A Contrastive Learning framework for Self-Supervised Pre-Training of 3D Point Cloud Networks with Visual Data

by

Andrej Janda

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Aerospace Science and Engineering
University of Toronto

Abstract

A Contrastive Learning framework for Self-Supervised Pre-Training of 3D Point Cloud Networks with Visual Data

Reducing the quantity of annotations required during supervised training is vital when labels are scarce and costly. This reduction is especially important for segmentation tasks involving 3D datasets, which are often significantly smaller, and more challenging to annotate, than their image-based counterparts. Self-supervised pre-training on large unlabeled datasets is one way to reduce the amount of manual annotations needed. Previous work has focused on pre-training with point cloud data exclusively; this approach often requires two or more registered views. In this thesis, we combine image and point modalities, by first learning self-supervised image features and then using these features to train a 3D model. By incorporating visual data, which is often included in many 3D datasets, our pre-training method requires a single scan of a scene only. We demonstrate that our pre-training approach, despite using single scans, achieves comparable performance to other multi-scan, point cloud-only methods.

Acknowledgements

Although this thesis names a single author, the work enclosed is a product of many interactions with friends and colleagues without which none of the following pages would have been possible. I would like to thank all those who supported me throughout my graduate studies. First and foremost I would like to thank Prof. Jonathan Kelly for taking a chance on me both as an undergraduate and graduate student. Throughout my studies, you were a source of technical support that kept me going on the right path. As much as research is an academic pursuit, it is also an emotional one, and your words really helped me get through the hardest days. Thank you also for shielding me from the complex project administration that allowed me to focus on my research. I also have to thank all my colleagues at the STARS Laboratory for all their advice, support and valuable feedback. I would like to especially thank Philippe Nadeau and Adam Hall for all of our philosophical talks on robotics and for creating an open space to bounce ideas around. An especially large thank you goes out to Brandon Wagstaff who not only helped me figure out core parts of my thesis but also helped edit much of the work I submitted for publication. Thank you also Edwin Ng for all your helpful comments and for being my partner on the ARIBIC project. I would like to thank my parents, Miroslav and Jaroslava, and my siblings Katherine and Martin for being there for me always. Thank you Bethany and Mazhar for all of our conversations and for believing in me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Maintaining an accurate representation of the environment is necessary for many tasks in robotics, such as navigation, planning, obstacle avoidance, and scene understanding. The two most common scene representations, particularly for scene understanding tasks, are *images*, and *point clouds*. These representations are built from data captured by specialized sensors for each modality. Images are most often captured using cameras, whereas point clouds can be captured from a variety of sensors, such as lidars and stereo cameras. Cameras are mature, abundant, and relatively inexpensive sensing devices. The images that cameras produce contain dense, feature-rich information, which has made visual data useful for robotics applications. Algorithms that process overlapping sequences of images have proven extremely effective in estimating position and orientation changes [36], recognizing objects [35], tracking the motion of features [48], stitching images together [37], and segmenting objects and structures [19]. However, the lack of depth information in images limits how well they can be used on their own to model the 3D environment. For instance, images do not provide information about distances or object sizes. Objects in images are also prone to occlusion. Therefore, images, on their own, without additional measurement information or processing, are insufficient for robotics tasks that require knowledge of metric distances in 3D space (e.g., for localizing objects within a 3D scene).

Modelling the 3D world directly with point clouds circumvents many of the limitations inherent to images. Unlike images, point clouds contain metric information about distances (between points). Individual point cloud scans from multiple viewpoints can also be stitched together to reduce occlusions. Recent advances in sensing have made 3D data increasingly more available, dense, and accurate. Algorithms that process point clouds for various perception tasks have also been very successful [8, 25, 42–44]. Point clouds are appealing for perception in part because they can provide a more cohesive

understanding of a scene than can be gained from images alone.

Working with point clouds is not without challenges, however. In contrast to images which are dense and easy to collect, point clouds are sparse, harder to collect, and relatively less abundant. Point cloud data is more difficult to acquire since the sensors that capture point clouds are not as common or as inexpensive as standard cameras. Localizing individual scans within the global scene adds additional complexity, owing to the cost of careful point cloud registration or the need to build a robust localization and mapping pipeline.

Point clouds are also notoriously hard to annotate. The annotation difficulty is a key limiting factor for many state-of-the-art *data-driven* scene understanding algorithms that require large, annotated datasets [8, 19, 25, 43]. Generating point cloud labels requires (human) annotators to manipulate the point clouds by zooming, panning, and rotating to select points of interest. Annotators then have to separate points that belong to a particular object from any background and other occluded points. In comparison, generating image annotations requires annotators to draw polygons or rectangular boxes around objects (or classes) of interest in 2D only. Manipulating point clouds and identifying all 'object' points makes point cloud annotation significantly more challenging than image annotation.

The difficulty of annotating 3D data has resulted in considerable effort and labelling times for existing datasets. For example, the SemanticKITTI dataset [4], which has 518 square tiles of 100 metres length each, required 1,700 hours for annotators to label. ScanNet [11], which has 1,600 reconstructed scenes of indoor rooms, took about 600 hours to label [63]. Despite the substantial labelling time, 3D datasets are still significantly smaller than comparable image-only datasets.

1.1 Reducing Reliance on Annotations

The extensive labelling effort required for 3D data is the reason we seek, herein, to reduce the volume of annotations necessary. Although cumbersome, annotations are integral to many accurate, data-driven modelling efforts. The need for annotations could be avoided altogether through the use of explicit classical algorithms (in some cases), but data-driven approaches have proven more apt at modelling the complexity in many vision and robotics tasks.

The dependence on annotations varies depending on the choice of learning approach. At one extreme, *supervised learning* relies entirely on ground-truth labels and uses the discrepancy between the current model prediction and the given label to iteratively op-

timize the model. At the other extreme, *unsupervised learning* has no dependence on labels and instead attempts to find patterns in the raw data, usually through clustering. Without labels, however, unsupervised learning is generally much more challenging. *Semi-supervised learning* offers a compromise by combining aspects of both supervised and unsupervised learning at the same time. Semi-supervised learning reduces the reliance on annotations while still learning the desired classification or regression model, for example. Alternatively, *few-shot* and *zero-shot* learning frameworks seek to generalize to new labels using few or zero annotations. *Transfer learning* uses model parameters pre-trained on a similar task with a larger dataset as the initialization for supervised learning on a smaller, target dataset. Finally, *self-supervised learning* leverages unsupervised training on a large unlabelled dataset to initialize the parameters of a given model, which is subsequently trained with supervised annotations on a downstream task.

Self-supervised learning, in particular, has shown great success in fields such as natural language processing [13]. These results have encouraged researchers to investigate whether self-supervised learning can be of benefit in other domains. A common and popular type of self-supervised learning is *contrastive learning*. Contrastive learning aims to minimize the distance, specified by a given metric, between model outputs for 'similar' inputs, and to maximize the distance between outputs for 'dissimilar' inputs. Contrastive learning has proven to be highly effective for image classification tasks [6,18,57], matching fully supervised pre-training in terms of performance [17]. Once a model has been pre-trained, it can be easily and quickly deployed on any number of related tasks to improve performance. Recently, results from [21, 59, 63] have demonstrated that the contrastive framework can be applied to 3D data to train point cloud networks. These methods train a network to produce point-specific 3D features that are similar between matching points and dissimilar between non-matching points. We choose to pursue self-supervised learning to reduce reliance on annotations, since the approach has proven successful in various domains including point cloud processing.

1.2 Contributions

A key limitation of existing 3D pre-training methods for segmentation, for example, is that they utilize point cloud data for pre-training while neglecting the information-rich images that are often available as part of 3D datasets. We propose a pre-training method that leverages images as an additional modality, by learning self-supervised image features that can be used to pre-train a 3D model. Our learning method is split into two separate stages. The first stage (Stage 1) learns image features using a self-supervised contrastive

learning framework. The second stage (Stage 2) applies the same contrastive learning framework to pre-train a 3D model by making use of the self-supervised 2D features learned in Stage 1. By incorporating visual data into the pre-training pipeline, we obtain a notable advantage: only a single point cloud scan and the corresponding images are required during pre-training. The use of a single scan obviates the need for two or more overlapping 3D views of a point cloud, which is required by many point-only approaches. The use of a single scan improves the scalability of our pre-training approach, since we require raw 3D data only, as opposed to multiple scans that have been aggregated using a robust mapping pipeline for data association.

Through extensive experimentation, we compare our pre-training approach with existing point cloud-only approaches on several downstream tasks and across several datasets. We find that our method performs competitively with methods that use only overlapping point cloud scans. However, we note that the downstream performance of all methods varied extensively between datasets and available label proportions. Achieving the best performance may require a search over different self-supervised learning algorithms. Since our method does not require registered point clouds, it can be implemented relatively easily and is a good first candidate for pre-training.

In short, this thesis makes the following contributions:

- we describe a self-supervised method for extracting visual features from images and using them as labels to pre-train 3D models via a contrastive loss;
- we provide visualizations demonstrating that the features follow structures, such as lines and surface patches, that are present in the input image and point cloud;
- we show that our approach simplifies self-supervised pre-training on point clouds relative to other pre-training strategies;
- we demonstrate that a model trained using features learned from raw images improves performance on 3D segmentation and object detection tasks;
- we include an in-depth analysis of how different pre-training methods, including our own, perform on several standard datasets and tasks including segmentation and object detection.

Notably, our method yields more consistent performance gains than other, related algorithms. We achieve results that are comparable to pre-training methods that use multiple overlapping point cloud scans, despite having access to single scans and images only.

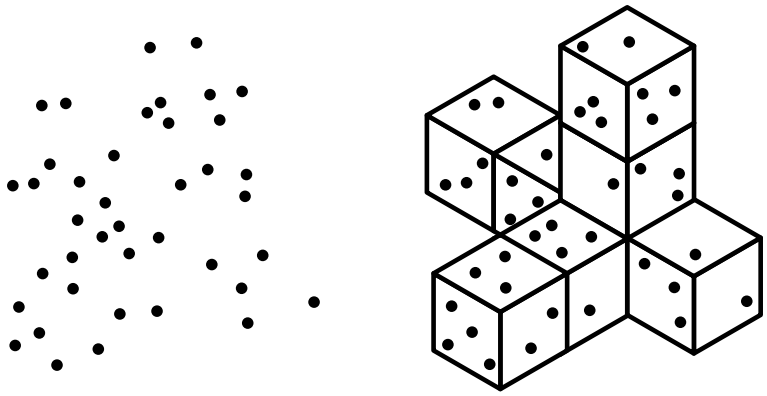
Chapter 2

Background

This chapter reviews the fundamental mathematical concepts and definitions required to understand the contents of the thesis. A brief overview of the data structures used to represent 3D scenes is provided first. Then, we give a short review of coordinate frames and transformations. Next, we discuss classical computer vision theory, with an emphasis on the relationship between point clouds, images, and features. Finally, recent advances in deep learning are detailed, in the context of point cloud and image processing; this is followed by a brief overview of self-supervised contrastive learning.

2.1 Data Structures for Scene Representations

The spatial form of the external world (i.e., a scene) can be represented by a variety of data structures with different dimensionalities. In this thesis, we consider 3D scenes and use either point cloud representations or images, which are 2D projections of the 3D world. The images we rely on are captured by a standard camera that focuses light rays onto a rectangular photosensor. The geometry of image formation is covered in Section 2.3. An image is represented as a three-dimensional tensor, denoted by $\mathcal{I} \in \mathbb{R}^{W \times H \times C}$, with width W, height H, and channel size C. Each image is made up of *pixels*, which are defined by integer or real values on a regularly-spaced grid, at coordinate $u \in [0, \ldots, W-1]$ along the x-axis and coordinate $v \in [0, \ldots, H-1]$ along the y-axis. As per convention, the u and v coordinates are indexed starting from the top left corner of the image. The channels of the image hold the colour information, with one channel per colour band. Standard colour images contain three channels, which define the red, green, and blue (RGB) intensity values for a given pixel; greyscale images contain a single (luminance) channel only. The image files used in this thesis store each channel value as an 8-bit positive integer, in the range from 0 to 255. An RGB colour image can be converted to a



Point Cloud Voxel Grid

Figure 2.1: Visualization of a point cloud and a corresponding voxel grid.

greyscale image using a weighted sum of the colour channels for each pixel, according to

$$Y = 0.299R + 0.587G + 0.114B, \tag{2.1}$$

where Y is the resulting luminance of the greyscale channel and R, G, and B are the red, green and blue channels of the colour image, respectively.[1] Colour images can also be augmented with an additional *depth* channel that stores the distance from the camera to the corresponding surface in the scene from which the light was reflected or emitted. This augmented image is referred to as an *RGB-D* image.

While an image represents the projection of 3D space onto a discrete 2D grid, a *point cloud*, denoted by $\mathcal{P}$ herein, can represent a 3D scene without quantization and without the need to project to a lower-dimensional space. Point clouds are sets of 3D points, where each point is defined relative to a given reference frame (see Section 2.2). Each point in $\mathcal{P}$ lies on a surface in the scene, (i.e., represents occupied space as opposed to free space). Point clouds are the direct output of sensors that acquire range measurements, or can be produced through the process of triangulation, for example, by stereo cameras. The main advantage of point clouds is that they encode metric distance information. Additionally, individual 3D measurements from different viewpoints can be stitched together to reconstruct the entire scene with fewer occlusions than for images (see Section 2.2). Although building a registration pipeline for scene reconstruction is challenging, the ability to do so when needed allows point clouds to more accurately model complex 3D environments. The point cloud data structure can also be augmented with other information, such as the colour or reflectivity of each point (analogous to the

[1] This conversion follows the ITU Radiocommunication Sector (ITU-R) BT.601 standard and extracts luminescence information more accurately than simply averaging the three colour channels together.

colour channels for images). A point cloud $\mathcal{P}$ is defined by a set of coordinates $\mathbf{r}_i$ and channel values $\mathbf{c}_i$,

$$\mathcal{P} = \left\{ \begin{bmatrix} \mathbf{r}_i \\ \mathbf{c}_i \end{bmatrix} \in \mathbb{R}^{3+C} \;\middle|\; \mathbf{r}_i \in \mathbb{R}^3 \,,\; \mathbf{c}_i \in \mathbb{R}^C \right\}, \tag{2.2}$$

where C is the number of channels assigned to each point (which may be zero).

It is also possible to discretize a point cloud into a *voxel grid*, which aggregates points into fixed-sized 'voxels' (i.e., *volume pixels*) in 3D space. A voxel grid can be represented as a four-dimensional tensor, $\mathcal{V} \in \mathbb{R}^{W \times H \times L \times C}$, with width W, height H, length L and channel size C. A non-empty voxel represents a cubic volume that is at least partially occupied by a physical object in the scene (i.e., for a voxel derived from a point cloud, non-empty voxels must contain at least one point). A visual comparison between a point cloud and a corresponding voxel grid is shown in Figure 2.1. When converting a point cloud to a voxel grid, the channel information stored by a specific voxel is derived from the channel values of the points that fall within the voxel. For colour values, the average colour of all points might be used, while for semantic labelling, the most common label among the points may be selected.

2.2 Coordinate Frames

Building a complete map of a 3D scene requires correctly merging many individual 3D scans. To aggregate 3D points from individual scans, the points must be expressed in a common reference frame. Following the formulation given in [3], a reference frame in three dimensions, denoted in vectrix form as $\underrightarrow{\mathcal{F}}_i$, is defined by three orthogonal unit vectors (i.e., axes),

$$\underrightarrow{\mathcal{F}}_i = \begin{bmatrix} \underrightarrow{1}_1 \\ \underrightarrow{1}_2 \\ \underrightarrow{1}_3 \end{bmatrix}, \tag{2.3}$$

where $\underrightarrow{1}_1 \cdot \underrightarrow{1}_2 = \underrightarrow{1}_1 \cdot \underrightarrow{1}_3 = \underrightarrow{1}_2 \cdot \underrightarrow{1}_3 = 0$. A vector, $\underrightarrow{r}$, which has a magnitude and direction independent of any reference frame, can be expressed as a set of coordinates in a particular reference frame through

$$\underrightarrow{r} = \underrightarrow{\mathcal{F}}_i^T \mathbf{r}_i = \begin{bmatrix} \underrightarrow{1}_1 & \underrightarrow{1}_2 & \underrightarrow{1}_3 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}. \tag{2.4}$$

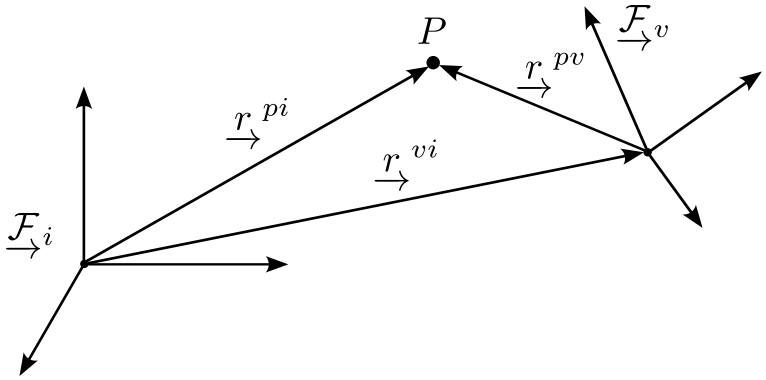


Figure 2.2: Visualization of the relationship between two reference frames and a point P. The point is shown relative to $\underrightarrow{\mathcal{F}}_i$ and $\underrightarrow{\mathcal{F}}_v$. Vectors $\underrightarrow{r}^{pi}$ and $\underrightarrow{r}^{pv}$ identify the point, while the vector $\underrightarrow{r}^{iv}$ connects the origin of $\underrightarrow{\mathcal{F}}_i$ to the origin of $\underrightarrow{\mathcal{F}}_v$.

Following the visualization in Figure 2.2, we can represent the location of a point P by

$$\underrightarrow{r}^{pi} = \underrightarrow{r}^{vi} + \underrightarrow{r}^{pv}, \tag{2.5}$$

where the superscripts on each vector are read from right (tail) to left (tip).

Note that the basis vectors of one frame (e.g., $\underrightarrow{\mathcal{F}}_i$) may be *rotated* with respect to the basis vectors of a second frame (e.g, $\underrightarrow{\mathcal{F}}_v$). Utilizing the vectrix representation again and taking the outer product, we obtain

$$\mathbf{C}_{iv} = \underrightarrow{\mathcal{F}}_i \cdot \underrightarrow{\mathcal{F}}_v^T. \tag{2.6}$$

The resulting 3×3 *rotation matrix*, $\mathbf{C}_{iv}$, is a member of the special orthogonal group (in three dimensions), which is the set defined as

$$SO(3) = \{\mathbf{C} \in \mathbb{R}^{3\times 3} \mid \mathbf{C}\mathbf{C}^T = 1,\ \det(\mathbf{C}) = 1\}. \tag{2.7}$$

The subscripts are read from right (initial frame) to left (final frame). Note that rotation matrices are orthonormal (i.e., $\mathbf{C}\mathbf{C}^T = 1$) and preserve handedness (i.e., $\det(\mathbf{C}) = 1$). It follows directly from orthogonality that the inverse of the rotation matrix is

$$\mathbf{C}_{vi} = \mathbf{C}_{iv}^{-1} = \mathbf{C}_{iv}^T. \tag{2.8}$$

Consider expressing the vector $\underrightarrow{r}^{pi}$ in terms of coordinates in reference frame $\mathcal{F}_i$.

Making use of Equations (2.5) and (2.6), we have

$$\mathbf{r}_i^{pv} = \mathbf{C}_{iv}\mathbf{r}_v^{pv}, \tag{2.9}$$

$$\mathbf{r}_i^{pi} = \mathbf{r}_i^{vi} + \mathbf{r}_i^{pv}, \tag{2.10}$$

$$\mathbf{r}_i^{pi} = \mathbf{r}_i^{vi} + \mathbf{C}_{iv}\mathbf{r}_v^{pv}, \tag{2.11}$$

where the subscripts indicate the frame in which the coordinates are expressed.

When two reference frames are rotated relative to each other and their origins are translated, the transformation between the frames can be represented by a *homogeneous transformation matrix*, $\mathbf{T} \in \mathbb{R}^{4\times4}$, that is a member of the special Euclidean group (in three dimensions),

$$SE(3) = \left\{ \mathbf{T} \in \begin{bmatrix} \mathbf{C} & \mathbf{r} \\ \mathbf{0}^T & 1 \end{bmatrix} \;\middle|\; \mathbf{C} \in SO(3), \mathbf{r} \in \mathbb{R}^3 \right\}. \tag{2.12}$$

The coordinates of a point expressed in two different reference frames are thus related by

$$\begin{bmatrix} \mathbf{r}_i^{pi} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{C}_{iv} & \mathbf{r}_i^{vi} \\ \mathbf{0}^T & 1 \end{bmatrix}}_{\mathbf{T}_{iv}} \begin{bmatrix} \mathbf{r}_v^{pv} \\ 1 \end{bmatrix}. \tag{2.13}$$

The inverse transformation can be computed as

$$\mathbf{T}_{iv}^{-1} = \mathbf{T}_{vi} = \begin{bmatrix} \mathbf{C}_{iv}^T & -\mathbf{C}_{iv}^T\mathbf{r}_i^{vi} \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{2.14}$$

With knowledge of the appropriate transform matrices, the points captured in individual point cloud scans can be merged into a common, world reference frame, $\mathcal{F}_i$, and concatenated together. The world reference frame is usually assumed to be defined by the first scan in a sequence. Given a set of $V \geq 1$ scans, the full set of points (i.e., the *point cloud* or $\mathcal{P}$) can be expressed in the world frame as

$$\mathcal{P} = \left\{ \mathbf{p}_i^{ji} \in \mathbb{R}^3 \;\middle|\; \begin{bmatrix} \mathbf{p}_i^{ji} \\ 1 \end{bmatrix} = \mathbf{T}_{iv} \begin{bmatrix} \mathbf{p}_v^{jv} \\ 1 \end{bmatrix}, \; \mathbf{p}_v^{jv} \in \mathcal{P}_v \right\}, \tag{2.15}$$

where each individual scan $v \in (1, \ldots, V)$, contains a set of points $\mathcal{P}_v = \{\mathbf{p}_v^{jv}\}_{j=1}^{M_v}$. Here, $\mathbf{p}_v^{jv} \in \mathbb{R}^3$ and M_v is the total number of points in scan (or cloud) $\mathcal{P}_v$. We note a slight abuse of notation above, where we mix alphabetic (i.e., i) and numeric (e.g., 1, 2, ...)

frame identifiers for notational simplicity. The homogeneous transform matrix from each scan frame to the world frame is denoted by $\mathbf{T}_{iv}$. Multiple point cloud scans are said to be *registered* if the scans are all expressed in a common frame.

2.3 Projective Geometry

Images and point clouds can be used together in a flexible way for many perception tasks because 3D points can be mapped directly to image pixels. The direct mapping from points to pixels is possible because an image is a *projection* of 3D space onto a 2D image plane. There are various types of projection operations or models, and we refer to the formulations discussed by Szeliski [51] for more details. The simplest projection model is an *orthographic projection*, which defines the 3D to 2D mapping by discarding the z-coordinate of each 3D point. However, this model does not accurately represent the way light is focused by a camera lens system. Instead, in the *ideal perspective projection* model, a light ray emitted from a 3D point passes through an infinitely small aperture (i.e., a *pinhole*) and intersects the image plane at a 2D point. This model is also commonly referred to as the pinhole camera model. Since the pinhole model is based on similar triangles, there is no interdependence between projection in the x and y directions, and hence the two axes can be treated independently. The diagram in Figure 2.3 visualizes perspective projection for a 1D camera (in the x direction). Using similar triangles, it can be shown that

$$x_c = f_x \frac{x_p}{z_p}, \tag{2.16}$$

where x_c represents the projection of the point (x_p, z_p) onto the image plane along the x-axis. The extension to the y-axis follows the same logic by substituting the point coordinate y_p.

To obtain distances in terms of pixel values, the focal length, f can be converted to pixels by dividing by the (metric) pixel size, s, which yields

$$f_u = \frac{f_x}{s_x}. \tag{2.17}$$

We distinguish values represented in pixel coordinate units with the subscript u for pixel values along the image plane x-axis and v for pixel values along the y-axis. Image coordinates are specified using ordered pairs of coordinates, denoted by (u, v). Note also that pixels may not be square, and hence the focal lengths can differ between axes. The distances to the *principal point*, where the optical axis pierces the image place, also vary

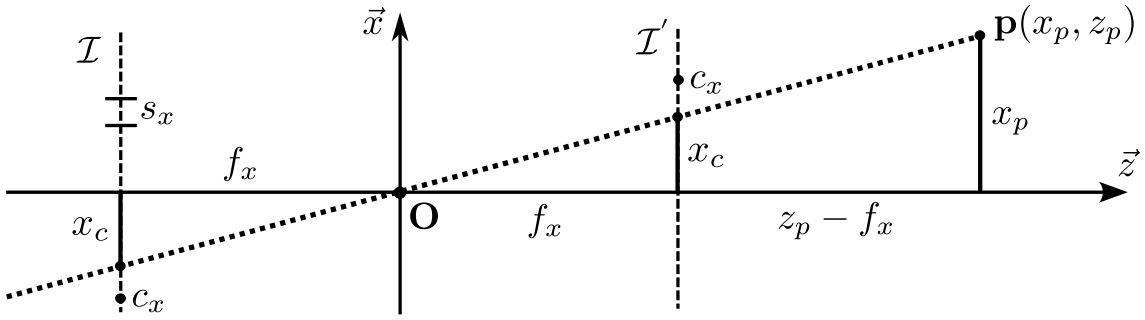


Figure 2.3: Visualization of a 1D perspective projection. The point $\mathbf{p}$ is projected onto the image plane $\mathcal{I}$. The light ray passes through the optical centre (aperture) $\mathbf{O}$; the distance between the image plane and origin (optical centre) is f_x. Each pixel on the image sensor has size s_x and the distance from the top of the image place to the principal point (x-axis) is c_x pixels. An equivalent formulation places a 'virtual' image plane in front of the optical centre (denoted as $\mathcal{I}'$) to avoid image inversion.

because the image sensor is usually rectangular. By applying the offset of the principal point and re-arranging, the pixel coordinates of a projected 3D point become

$$u = \frac{1}{z_p}(f_u x_p + c_u z_p), \quad v = \frac{1}{z_p}(f_v y_p + c_v z_p). \tag{2.18}$$

Equation (2.18) can be further re-arranged to define the projection operation in matrix form,

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z_p} \underbrace{\begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}}_{\mathbf{p}}, \tag{2.19}$$

where $\mathbf{K}$ is a 3×3 *intrinsic calibration* matrix. Re-arranging Equation (2.19) yields

$$\mathbf{p} = z_p \mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \tag{2.20}$$

which is the inverse projection that maps pixel coordinates back to 3D point coordinates To solve Equation (2.20), we require knowledge of z_p. Since images lack this information, a unique solution for the inverse projection cannot usually be computed. However, for the special case of RGB-D images, z_p can be extracted from the depth channel.

The perspective projection model assumes that light rays travel in straight lines, through the aperture, and intersect the image plane. In reality, all lenses introduce some amount of distortion, or deviation of the path of each light ray from a straight line. If

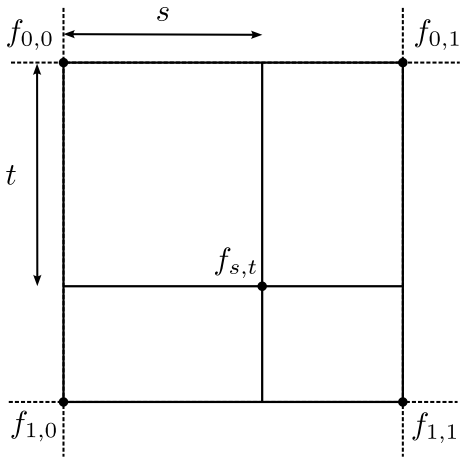


Figure 2.4: Visualization of bilinear interpolation at fractional pixel coordinates. There are four pixels, one at each corner of the square, with known intensity values f. The fractional pixel coordinates $f_{s,t}$ are $s \in [0,1]$ and $t \in [0,1]$ relative to $f_{0,0}$, in the horizontal and vertical directions, respectively.

this deviation is not accounted for, it will introduce an error when computing both the forward and backward projections. The most common type of lens distortion is radial distortion, which can be modelled as a transform of the image points inwards, towards the image centre (i.e., barrel distortion), or outward, away from the image centre (i.e., pincushion distortion). The effects of radial distortion can be approximated by

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4), \tag{2.21}$$

$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4), \tag{2.22}$$

where $r_c^2 = x_c^2 + y_c^2$ is the radial distance of the point (x_c, y_c) that is computed *prior* to applying the intrinsic transform matrix from the centre of projection on the ideal (distortion-free) image plane and κ_1 and κ_2 are radial calibration coefficients.

Image operations such as perspective projection or rescaling often result in fractional pixel values. Usually, this is not a problem, since fractional values can be rounded to the nearest whole number. However, this rounding operation is inaccurate by definition, and interpolation using the values of nearby pixels is sometimes necessary. The simplest interpolation method is to apply a first-order (linear) approximation along each axis, as part of *bilinear interpolation* (see Figure 2.4). Following the formulation in Szeliski [52], the value of the (fractional) pixel $f_{s,t}$ in the diagram can be computed as

$$f_{s,t} = (1-s)(1-t)f_{0,0} + s(1-t)f_{1,0} + (1-s)tf_{0,1} + stf_{1,1}, \tag{2.23}$$

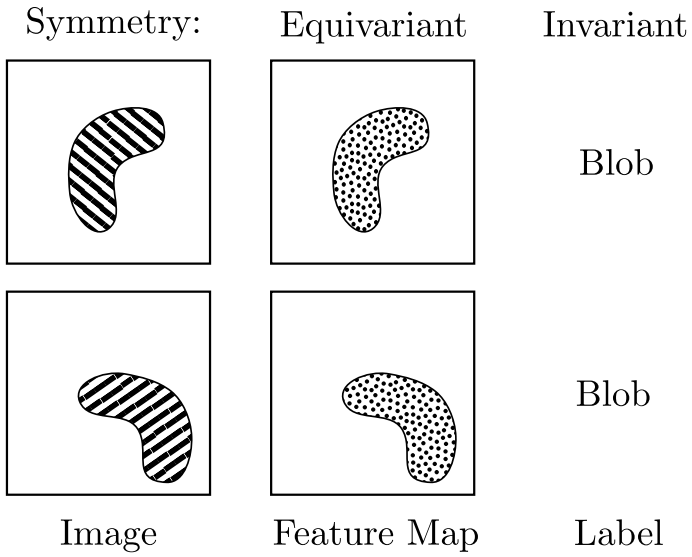


Figure 2.5: Visualization of the concepts of equivariance and invariance.

where s and t represent the horizontal and vertical distances from the top left pixel, respectively, and $f_{i,j}$ defines the intensity (or colour) value of a pixel in one of the four corners. The contribution of the pixel value at each corner is scaled according to the area of the rectangle directly opposite.

2.4 Features

Extracting essential information from data, in the form of *features*, plays a key role in segmentation and object detection tasks. Here, a feature is a summary representation computed from all or a portion of the data. Feature extraction is a common first step in scene understanding algorithms that process features for downstream decision-making. Designing features that make decision-making accurate is, therefore, an important aspect of algorithm development and one of the core objectives of this thesis.

Depending on the task, features can summarize either the *global* information about a single data point, or the *local* surrounding region. Local features are useful in scene segmentation since objects in a scene typically 'cover' more than a single pixel (or 3D point in the case of a point cloud), and so valuable contextual information can be acquired from the surroundings of a single data point. Meanwhile, global features are useful in high-level scene classification since they summarize information about the entire scene.

We seek certain desirable properties in both feature types. The first property is that a feature should form a *compact* and *succinct* representation of the underlying data, which is important for the computational efficiency of downstream feature processing. Second,

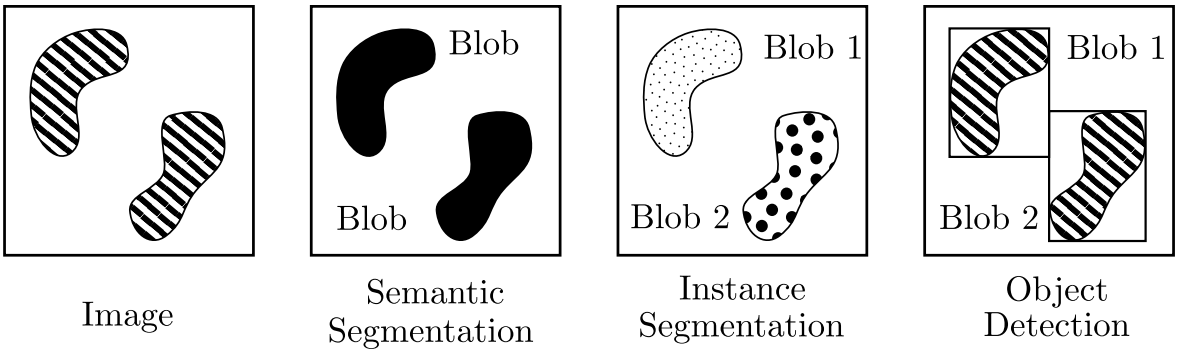


Figure 2.6: Visualization of scene understanding tasks.

a feature should usually exhibit *symmetry*, either in the form of *invariance* or *equivariance*, to transformations of the input data. Symmetry allows for algorithms to generalize to larger input spaces. Examples of possible input transformations include translations, rotations, and scaling. An invariant feature remains unchanged by a given transformation. Invariance is desirable for tasks such as image classification, where transformations, such as rotations, should not influence the final image class. An equivariant feature, on the other hand, undergoes a transformation in the output space that is equivalent to the transformation of the input space. A convolution (see Section 2.6.2) is an example of an operation that is equivariant to input translations. Equivariance is desirable for tasks such as image segmentation, where the segmented pixels corresponding to a particular object should be 'grouped' along with the object.

There exist many possible feature representations. In certain cases, a *metric* may also be defined on the feature space, allowing one to measure distances between features. By defining a feature space and an accompanying metric on that space, features that are similar according to the metric can be grouped together or compared. In this thesis, a feature (image or point cloud) is represented by a vector $\mathbf{f} \in \mathbb{R}^n$ with the standard Euclidean or ℓ^2 norm applied as the distance metric,

$$\ell^2(\mathbf{f}_1, \mathbf{f}_2) = ||\mathbf{f}_1 - \mathbf{f}_2||^2, \ \forall \mathbf{f}_1, \mathbf{f}_2 \in \mathbb{R}^n. \tag{2.24}$$

2.5 Tasks and Metrics for Scene Understanding

Although features can be informative, they are not particularly useful on their own. Instead, features are utilized for tasks such as object detection, semantic segmentation, and instance segmentation. Therefore, we gauge the quality of our pre-trained features by evaluating how they impact the performance of these downstream tasks. This sec-

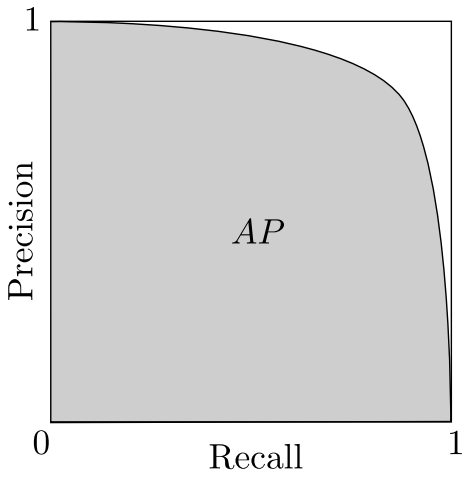


Figure 2.7: Visualization of a precision-recall curve. The shaded area under the curve is the average precision (AP).

tion gives a short description of each task along with relevant evaluation metrics. A visualization of the different scene understanding tasks is shown pictorially in Figure 2.6.

First, the goal of object detection is to find instances of specific objects and tightly fit a bounding box around each object. A detection is considered correct if it meets a defined threshold of the intersection over union (IOU) metric. Intersection over union is computed as a ratio between the area or volume of the intersection between the predicted and ground truth bounding boxes and the area or volume of their union. A detection is commonly considered to be 'correct' when the IOU exceeds 50% and is used to compute a mean average precision (mAP) metric. To compute the mAP we first need to compute the precision and recall scores, defined as

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}, \tag{2.25}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}, \tag{2.26}$$

across all possible confidence thresholds, to build a precision-recall curve. The area under this curve is the average precision, and when further averaged over all possible classes it becomes the mean average precision. A visualization of a precision-recall curve along with the area that represents the average precision is shown in Figure 2.7. An extension of object detection is the task of semantic segmentation, where the goal is to label each point or pixel in a scene as belonging to a predefined class (including structures such as walls), instead of drawing bounding boxes around objects. The evaluation metric used is the mean intersection over union (mIOU), which is the intersection of the predicted

and ground truth labels over their union, averaged across all available classes. Lastly, the task of instance segmentation aims to identify specific instances of objects. Like semantic segmentation, instance segmentation labels the raw data directly as belonging to a specific instance of an object class. Instance segmentation uses the mAP metric, with a correct classification thresholded by the IOU metric.

2.6 Learning-Based Methods

The complexity of scene understanding makes the problem of 'hand-modelling' infeasible. *Learning-based* approaches have proven to be a successful and practical alternative. Learning-based methods work by optimizing models to mimic the input-output relationship defined by sample data points (i.e., training data). This section provides an overview of the learning-based methods that are used in this thesis.

2.6.1 Deep Neural Networks

Deep neural networks (DNNs) are able to model complex nonlinear problems in an end-to-end fashion (meaning from model input to output) and have been highly successful for segmentation and object detection tasks. This section of the thesis provides an overview of DNNs, following the outline from Szeliski [53].

DNNs are built from consecutive layers of 'neurons,' where each neuron computes a weighted sum of its inputs that is then shifted by a scalar bias value. The result is then passed through a non-linear activation function. The output of an individual layer i, denoted as $f_{\boldsymbol{\theta}_i}$, is

$$f_{\boldsymbol{\theta}_i} = h(\mathbf{W}_i^T \mathbf{x}_i + \mathbf{b}_i), \tag{2.27}$$

where h represents a non-linear activation function applied element-wise and $\boldsymbol{\theta}_i$ represents the model parameters, which are the weight matrix $\mathbf{W}_i \in \mathbb{R}^{n \times m}$ and bias vector $\mathbf{b}_i \in \mathbb{R}^n$. Individual layers are stacked together such that the output of one layer becomes the input to the next (see Figure 2.8). A helpful way to express the output of multiple consecutive layers is via the composition operator,

$$(f_{\boldsymbol{\theta}})(\mathbf{x}) = (f_{\boldsymbol{\theta}_M} \circ \cdots \circ f_{\boldsymbol{\theta}_1})(\mathbf{x}). \tag{2.28}$$

The parameters of each layer are summarized into a single variable $\boldsymbol{\theta} = \{\boldsymbol{\theta}_i\}_{i=1}^M$ where M is the number of layers. For regression problems, the output of the network can be used directly without further modification. For classification problems, the continuous model

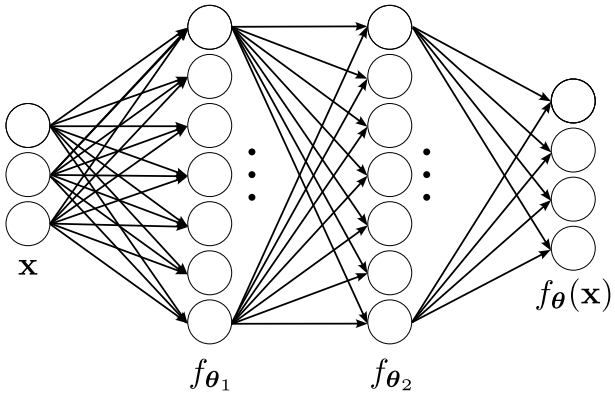


Figure 2.8: Visualization of a deep neural network.

output has to be converted to a discrete classification. This conversion is commonly done by numbering each class and representing the class index using a *one-hot encoding*. A one-hot encoding vector has a length equal to the total number of classes with a one at the position corresponding to the identified class index and zeros everywhere else. The model output is then made to match the size of the one-hot encoding vector, K, and converted into a probability distribution across all classes using the *softmax* function

$$y_i = \frac{e^{x_i}}{\sum_j^K e^{x_j}}, \tag{2.29}$$

where the predicted class is that with the highest probability (i.e., closest to one).

The weights and biases of the network are the learnable parameters that are adjusted according to the task. The easiest and most common method for tuning a model is through *supervised* learning, whereby a set of input-output pairs are given. The model is optimized such that the model outputs match the desired outputs (targets) as closely as possible. Optimization is based on a selected scalar loss function $\mathcal{L} : \mathbb{R}^K \to \mathbb{R}$, where K is the model output size. The total loss is then computed as the average of the loss from each input-output pair,

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i, \tag{2.30}$$

where we use $\mathcal{L}_i$ to denote the loss from one of the N samples.

For classification problems, a common practice is to use a cross-entropy loss

$$\mathcal{L}_{CE} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i \log(f_{\boldsymbol{\theta}}(\mathbf{x}_i)), \tag{2.31}$$

where $\mathbf{y}_i \in \mathbb{R}^K$ represents the target output given input $\mathbf{x}_i$. For regression problems, an

ℓ^2 loss is used

$$\mathcal{L}_2 = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - f_{\boldsymbol{\theta}}(\mathbf{x}_i)\|^2. \tag{2.32}$$

Training the network can now be formulated as an optimization problem,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}). \tag{2.33}$$

This optimization problem is almost always non-convex and riddled with local minima. The most common optimization method, known as *gradient descent*, iteratively takes steps in the direction opposite to the gradient of the loss function with respect to the network parameters, where the gradient is computed at the current operating (parameter) point. This method is also known as *steepest descent*. However, like all methods based on linearization, moving too far in the steepest descent direction may not yield the lowest value of the loss function. Selecting a step size ϵ for each gradient-based update that is neither too big nor too small is crucial. Updating the parameters is done simply using a step opposite to the gradient direction

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \epsilon \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i}. \tag{2.34}$$

Since the network is a composition of the individual activation functions across each layer, the complete gradient is simply a product of gradients from the previous layers, defined using the chain rule,

$$\frac{\delta \mathcal{L}}{\delta \boldsymbol{\theta}_i} = \frac{\delta \mathcal{L}}{\delta f_{\boldsymbol{\theta}_M}} \cdot \frac{\delta f_{\boldsymbol{\theta}_M}}{\delta f_{\boldsymbol{\theta}_{M-1}}} \cdots \frac{\delta f_{\boldsymbol{\theta}_{i+1}}}{\delta \boldsymbol{\theta}_i},\ i \in \{1, \ldots, M\}. \tag{2.35}$$

The parameters are updated until either a predefined maximum time or iteration count is reached or the loss stops decreasing for an extended period. Many different algorithms for network optimization exist, but the most common is stochastic gradient descent (SGD). SGD works by randomly grouping data into smaller sets called *mini-batches* and computing the gradient across this mini-batch only. Optimization requires setting various hyper-parameters such as the step size and a momentum term.

The activation function must be nonlinear to ensure that the network can model nonlinear input-output relationships. The most-used activation function is the rectified linear unit (ReLU) which is defined as

$$h(y) = max(0, y). \tag{2.36}$$

However, the ReLU activation suffers from a 'zero gradient' problem when the input is negative, preventing any further parameter updates for that neuron and for any connected neurons in other layers. To avoid this zero gradient issue, the leaky ReLU function uses a scaling value, α, to ensure the slope is nonzero,

$$h(y) = \begin{cases} y & y > 0 \\ \alpha y & y < 0, \end{cases} \tag{2.37}$$

while the exponential linear unit applies an exponential function as part of the activation,

$$h(y) = \begin{cases} y & y > 0 \\ \alpha(\exp(y) - 1) & y < 0, \end{cases} \tag{2.38}$$

for negative inputs.

Applying the additional step of normalizing the inputs before applying the activation function is also common. Normalization prevents an imbalance in the magnitude of the parameter weights between different layers, which can make training extremely sensitive to even small update steps. The most common type of normalization is *batch normalization*, which rescales all of the outputs of a layer in a mini-batch such that the set has zero mean and unit variance.

2.6.2 Convolutional Neural Networks in 2D

Since the impressive breakthrough in performance shown by AlexNet [28], which in 2012 beat human performance on the ImageNet benchmark [12], most image-based computer vision algorithms have adopted the convolutional neural network (CNN) as part of their core architecture. Unlike fully-connected networks that have individual weights for each element of the input, convolutions have the advantage of being *equivariant* to translations of pixels in the input image. This is important because the same clusters of pixel values can appear together anywhere within an image (e.g., consider an edge at a specific orientation), and the model needs to behave in the same way regardless of where in the image the exact pattern appears. Convolutional neural networks apply small filters, known as *kernels*, to the inputs by 'sliding' them across the image, similar to the convolutional operation used in signal processing. The output of each filter is computed using the set of all channels of the input, denoted by $\mathcal{C}$. Stacking multiple kernels together forms a *convolutional layer*, where each stacked kernel output represents a feature vector. Convolutional neural networks are composed of multiple sequential convolutional layers. Following the

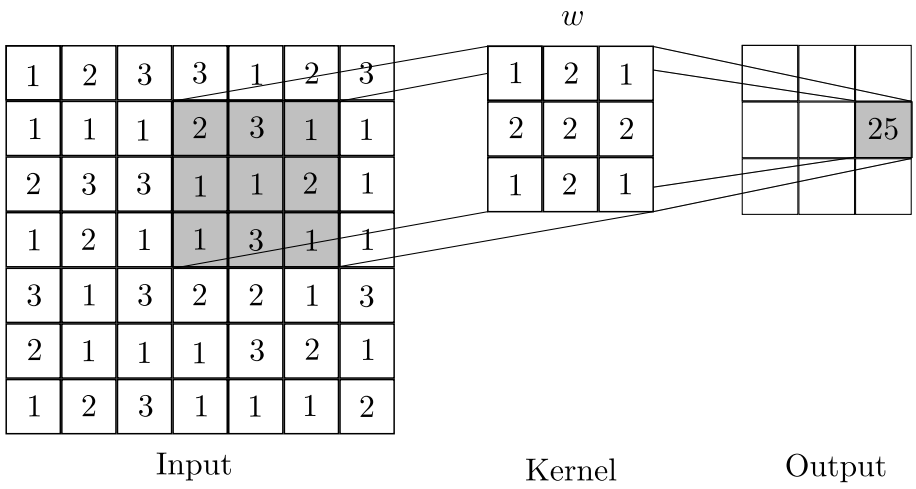


Figure 2.9: Visualization of a 2D convolution. The input is size 7×7 and the kernel size is 3×3 with a stride of 2 and padding of 0. The resulting output size is 3×3.

formulation given in Szeliski [53], the output of a single convolutional kernel at input coordinates i and j can be expressed as

$$y(i,j) = \sum_{c \in \mathcal{C}} \sum_{(k,l) \in \mathcal{N}} \mathbf{w}(k,l,c)\, \mathbf{x}(i+k, j+l, c) + b, \tag{2.39}$$

where $\mathbf{w}$ and b represent the kernel weights and bias respectively and $\mathcal{N}$ represents the set of 2D kernel offsets. The output y can subsequently be passed through an activation function as described in Section 2.6.1. A visualization of an example of 2D convolution is shown in Figure 2.9. Each filter can move in positive increments of whole pixel coordinates, referred to as the *stride* and denoted as s, to reduce the spatial dimension of the output. When applying a convolutional filter of size greater than one, it will extend beyond the boundary of the image. To deal with this situation, the image is often padded on all sides with zeros or with the value of the nearest-neighbour pixel at the boundary. The output size for an individual dimension, d_{out}, can be computed using

$$d_{out} = \left\lfloor \frac{d_{in} - k + 2p}{s} \right\rfloor + 1, \tag{2.40}$$

where s represents the stride, p represents the number of padding pixels, k represents the kernel size, and d_{in} represents the input dimension size. Note that all variables are measured along a single axis. Positional invariance of features in a small local window is achieved through the *pooling* of outputs in the window. Note that the translational invariance of local pooling means that the output is no longer equivariant within the window. Local pooling is often performed by simply taking the maximum value of the

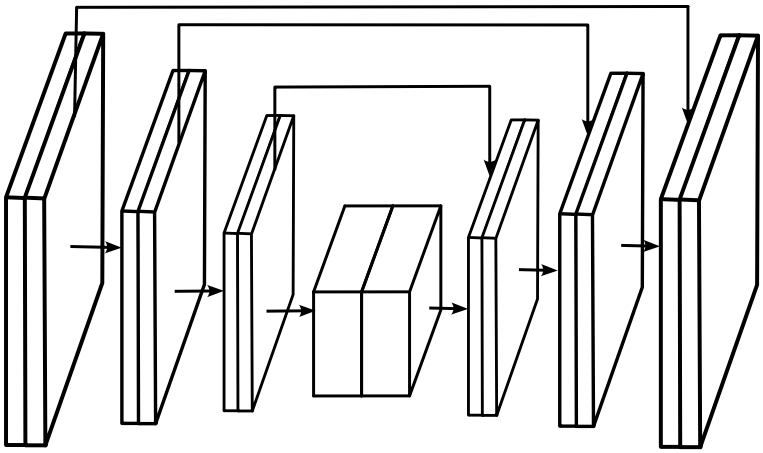
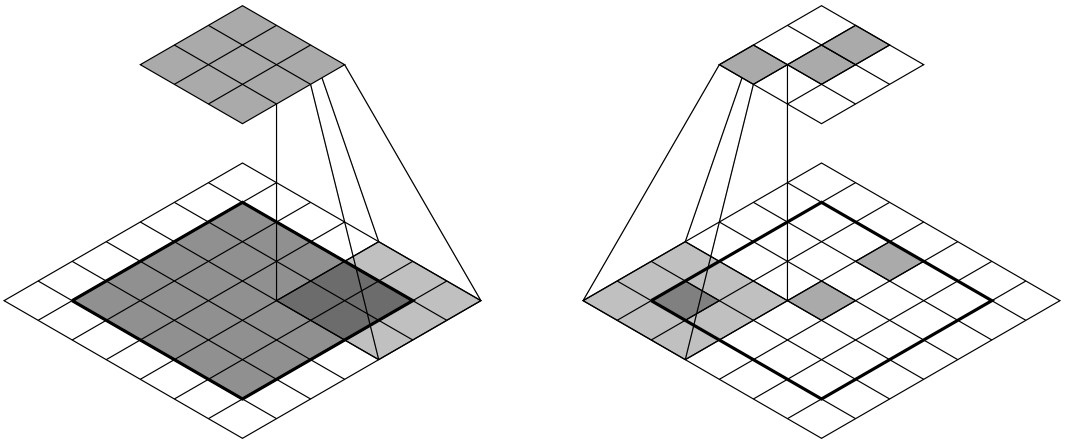


Figure 2.10: Visualization of the generic UNet architecture.

outputs within the window, otherwise known as *max pooling*.

Given the definition of the convolution operation above, the output size can only stay the same or decrease. In many cases, it is advantageous for the output to be larger than the input, in which case a *transposed convolution* can be applied. A transposed convolution inserts $s-1$ rows and columns between each pixel of the input layer, where s represents the upsampling stride. Then, a regular convolution operation with a stride of 1 is performed over the new input, resulting in an increase in the output dimension compared to the input. Similarly, a discrete upsampling operation can achieve a similar size increase by interpolating over the output using the input values. In this thesis, we use simple nearest-neighbour interpolation, which maps each pixel in the upsampled image back to the original image and selects the pixel in the original with the nearest pixel coordinates.

The deeper a convolutional network is, the more features it can extract. However, deep networks suffer from the *vanishing gradient problem*, where the gradient magnitude typically reduces as the network size increases. ResNet [20] mitigates the impact of vanishing gradients by learning residuals that are added to the input, as opposed to learning the output directly. This allows a signal to pass through the layers relatively unaltered so that its gradient does not diminish significantly. Groups of layers where the input is bypassed are referred to as *blocks*. There exist many configurations of the ResNet architecture that vary in terms of the number of blocks that are employed. Importantly, larger ResNets (ResNet50/101/152) build a *bottleneck* layer into their blocks, which reduces the number of parameters per block. It is common to leverage different versions of the ResNet backbone, trained on the large ImageNet dataset, as the early-stage feature extractor for state-of-the-art image-based semantic segmentation networks.



Dense Convolution Sparse Convolution

Figure 2.11: Visualization of dense and sparse 2D convolution operations.

An outstanding issue with ResNet is that it suffers from being sequential, meaning that downstream layers have no direct access to the raw outputs from upstream layers and cannot take advantage of features at multiple scales. Feature pyramid network (FPN) [32] architectures address this drawback by implementing feature pyramids, which are a popular method for handling objects at multiple scales in classical compute vision algorithms. FPNs make use of a typical CNN with decreasing filter dimensions and then add an adjacent top-down pathway that sequentially up-samples the outputs of the smallest filters. FPNs use lateral *skip connections* to directly connect the output of equally-sized layers from the down-sampling pyramid to the up-sampling pyramid. This structure resembles two pyramids stacked back-to-back (or top-to-top). Importantly, predictions are made at all levels from the smallest layer to the largest layer on the up-scaling side. The UNet [46] architecture modifies the feature pyramid network to use the output of the last layer of the up-scaling pyramid only. This approach has proven successful for many kinds of image and point cloud segmentation tasks. A visualization of a generic UNet is shown in Figure 2.10.

2.6.3 Convolutional Neural Networks in 3D

The advantages of using convolutions for image data, such as parameter efficiency and equivariance, are equally desirable when processing point clouds. However, using convolutions on point clouds brings a few challenges. The first issue is that the convolutions, when applied to images, assume discrete input coordinates (i.e., pixels), while points in point clouds do not generally have integer coordinates. To handle this discrepancy,

point clouds are frequently converted to voxel grids (see Section 2.1). The second issue is that, with each new dimension, the space and time complexity of forward passes through the network grow exponentially. Processing larger scenes at high spatial resolutions, therefore, becomes computationally infeasible. A key observation for improving computational efficiency is that, although there may be many voxels, most of them are empty. Any operation on empty voxels has no effect on the output and so empty voxels can be discarded immediately. Performing a convolution solely on occupied elements is referred to as a *sparse convolution*. A visual comparison between a normal dense convolution and a sparse convolution in 2D is shown in Figure 2.11.

2.6.4 Self-Supervised Contrastive Learning

Although supervised learning has proven immensely successful, collecting labels is often tedious, expensive, and time-consuming. Labelling can also severely limit the scalability of a dataset, which further restricts the number of examples that can be used to train a model. For this reason, self-supervised learning techniques seek to utilize unlabelled data to learn representations that can be used either directly or on other downstream, fully-supervised tasks.

In this thesis, we utilize a self-supervised technique referred to as *contrastive learning*. To describe contrastive learning, we follow the formulation from [29]. Contrastive learning aims to produce features that are distinguishable between unique inputs. The objective function encourages the outputs of a model, given similar input data points (i.e., positive samples), to be close together. Additionally, the objective function encourages the outputs produced from dissimilar data points (i.e., negative samples) to be far apart. The 'closeness' of model outputs is measured according to the metric defined by the feature space (see Section 2.4). Defining which data points are similar and which are dissimilar depends on the objective task that is being trained. This objective is otherwise known as the *pretext task*. For example, image colourization [62] seeks to find the original colours of an image that has been converted to greyscale (e.g., similar to colourizing old movies). The same pixel coordinate between the colour and greyscale images forms a positive pair while corresponding pixel coordinates from different images form negative pairs. The most common pretext task, and the one used in this thesis, is instance discrimination [57], whereby each input individually, regardless of the input type (e.g., image, point cloud) is considered a separate class. Instance discrimination generates positive pairs by applying augmentations to a query point and generates negative pairs by sampling from all other available data points.

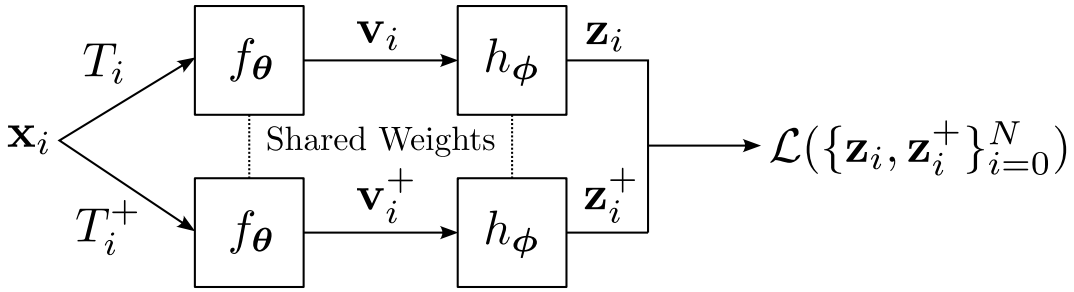


Figure 2.12: Visualization of a typical contrastive learning framework.

The points are subsequently fed through a model that learns to discriminate using a contrastive loss. More formally, a query point $\mathbf{x}_i \in \mathbb{R}^N$ is first sampled from the dataset. The query point is then augmented by sequentially applying one or more individual transformations $t : \mathbb{R}^N \to \mathbb{R}^N$, forming a composite function,

$$T(\mathbf{x}_i) = (t_1 \circ \cdots \circ t_S)(\mathbf{x}_i), \tag{2.41}$$

where all transformation hyperparameters are sampled randomly. Whether the input is a point in a point cloud or a pixel in an image, the same transformation is applied to all data points in the scene. Applying two separate transformations to the query point results in the positive pair $(T(\mathbf{x}_i), T^+(\mathbf{x}_i))$. Each member of the pair then follows a separate *pathway* through the network or model, as shown in Figure 2.12. Conversely, negative points are sampled by taking any other point not derived from $\mathbf{x}_i$, regardless of its augmentation.

All points are subsequently fed through an encoder $f_{\boldsymbol{\theta}}$ to obtain a feature $\mathbf{v} = f_{\boldsymbol{\theta}}(\mathbf{x})$, where $\mathbf{v} \in \mathbb{R}^D$. This encoder forms the backbone of the model that we are trying to initialize. The feature is then passed through a decoder $\mathbf{z} = h_{\boldsymbol{\phi}}(\mathbf{v})$, where $\mathbf{z} \in \mathbb{R}^M$ and $M \leq D$. However, if the features are desired directly then the decoder can simply apply the identity transform. The decoder outputs are normalized such that $\|\mathbf{z}\|^2 = 1$ to improve the stability of the gradient update during training. Once pre-trained, only the encoder parameters $\boldsymbol{\theta}$ are retained as part of the initialized backbone; the decoder parameters $\boldsymbol{\phi}$ are discarded.

The most common and successful contrastive objective function is the InfoNCE (Info Noise Contrastive Estimation) loss function [39]. The loss is defined over the set of query points as

$$\mathcal{L} = -\sum_{i=1}^{N} \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_i^+ / \tau)}{\exp(\mathbf{z}_i \cdot \mathbf{z}_i^+ / \tau) + \sum_j^K \exp(\mathbf{z}_i \cdot \mathbf{z}_j^- / \tau)}, \tag{2.42}$$

where $\tau \in (0, 1]$ represents the temperature parameter that controls the smoothness of the latent (encoded) representations, and K is a hyperparameter that determines the number of negative features to sample. For simplicity, it is common to set $K = N$ and take all negative samples as query points. The similarity between features is computed as the dot product, although other suitable distance functions exist.

Chapter 3

Related Work

This thesis builds upon work on self-supervised learning from image and point cloud data. The present chapter provides a review of existing self-supervised contrastive learning methods used to initialize models for downstream scene understanding tasks. We first cover methods that operate on either images or point clouds separately. Then, we discuss methods that utilize both modalities together.

3.1 Contrastive Learning in 2D

This section covers the algorithms used for learning self-supervised features from raw 2D data, which comprises the first stage of our method. A 2D image feature, in this context, is the output of an encoder network that has been trained using a contrastive objective and that follows the formulation discussed in Section 2.4. Importantly, the methods described in this section are also used as building blocks for self-supervised learning on other data types including point clouds.

We first cover algorithms that learn a single feature vector for each image (i.e., image-level), since this is the format most often used to develop and evaluate new contrastive architectures. We then examine extensions of image-level contrastive algorithms to those that learn a feature vector for each pixel in an image (i.e., pixel-level).

Almost all of the contrastive algorithms covered in this section use the same method to generate positive-pair correspondences. If an algorithm deviates from this approach, we indicate this distinction clearly in the text. Positive pairs are generated by randomly augmenting two separate random crops of the same image. Augmentations can generally be split into two types: coordinate transforms (e.g., horizontal image flips) and colour transforms (e.g., from colour to greyscale). Each augmentation has a pre-defined probability of being applied to the data point during training. New positive pairs are

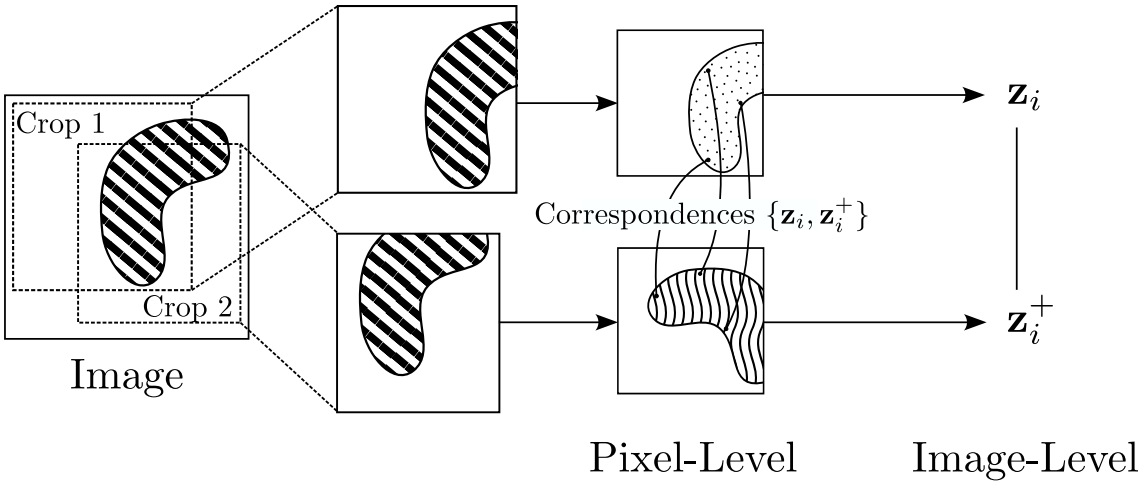


Figure 3.1: Generating 2D feature correspondences for contrastive learning. Pixel-level contrastive learning compares and contrasts features vectors defined for individual pixels within an image, while image-level contrastive learning compares and contrasts features defined over full images.

generated at each iteration, with all of the augmentation parameters sampled randomly. A visualization depicting how positive pairs are produced from a single image is shown in Figure 3.1. The positive samples are passed separately down one of two different data pathways, each of which applies an encoder to produce a feature vector, as shown in Figure 2.12. The exact form or architecture of each pathway depends on the algorithm, as shown in Figure 3.2. The encoder is implemented as a 2D CNN, denoted by $g_{\boldsymbol{\theta}}$ in Figure 3.2, while the resulting feature vector is denoted by $\mathbf{z}$. For pixel-level tasks, the encoder CNN computes one feature vector for every pixel; for image classification tasks, the CNN computes a single feature vector for the entire image. During image-level pre-training, the image-level feature vectors derived from the two augmented image crops form a positive pair used in the contrastive loss or objective function. During pixel-level pre-training, pixels in the two cropped images that originate from the same coordinates in the original (full) image are selected as positive pairs. The pretext task used in the algorithms discussed here is instance discrimination (see Section 2.6.4 for more details). A visual comparison of the most common 2D contrastive learning architectures is provided in Figure 3.2. The next section covers each of these architectures in detail.

3.1.1 Learning Image Features

Contrastive learning is a useful form of pre-training for image classification. Perhaps the best-known algorithm for self-supervised image feature learning is SimCLR [6]. SimCLR contrasts the similarity of a positive image pair with the dissimilarity of the query point

to other images in the training batch using an InfoNCE loss, defined by Equation (2.42). The SimCLR architecture follows the structure found in Figure 2.12. SimCLR achieves performance gains on downstream image-classification tasks that are comparable to supervised pre-training. The SimCLR architecture also forms the basic structure for all other contrastive learning algorithms, each of which modifies SimCLR to address various shortfalls.

One shortfall of SimCLR is the dependence on negative samples, or those samples whose features should be *dissimilar* to the query point (see Section 2.6.4). Contrastive architectures that apply the InfoNCE loss benefit greatly from large batches of negative samples at each iteration. This is because, as training progresses, the model outputs for negative samples become more dissimilar to the output for the query point, leading to gradients with very small magnitudes and stagnation. Using large negative batch sizes yields a higher chance of selecting a sufficient quantity of negative features that remain 'close' to the query point. However, due to resource limitations, generating large batches of feature vectors is not always feasible. One method to efficiently increase the negative sample batch size is to store previous feature encodings in a memory bank so that they do not have to be recomputed [57]. However, since the parameters of the encoder are updated at each iteration, the previous encodings become less representative of the current encoder state. Encodings are therefore only kept for a few iterations. An extension of using memory banks is to use a momentum encoder architecture (MoCo) [18]. The MoCo architecture is also based on a memory bank, but the features in the bank come from a separate encoder, $g_{\boldsymbol{\psi}}$. This separate encoder has the same structure as $g_{\boldsymbol{\theta}}$ but with parameters $\boldsymbol{\psi}$ that are updated at step $k+1$ according to a moving average of the current parameters $\boldsymbol{\theta}_k$ and $\boldsymbol{\psi}_k$,

$$\boldsymbol{\psi}_{k+1} \leftarrow m\,\boldsymbol{\theta}_k + (1-m)\,\boldsymbol{\psi}_k, \tag{3.1}$$

where $m \in (0,1)$ represents the blending coefficient between the current encoder and momentum encoder parameters. Backpropagation updates are stopped for this separate encoder $g_{\boldsymbol{\psi}}$ by applying a *stop-gradient* operation to prevent gradient information from propagating further through the network. Compared to storing feature vectors from the target model in a memory bank, storing features from the momentum encoder greatly improves downstream image classification performance. The reason for the performance improvement is not well understood, but one hypothesis is that the momentum encoder produces a more stable gradient than the model itself [41].

Without labels, it is possible that some negative samples may belong to the same

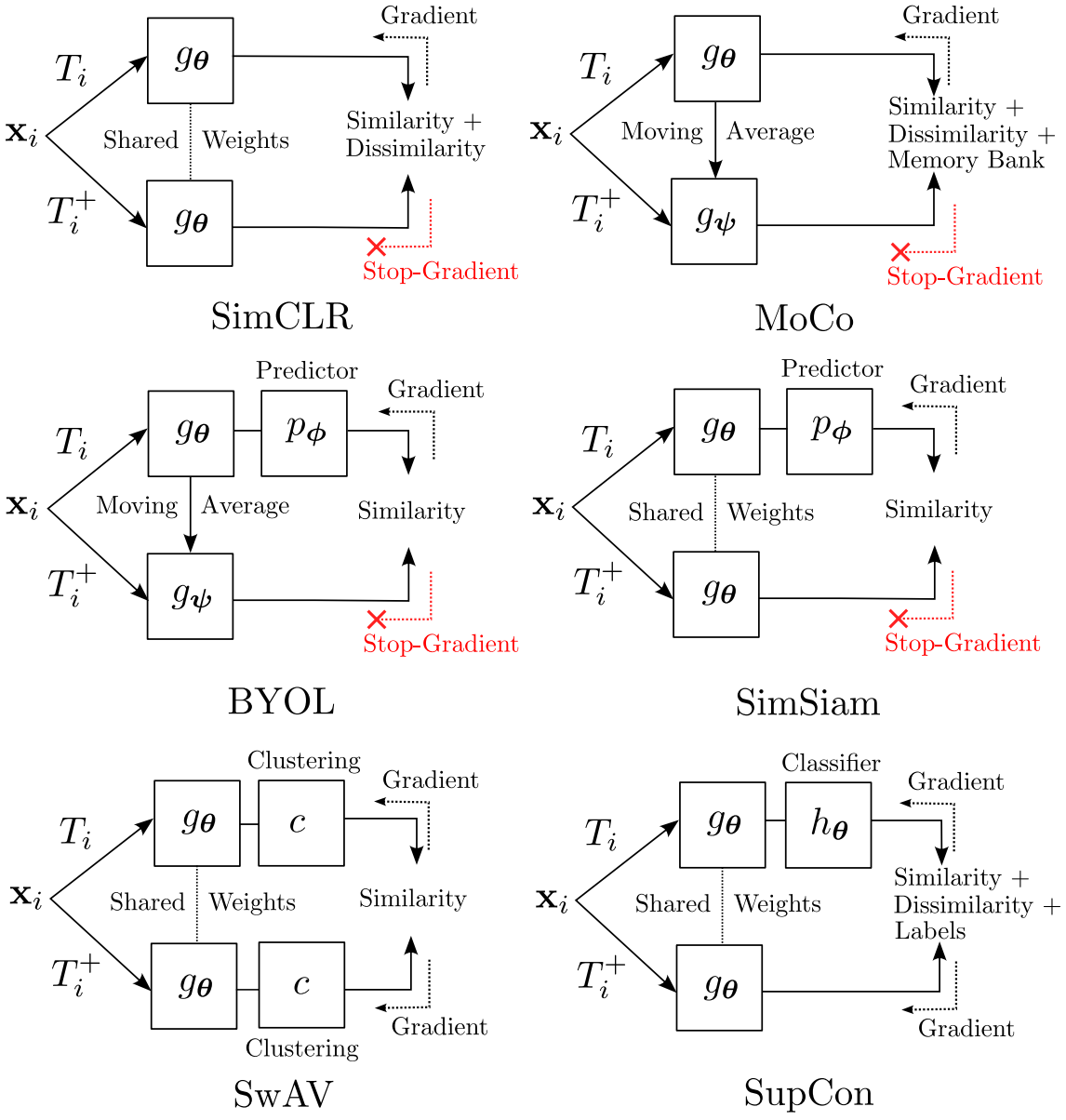


Figure 3.2: Comparison of contrastive learning architectures. Applying the *gradient* operation along a pathway modifies parameters upstream of the loss function through (stochastic) gradient descent. Applying the *stop-gradient* operator along a pathway prevents any upstream parameter updates.

class as the positive samples. Pushing these features apart is therefore counter-productive during training. One way to deal with negative samples is to avoid using them altogether and instead to focus on positive samples only. This is the approach taken by the Bootstrap Your Own Latent (BYOL) [17] algorithm. BYOL uses the same dual encoder architecture as MoCo, but does not include a memory bank. BYOL adds a predictor module, p_{ϕ}, that learns to predict the feature vector of one encoder from the output of the other. Also, BYOL only uses positive pairs in the objective function, and so is not restricted by negative sampling. SimSiam [7] modifies BYOL by sharing weights between the two encoders instead of using a moving average. The authors of [7] remark that this change does not affect downstream performance. Instead, they find that the key component of BYOL is the stop-gradient operation that prevents the second encoder from being updated via backpropagation. Another contrastive learning algorithm that forgoes using negative samples is Swapping Assignments between Views (SwAV) [5]. SwAV iteratively clusters feature vectors into groups, called prototypes, for each batch and compares these prototypes against a dictionary of existing (current) prototypes. SwAV learns which prototypes best represent the data and how to generate features close to these prototypes, regardless of the input transformation. Similarly, Prototypical Contrastive Learning (PCL) [30] applies a different clustering algorithm that avoids sampling positive-negative pairs from the same cluster.

An alternative technique to deal with negative samples is to design specific negative sampling strategies. Supervised Contrastive Learning (SupCon) [27] performs both supervised learning and self-supervised feature learning concurrently. The predicted class labels learned during training, also called pseudo-labels, are used to select negative samples that belong to a different class than the positive pair. SupCon also extends the InfoNCE loss to compare multiple positive points at the same time. Debiased sampling [10] accounts for the likelihood that a negative and positive sample belong to the same class by adjusting the loss function. Hard negative sampling [45] extends debiased sampling by picking negative samples that are closer to the positive sample in the selected metric space (e.g., the ℓ^2 norm). Lastly, hard negative mixing [26] selects negative samples that are close to the positive sample and then generates new samples using linear combinations of close negative samples.

3.1.2 Learning Pixel Features

Contrastive learning is also a valuable pre-training strategy for pixel-level scene understanding tasks. The success of pixel-level pre-training motivates and guides our own

work, which explores how this feature-based approach performs when applied to train models for 3D scene understanding tasks. One such algorithm is DenseCL [56], which applies the same framework as MoCo but, in addition to computing an InfoNCE loss for image-level features, adds a term to the loss function that determines an InfoNCE loss from features derived for individual pixels. The pixel features are computed using another CNN network built upon the image-level encoder. Unlike all other algorithms covered in this section, DenseCL actually learns which samples form positive pairs, instead of using known correspondences. Selecting positive pairs is done *after* passing both augmented images through the model. Pixel-level model outputs from different augmentations are compared using a Euclidean distance metric. Each pixel-level feature derived from the first augmented image is assigned the 'closest' pixel-level feature derived from the second augmented image. The authors of DenseCL demonstrate that the addition of individual pixel features to MoCo greatly improves downstream performance compared to using image-level features alone. PixContrast [60] follows a similar approach as DenseCL, except that the loss does not involve any image-level features; positive pairs are identified using the known correspondences between pixels in the augmented images. PixContrast also adds a 'pixel propagation module' that encourages smoothed feature representations by enforcing any similarity between features that are derived from pixels that occupy distant parts of the image. Lastly, ContrastiveSeg [55] modifies MoCo by training self-supervised features and performing supervised semantic segmentation concurrently. The semantic segmentation labels (learned by the model via supervised training) are used to select positive and negative pairs. Given the semantic labels from the model, features from the same semantic class are selected as positive pairs, while features from different semantic classes are selected as negative pairs. Applying supervised learning in conjunction with feature learning is similar to the method applied in SupCon.

3.2 Contrastive Learning in 3D

The same contrastive pre-training techniques used for images can often be applied to 3D point clouds. Self-supervised pre-training algorithms for point clouds roughly follow the same positive-pair generating procedure outlines for images in Section 3.1. Augmentations are applied to a point cloud (with colour information) that is either captured from a single view or registered using multiple views (see *point cloud registration* in Section 2.2). The augmentations may include coordinate transforms, for example, rotations, and colour transformations (see Appendix A). Using registered point cloud pairs has the added advantage of encouraging the learned features to become invariant to slightly dif-

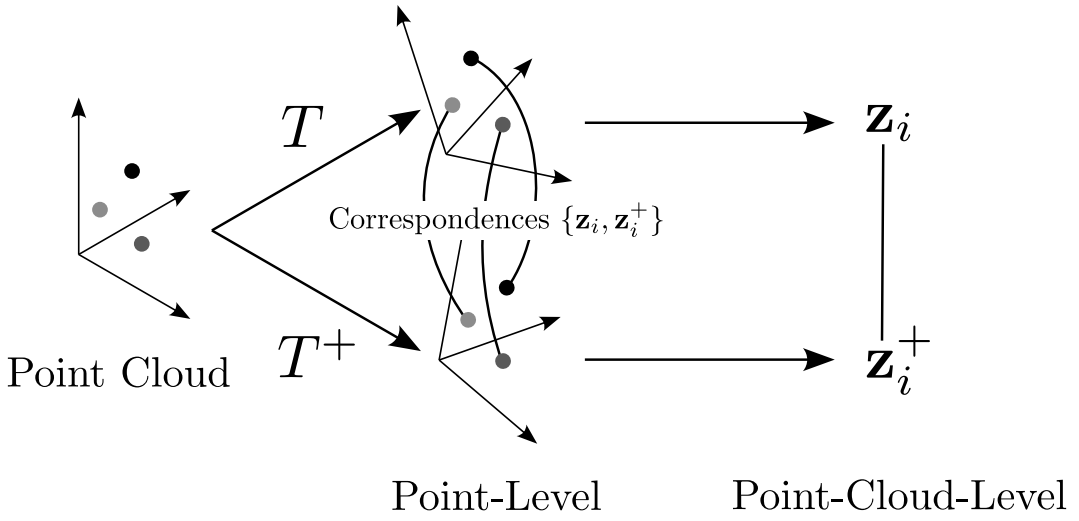


Figure 3.3: Generating 3D feature correspondences for contrastive learning. Contrastive learning at the point level compares and contrasts features vectors defined for individual points within a point cloud, while point cloud-level contrastive learning compares and contrasts features defined over full clouds.

ferent viewpoints. Two augmented point clouds, each considered as a single data point, form a positive pair for training. Alternatively, algorithms that learn features for individual points over multiple scans select positive pairs that are spatially close together in a common world frame, but that belong to different scans. A visualization of how positive pairs are generated for a point cloud is shown in Figure 3.3.

3.2.1 Learning Point Cloud Features

One approach to contrastive learning for point clouds is to learn a single feature for each complete scan. This is the approach taken by DepthContrast [63], which demonstrated that global feature descriptors from individual scans can be effective for downstream point-level scene understanding tasks. However, global feature descriptors produce a single feature vector per scan, while the same scan can generate multiple point-level features. As a result, DepthContrast must process many more scans per iteration than equivalent point-level methods to maintain a sufficient quantity of features for effective contrastive learning. Since processing a large number of scans per pass is computationally expensive, DepthContrast employs a momentum encoder and memory bank (as done by MoCo), allowing for larger batches of negative samples. Despite leveraging the momentum encoder, DepthContrast still requires significantly more computing resources than comparable point-level methods.

3.2.2 Learning Point Features

Learning point-level features is a common choice for 3D contrastive learning algorithms because a single scene can generate a large number of positive and negative pairs. Often, state-of-the-art 3D contrastive learning algorithms utilize overlapping scans instead of a single scan. Fully Convolutional Geometric Features (FCGF) [9] is an early algorithm that uses overlapping scans; the results in [9] demonstrate the usefulness of contrastive learning for point cloud registration. The features learned by FCGF are uniquely identifiable between scans, which allows features with the same coordinates in the world frame to be matched correctly. FCGF was subsequently adapted by PointContrast [59] for use in semantic segmentation tasks. Results from the PointContrast work show that the same features that proved useful for point cloud registration are also useful to initialize weights for downstream semantic segmentation tasks. However, many of the negative samples in PointContrast are taken from spatially distant points, which are easy to distinguish and thus contribute minimally to the gradient at each training step. Contrastive Scene Contexts (CSC) [21] uses a partitioning scheme to select points that are closer in 3D space to enable more efficient contrastive learning. The space around each point is split into spherical segments that are defined by ranges of angles (along an arc) and a minimum and maximum distance from the point. For each partition, a separate loss is computed, where all negative samples are selected from that partition only. The individual losses are then averaged to form a complete loss value for the query point. The model pre-trained with CSC achieves 89% of the fully-supervised instance segmentation accuracy and 90% of the fully-supervised semantic segmentation accuracy (without pre-training) on the ScanNet dataset, using just 0.1% of the available labels during supervised training.

Jiang et al. [24] adapt the technique from SupCon by performing both supervised and self-supervised learning in parallel. The authors of [24] develop a model that contains both an encoder and classification head. The model is trained with full supervision using a cross-entropy loss on a subset of the labelled data during a pre-defined warm-up period. Afterward, a self-supervised InfoNCE loss is added to train the encoder component of the model (jointly with the supervised loss). The pseudo-labels generated from the classification head are used to select positive points, defined as those points that have the same pseudo-label, and negative points, defined as those points that have different pseudo-labels. This prevents negative samples from being selected from the same class.

SegContrast [38] uses a hand-engineered (i.e., non-learning-based) clustering algorithm [14] to segment a scene. The segmentation 'approximates' individual objects; negative pairs are selected such that each member belongs to a different segment. The advantage is that no labels are required for pseudo-label generation.

The method used in this thesis for learning 3D features most closely resembles PointContrast in terms of its operation. PointContrast requires two overlapping registered scans, however, while our method uses a single scan and an image from the same viewpoint (captured concurrently). Operating with single scans individually eliminates any reliance on a sophisticated point cloud registration pipeline, simplifying the data collection process in many cases.

3.3 Multi-Modal Contrastive Learning

The pre-training method presented in this thesis leverages images and point clouds to improve performance on downstream 3D tasks. In this section, we cover methods that also leverage self-supervised pre-training across different modalities. Note that all the algorithms discussed in this section operate solely at the point and pixel level.

We first cover algorithms that pre-train models for image understanding tasks. Pri3D [22] relies on standard perspective projection (see Section 2.3) to produce pixel-point pairs. The pixel-point pairs that map to the same physical 3D location in a common frame are subsequently used as positive pairs in the (pixel-only) contrastive loss. In addition to the 2D loss, Pri3D also employs a 2D-to-3D loss where each pixel and back-projected point forms a positive pair and all other 3D points are used as negative samples. SimIPU [31] differs from Pri3D in that it clusters 3D points into local regions prior to pixel-based matching. The authors of [31] show that SimIPU outperforms other 2D pre-training techniques on an outdoor autonomous dataset, even beating DenseCL (see Section 3.1.2).

Other works have shown that features learned from images can be useful for many 3D tasks. CrossPoint [1] applies contrastive learning to global scene features generated from synthetic point clouds of computer-modelled objects and corresponding rendered images. The CrossPoint algorithm computes three different feature vectors: two vectors are produced by applying a 3D feature extraction model to two different transformations of the synthetic point cloud, where each point cloud produces a single global feature vector, while the third feature vector is derived by applying a 2D feature extraction model to the rendered image of the object. The full contrastive loss has two parts, a contrastive loss between the augmented point cloud feature vectors and a contrastive loss between the image feature vector and the point cloud feature vectors. A major limitation of CrossPoint is that it operates on synthetic object-centric datasets and has not been shown to scale effectively to real-world 3D scans. Superpixel-driven Lidar Representations (SLidR) [47] uses self-supervised 2D features to train a 3D model by projecting

each 3D point to a feature coordinate in the image. Pixel features are learned through a contrastive loss with positive pairs selected by a 'classical' (i.e., non-learning) image segmentation algorithm. A similar algorithm is P4Contrast [33], which performs sensor fusion and self-supervised pre-training on fused 2D-3D inputs. P4Contrast fuses both modalities during pre-training and at runtime, and hence always requires access to images. This means P4Contrast cannot operate on colourized point clouds (by themselves) at runtime.

Pixel-to-Point Knowledge Transfer [34] also projects point cloud data into images. The method first pre-trains a 2D model on ImageNet. The 2D model is subsequently frozen (i.e., all parameters fixed) and the pixel-level features are matched to point-level features using the projected point-pixel pairs. A contrastive InfoNCE loss is then utilized to pre-train the 3D model. The contrastive loss uses the features from the point-pixel matches as positive pairs and treats the features derived from all other 3D points as negative samples. We follow roughly the same process in this thesis, with the differences being the specific model architectures involved and the datasets we use for pre-training and evaluation. We also pre-train our 2D model on images specific to the target dataset, while Pixel-to-Point Knowledge Transfer pre-trains on ImageNet only. The advantage of pre-training on images related to the target environment (e.g., indoor office scenes), on top of general datasets, is that the features learned are optimized to better match the target (image and point cloud) dataset.

Chapter 4

Methodology

This chapter describes our method for self-supervised contrastive learning using both 2D and 3D data, applied to downstream 3D scene understanding tasks. Our method can be split into two distinct and sequential stages. The first stage applies a 2D CNN to generate image features at the pixel level, based on a contrastive loss on the individual pixels. The second stage then uses these image features to train a 3D model. A visualization of both stages is provided in Figure 4.1. Below, we describe our model architectures for feature extraction and downstream scene understanding tasks. We then explain both of our pre-training stages in detail. For more details about specific augmentation implementations, please refer to Appendix A.

4.1 Perception Models

We use self-supervised contrastive learning to pre-train models prior to their use in downstream 3D scene understanding tasks. In this section, we cover the model architecture used for extracting point-level features from point clouds. We then cover downstream models that leverage this feature extractor for 3D semantic segmentation, instance segmentation, and object detection tasks.

Feature Extraction For point cloud feature extraction, we first convert an input point cloud to a voxel grid and use a 3D convolutional neural network to compute a feature for each voxel. Specifically, we use the 3D UNet model presented in Choy et al. [8] that computes sparse convolutions on the voxel grid. We select this architecture because of its popularity among contrastive learning methods (see Section 3.2).

Semantic Segmentation The features from the 3D UNet are used for semantic segmentation. Features are passed through a single convolutional layer that reduces the channel size of the feature to the number of classes being learned. All other dimensions are left unchanged. The output of the last convolutional layer is then passed through a softmax function (see Section 2.6.1) to obtain a class label for each voxel.

Instance Segmentation For the instance segmentation task, we use the architecture from PointGroup [25]. PointGroup first learns the offset between each point and the 3D centroid of the object to which the point belongs. Object centroids are approximated during training by computing the average coordinates of all points belonging to the same object instance. PointGroup then shifts each point by the predicted offset so that the points belonging to the same object are grouped together. By moving the points together, PointGroup is able to cluster points more easily in the model that it produces. The proposed clusters are evaluated by a separate scoring network, which uses a small UNet to extract a single feature vector from each cluster. The feature is passed through a fully connected network with sigmoid activation functions (see Section 2.6.1) to obtain a final score. Clusters with low scores are pruned. Each point is then assigned to the cluster with the nearest centroid to the point (in terms of the Euclidean distance). We modify the feature extraction backbone of PointGroup to use the 3D UNet presented in [8] so that the backbone is standardized across all tasks.

Object Detection For object detection, we use VoteNet [42], since it is a standard among contrastive learning algorithms (see Section 3.2). VoteNet also predicts the centroid of the object instance that a point belongs to, as is done by PointGroup. The predictions are grouped into a fixed number of clusters based on the spatial proximity of each instance centroid prediction. The bounding box size, location, orientation, and semantic class are computed by passing the set of aggregated features from each cluster through a separate network. We modify the feature extraction backbone of VoteNet to also utilize the same 3D UNet used for all of our other scene understanding tasks (i.e., the Minkowski UNet [8]).

4.2 Stage 1 - Learning Image features

In this section we describe our method for generating 2D features from unlabeled images—this step corresponds to *Stage 1* of Figure 4.1. We follow the ResUNet architecture from Godard et al. [16], which is a UNet-style network architecture commonly used for monoc-

ular depth estimation. The backbone consists of a ResNet34 network with its final and intermediate encodings passed into the decoder via skip connections. We modify the decoder to compute a 16-dimensional feature vector for each pixel in the input image, instead of a depth estimate. A diagram showing our modified network is provided in Figure 4.3. The use of a 2D model also allows us to leverage other sources of data, for example, we pre-load the weights of the encoder from a model trained on the large ImageNet corpus [12].

To pre-train the full model, images are selected from a desired pre-training dataset. We follow roughly the same data augmentation and use the same InfoNCE loss function as SimCLR, except that we compare pixel-level features instead of image-level features. Positive samples are generated by randomly cropping an image and then applying the following transformations sequentially: resized crop, horizontal flip, image-wide colour-to-greyscale conversion, gaussian blurring, and colour jitter. The augmentation procedure follows the description in Section 3.1 and the visualization in Figure 3.1. All transformation parameters are randomly sampled at each iteration. Each transformation has a pre-defined random chance of being applied to the input. Only pixels identified within the overlap region of the two crops are used in the objective function. Pixels that map back to the same coordinates in the original image are considered as positive samples, while all others, including those from other images in a batch, are considered as negative samples. These positive and negative samples are passed into the contrastive loss function described in Equation (2.42). As discussed in Section 2.3, the resizing operation results in fractional pixel coordinates when the augmented image coordinates are mapped back to the original image. This means that positive pairs must be selected according to the Euclidean distance between fractional pixel coordinates. Those pairs with a distance below a given threshold (e.g., 1.5 pixels in our work) are considered to form a positive pair.

In order to find positive pairs, we keep track of how the coordinates of the original image are modified by the transformations applied to each augmented view. We build a separate image with the same shape as the original but with the pixel values representing (or encoding) coordinates rather than colour. We pass this *coordinate image* through the same transformations as the augmented image. The coordinate image output represents the fractional pixel coordinates of the augmented image pixels in the original image. We then find which pixels from the two augmentations that map to the same point in the original image. This search can be performed efficiently using a k-d tree. Pixels that are not from the overlap region of the two crops are discarded. Note that positive pair selection is discrete and non-differentiable and hence does not appear in the loss function.

Pseudocode for the InfoNCE loss is listed in Algorithm 1, while pseudocode for Stage 1 of our pipeline is listed in Algorithm 2. One potential source of confusion in the pseudocode for the InfoNCE loss function is that negative samples are not explicit inputs into the loss calculation. Instead, negative samples for a given query point are derived from the set of all other positive samples. Defining negative samples in this way is common practice in most algorithms using an InfoNCE loss. Examples of algorithms that use more sophisticated approaches for producing negative samples can be found at the end of Section 3.1. The loss can be calculated efficiently by assembling the feature vectors into two matrices, one for the query points, z_1, and another for the positive samples, z_2. Each column of these matrices defines an individual feature vector. The columns are normalized according to the ℓ^2 metric, which improves the training stability (as discussed in Section 2.6.4). The similarity between feature vectors in z_1 and z_2 is then determined by a single matrix multiplication, where the similarity scores are defined by the vector dot product (i.e., rows times columns). We refer to the output as a *similarity matrix*. Subsequently, the Stage 1 InfoNCE loss is computed according to Equation (2.42). First, each element in the similarity matrix is transformed by the following sequence of operations: 1) inverse scaling by a temperature parameter τ and 2) application of an exponential function. These operations are applied to each dot product (i.e., similarity score) in the InfoNCE loss. Second, the similarity between positive pairs is extracted from the diagonal elements of the updated similarity matrix. The diagonal elements of the similarity matrix represent the similarity between feature vectors in matrices z_1 and z_2 at the same index position (which we have defined as a positive pair). Conversely, negative samples are defined as all other points in z_2. Thus the denominator of the InfoNCE loss is computed by summing the rows of the similarity matrix. This result represents the sum of similarities between query points and all other points in z_2. Finally, each positive similarity value (diagonal entry) is divided by the sum of similarities to all other points in z_2 and passed through a logarithm function. The loss is averaged over all query points.

4.3 Stage 2 - Learning Point Features

In this section, we describe our approach for pre-training a 3D point-level feature extraction model using pixel-level features from *Stage 1*—this step corresponds to *Stage 2* of Figure 4.1. Mapping between 3D points and 2D features is done via perspective projection using the pinhole model (see Section 2.3). We use the same 3D model as in [59] and treat the final 1×1 convolution as the decoder, which we initialize from scratch for training on downstream tasks. A diagram of this network can be found in Figure 4.2.

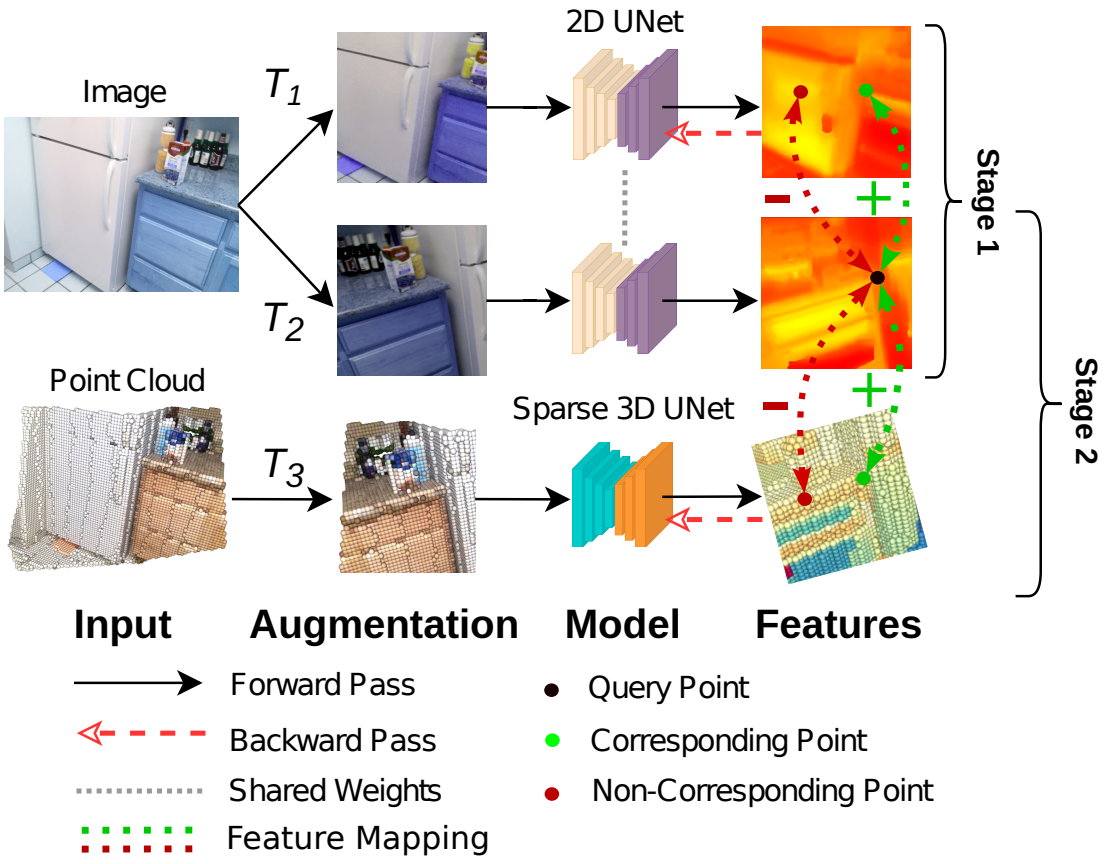


Figure 4.1: Overview of our multi-modal contrastive learning framework. The first stage trains a 2D model on images at the pixel level. The next stage then uses the learned image image features as targets for pre-training a 3D model.

We use the same contrastive loss as defined by Equation (2.42) and follow the same notation and positive and negative sampling strategies as defined in Section 2.6.4. The query point features, $\mathbf{z}_i$, are obtained from the point-level output of the 3D model. The corresponding positive samples, $\mathbf{z}_i^+$, are obtained from the feature vectors of the 2D model using the point-pixel pairs computed in the perspective projection. Negative samples, $\mathbf{z}_i^-$, are any outputs from the 3D model not derived from the query point. Note that the feature output of the 3D model is set to match the pixel-level features of the 2D model. Stage 2 can be visualized by referring to Figure 2.12, where the 3D query points are passed through the 3D model along the top pathway, while 2D pixel values (acting as positive sample points) are passed through the 2D model along the bottom pathway. We find that using a contrastive loss where pixel features serve as positive samples is the most effective method to pre-train the 3D network using 2D features. Alternatively, the features of each point-pixel pair could be compared directly in a different loss function such as an ℓ^2 loss (See Section 2.6.1), without the use of negative samples. However, we found that an ℓ^2 loss did not improve performance on downstream tasks. This suggests that the negative samples play an important role in the learning of high-quality features.

During Stage 2 training, our 2D network model is held frozen. The fixed 2D features then act as a target for the 3D model to learn. Images are augmented before being passed through the 2D model by sequentially applying the following transformations: resizing, square centre cropping, and colour normalization. Note that these image transformations are deterministic with no random component. Each point cloud is also augmented so that the model learns to be invariant to differences in orientation, point density and colour fluctuations. We transform point clouds with a random horizontal flip and random rotation around the z axis while each point colour is transformed using auto-contrast blending, colour translation, and colour jitter. The colour transformations do the following: auto-contrast blending mixes the original input with a modified version of the input that maximizes contrast by re-scaling the colours (so that the lowest value is totally black and the highest value is bright white), colour translation shifts all the colours by a set amount, and colour jitter adds random noise to the colour of each data point. The types of augmentations applied are different for images and point clouds because the modalities differ. Pseudocode detailing how these augmentations are implemented can be found in Sections A.2 and A.3. Pseudocode for Stage 2 is listed in Algorithm 3.

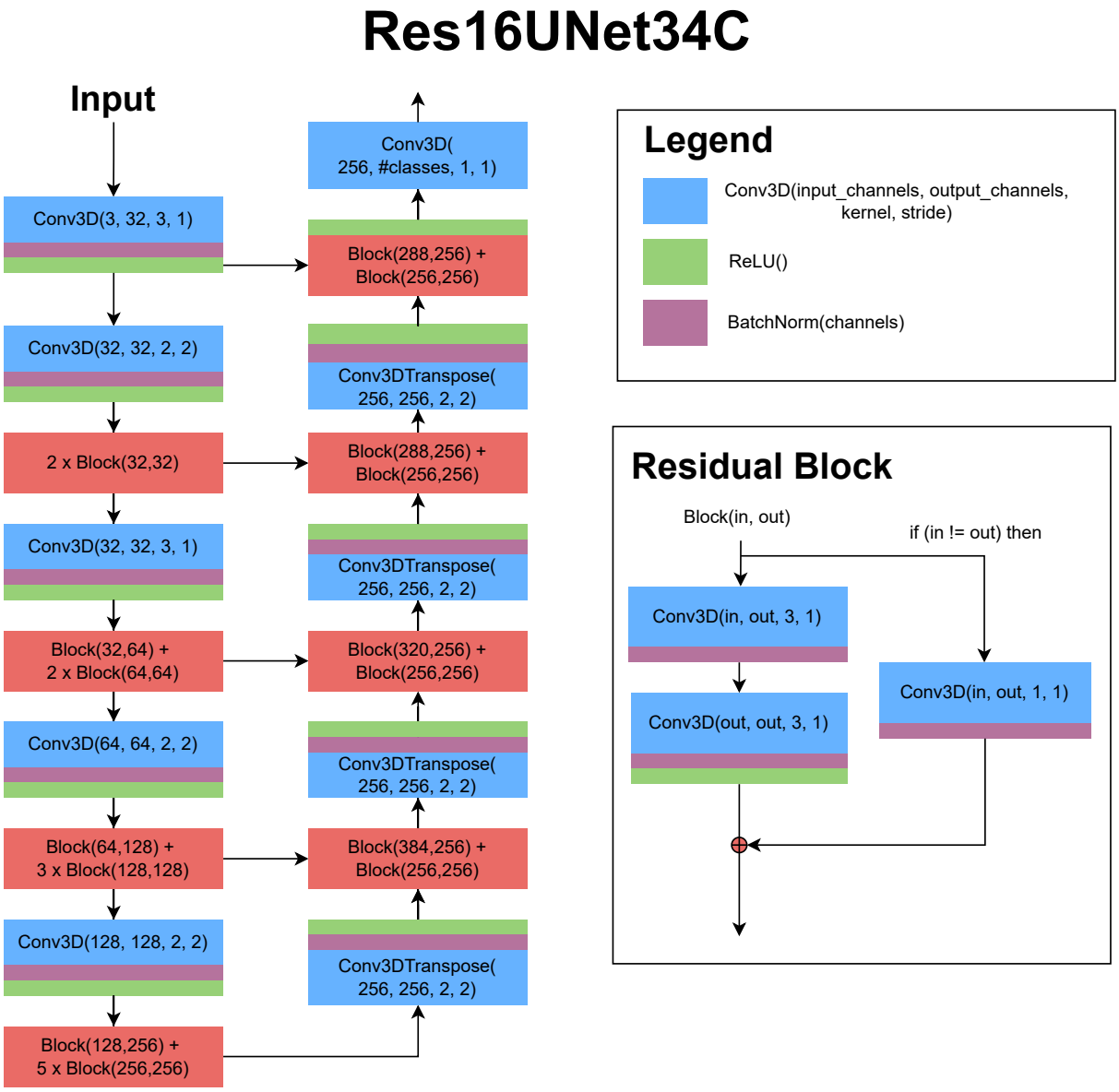


Figure 4.2: Network diagram of a 3D Res16UNet34C model. This network is used as the backbone for all 3D tasks. The model computes sparse 3D convolutions as described in Section 2.6.3. All kernels and stride parameters are 3D and symmetrical. The kernel used is a hypercube (i.e., the same size in all dimensions).

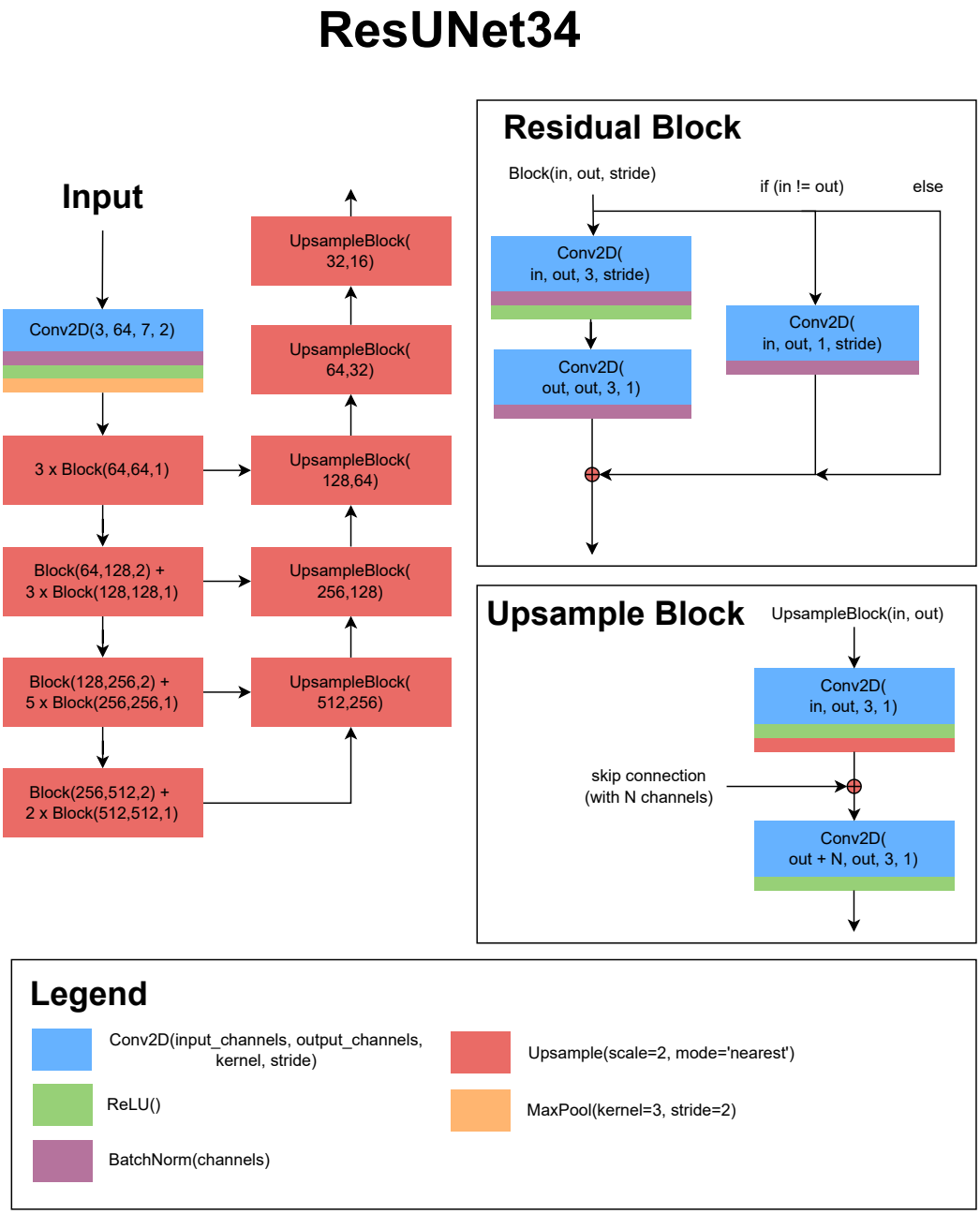


Figure 4.3: Network diagram of a 2D ResUNet34 feature extractor. This network is used to extract features from 2D images. All convolutions, kernels and strides are expressed in two dimensions. The last two layers have no skip connections and only reduce the size of the feature channel.

Algorithm 1 Contrastive Loss Pseudocode

```
# m: number of samples
# tau: smoothing parameter
# z1: features of query points
# z2: features of positive samples corresponding to z1
# Note that z1 and z2 are matrices with individual
# features as columns
def loss(z1, z2):

    # L2 normalization along columns
    z1 = normalize_along_columns(z1)
    z2 = normalize_along_columns(z2)

    # Compute the similarity between augmented data points
    similarity = matrix_multiplication(z1, transpose(z2))
    similarity = similarity / tau

    # Apply element-wise exponential
    similarities = exp(similarities)

    # Select positive labels by similarity
    # between the same points, which corresponds
    # to diagonal entries in the similarity matrix
    positive_similarities = get_diagonal_entries(similarities)

    # Select negative labels as all points
    # for each query point
    negative_similarities = sum_along_rows(similarities)

    # Compute loss in steps. Note that the division
    # and logarithm are applied element-wise.
    # The last step averages across all query points
    loss = positive_similarities / negative_similarities
    loss = log(loss)
    loss = sum(loss) / m
    return loss
```

Algorithm 2 Stage 1 Pseudocode

```
# f: model
# tau: smoothing parameter
# m: number of pixels to sample per positive image pair

for images in loader: # load a set of images
   L = 0 # Zero the loss
   for image in images:

      # Apply random augmentations while keeping track
      # of the coordinates of each augmented pixel
      # in the original image
      x1, coordinates1 = augment(image)
      x2, coordinates2 = augment(image)

      # Model outputs
      z1, z2 = f(x1), f(x2)

      # Match pixels between augmentations (using a KD-tree)
      matches1, matches2 = match(coordinates1, coordinates2)
      z1, z2 = z1[matches1], z2[matches2]

      # Randomly sample m pixels
      indices = random(m)
      z1, z2 = z1[indices], z2[indices]

      # Add to loss
      L += loss(z1, z2)

   L.backward() # Back-propagate
   update(f) # Update model parameters
```

Algorithm 3 Stage 2 Pseudocode

```
# f: 2D model
# h: 3D model
# m: number of point-pixel pairs to sample per image-scan pair

for pairs in loader:
   L = 0 # Zero the loss

   # Load each image and scan
   for image, scan in pairs:

      # Apply random augmentations depending on the modality
      # Image augmentations keep track of the augmented
      # pixel coordinates in the original image
      x1, coordinates = augment_2D(image)
      x2 = augment_3D(scan)

      # Get model outputs
      z1, z2 = h(x1), f(x2)

      # Upsample image features to match original
      # pixel coordinates
      z1 = interpolate(z1, coordinates)

      # Get matches between image pixels and scan points,
      # generated either from stereo matching
      # or by projecting 3D points into the image
      pixel_matches, point_matches = match(image, scan)
      z1, z2 = z1[pixel_matches], z2[point_matches]

      # Randomly sample m point-pixel pairs
      indices = random(m)
      z1, z2 = z1[indices], z2[indices]

      # Add to loss
      L += loss(z1, z2)

   L.backward() # Back-propagate
   update(f) # Update model parameters
```

Chapter 5

Results

In this chapter, we benchmark the performance of our pre-trained model on a variety of popular datasets and tasks. We compare our method to state-of-the-art baselines on three different downstream tasks: semantic segmentation, instance segmentation, and object detection. An overview of our results, across different datasets and tasks, can be found in Table 5.1.

5.1 Datasets

We use three indoor stereo datasets that are commonly used to benchmark 3D contrastive learning [21, 24, 59, 63] together with an extra outdoor lidar dataset [4]. In this section, we describe each dataset in detail. We use ScanNet to pre-train our backbone network as it is the largest indoor dataset available. ScanNet shares many of the same classes and types of indoor scenes as S3DIS and SUNRBGD and therefore is a good candidate for demonstrating the impact of pre-training on a large unlabelled dataset. We also use ScanNet to pre-train our SemanticKITTI model, in order to evaluate how learned 3D features can generalize to substantially different environments and sensor types.

ScanNet [11]: The main indoor dataset is ScanNet, which comprises of roughly 1,600 reconstructed scenes of indoor environments. The raw data comprises individual RGB-D images. The depth channel from the RGB-D output is converted into a point cloud using Equation (2.20). ScanNet scenes are mostly of individual rooms and range in size from 2 metres to 10 metres on a side, with a standard ceiling height of about 3 metres. The various rooms contain 20 different classes of labelled objects. We use the dataset-defined training and validation splits, and use the validation set as our test set.

S3DIS [2]: S3DIS is a much smaller dataset than ScanNet and only comprises of 300 reconstructed scenes. However, the size of these scenes varies quite drastically; some are of rooms and others are of entire auditoriums. The scenes are mainly of indoor office environments and each scan is an RGB-D image (see Section 2.1). There are 13 different semantic classes and all scenes contain point-level semantic and instance labels. Following [21, 59], we use the Area 5 split for our validation and test sets.

SunRGBD [23, 49, 50, 58]: SunRGBD is a 3D dataset of indoor office environments. SunRGBD contains roughly 10,000 RGB-D scans and corresponding 3D bounding box annotations for 10 different classes, with labels similar to those of ScanNet and S3DIS. There is no scene reconstruction available. We use the dataset-defined training and validation split.

SemanticKITTI [4, 15]: SemanticKITTI contains roughly 25,000 laser scans of outdoor driving environments. It has 20 different classes with 3D semantic and instance labels. The scans are of a full rotation around the vehicle, going out to a range of about 20 metres, with the spatial resolution decreasing with distance. We use sequences 1-7 and 9-10 as our training set and sequence 8 as our validation set.

5.2 Baselines

To verify the effectiveness of our method, we compare against three state-of-the-art baseline algorithms: PointContrast [59], Contrastive Scene Contexts (CSC) [21], and DepthContrast [63]. Both PointContrast and Contrastive Scene Contexts require pairs of scans with known poses and at least 30% overlap, while DepthContrast and our method operate on single scans only.

Due to our own resource limitations, we run DepthContrast with a batch size of 32 on a single graphics processing unit (GPU), instead of with a batch size of 1,024 split across 32 GPUs. We run DepthContrast for 40 epochs instead of 400, which still takes twice as long as any other method. This should serve as a more fair comparison between algorithms when access to large compute clusters is not possible. Where applicable, we also compare against a fully-supervised backbone to give an idea of a reasonable upper-bound on performance improvement.

5.3 Implementation Details

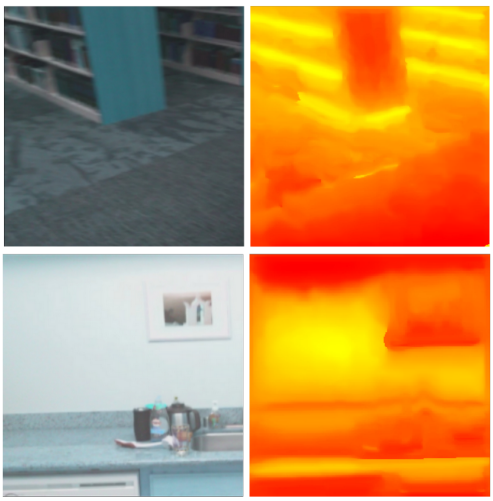
Our model is built using the PyTorch [40] package. We use PyTorch for all deep learning tasks, including data loading, model building, and backpropagation during training.

Model Our backbone network is the standardized UNet described in Section 4.1, which is implemented with the sparse convolutional library developed by [8]. The backbone, shown in Figure 4.2, uses a Res16UNet34 structure with non-bottleneck blocks and a maximum feature embedding size of 256. The outputs of the model are used directly for semantic segmentation. Instance segmentation is done using the same backbone; extracted features are subsequently fed into PointGroup [25]. Object detection is performed with VoteNet [42].

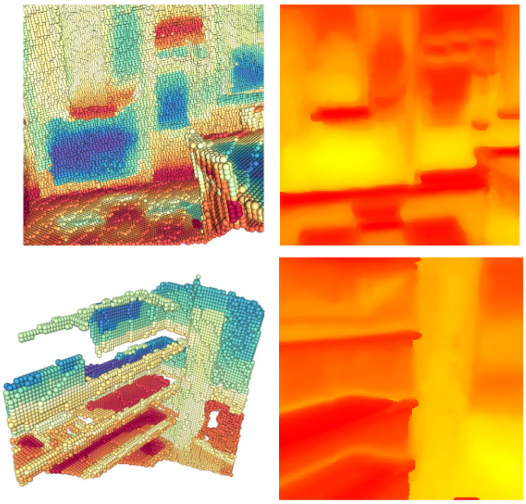
Pre-training of the 2D Backbone For pre-training of the 2D backbone on images, we use a batch size of 64 image pairs. For each pair, we sample 4,092 pixels to contrast in our loss function (see Equation (2.42)). The total loss for each training iteration is the sum of the loss for each image pair. The loss function uses a τ of 0.4, a learning rate of 0.01, an SGD optimizer with a momentum of 0.9, dampening of 0.1, and a weight decay of 0.004. We decrease the learning rate according to an exponential scheduler with an exponential rate of 0.99. Positive samples are generated using the transformations described in Section 4.2. We pre-train the image network for 20,000 iterations.

Pre-training of the 3D Backbone Pre-training of the 3D backbone is carried out on the ScanNet dataset. We use a batch size of 8 scan-image pairs and select 2,000 point-pixel correspondences per pair. This is a significantly smaller number than for 2D pre-training because of the increased compute and memory usage required by point cloud data (compared to image-data). The images are centre-cropped to 224×224 pixels to conform to the expected input size of the 2D model. We voxelize the points, with a voxel size of 5 cm per side. The hyper-parameters are the same as those for 2D training except that the learning rate is set to 0.1. The 3D model is pre-trained for 20,000 iterations.

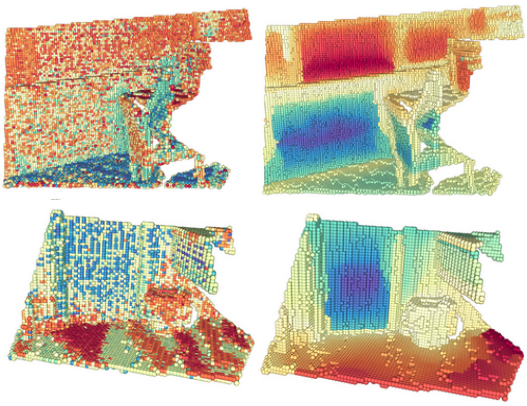
Downstream Tasks All downstream tasks are run using the pipeline and parameters from CSC [21]. Note that we obtain slightly different numbers from those originally reported in [21]. We believe this is due to a required update to the core sparse convolution library (to run on our newer GPU). We also use a single GPU instead of eight. In this case, all pre-training was run from scratch.



(a) Input image on the left with corresponding feature visualization on the right.

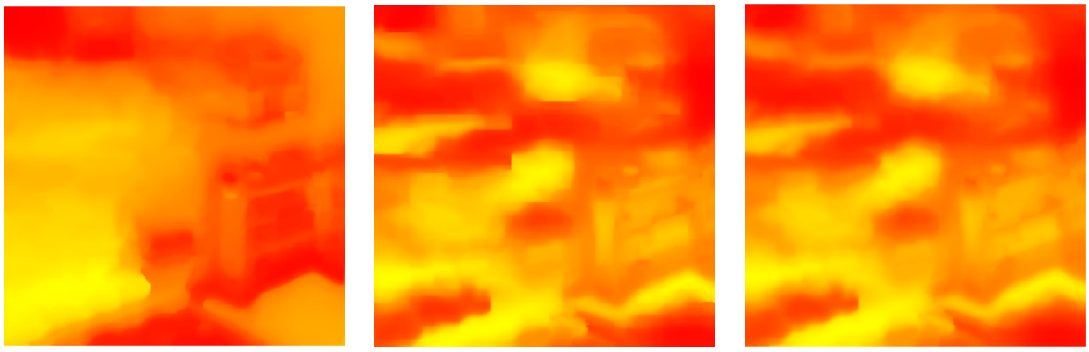


(b) Point features on the left with corresponding image features on the right.



(c) Point features before (left) and after (right) pre-training with our method.

Figure 5.1: Visualization of 2D and 3D feature vectors.



(a) Original (b) Closest (c) Bilinear

Figure 5.2: Comparison of feature interpolation methods for upsampling the representation to the original image size.

5.4 Visualization of Image Features

Before pre-training a 3D model with 2D features, we first verify that the 2D features learned using our 2D pre-training scheme have some connection to the original image. We do so qualitatively following the approach from [9]. We bring each pixel feature into a 1D colour space using the t-SNE [54] algorithm and build a heat map image using the pixel's original coordinates. Figure 5.1a shows a comparison between the visualizations and the original image. The visualizations show a clear mapping between input image and output features. The heatmaps tend to 'highlight' structures such as lines and surface patches that are present in the input image. Key parts of the image share similar feature vectors (e.g., individual shelves, tables and chairs).

Using the same technique, we also visualize what the interpolation of the features looks like in Figure 5.2. We compare the following interpolation schemes: closest pixel and bilinear interpolation. Using closest-pixel interpolation produces abrupt changes between pixels whereas the bilinear interpolation yields a much smoother result. Note that the visualizations are a 1D approximation of a higher-dimensional feature space, so small changes to the input may cause small deviations and colour changes. However, the general structural components should be distinguishable.

5.5 Visualization of Point Features

Similar to the visualization of 2D features, we visualize point features from individual scans by mapping them into a common color space using the t-SNE algorithm. The features are the outputs of models, having passed through both stages of our pre-training

framework. Figure 5.1b shows the relation between the 2D and 3D features. There is a clear mapping between the image and the corresponding point cloud heatmaps. This correlation verifies that the 3D model has indeed learned to 'mimic' the features of the 2D model without relying on the image. Parts of the scene, such as walls and shelves, have consistent features that are present across both modalities. Figure 5.1c shows the difference between randomly initialized and pre-trained features. Features that are pre-trained follow visible object boundaries.

5.6 Semantic Segmentation

Our first test case is pre-training our model on ScanNet and fine-tuning on S3DIS. Results are found in Table 5.1, with a detailed class breakdown found in Table 5.2. Fully-supervised pre-training on ScanNet has a drastic impact on the final downstream performance (+5.1 mIOU) and is used as a rough upper bound on expected performance. Almost all pre-training algorithms improve downstream results, except DepthContrast. Our algorithm is comparable to PointContrast in terms of performance, since both methods achieve an mIOU increase of more than 1% but perform worse than CSC.

When ScanNet is used for both pre-training and supervised learning (under the ScanNet column in Table 5.1), all methods fail to improve semantic segmentation. Although we see no improvement on ScanNet, we find that pre-training methods do improve performance when the number of annotations during supervised training on downstream tasks is reduced, as seen in Figure 5.4. Using varying proportions of labels for the raw data simulates the use case where only a portion of the data collected are annotated. When ScanNet is restricted to only 5% of the raw data labels, all methods show an improvement.

Using the model pre-trained on ScanNet shows performance improvements on SemanticKITTI as well, even though the datasets are quite different. In Figure 5.6, we test the effect of varying the amount of labelled data on SemanticKITTI. We find that in almost all scenarios, pre-training methods are able to help improve performance. Our method, when applied to SemanticKITTI, however, falls short of all other algorithms. Interestingly, the performance gain of pre-training algorithms improves as more labelled data for downstream supervised training becomes available. Since there is no overlap between the pre-training and training datasets, increasing the ratio of labelled to unlabelled data does not expose the model to previously-seen samples. Therefore, when pre-training and training datasets have no overlap, pre-training offers a performance improvement even when labels are plentiful. The performance improvements on S3DIS

and SemanticKITTI suggest that exposing models to different environments and sensor setups helps to learn features that are better able to generalize.

To visualize the impact that pre-training has on downstream semantic segmentation, in Figure 5.3 we colourize scenes from S3DIS with their semantic labels. The visualization shows that the segmentation of the model without pre-training is already quite good and that there is a minimal difference between the different pre-training methods. The differences that appear are usually confined to the boundaries of walls and objects. Although the metrics show clear differences, these differences may play only a small practical role in the real world.

5.7 Instance Segmentation

As in the previous section, we evaluate the performance of pre-training on ScanNet and training on S3DIS for instance segmentation. Results are found in Table 5.1 with a detailed class breakdown found in Table 5.3. Almost all pre-training methods are able to come close to or even surpass supervised results with the exception—again—being DepthContrast. The best-performing method is CSC, which achieves a performance boost of 4.8 mAP@0.5. Our own method performs similarly to supervised pre-training, with a boost of 2.8.

As with semantic segmentation, all methods struggle to improve on ScanNet instance segmentation, even when access to labelled data is limited as shown in Figure 5.4. In most cases, the algorithms perform substantially worse than training from scratch. This is in spite of the boost on semantic segmentation with the same data ratios.

5.8 Object Detection

Our last test case is the downstream task of object detection. Results for the ScanNet and SunRBGD datasets are found in Table 5.1. This task is interesting because, unlike segmentation, it operates at an object level instead of a point level (which is our pre-training objective). All methods besides DepthContrast show a strong improvement on both datasets. CSC exhibits the best performance with a boost on SunRGBD of 3.1 mAP, while our method attains the largest boost on ScanNet with an increase of 2.5 mAP. For the segmentation task, using the same dataset for both pre-training and the downstream task yields no performance gain. However, object detection on ScanNet shows strong improvements even when access to the full dataset is available. This outcome suggests

that pre-trained point-level features generalize well to object-level tasks when using the same raw data for both.

5.9 Varying Overlap

A key distinguishing advantage of single-scan methods (i.e., our method and DepthContrast) versus multi-scan methods (i.e., PointContrast and CSC) is that single-view algorithms do not require a robust point cloud registration pipeline. In terms of single-scan methods, we believe that ours is superior owing to the extreme computational complexity and the deserved poor performance of DepthContrast. However, before definitively concluding that our method is the best for single-scan pre-training, we wanted to explore what effect varying the overlap region of multiple scans had on the performance of PointContrast and CSC. Specifically, we wanted to see whether the multi-scan methods can perform just as well when given access to single scans only. We tested two scenarios: one where the minimum overlap was raised from 30% to 70% and another where the same scan was used for both pathways (i.e., minimum 100% overlap). By increasing the minimum overlap ratio, the area around points from different scans contains less information, because the matched scans are from almost the same viewpoint. For this reason, restricting the overlap ratio should decrease the performance of multi-view algorithms. Tabular results for S3DIS are found in Tables 5.2 and 5.3, while plots of performance on varying ScanNet data portions are found in Figure 5.5. The effect is highly variable. Generally, using the same scan has a negative effect and performance is substantially worse than when using an overlap of 30%. However, in some rare cases such as semantic segmentation with only 5% of the labelled ScanNet data, using the same scan with CSC achieves the best results of any method. At 40% labelled ScanNet data, the performance gain of using CSC with a single scan performs much worse than using CSC with a minimum overlap of 30%. Using a larger minimum overlap of 70% has a positive effect in many situations, such as for S3DIS semantic segmentation and instance segmentation, but again the results are highly variable. This variability further emphasizes the difficulty of finding features that are meaningful across multiple tasks and datasets. Small variations in an algorithm or a dataset can yield wildly different results. Pre-training may require a large amount of experimentation and fine-tuning for a particular dataset to achieve the best performance. Our method does not require overlapping scans at all, which is an advantage since the choice of overlap ratio can lead to large variability for downstream tasks. As we discuss in Section 5.11, we find that our method does achieve more consistent gains in performance when compared to other methods.

Pre-Training Method		S3DIS		ScanNet			KITTI	SUNRGBD
		Semantic	Instance	Semantic	Instance	Object	Semantic	Object
	Scratch	65.1	53.0	67.4	49.0	35.2	41.0	32.0
	Supervised	70.2 (+5.1)	56.2 (+3.2)	–	–	–	–	–
Multi-Scan	PointContrast [59]	66.2 (+1.1)	54.8 (+1.8)	66.9 (-0.5)	49.1 (+0.1)	36.7 (+1.5)	42.1 (+1.1)	34.2 (+2.2)
	CSC [21]	**69.0 (+3.9)**	**57.8 (+4.8)**	67.6 (+0.2)	**49.3 (+0.3)**	36.1 (+0.9)	**43.0 (+2.0)**	**35.1 (+3.1)**
Single-Scan	DepthContrast [63]	64.9 (-0.2)	52.3 (-0.7)	67.4 (+0.0)	48.7 (-0.3)	33.9 (-1.3)	42.0 (+1.0)	32.9 (+0.9)
	Ours	66.5 (+1.4)	55.8 (+2.8)	**67.7 (+0.3)**	48.5 (-0.5)	**37.7 (+2.5)**	42.0 (+1.0)	33.1 (+1.1)

Table 5.1: Downstream performance comparison of pre-training methods. All methods use weights pre-trained on ScanNet. The difference in brackets is with respect to training from scratch. Semantic segmentation uses the mIOU metric, while both instance segmentation and object detection tasks use the mAP@0.5 metric with a minimum correct overlap ratio of 0.5. The best self-supervised pre-training result for each task and dataset combination is highlighted in bold.

Pre-Training Method	**Min Overlap**	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter	mIOU
Scratch	-	90.34	96.84	80.63	0	25.5	56.95	62.93	75.14	86.21	65.77	72.12	76.91	56.4	65.06
Supervised	-	91.81	96.74	84.11	0.21	34.68	57.64	81.53	77.63	89.82	85.76	76.03	79.27	57.67	70.22 (+5.16)
PointContrast	30%	91.97	**96.67**	82.4	0	17.84	56.49	75.39	77.63	**87.35**	63.3	73.53	**79.55**	58.76	66.22 (+1.16)
PointContrast	70%	91.85	96.56	83.55	0	28.25	**59.36**	73.53	**78.51**	88.28	79.95	**74.61**	76	61.08	68.58 (+3.52)
CSC	30%	92.34	96.02	83.53	0	40.88	57.73	72.11	77.5	88.15	74.61	74.52	76.45	**62.64**	68.96 (+3.9)
CSC	70%	**92.68**	96.63	**83.99**	0	**42.37**	55.95	**77.17**	75.74	**89.57**	**82.74**	73.45	80.67	60.55	**70.12 (+5.06)**
PointContrast	Single-Scan	91.03	96.27	82.8	0	31.53	57.27	69.33	76.97	88.83	45.49	71.02	77.09	57.06	64.97 (-0.09)
CSC	Single-Scan	91.68	96.59	83.21	0	40.38	58.12	70.42	76.77	87.13	69.33	68.77	**82.34**	58.45	67.94 (+2.88)
DepthContrast	Single-Scan	90.99	95.45	80.99	0	32.52	51.94	61.57	74.55	87.28	71.62	71.34	67.71	57.85	64.91 (-0.15)
Ours	Single-Scan	90.8	96.57	82.52	**0.110**	30.94	56.94	68.71	75.12	88.48	71.7	72.17	73.783	56.35	66.48 (+1.42)

Table 5.2: S3DIS semantic segmentation results (mIOU). The best self-supervised pre-training result for each task and dataset combination is highlighted in bold.

Pre-Training Method	**Min Overlap**	beam	column	window	door	table	chair	sofa	bookcase	board	clutter	mAP@0.5
Scratch	-	0	12.8	61.7	92.0	51.4	87.3	62.4	38.5	87.0	37.1	53.0
Supervised	-	0	26.0	74.3	84.5	43.3	87.3	81.8	37.4	85.4	41.2	56.2 (+3.2)
PointContrast	30%	0	24.7	67.9	90.7	43.8	**92.0**	72.7	32.3	85.6	38.0	54.8 (+1.8)
PointContrast	70%	0	22.6	**71.6**	94.6	45.8	87.9	71.7	37.0	81.0	**40.9**	55.3 (+2.3)
CSC	30%	0	32.9	69.2	93.0	47.6	87.5	80.9	39.6	87.5	40.4	57.8 (+4.8)
CSC	70%	0	27.2	76.9	95.4	42.2	90.3	**90.9**	**42.6**	84.7	38.7	**58.9 (+5.9)**
PointContrast	Single-Scan	0	19.8	70.7	92.1	48.4	90.5	81.8	38.4	**90.4**	34.1	56.6 (+3.6)
CSC	Single-Scan	0	32.0	69.3	**96.1**	38.7	89.4	87.6	38.8	81.3	38.8	57.2 (+4.2)
DepthContrast	Single-Scan	0	**40.3**	68.4	74.9	42.2	87.8	62.4	33.3	78.1	35.5	52.3 (-0.7)
Ours	Single-Scan	0	35.0	70.0	83.4	**53.9**	88.9	71.7	42.2	76.1	36.7	55.8 (+2.8)

Table 5.3: S3DIS instance segmentation results (mAP@0.5). The best self-supervised pre-training result for each task and dataset combination is highlighted in bold.

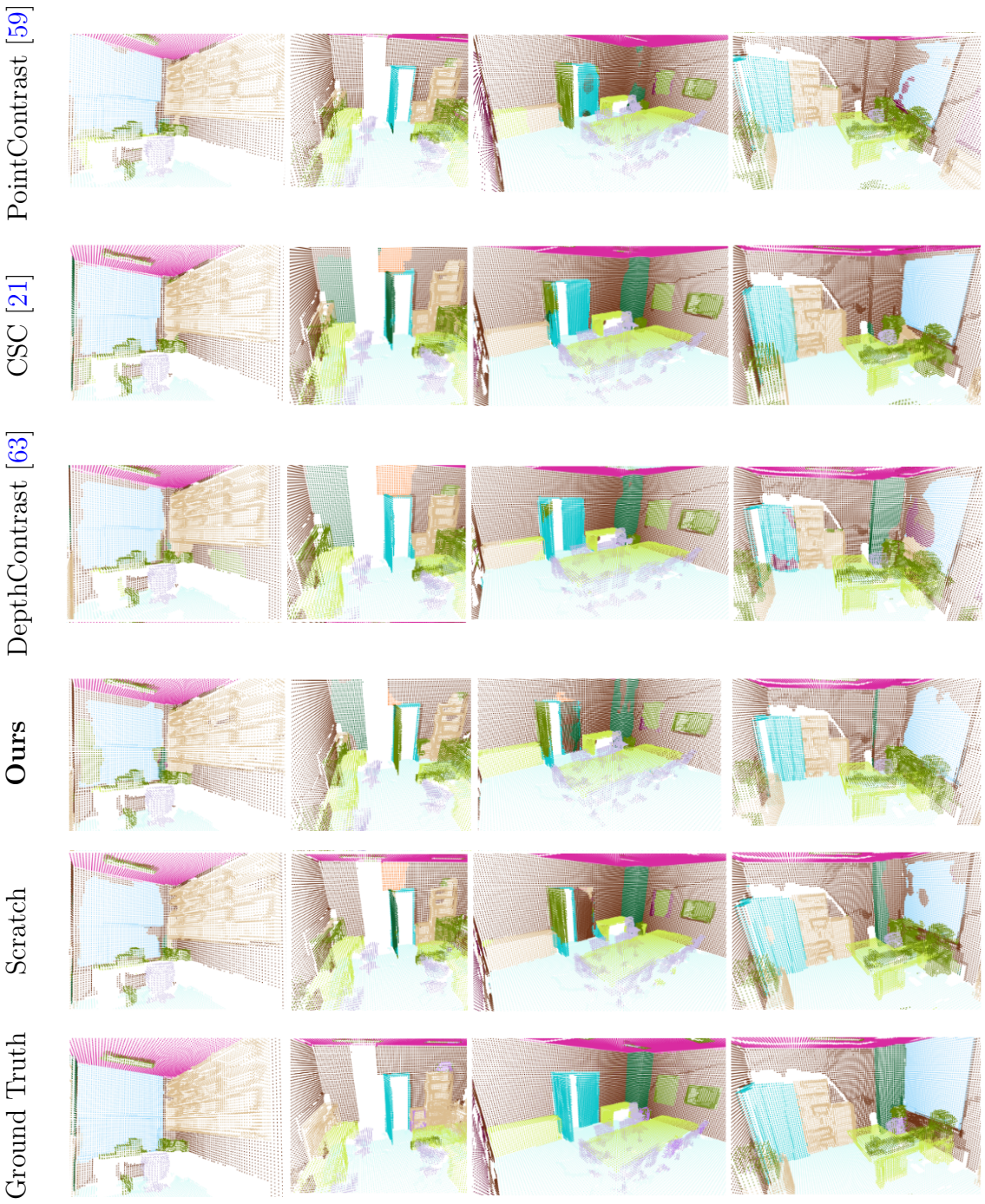


Figure 5.3: Visual comparison of semantic segmentation performance. All results are from pre-training on ScanNet and running on the S3DIS dataset.

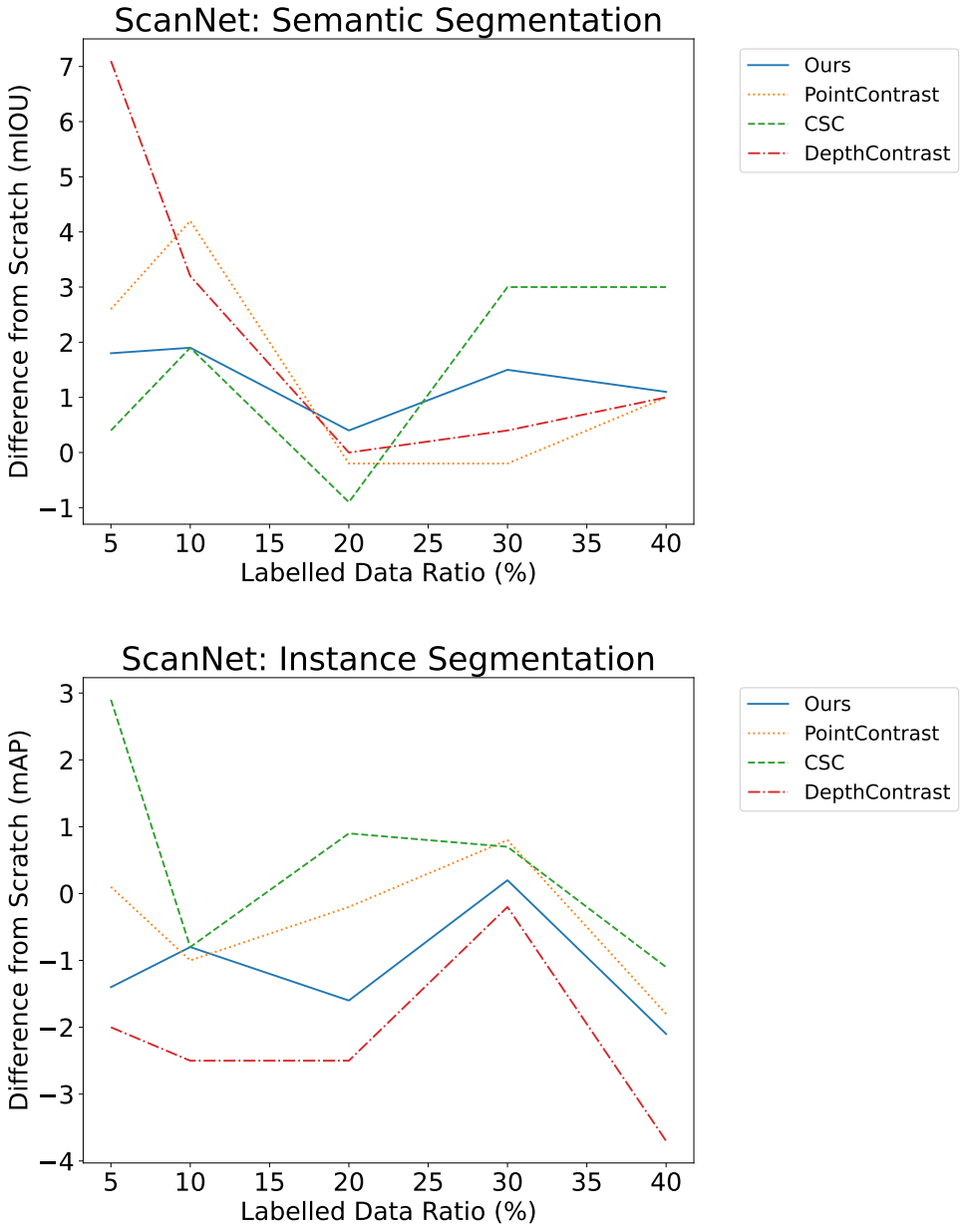


Figure 5.4: Performance on ScanNet over varying labelled data proportions. The performance is measured relative to training from scratch (with no pre-training). Positive differences indicate better performance.

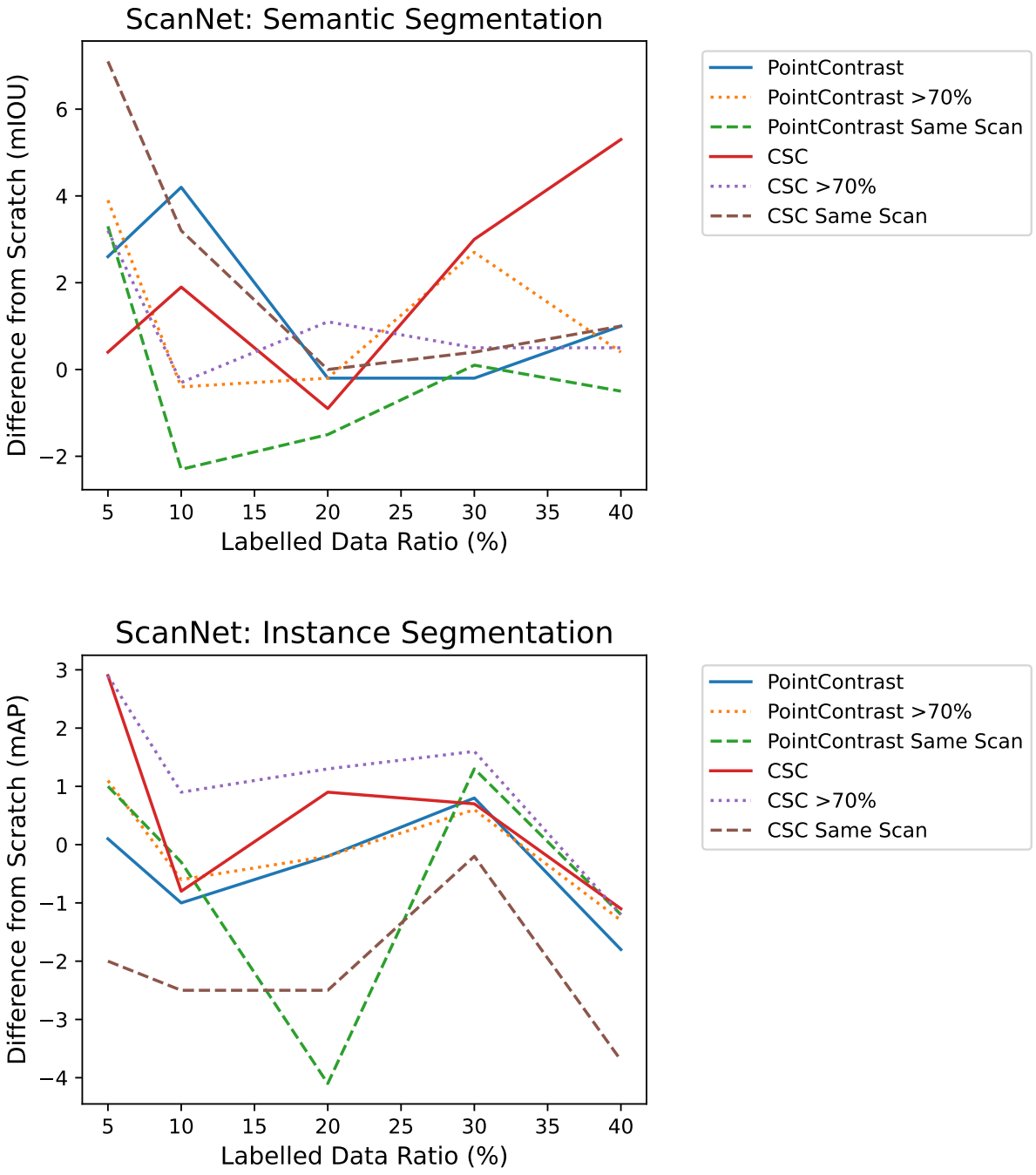


Figure 5.5: Effect of point cloud overlap on downstream performance. The performance is measured relative to training from scratch (with no pre-training). Positive differences indicate better performance.

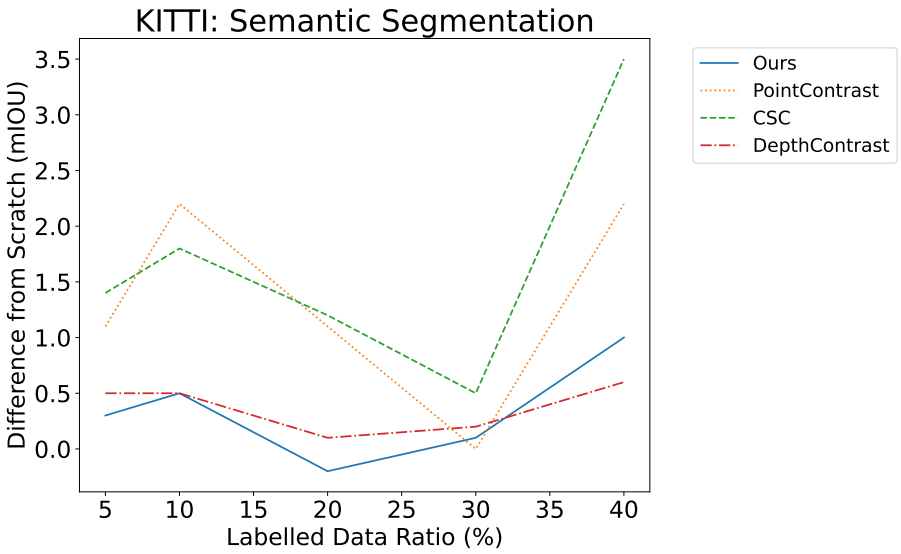


Figure 5.6: Performance on SemanticKITTI across varying labelled data proportions. The performance is measured relative to training from scratch (with no pre-training). Positive differences indicate better performance.

5.10 Pre-Training on Out-of-Distribution Data

We find that using a pre-training dataset that is out-of-distribution relative to the downstream dataset leads to more consistent and greater performance gains than using the same dataset for both pre-training and fine-tuning. Table 5.1 shows that using model weights pre-trained on ScanNet offers very little performance improvement when fine-tuned on ScanNet. However, those same weights result in much larger gains when applied to a different dataset such as S3DIS or SemanticKITTI. A similar discrepancy is found when varying the proportion of labelled data available. Using weights pre-trained and fine-tuned on ScanNet, we find that the biggest performance gain occurs when the proportion of labelled data available during fine-tuning is the most restricted (shown in Figure 5.4). The same weights have the opposite effect when fine-tuning on SemanticKITTI, which sees the largest performance boost as the proportion of labelled data increases, (shown in Figure 5.6). The discrepancy suggests that the relationship between the data available during pre-training and the data available during fine-tuning is a key contributing factor to the effectiveness of pre-training. A potential explanation is that using the same dataset already exposes the network to the same information, hence the backbone is not gaining as much new information as it could from a different dataset.

5.11 Inconsistencies in Performance Boost

In this section, we investigate the relationship between unlabelled and labelled data proportions on downstream performance, by limiting the available label amounts (details shown in Figure 5.4). While we expect the benefits of pre-training to increase during downstream supervised training in situations with fewer labels available, we find that the actual performance gains are inconsistent. For example, CSC, which achieves a 3% boost using 30% of the labelled data, has reduced downstream performance with 20% of the labelled data. Interestingly, DepthContrast, which is the least effective on S3DIS, is the most effective on ScanNet at 5% of the labelled data. Performances are also unpredictable across different tasks. Although most techniques have a positive impact on semantic segmentation, they have a mostly negative impact on instance segmentation, as shown in Figure 5.4. We also see in Section 5.9 that changing parameters such as the minimum overlap ratio between scans has a dramatic and inconsistent impact on downstream performance.

To demonstrate the inconsistency that pre-training has on downstream accuracy, we plot the distribution of semantic segmentation accuracy with varying amounts of labelled ScanNet data in Figure 5.7. The mean improvement of our method is higher than both PointContrast and DepthContrast. Our method also has a much smaller variance and greater lower bound on performance gain. Therefore, given a dataset with a limited amount of labelled data, our method is more likely to offer a consistent performance boost than other pre-training approaches.

Our results give no definitive answer as to which algorithm works best for a specific task or dataset. Since there is no way (at present) to accurately measure the quality of features produced from a pre-training algorithm, the only way to quantitatively evaluate features is to compare downstream performance. Downstream performance is highly variable and without knowing what makes a 'good feature' for these tasks, we are forced to proceed with development following a mostly trial-and-error methodology. Therefore, during algorithm development on a new dataset, experimentation may be required to see which method can improve downstream performance the most. However, our method may be a good first candidate given its relatively large performance gain, consistency across dataset sizes, and the simplicity gained from using individual scans.

5.12 Training Speed-up

Figure 5.8 shows the validation curves versus the training steps and compares training from scratch versus other methods of training. All techniques show an immediate improvement early on in the training cycle. This is potentially useful during development for downstream tasks: the final performance can be determined roughly within the first 2,000 cycles, instead of 20,000. For S3DIS for example, algorithms like PointContrast can reach a performance level that is within 1 mIOU of the final from-scratch performance in just 1,000 cycles. This early improvement observation should potentially drastically reduce training and development time.

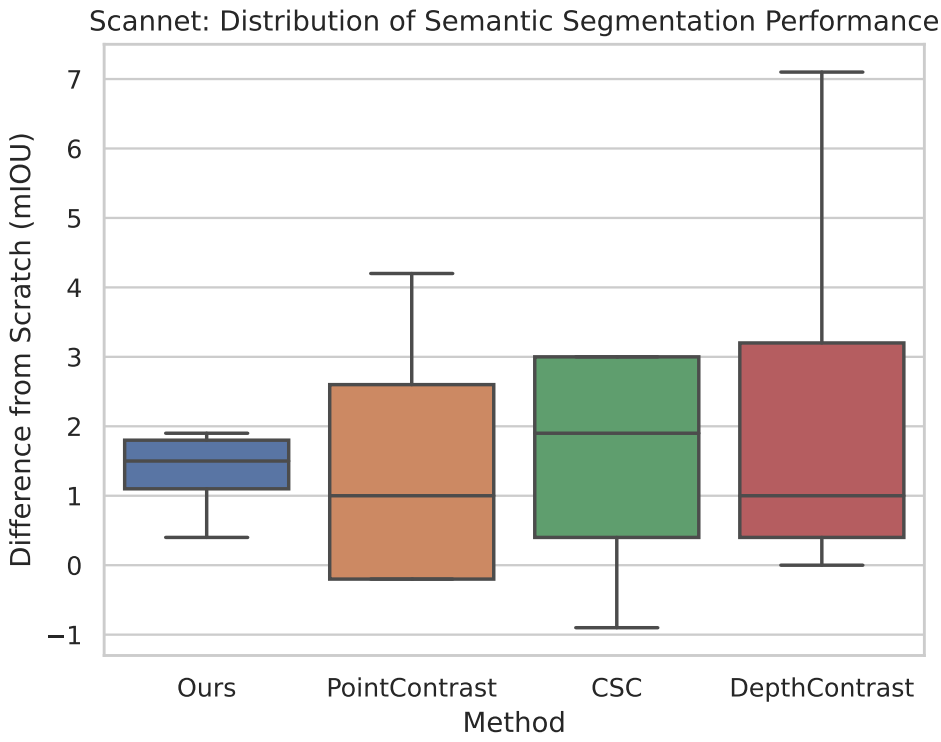


Figure 5.7: Distribution of performance improvements by pre-training methods. All differences are computed with respect to scratch performance. Measurements were taken on ScanNet semantic segmentation across labelled data ratios used in Figure 5.4.

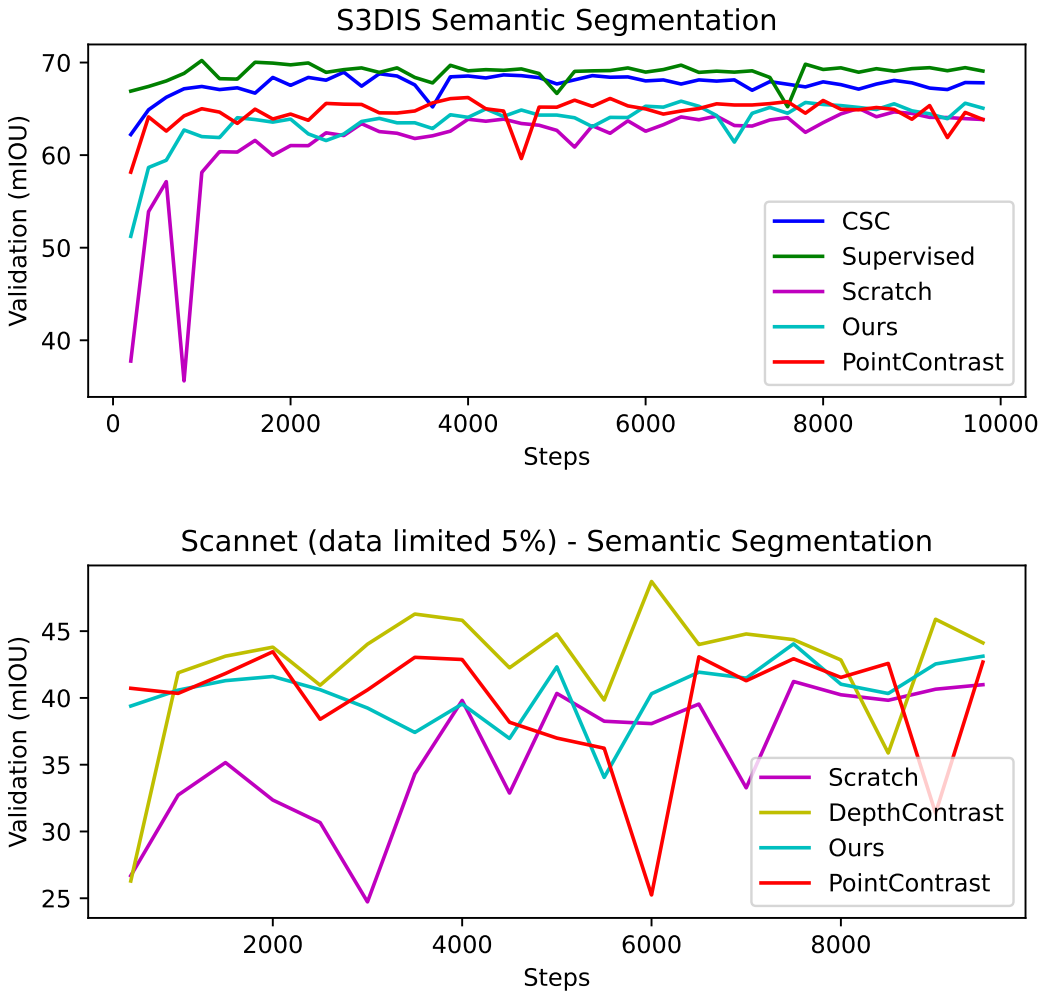


Figure 5.8: Comparison of validation performance over training steps. Models were pre-trained on ScanNet and fine-tuned for semantic segmentation on S3DIS (top) and ScanNet (bottom) with only 5% labelled data.

Chapter 6

Conclusions

In this thesis, we presented a method for transferring self-supervised features derived from dense images to models that operate on sparse point clouds. Point clouds are a vital modality for scene understanding tasks owing to their ability to represent scenes in 3D space. However, point clouds have many drawbacks, including being extremely sparse and diffiult to annotate. Alternatively, images offer a rich source of scene information. Images are often included with 3D datasets but are overlooked by many point cloud processing approaches. In addition, the difficulty of annotating point clouds can be alleviated through self-supervised feature learning, which leverages unlabelled raw data and is especially effective on image-related tasks. The density of image data along with the proven usefulness of image features led us in this work to explore how image features may be used to improve performance on point cloud-related tasks. To verify that performance gains are possible, we compared our method against three competing pre-training methods on three different downstream tasks and datasets.

6.1 Contributions

The core contribution of this thesis is a framework for learning dense pixel-level features from raw unlabelled images, which are then used as targets in a contrastive loss to pre-train a 3D model. We modified the InfoNCE loss function to work with multi-modal image and point cloud inputs, where a 3D model was trained from frozen 2D features. We visualized these features and found that the 2D features were successfully 'mimicked' by the 3D model. We then initialized our backbone model parameters using these features and fine-tuned the model for downstream tasks.

There were several key findings from our experiments. First, we found that the performance of existing methods was inconsistent relative to our own; specifically, our approach had a smaller variance in terms of accuracy when training on different amounts of labelled data from a given dataset. Second, we found—for all methods that we evaluated—that using the same data for pre-training and training yields no performance improvement; instead, larger amounts of data (or a different dataset entirely) are required for pre-training to improve downstream performance. Finally, we showed that incorporating visual data into the pre-training procedure is a viable strategy to reduce the requirement for registered point clouds as part of pre-training. By relaxing this requirement, our method scales more readily when compared to other pre-training techniques that require multiple (registered) scans for contrastive learning. With this improved scalability, more consistent performance across a number of downstream tasks, and the obviation of the need for registered point clouds, we believe that our approach is a reasonable starting point when choosing a pre-training strategy for 3D data.

6.2 Potential Improvements

The most challenging aspect of using an InfoNCE loss is selecting negative samples to contrast against a given query point. Usually, negative samples are selected randomly from all the input data points. Random sampling often selects points that either belong to the same class or contribute very little to feature learning. One option to choose more informative negative samples is to explore negative mining techniques, such as debiased sampling [10], hard negative sampling [45] and hard negative mixing [26]. These methods were originally developed and tested on images and have not been applied to point cloud data. Through initial experimentation, we found that these methods did not work well on point clouds and that further tuning and development is needed. One potential explanation for the lack of success is that the mining techniques have no knowledge of specific classes and so cannot prevent negative pairs from being drawn from the same class. A promising alternative is to use pseudo-labels generated from a network previously trained in a supervised manner, as is done by SupCon [27]. Alternatively, negative samples could be removed altogether by employing contrastive algorithms that rely on positive samples only (e.g., BYOL [17]).

Another improvement would be to experiment with datasets collected in different environments during pre-training. Perhaps the best example would be pre-training and training on outdoor autonomous driving datasets. Outdoor lidar-based datasets present a good use case for self-supervised learning since the lidar scans are usually much more sparse than indoor scans produced by stereo cameras.

6.3 Future work

Beyond the potential improvements discussed in the previous section, there exist other applications of self-supervised feature learning that could prove promising. We found that self-supervised features can improve performance on downstream supervised learning tasks, but that the improvements were minimal, highly inconsistent, and only useful when the datasets were extremely small. Instead, self-supervised features may prove much more useful in tasks where the objective is to learn with very few examples of each class. Specifically, self-supervised features can be useful for few-shot or zero-shot learning, where access to labels is limited to just a handful for each class. Good features for these applications are vital and may prove beneficial when large supervised datasets are unavailable for learning.

The self-supervised techniques could also be used more extensively in unsupervised learning for scene understanding tasks. For example, unsupervised learning can be used to detect similar objects (e.g., chairs) between frames [61]. In [61], the authors show that a self-supervised contrastive loss can be applied to train a network that automatically segments out objects without requiring any pixel-level annotations. The approach in [61] could be extended to operate on entire reconstructed point clouds by automatically identifying similar regions belonging to the same object type. Extending contrastive learning to unsupervised segmentation of entire scenes is non-trivial, however, but this should be possible in theory and could be very impactful.

Appendices

Appendix A

Augmentations

This appendix provides pseudocode for each algorithm applied to transform (augment) images and point clouds as part of our contrastive training process.

A.1 Image Transformations

Algorithm 4 Gaussian Blur

```
# sigma: variance of Gaussian kernel
# H, W: height and width of kernel
kernel = zeros(W, H)
for x,y in kernel.shape:
    kernel[x,y] = gaussian(x, y, sigma)
image = convolve(image, kernel)
return image
```

Algorithm 5 Horizontal Flip

```
# H, W: height and width of image
for i,j in image.shape:
    image[i,j] = image[i, W - j - 1]
return image
```

Algorithm 6 Vertical Flip

```
# H, W: height and width of image
for i,j in image.shape:
    image[i,j] = image[H - i - 1,j]
return image
```

Algorithm 7 Random Resized Crop

```
# Hi, Wi: height and width of input image
# Hf, Wf: height and width of desired output image
# scale_1, scale_2: range of cropped sizes
# ratio_x, ratio_y: range of aspect ratios
while w < Wi and h < Hi
    target_area = rand(scale_1, scale_2) * Wi * Hi
    log_ratio = (log(ratio_x), log(ratio_y))
    aspect_ratio = exp(rand(log_ratio))
    w = int(sqrt(target_area * aspect_ratio))
    h = int(sqrt(target_area / aspect_ratio))
i = rand(0, Wi - w)
j = rand(0, Hi - h)
image = image[j:j + h,i:i + w]
image = resize_with_interpolation(image, Hf, Wf)
return image
```

Algorithm 8 Centre Resized Crop

```
# Hi, Wi: height and width of input image
# Hf, Wf: height and width of desired output image
# ratio_x, ratio_y: range of aspect ratios
aspect_ratio = Hi / Wi
if aspect_ratio < min(ratio_x, ratio_y):
    w = Wi
    h = int(Hi / min(ratio_x, ratio_y))
else if aspect_ratio > max(ratio_x, ratio_y):
    w = int(Wi / max(r_x, r_y))
    h = Hi
else:
    w = Wi
    h = Hi
i = int((Wi - w) / 2)
j = int((Hi - h) / 2)
image = image[j:j + h,i:i + w]
image = resize_with_interpolation(image, Hf, Wf)
return image
```

A.2 Point Cloud Transformations

Algorithm 9 Random Dropout

```
indices = rand(length(point_cloud), dropout_ratio)
return point_cloud[indices]
```

Algorithm 10 Random Rotate

```
theta = zeros(3)
for i in range(3):
    theta[i] = rand(pi)
rotation = euler_angles_to_rotation_matrix(theta)
point_cloud = matrix_multiplication(rotation, point_cloud)
return point_cloud
```

Algorithm 11 Axis Flip

```
# axis: axis that is being flipped
axis_max = max(point_cloud[:, axis])
point_cloud[:, axis] = axis_max - point_cloud[:, axis]
return point_cloud
```

A.3 General Colour Transformations

Algorithm 12 Chromatic Translation

```
# colours: array of R G B colour elements
# ratio: ratio of translation
translation = (rand(rows=1, cols=3) - 0.5) * 255 * 2 * ratio
colours = colours + translation
colours = clip(colours, min_val=0, max_val=255)
return colours
```

Algorithm 13 Chromatic Auto-Contrast Blending

```
# colours: array of R G B colour elements
l = min(colours)
h = max(colours)
scale = (h - l) / 255
contrast = (colours - l) * scale
blend = rand()
colours = (1 - blend) * colours + blend * contrast
return colours
```

Algorithm 14 Chromatic Jitter

```
# colours: array of R G B colour elements
# std: standard deviation of noise to be added
noise = rand(rows = length(colours), cols=3)
noise = noise * std
colours = colours + noise
colours = clip(colours, min_val = 0, max_val = 255)
return colours
```

Bibliography

[1] Mohamed Afham, Isuru Dissanayake, Dinithi Dissanayake, Amaya Dharmasiri, Kanchana Thilakarathna, and Ranga Rodrigo. CrossPoint: Self-supervised cross-modal contrastive learning for 3D point cloud understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9902–9912, 2022.

[2] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3D semantic parsing of large-scale indoor spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1534–1543, 2016.

[3] Timothy D. Barfoot. *State Estimation for Robotics*, chapter 6: Primer on Three-Dimensional Geometry, pages 165–204. Cambridge University Press, 2017.

[4] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jürgen Gall. SemanticKITTI: A dataset for semantic scene understanding of lidar sequences. In *International Conference on Computer Vision (ICCV)*, pages 9296–9306, 2019.

[5] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9912–9924. Curran Associates, Inc., 2020.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 2020.

[7] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15745–15753, 2021.

[8] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D spatio-temporal convnets: Minkowski convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3070–3079, 2019.

[9] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully convolutional geometric features. In *International Conference on Computer Vision (ICCV)*, pages 8957–8965, 2019.

[10] Ching-Yao Chuang, Joshua D. Robinson, Yen-Chen Lin, Antonio Torralba, and Stefanie Jegelka. Debiased contrastive learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 8765–8775. Curran Associates, Inc., 2020.

[11] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2432–2443, 2017.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, volume 1, pages 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics.

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231. AAAI Press, 1996.

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[16] Clement Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. Digging into self-supervised monocular depth estimation. In *International Conference on Computer Vision (ICCV)*, pages 3827–3837, 2019.

[17] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap your own latent - A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 21271–21284. Curran Associates, Inc., 2020.

[18] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735, 2020.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *International Conference on Computer Vision (ICCV)*, 2017.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[21] Ji Hou, Benjamin Graham, Matthias Nießner, and Saining Xie. Exploring data-efficient 3D scene understanding with contrastive scene contexts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15582–15592, 2021.

[22] Ji Hou, Saining Xie, Benjamin Graham, Angela Dai, and Matthias Nießner. Pri3D: Can 3D priors help 2D representation learning? In *International Conference on Computer Vision (ICCV)*, pages 5673–5682, 2021.

[23] Allison Janoch, Sergey Karayev, Yangqing Jia, Jonathan T. Barron, Mario Fritz, Kate Saenko, and Trevor Darrell. A category-level 3-D object dataset: Putting the kinect to work. In *International Conference on Computer Vision (ICCV)*, pages 1168–1174, 2011.

[24] Li Jiang, Shaoshuai Shi, Zhuotao Tian, Xin Lai, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Guided point contrastive learning for semi-supervised point cloud semantic segmentation. In *International Conference on Computer Vision (ICCV)*, pages 6403–6412, 2021.

[25] Li Jiang, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. PointGroup: Dual-set point grouping for 3D instance segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4866–4875, 2020.

[26] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard negative mixing for contrastive learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 21798–21809. Curran Associates, Inc., 2020.

[27] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 18661–18673. Curran Associates, Inc., 2020.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25. Curran Associates, Inc., 2012.

[29] Phuc H. Le-Khac, Graham Healy, and Alan F. Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 8:193907–193934, 2020.

[30] Junnan Li, Pan Zhou, Caiming Xiong, and Steven C. H. Hoi. Prototypical contrastive learning of unsupervised representations. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021.

[31] Zhenyu Li, Zehui Chen, Ang Li, Liangji Fang, Qinhong Jiang, Xianming Liu, Junjun Jiang, Bolei Zhou, and Hang Zhao. SimIPU: Simple 2D image and 3D point cloud unsupervised pre-training for spatial-aware visual representations. *Association for the Advancement of Artificial Intelligence Conference (AAAI)*, 36(2):1500–1508, 2022.

[32] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[33] Yunze Liu, Li Yi, Shanghang Zhang, Qingnan Fan, Thomas Funkhouser, and Hao Dong. P4Contrast: Contrastive learning with pairs of point-pixel pairs for RGB-D scene understanding. arXiv:2012.13089v1 [cs.CV], 2020.

[34] Yueh-Cheng Liu, Yu-Kai Huang, Hung-Yueh Chiang, Hung-Ting Su, Zhe-Yu Liu, Chin-Tang Chen, Ching-Yu Tseng, and Winston H. Hsu. Learning from 2D: Contrastive pixel-to-point knowledge transfer for 3D pretraining. arXiv:2104.04687v3 [cs.CV], 2021.

[35] David G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[36] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[37] Yoshikuni Nomura, Li Zhang, and Shree K. Nayar. Scene collages and flexible camera arrays. In *Eurographics Symposium on Rendering (EGSR)*. The Eurographics Association, 2007.

[38] Lucas Nunes, Rodrigo Marcuzzi, Xieyuanli Chen, Jens Behley, and Cyrill Stachniss. SegContrast: 3D point cloud feature representation learning through self-supervised segment discrimination. *IEEE Robotics and Automation Letters (RA-L)*, 7(2):2116–2123, 2022.

[39] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. arXiv:1807.03748v2 [cs.LG], 2018.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035. Curran Associates, Inc., 2019.

[41] Trung Pham, Chaoning Zhang, Axi Niu, Kang Zhang, and Chang D. Yoo. On the pros and cons of momentum encoder in self-supervised visual representation learning. arXiv:2208.05744v1, 2022.

[42] Charles R. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas. Deep hough voting for 3D object detection in point clouds. In *International Conference on Computer Vision (ICCV)*, pages 9276–9285, 2019.

[43] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[44] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30. Curran Associates, Inc., 2017.

[45] Joshua D. Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In *International Conference on Learning Representations (ICLR)*, 2021.

[46] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241. Springer International Publishing, 2015.

[47] Corentin Sautier, Gilles Puy, Spyros Gidaris, Alexandre Boulch, Andrei Bursuc, and Renaud Marlet. Image-to-lidar self-supervised distillation for autonomous driving data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9891–9901, 2022.

[48] Jianbo Shi and Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[49] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference Computer Vision (ECCV)*, pages 746–760, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[50] Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao. SUNRGBD: A RGBD scene understanding benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 567–576, 2015.

[51] Richard Szeliski. *Computer Vision: Algorithms and Applications*, chapter 2: Image Formation, pages 51–60. Springer Cham, 2nd edition, 2022.

[52] Richard Szeliski. *Computer Vision: Algorithms and Applications*, chapter 3: Image Processing, pages 118–119. Springer Cham, 2nd edition, 2022.

[53] Richard Szeliski. *Computer Vision: Algorithms and Applications*, chapter 5: Deep Learning, pages 239–336. Springer Cham, 2nd edition, 2022.

[54] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[55] Wenguan Wang, Tianfei Zhou, Fisher Yu, Jifeng Dai, Ender Konukoglu, and Luc Van Gool. Exploring cross-image pixel contrast for semantic segmentation. In *International Conference on Computer Vision (ICCV)*, pages 7283–7293, 2021.

[56] Xinlong Wang, Rufeng Zhang, Chunhua Shen, Tao Kong, and Lei Li. Dense contrastive learning for self-supervised visual pre-training. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3023–3032, 2021.

[57] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3733–3742, 2018.

[58] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. SUN3D: A database of big spaces reconstructed using SfM and object labels. In *International Conference on Computer Vision (ICCV)*, pages 1625–1632, 2013.

[59] Saining Xie, Jiatao Gu, Demi Guo, Charles R. Qi, Leonidas J. Guibas, and Or Litany. PointContrast: Unsupervised pre-training for 3D point cloud understanding. In *European Conference Computer Vision (ECCV)*, pages 574–591, Cham, 2020. Springer International Publishing.

[60] Zhenda Xie, Yutong Lin, Zheng Zhang, Yue Cao, Stephen Lin, and Han Hu. Propagate yourself: Exploring pixel-level consistency for unsupervised visual representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16679–16688, 2021.

[61] Cheng-Kun Yang, Yung-Yu Chuang, and Yen-Yu Lin. Unsupervised point cloud object co-segmentation by co-contrastive learning and mutual attention sampling. In *International Conference on Computer Vision (ICCV)*, pages 7335–7344, 2021.

[62] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference Computer Vision (ECCV)*, pages 649–666. Springer International Publishing, 2016.

[63] Zaiwei Zhang, Rohit Girdhar, Armand Joulin, and Ishan Misra. Self-supervised pretraining of 3D features on any point-cloud. In *International Conference on Computer Vision (ICCV)*, pages 10232–10243, 2021.