data, formally (x, y, z) floating-point value tuples for each data point, their output are different. In the

first case, the model needs to output color data for each point received, formally a mapping $(x, y, z) \rightarrow (r, g, b)$, where the mapping result is a three-value tuple encoding red, green and blue colors according to the RGB additive color standard.
The output differentiation is not the only reason for which the model branching is required. In fact, as described in 3.2, the deeper is the layer within the neural network model, the more specialized are the features extracted. The first layers contain neurons that activate when basic geometric features are detected, while the deeper ones are more specialized in extracting more high-level features which are more directly linked to the desired output. This means that the very last layers within a model that have been trained to perform, for instance, shape classification cannot be expected to perform well in colorization at the same time. Output differentiation and layer specialization are the principal reasons that architectures that perform auxiliary tasks need to branch off in order to solve both tasks correctly. The branching point needs to be chosen such as the layers that are not deep enough to be specialized are shared, while the deeper layers that have learned to extract specialized features for the task are not. Sharing the backbone of the model but not the specialized deep part is essential as it can benefit from both of the branches' propagated signal. In factc during the backpropagation phase, the gradients from both branches will be propagated first through their own branch, and then merged together in order to be passed through the shared backbone. Figure 4.1 shows a graphic representation of a generic model architecture designed to perform both the auxiliary colorization task as well as the main task. Figuring out the correct branching point as well as how deep the new branch needs to be is an architectural design challenge on its own.

Technical challenges. A new branch within the model means a bigger model in terms of parameters. Model parameters refer to weights and biases contained in all the layers, which can easily sum to millions in most common models, and even tens of billions in the most ambitious ones. If the model at hand is already big, adding a hefty percentage in model parameters with another branch might not be always feasible, since training large models presents many technical constraints.

4.1.2 Self-Supervision as pretext task (Multistage)

Compared to the multi-stage approach, the model usually needs only a few adjustments in order to be trainable on the pretext task alone, and it hardly involves any change in the number of parameters. Separating the two tasks in two different training phases also allows for more freedom when designing the pretext, as it allows problem separation.
In this case, passing from a multi-task approach to a multi-stage one was especially beneficial since it allowed for the colorization pretext task to be addressed with the help of Generative Adversarial Networks (Section 3.3.4), which ended up to be the turning point for this thesis work. While not impossible, the adoption of GANs for colorization in a multi-task manner would have been both an architectural challenge as well as a technical one in terms of model size. As mentioned in the previous chapter, multi-staging the pipeline comes with some costs, the most obvious one of having to do double the training necessary, as opposed to only one, in order to be able to verify the results. Given that

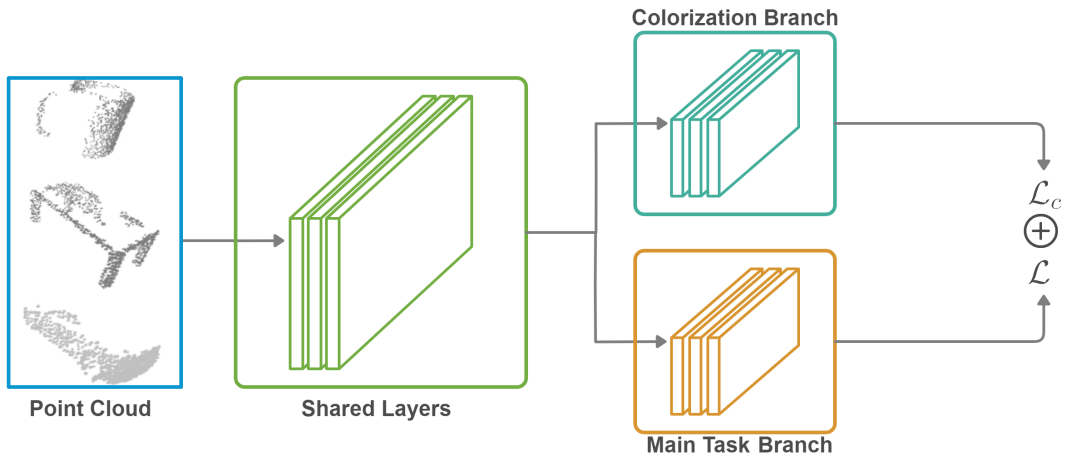


Figure 4.1: Visual representation of a generic model architecture able to perform point cloud colorization together with a supervised task in a multi-task fashion. The first layers of the model are shared, while the deeper ones are divided in two branches, each receiving low-level features from the shared layers and trying to optimize their own losses. During backpropagation, the gradients follow the opposite direction of the arrows as they merge when passing through the shared layers.

single training runs can sometimes spread over multiple days, developing a fruitful training process ended up being a lengthy process. While the multi-stage approach still has its disadvantages, in this case the advantage of handling less complex models ended up outweighing them. In addition, decoupling the two tasks had also posed the occasional advantage of being able to reuse the model trained on colorization for multiple runs on the downstream task, either to better tune the training or in different few-shot scenarios, where the training data for the downstream task has been purposely reduced (Section **??**).

4.2 Datasets

There are two 3D point cloud datasets that have played an elemental role in this thesis work for the colorization as well as the part segmentation task.

4.2.1 DensePoint

The DensePoint dataset is a synthetically-generated 3D object point cloud dataset presented in [8], which has been constructed using information from both ShapeNet [10] and ShapeNetPart [51] datasets to contain both colored and part segmented 3D object point clouds. While the original ShapeNet contains over 50,000 colored mesh models across 55 categories and the later published ShapeNetPart contains over 30,000 point clouds with part labels, DensePoint is the result of an intersection of the two datasets, which ended

up containing over 10,000 3D point clouds that contain both color information as well as part labels, which makes it the perfect choice for the purpose of this thesis.

Table 4.1: DensePoint dataset broken down into its object categories and number of part labels for each category.

Category	Guitar	Knife	Pistol	Lamp	Chair	Table	Mug	Car
No. instances	611	266	166	790	1990	3860	66	402
No. Parts	3	2	3	4	5	3	2	4
Category	Bag	Cap	Earphone	Laptop	Skateboard	Rocket	Motorbike	Plane
No. instances	57	31	36	338	127	29	159	1492
No. Parts	2	2	3	2	3	3	6	4

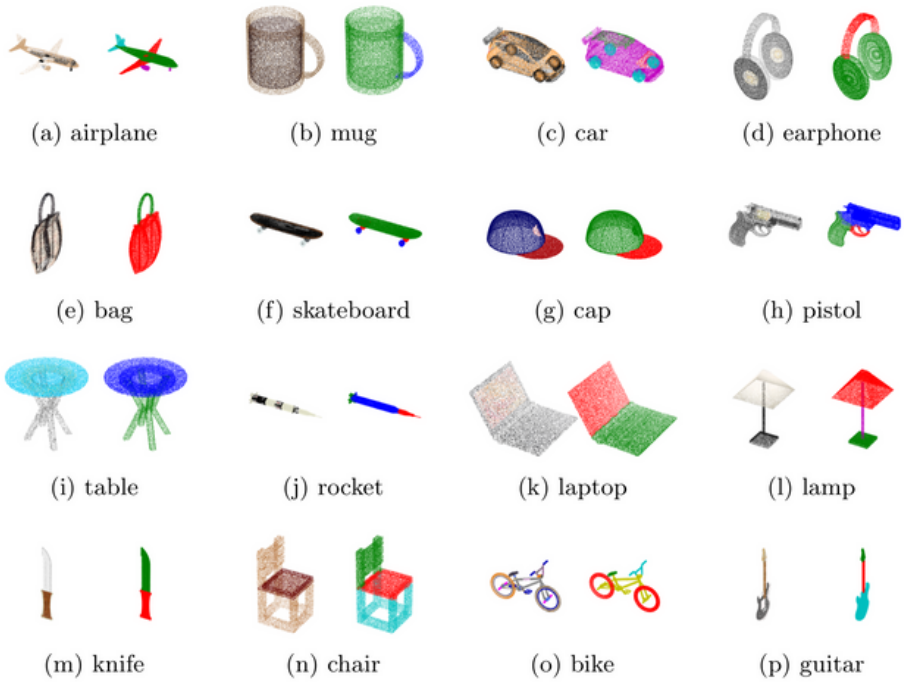


Figure 4.2: Visualization of the type of point clouds included in the DensePoint dataset, one example per object category. On the left RGB colored point cloud, on the right the color serves to segment the object into its individual parts.

From a practical view, the dataset contains individual object point cloud `.ply` files, each containing the following:

- (x, y, z) spacial information as normalized floating-point values on a unit sphere (range $[-1, 1]$)
- (r, g, b) color information as integer values in range $[0, 255]$ which had to be normalized to $[-1, 1]$ for training

- object part label index (range $[0, 49]$) for each data point
- object category label index (range $[0, 15]$) for each point cloud

Each point cloud counts tens of thousands of data points each, therefore in order to normalize the number of data points across all the objects in the dataset and for the purpose of model training, each shape point cloud was undersampled in two resolutions: 2048 and 4096 data points only.

4.2.2 ScanNet

ScanNet [12] is an ambitious project which resulted in a dataset consisting of real, indoor scenes containing both spatial as well as color information. The dataset contains 707 scenes distributed over 1513 3D scans, all composed from 2.5M RGB-D images (see Section 2.2.1 about RGB-D technology). The resulting 3D point clouds represent therefore not individual objects, but entire three-dimensional scenes, each of them with colored information, and semantic object labels.

The reason this dataset was considered for this thesis work is due to the very large amount of realistically colored point clouds. In fact, while the DensePoint dataset (Section 4.2.1) containing individual 3D objects point clouds that are not only colored, but also segmented into parts allows for both the colorization and part segmentation task to be performed on the same point clouds, it has to be noted that the objects' color are of synthetic origin, meaning that the colors represented are not always realistic, nor consistent within the category. Instead, color information from the ScanNet dataset is acquired from real scenes, meaning that objects of the same type are expected to be similarly colored. The presence of such color consistency implies that a colorization task on such data might represent an easier task for a model to learn. The insight is that despite the data from ScanNet being from a different distribution than the one the downstream task is performed on, the realistic color information alone should be able to give an edge during the colorization task, and if so the resulting data representation learned during the pretext training can be carried over to improve the performance in part segmentation tasks performed on objects from DensePoint.
The accessible contents of the dataset include multiple RGB-D sensor streams with color, depth and camera poses information as well as derived point clouds with RGB color and semantic labels. For the purpose of this thesis, the following information was retrieved for each of the 1513 scans contained in the dataset:

- (x, y, z) spacial information as floating-point values in original scale
- (r, g, b, α) color information as integer values in range $[0, 255]$
- per-point semantic object label

Since the single point cloud scenes were representing room-size scenes rather than object-size shapes, the complexity and the amount of data points (millions per single cloud) was unnecessarily high and far too much to be feasible using them. At the same

time, undersampling a whole scene to a more feasible resolution meant that the density of points per object had to be way too low for the model to pick up shape information. The solution therefore ended up being dividing the scene into $n \times n \times n$ voxels, each of them undersampled to more feasible resolutions (2048 and 4096). The n value had to be big enough that objects could be included entirely, but small enough so that data points density could still allow for the model to pick up useful information about object shapes. Values of $1m$ and $2m$ ended up being chosen for the experiments.

Figure 4.3: A few scene visualizations from ScanNet. Colors represent semantic segmentation labels. Source: [12].

Finally, as it can be observed in Figure 4.3, objects are not homogeneously distributed in the rooms: there are plenty of scene portions that have no object at all but empty floor, and areas that are instead very object-dense. In order to assure a certain data quality standard and avoid cases where the sampled voxel would contain no useful objects to be colorized, as well as making sure that in voxels that do contain objects, their point density is as high as possible, a preprocessing step has been applied to all scenes in order to strip the clouds of points that don't belong to objects at all. For this purpose, data points deemed to be of low interest, namely those semantically labeled as `'wall'`, `'ceiling'`, `'floor'` and `'blinds'`, were removed from the scene point cloud, as such areas are generally mono-colored and contain no useful shape information. As the filtering

process would end up creating multiple empty spots within the scene, portioning had to be done by checking if the resulting voxel sampled from the point cloud would have at least M points for it to be used for the training, with $M = \{2048, 4096\}$ depending on the desired resolution. Furthermore, as the α opacity value was considered to be unnecessary, it was dropped from the points cloud data. Figure 4.4 shows the resulting scene after the filtering of unwanted data points.

The result of the described process were $n \times n \times n$ voxels of point cloud data, each containing:

- (x, y, z) spacial information as floating-point values, normalized in original scale
- (r, g, b) color information as integer values in range $[0, 255]$, which had to be normalized to $[-1, 1]$ during training
- either 2048 or 4096 data points, depending on the desired resolution

4.3 Evaluation metrics

In order to evaluate the performance of a model on part segmentation tasks, two metrics were employed:

- *Accuracy.* As a standard way to define classification results, an accuracy has been employed, interpreting the part segmentation results as a point-wise classification over 50 classes. The accuracy score has been computed as

$$\mathbf{A} = \frac{\#\text{ of correctly classified points}}{\#\text{ of points}} \tag{4.1}$$

- *Intersection-over-Union.* 3D Object part segmentation is a process of dividing an object shape in different segments, where each can be labeled as an individual part within the object. To better evaluate the quality of segmentation within a single object, mean Intersection-over-Union (mIoU) has been employed. For each part type in a certain object category, IoU score is computed as a ratio between the points that have been correctly classified (intersection) and the union of groundtruth and predicted points for that part type. To get the mIoU for a shape, IoUs for all part types in that shape category are averaged. This type of metric is very common in object detection tasks, as it accounts for the fact that the segmentation of the image (or point cloud) to detect the target will hardly perfectly coincide with the ground truth one, yet being able to measure how close it gets is useful. Differently from the overall accuracy, this metric compensates for different class frequencies, which will result in lower scores with respect to the overall accuracy metric, as the cloud point objects dataset is not balanced.

(a)

(b)

Figure 4.4: An example of a ScanNet scene point cloud (a) that has had data points that do not belong to objects filtered out (b).

Chapter 5

Experiments and Results

This chapter collects the main experiments and their results relative to the colorization outcome and how well does the pretext translate into the downstream part segmentation task. For a more detailed review of the datasets and metrics mentioned, refer to Chapter 4.

5.1 Environment

All model architectures as well as training scripts were written in Python using the open source machine learning framework PyTorch v1.8. In terms of hardware, all of the neural network training tasks were performed on a NVIDIA Titan V GPU, provided by Politecnico di Torino. The high-end GPU was essential, as training big neural network models on regular consumer hardware would be unfeasible.

5.2 Colorization Architecture

Colorization tasks were formulated as a self-supervised generative problem. Following the approach in [8], a generative adversarial network (see 3.3.4) system was designed in order to generate colors by having as input the only the data points within the point cloud.

As shown in figure 5.1, two models are employed for the task:

- Generator model, trained to generate per-point RGB color values by being feed normalized point clouds. Since the purpose of the colorization is to transfer the trained model to part segmentation, the generator has to share most of its architecture with the final segmentation network.

- Discriminator model, trained to evaluate the authenticity of the generated colors. Both generated and real colors, as well as the original point cloud are received as input, and a binary classification result is returned as output.

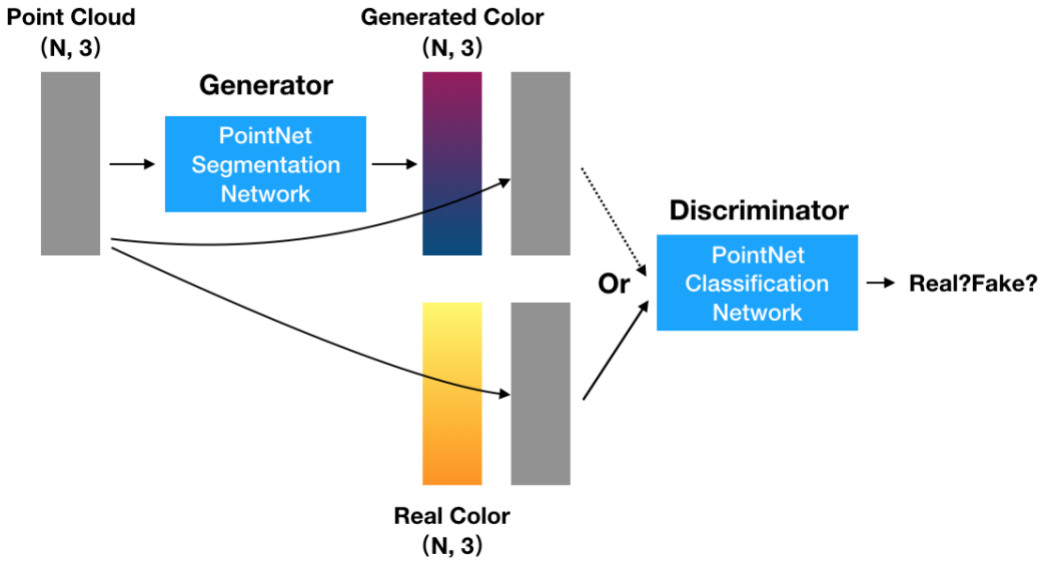


Figure 5.1: Architecture of the GAN model for color generation. The generator takes regular point clouds as input, and outputs RGB color for each point. Generated color and original point cloud are concatenated before being forwarded to the discriminator, along with the real color and point cloud, for discrimination.

As mentioned, when adapting the network segmentation to perform color regression architecture modifications has to be kept low:

- Adapting the PointNet model only required to adjust the number of neurons of the output layer and the addition of a Tanh activation function in order to keep outputs in range $[-1, 1]$.
- PointNet++ part segmentation network also makes use of shape labels in order to facilitate the task. To maintain this feature, it would be necessary that all input point clouds also be labeled within object categories. While this type of labels are available for datasets like DensePoint, scene datasets like ScanNet that were also used for colorization don't necessarily contain them. Therefore, object category input was removed and the model adjusted accordingly. As with the PointNet, the output layer dimensions were adjusted and a Tanh non-linearity was added as well.

Objective function

As mentioned previously, the goal of the generator is to produce plausible point-wise color values for point clouds, while the discriminator needs to learn to discriminate real colors from the fake ones. Following [8], a combination of conditional GAN loss defined as

$$L_{cGAN}(G, D) = E_{x,y}[\log(x, y)] + E_{x,z}[\log(1 - D(x, G(x, z)))], \tag{5.1}$$

and L1 loss defined ass

$$L_{l_1}(G) = E_{x,y,z}\left[\|y - G(x,z)\|_1\right] \tag{5.2}$$

which serves the purpose of driving the generator not only to produce colors that can fool the discriminator, but also colors that are not too far off with respect to the original ones. In the loss function, x is the input $N \times 3$ point cloud, and y the output containing per-point RGB color. To approximate the presence of z the random input usually used for traditional GANs, the dropout layer is kept activated during evaluation as well in order to introduce variation in the color generated. The resulting object function is therefore

$$G^* = \arg\min_G \max_D L_{cGAN}(G,D) + \lambda L_{l_1}(G), \tag{5.3}$$

where the discriminator aims at minimizing the conditional GAN loss and L1 loss, and the discriminator tries to maximize it. The purpose of the λ value is to adjust the weight of the L1 loss relative to the conditional GAN one.

5.3 Colorization on Object Point Clouds

Training generative adversarial model is a complex process and poses many challenges, as two models need to be balanced during the whole process and training stability is difficult to achieve. If the generator is a fast learner and the discriminator fails to keep up, then the training will hardly converge to anything useful, as the feedback from the discriminator will get less and less meaningful over time.

A stable training process was achieved with the following hyperparameters:

- Adam optimization algorithm for both the generator and discriminator
- learning rates of 0.0001 for the discriminator and 0.001 for the generator, decayed by a factor of 0.5 every 10 epochs
- a total of 4096 data points per point cloud
- batch size of 16 point clouds
- training for 200 epochs

Figure 5.2 shows a few examples of the results.

5.3.1 Results in Object Part Segmentation

As previously mentioned, weights from all but the final classification layer were transferred to the part segmentation task, which was performed on object point clouds from the DensePoint dataset.

The following hyperparameters were used during both baseline and colorization transfer training:

Figure 5.2: A few example of object point cloud colorizations from PointNet.

- Adam optimization algorithm
- learning rate of 0.001, decayed by a factor of 0.5 every 20 epochs
- a total of 4096 data points per point cloud
- batch size of 16 point clouds
- training for 200 epochs

Table 5.1 collects the results on object art segmentation, both for PointNet and PointNet++, with and without the colorization task. It can be noted that while the colorization task does bring a performance boost for the PointNet model (almost 1% in mIoU), it fails with the PointNet++ model.

Table 5.1: Results on object part segmentation, comparing the performance of the baseline model with the ones achieved thanks to the colorization pretext done on objects.

Model	Overall accuracy [%]	mIoU [%]
PointNet (baseline)	92.94	82.45
PointNet colorization (objects)	93.02	83.57
PointNet++ (baseline)	93.56	84.92
PointNet++ colorization (objects)	92.86	83.75

Results with Decreased Training Data

Table 5.2 shows performance results on part segmentation where the training data has been reduced to multiple percentages. The colorization PointNet model is able to maintain an advantage over the baseline.

Table 5.2: Part segmentation results where the training data for the downstream has been reduced to from 1% to 50%.

Model	1%	2%	10%	20%	30%	50%
	Overall accuracy [%]					
PointNet (baseline)	70.26	81.12	89.6	90.59	91.83	92.07
PointNet colorization (objects)	80.17	84.62	89.76	91.51	91.88	92.22
PointNet++ (baseline)	80.53	84.81	90.42	91.45	92.63	92.79
PointNet++ colorization (objects)	61.05	73.04	85.69	88.7	90.0	91.78
	mIoU [%]					
PointNet (baseline)	58.12	70.67	77.68	79.48	81.91	81.14
PointNet colorization (objects)	69.24	74.52	78.73	80.35	81.13	82.07
PointNet++ (baseline)	72.07	75.06	80.32	81.98	83.04	83.32
PointNet++ colorization (objects)	49.86	61.12	74.79	74.48	78.15	81.95

5.4 Colorization on Real Scenes

Point clouds from ScanNet dataset of indoor scenes was used to train the models on colorization, with the same training configuration and hyperparameters as the ones used in Section 5.3. The data was prepared according to the modalities described in 4.2.2. Squares of dimension $(2 \times 2 \times 2)m$ were extracted and sampled to 4096 points as input point cloud for the generator.

5.4.1 Results in Object Part Segmentation

Displayed in 5.3 are the part segmentation results comparing baseline performances with the ones achieved with colorization on scenes. Although the data domain is not the same, the results are in line with the ones achieved with colorization pretext on object point clouds.

Table 5.3: Results on object part segmentation, comparing the performance of the baseline model with the ones achieved thanks to the colorization pretext done on real scene point clouds.

Model	Overall accuracy [%]	mIoU [%]
PointNet (baseline)	92.94	82.45
PointNet colorization (scenes)	93.15	83.43

Results with Decreased Training Data

Collected in 5.4 are the results regarding part segmentation performances with reduced data. Again, while for the very low percentages the scores seem to be lower than the ones

achieved with colorization on object point clouds, they are still higher when compared to the baseline results.

Table 5.4: Part segmentation results where the training data for the downstream has been reduced to from 1% to 50%.

Model	1%	2%	10%	20%	30%	50%
	Overall accuracy [%]					
PointNet (baseline)	70.26	81.12	89.6	90.59	91.83	92.07
PointNet colorization (scenes)	71.91	82.01	90.18	91.33	91.77	92.23
	mIoU [%]					
PointNet (baseline)	58.12	70.67	77.68	79.48	81.91	81.14
PointNet colorization (scenes)	59.36	71.11	78.84	81.20	81.48	82.47

Chapter 6

Conclusion

The objective of this thesis work was to develop a self-supervised point cloud colorization method in order to investigate its potential in improving the performance of deep neural network models on object part segmentation tasks. The proposed method based on the Generative Adversarial paradigm to generate color information from point clouds has proven to be able to drive the model into learning good data representation that can be exploited for downstream tasks, even when the pretext data is from a different domain. Of equal importance is the robustness that the model gains by being able to maintain an edge in scenarios where downstream training data is scarce, as building annotated 3D point cloud datasets can still be a challenge.

Future work could aim at finding ways to improve even more the colorization capabilities, and to extend the paradigm to one that can be applied to different models that have each taken different approaches towards point clouds.

Bibliography

[1] A busy street scene captured by velodyne's alpha puck, 2019. URL https://reut.rs/2NHMIXx. Courtesy Velodyne/Handout via REUTERS. [Online; accessed November 24, 2021].

[2] Super mario all-stars: Mario, 2020. URL http://pixelartmaker.com/art/aabc52982279e8e. [Online; accessed November 24, 2021].

[3] Eman Ahmed, Alexandre Saint, Abd El Rahman Shabayek, Kseniya Cherenkova, Rig Das, Gleb Gusev, Djamila Aouada, and Bjorn Ottersten. A survey on deep learning advances on different 3d data representations, 2019.

[4] Antonio Alliegro, Davide Boscaini, and Tatiana Tommasi. Joint supervised and self-supervised learning for 3d real-world challenges, 2020.

[5] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

[6] Saifullahi Aminu Bello, Shangshu Yu, Cheng Wang, Jibril Muhmmad Adam, and Jonathan Li. Review: Deep learning on 3d point clouds. *Remote Sensing*, 12(11), 2020. ISSN 2072-4292. doi: 10.3390/rs12111729. URL https://www.mdpi.com/2072-4292/12/11/1729.

[7] Y. Bengio. Learning deep architectures for ai. *Foundations*, 2:1–55, 01 2009. doi: 10.1561/2200000006.

[8] Xu Cao and Katashi Nagao. Point cloud colorization based on densely annotated 3d shape dataset, 2018.

[9] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features, 2019.

[10] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[11] Javier Civera and Seong Hun Lee. Rgb-d odometry and slam. *Advances in Computer Vision and Pattern Recognition*, page 117–144, 2019. ISSN 2191-6594. doi: 10.1007/978-3-030-28603-3_6. URL http://dx.doi.org/10.1007/978-3-030-28603-3_6.

[12] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[13] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction, 2016.

[14] Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks, 2015.

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. doi: 10.1109/CVPR.2012.6248074.

[16] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations, 2018.

[17] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[18] Xian-Feng Han, Jesse Jin, Ming-Jie Wang, and Wei Jiang. Guided 3d point cloud filtering. *Multimedia Tools and Applications*, 77, 07 2018. doi: 10.1007/s11042-017-5310-9.

[19] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949. ISBN 0-8058-4300-0.

[20] Long Hoang, Suk-Hwan Lee, Oh-Heum Kwon, and Ki-Ryong Kwon. A deep learning method for 3d object classification using the wave kernel signature and a center point of the 3d-triangle mesh. *Electronics*, 8(10), 2019. ISSN 2079-9292. doi: 10.3390/electronics8101196. URL https://www.mdpi.com/2079-9292/8/10/1196.

[21] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)*, 35(4):110:1–110:11, 2016.

[22] Hanbyul Joo, Tomas Simon, Xulong Li, Hao Liu, Lei Tan, Lin Gui, Sean Banerjee, Timothy Scott Godisart, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. Panoptic studio: A massively multiview system for social interaction capture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[23] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model of component-based shape synthesis. *ACM Transactions on Graphics*, 31, 07 2012. doi: 10.1145/2185520.2185551.

[24] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models, 2017.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[26] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. doi: 10.1038/nature14539.

[27] Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

[28] Yangyan Li, Soeren Pirk, Hao Su, Charles R. Qi, and Leonidas J. Guibas. Fpnn: Field probing neural networks for 3d data, 2016.

[29] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/f5f8590cd58a54e94377e6ae2eded4d9-Paper.pdf.

[30] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[31] 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, 2015. doi: 10.1109/IROS.2015.7353481.

[32] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles, 2017.

[33] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting, 2016.

[34] Charles R. Qi, Hao Su, Matthias Niessner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and multi-view cnns for object classification on 3d data, 2016.

[35] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017.

[36] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from rgb-d data, 2018.

[37] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

[38] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[39] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. Kinect range sensing: Structured-light versus time-of-flight kinect, 2015.

[40] Jonathan Sauder and Bjarne Sievers. Self-supervised deep learning on point clouds by reconstructing space, 2019.

[41] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network, 2020.

[42] Richard Socher, Brody Huval, Bharath Bath, Christopher D Manning, and Andrew Ng. Convolutional-recursive deep learning for 3d object classification. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/3eae62bba9ddf64f69d49dc48e2dd214-Paper.pdf.

[43] Shuran Song and Jianxiong Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images, 2016.

[44] Minhyuk Sung, Vladimir G. Kim, Roland Angst, and Leonidas Guibas. Data-driven structural priors for shape completion. *ACM Trans. Graph.*, 34(6), oct 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818094. URL https://doi.org/10.1145/2816795.2818094.

[45] Ali Thabet, Humam Alwassel, and Bernard Ghanem. Mortonnet: Self-supervised learning of local features in 3d point clouds, 2019.

[46] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, 2015.

[47] Karlijn Willems. Keras tutorial: Deep learning in python, 2019. URL https://www.datacamp.com/community/tutorials/deep-learning-python. [Online; accessed November 10, 2021].

[48] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes, 2015.

[49] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes, 2015.

[50] Hao Ye, Zuxuan Wu, Rui-Wei Zhao, Xi Wang, Yu-Gang Jiang, and Xiangyang Xue. Evaluating two-stream cnn for video classification. *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, Jun 2015. doi: 10.1145/2671188.2749406. URL http://dx.doi.org/10.1145/2671188.2749406.

[51] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016.

[52] Ling Zhang and Zhigang Zhu. Unsupervised feature learning for point cloud by contrasting and clustering with graph convolutional neural network, 2019.

[53] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization, 2016.

[54] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. State of the art on 3d reconstruction with rgb-d cameras. *Computer Graphics Forum*, 37:625–652, 05 2018. doi: 10.1111/cgf.13386.