Università degli Studi di Padova

Department of Information Engineering
University of Padova

UNIVERSITÀ DEGLI STUDI DI PADOVA

Department of Information Engineering

Master Degree in Computer Engineering

Exploiting Retentive Networks in 3D LiDAR Semantic Segmentation

Supervisor
Prof. Alberto Pretto

Co-Supervisor
Dr. Daniel Fusaro

Candidate
Simone Mosco

Academic Year 2022-2023
Graduation Date 14 December 2023

Ciao Nonno

Abstract

The advent of LiDAR technology has revolutionized the fields of autonomous driving, robotics, and environmental monitoring by providing precise 3D point cloud data. Semantic segmentation of LiDAR point clouds is an essential task for understanding the environment and facilitating intelligent decision-making in these applications. This Master's thesis introduces a cutting-edge approach, termed **RangeRet**, for 3D LiDAR Semantic Segmentation [13], which leverages the potential of range images to achieve real-time performance. It also exploits the Retentive Networks [35], a novel Natural Language Processing (NLP) architecture designed for Large Language Models. Through a comprehensive study of semantic segmentation in the context of 3D LiDAR data and state-of-the-art methods, the proposed approach introduces a lightweight network crafted to address key limitations in existing methods. It emphasizes efficiency, memory management, and real-time usage while achieving promising results. The implementation of the Retentive Networks in computer vision tasks serves as an alternative to the established Transformers [39] architecture, aiming to better capture geometric and spatial information in two-dimensional objects. The evaluation of the proposed method on benchmark datasets, such as SemanticKITTI, involves analyzing accuracy, efficiency, and generalization across diverse scenarios. In the final section, the thesis conducts an ablation study, systematically dissecting the proposed method by isolating and evaluating individual components. This process identifies the contribution of each architectural element, provides insights into the network's robustness, and highlights key factors influencing performance.

Abstract

L'introduzione della tecnologia LiDAR ha rivoluzionato i settori della guida autonoma, della robotica e del monitoraggio ambientale, fornendo nuovi dati di 3D point cloud. LiDAR Point Cloud Semantic Segmentation è un processo essenziale per comprendere l'ambiente e facilitare il decision-making in queste applicazioni. La tesi di laurea presenta un nuovo approccio, denominato **RangeRet**, nell'ambito di 3D LiDAR Semantic Segmentation [13], il quale sfrutta il potenziale delle range image per ottenere prestazioni in tempo reale. Impiega inoltre Retentive Networks [35], una nuova architettura nell'ambito del Natural Language Processing (NLP) progettata per Large Language Models. Attraverso uno studio approfondito nell'ambito della 3D Semantic Segmentation, in particolare con dati generati da sensori LiDAR e di approcci stato dell'arte, il metodo proposto introduce un lightweight network progettato per affrontare i limiti nei metodi esistenti. Evidenzia l'efficienza, la gestione della memoria e l'utilizzo in tempo reale ottenendo risultati interessanti. L'implementazione di Retentive Networks nell'ambito della computer vision rappresenta un'alternativa alla ben più nota architettura dei Transformers [39], con l'obiettivo di sfruttare al meglio le informazioni geometriche e spaziali negli oggetti bidimensionali. Il metodo proposto è stato testato con i più utlizzati dataset, come SemanticKITTI, analizzando i risultati e particolare attenzione ad accuratezza, efficienza e capacità di generalizzare su diversi scenari. Nella sezione finale, la tesi presenta un ablation study, analizzando ogni componente del metodo, isolandoli tra loro. Questo processo permette di identificare il contributo di ogni elemento della rete, fornendo maggiori informazioni sulle capacità della rete e mettendo in evidenza i module e le modifiche che influenzano maggiormente le prestazioni.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This Master thesis presents and describes **RangeRet**, a cutting-edge method for 3D LiDAR Semantic Segmentation, exploiting range view images and Retentive Networks [35], a novel architecture for Large Languale Models (LLMs).

RangeRet proposes the adoption of a pure Retentive Networks architecture for computer vision tasks, utilizing images instead of text, much like established Transformers [39] implementations such as the Vision Transformers [9] or Pyramid Vision Transformers [41] that have been previously experimented with in the field. The primary objective in employing range images lies in the pursuit of real-time performance, enabling the approach to operate effectively in real-world scenarios, facilitating a deeper understanding of the surrounding environment.

3D LiDAR Point Cloud Semantic Segmentation is a challenging task that has gained significant interest in recent years, primarily due to its applications in fields like autonomous driving and robotics. The core objective of Semantic Segmentation is to assign a class label to each individual point within the point cloud. However, the inherent sparsity and irregular structure of LiDAR point clouds have spurred the development of various approaches that tackle this task in diverse ways. The proposed method seeks to address this challenge by projecting point clouds onto a 2D plane, harnessing range view images to attain real-time performance. Nevertheless, this approach introduces the complexities of re-projection issues and potential information loss compared to methods that operate directly on the native point cloud data.

The described approach was significantly influenced by the State of The Art work of RangeFormer [21], which served as a primary source of inspiration. In addition, we followed a road-map that involved studying Transformers [39], Vision Transformers (ViT) [9] and Pyramid Vision Transformers (PVT) [41] to adapt Retentive Networks for vision tasks related to point clouds. These influences guided this approach, particularly in the decision to use 2D projections to achieve real-time performance and in strategies for handling the inherent sparsity and irregular structures within point cloud data. This integration of insights allowed to effectively address the challenges of 3D LiDAR Point Cloud Semantic Segmentation.

Through an in-depth examination of the design choices, experimental outcomes, and the implications of this method, this thesis aims to provide a holistic understanding of the capabilities and limitations of RangeRet, while providing every implementation choice and their correspondent justifications.

This thesis follows a structured approach to provide readers with a comprehensive understanding of 3D Semantic Segmentation. It begins with an introductory overview of the topic, including an exploration of the models and architectural concepts that have influenced the development of the method discussed. The subsequent sections delve into the model's structure and offer an in-depth analysis of its core components. The thesis

concludes by presenting a detailed account of the experiments conducted and the process of parameter tuning.

Chapter 2 introduces the central theme and the objective that RangeRet aims to tackle, which is LiDAR Point Cloud Semantic Segmentation. This chapter provides a thorough overview of Point Cloud Semantic Segmentation, and, most importantly, LiDAR Semantic Segmentation. It offers insights into the different approaches developed to address this task. Moreover it encapsulates the study carried out to develop the proposed method in a road-map style. The related architectures and methods that inspired the development of RangeRet are described and illustrated, along with their strengths and weaknesses.

In Chapter 3, a thorough exposition of the fundamental architecture employed in the discussed method will be provided, with meticulous elaboration on each component, including an in-depth discussion of the significant implementation decisions.

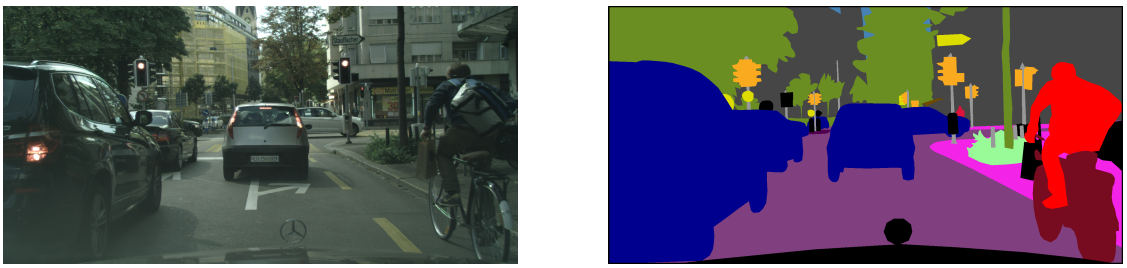
Chapter 4 serves as a comprehensive repository of information concerning the dataset, performance metrics, methodological outcomes, and a detailed comparison among the various technical implementations.

Concluding this research project, Chapter 5 presents an ablation study, delving into the critical components of the network to evaluate their individual contributions to the enhancements and overall performance.

1.1 Semantic Segmentation

Semantic segmentation is a fundamental task in computer vision and has a wide range of applications, including object detection, scene understanding, and autonomous navigation for robots and self-driving cars. It aids in understanding the spatial layout of objects within an image, making it a crucial step in interpreting visual data.

Its goal is to partition an image or scene into different regions based on the understanding of the objects and their categories, essentially providing a fine-grained analysis of visual data. More precisely, it involves assigning a semantic label to every pixel in an image or voxel in 3D volume, corresponding to what it represents. Since the predictions involve every pixel or voxel, semantic segmentation is referred to as a dense prediction task. It can also be thought as an image classification task at pixel/voxel level.



(a) Input RGB Image (b) Semantic Labels

Figure 1.1: Semantic Segmentation Sample from Cityscapes [8]

Consider as input an RGB color image of size $(H, W, 3)$ or a grayscale image of size $(H, W, 1)$, where H and W represent the height and width, respectively. The output of a segmentation task corresponds to a segmentation map of size $(H, W, 1)$ where each pixel contains a class label represented as an integer value.

However, when dealing with semantic segmentation, some challenges arise. Dense prediction tasks which involve pixel-level or voxel-level require high computational resources

and memory. Handling class imbalance is crucial, as some classes might be rare or underrepresented in the training data. Handling occlusions, fine-grained object boundaries, and variations in lighting conditions and viewpoint are also challenging aspects to deal with.

Recent advancements in deep learning, such as fully convolutional networks (FCNs) and more sophisticated architectures have greatly improved accuracy and efficiency of semantic segmentation in different fields. Moreover, transfer learning and pre-trained models also made easier to achieve good results on smaller datasets.

Semantic segmentation is applied in various real-world applications, including object recognition and tracking, autonomous driving for road understanding and identifying pedestrians or vehicles, medical image analysis, augmented reality and virtual reality for environment enhancement and many more. In this thesis, we will discuss about semantic segmentation applied to 3D point clouds, especially for LiDAR outdoor point cloud applied to autonomous driving field.

Point Cloud Semantic Segmentation is a computer vision task that involves assigning semantic labels to individual points in a 3D point cloud, a set of data points in a three-dimensional coordinate system. 3D LiDAR Semantic Segmentation is a specific application of semantic segmentation that involves the use of 3D LiDAR sensor data to classify and label individual points in 3D point clouds.

Chapter 2

Related Work

This Chapter describes the main task of 3D point cloud segmentation, introducing the topic of point cloud and LiDAR semantic segmentation. In particular, it will offer an overview of the different approaches when dealing with 3D data, starting from projection-based methods that exploit 2D networks, to discretization-based that use 3D grid and 3D convolution operator. Moreover, for each approach, some relevant methods for LiDAR semantic segmentation task will be briefly introduced and explained. In addition, we will present the approaches and methods that have not only influenced but also provided the foundation for the development of RangeRet. A road-map that starts with Transformers, the great architecture for Large Language Models in Natural Language Processing field and their implementations in computer vision tasks, Vision Transformers and Pyramid Vision Transformers. Then a brief overview of the State of the Art method RangeFormer for 3D semantic segmentation using range images, that inspired the development of the proposed method. Finally a description of Retentive Networks, a novel architecture, stated as the successor of Transformers and core of the proposed project.

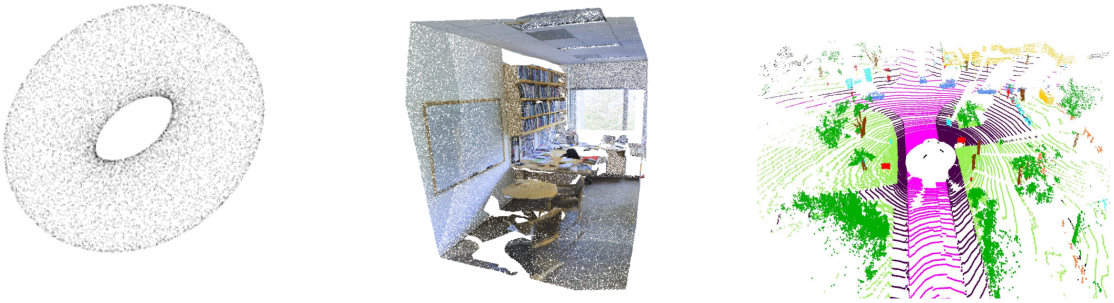
2.1 Point Cloud Semantic Segmentation

3D Point Cloud Semantic Segmentation is a specialized branch of computer vision that deals with the task of assigning semantic labels to individual points within a 3D point cloud. With respect to the 2D semantic segmentation task previously introduced, we employs 3D points in place of pixels and point clouds, i.e. a set of 3D points, instead of images. The task can be also interpreted as classifying point clouds into different subsets such that the points in the same isolated and meaningful region belong to the same class or object, exhibiting analogous properties. It enables the identification and classification of objects and their parts within a 3D scene, requiring the understanding of both global geometry structure and fine-grained details of each point.

A **Point cloud** is a collection of data points in a 3D space, where each point represent a specific coordinate in the space. Points are defined by their X, Y, Z coordinates and may also include additional attributes such as color, intensity or other attributes. Point clouds are commonly generated through various sensing technologies, with the most popular method being LiDAR (Light Detection and Ranging) but they can also be created using technologies like structured light.

Point cloud can be used in different ways, to represent a single 3D object, an indoor setting or outdoor environment as shown in Figure 2.1. This thesis will employ outdoor LiDAR point cloud, whose attributes correspond to the coordinates X, Y, Z and the remission or intensity property.

3D semantic segmentation is a challenging task because of high redundancy, uneven



(a) 3D object point cloud (b) Indoor point cloud (c) Outdoor point cloud

Figure 2.1: Types of point clouds

sampling density and lack of explicit structure of point cloud data. It is possible to precisely determine shape, size and other properties of objects in 3D data, however segmenting point clouds is not a trivial task. The point cloud data is usually noisy, inherently sparse, unstructured, the sampling density of points is uneven and the surface shape can be arbitrary with no statistical distribution pattern. Moreover, due to limitations in 3D sensors, the background is entangled with the foreground, making it challenging to achieve a clear separation between the two. Furthermore, achieving a deep learning model that combines computational efficiency with a small memory footprint for semantic segmentation is challenging, especially considering the requirements for real-world scenarios where resource constraints and real-time processing are crucial.

Point cloud semantic segmentation is of paramount importance in various domains, including autonomous navigation for environment understanding, robotics for object manipulation and human-robot interaction, urban planning, augmented reality, and environmental monitoring. It plays a pivotal role in enabling machines to comprehend their surroundings in 3D space.

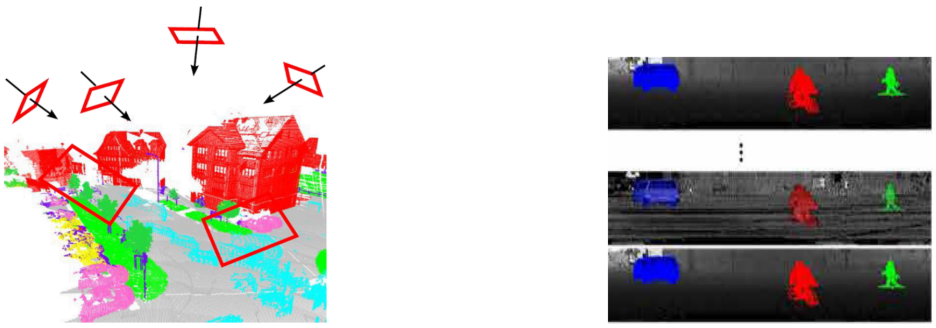
Recent advancements in 3D point cloud semantic segmentation have been driven by more powerful hardware and the development of specialized algorithms to cope with the previously mentioned difficulties. Additionally, the use of advanced sensor technologies such as LiDAR, has further enhanced the quality of point cloud data.

There exists four different paradigms [13] for semantic segmentation on point cloud: projection-based, discretization-based, point-based and hybrid methods. In the next sections, the main ideas of each approach will be described, along with an in-depth analysis of proposed method for 3D LiDAR semantic segmentation in the field of autonomous driving.

2.1.1 Projection-based Methods

Projection-based methods usually project a point cloud into 2D images such as multi-view and spherical images. This approach facilitates the application of existing deep learning models designed for 2D images to 3D point clouds. The intermediate segmentation results are then projected back to the raw point cloud.

Multi-view Representation enables the comprehensive analysis of three-dimensional scenes from multiple viewpoints, capturing a 3D scene often by simulating the perspective of multiple virtual cameras. This results in a collections of 2D images of views of the 3D scene. Therefore, the 3D point cloud is projected onto 2D images from different viewpoints, where each image view provides distinct spatial information. Then, an existing deep learning model can be applied to the 2D images in order to perform semantic



(a) Multi-View Representation (b) Spherical Representation

Figure 2.2: Projection-based Methods

segmentation and predict the class labels.

Multi-view representation captures a more complete view of the scene, reducing ambiguities and improving the accuracy of semantic segmentation. The approach is less sensitive to occlusions and variations in point cloud density, moreover it can adapt to different point cloud sizes, offering scalability for diverse applications. Overall, the performance is sensitive to viewpoint selection, especially when dealing with real-word data from different sensors, processing multiple views can be computationally intensive and finally, this methods does not fully exploit the geometric and structural information, introducing information loss due to the projection step.

Spherical Representation are created by projecting 3D point clouds onto a 2D surface, effectively mapping the 3D space onto a 2D spherical coordinate system. Compared to single view projection, spherical projection retains more information and is suitable for the labeling of LiDAR point cloud, since it provides a structured and regular representation of the 3D point cloud. Still the projection inevitably introduces several problem as information loss, discretization errors and occlusions. Furthermore, it may suffer from distortion effect[1], especially for point that are not evenly distributed in the 3D space, and it still needs to be accurately re-projected back, which is a critical challenge.

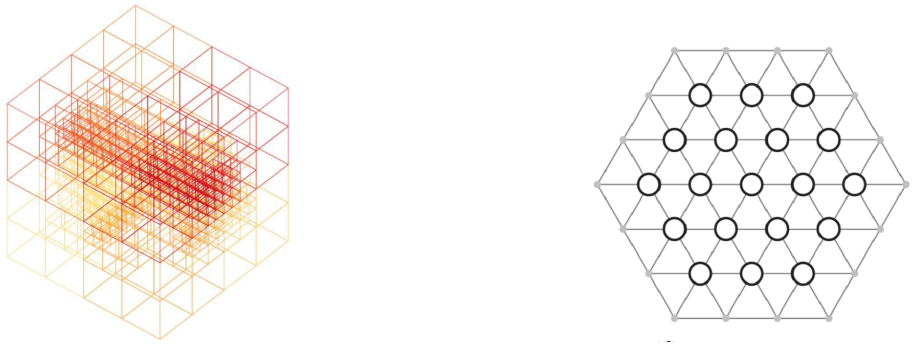
2.1.2 Voxelization-based Methods

Voxelization-based methods usually convert a point cloud into a dense/sparse discrete representation, such as volumetric and sparse permutohedral[2] lattices to facilitate the process of assigning semantic labels. The process of voxelization divides the 3D point cloud into cubic volumetric elements, known as voxels, to create a structured grid (Fig. 2.3). Features are then extracted from each voxel to capture information about the point within it and 3D convolutional networks can be used for processing the voxelized point cloud to perform semantic segmentation.

Dense Discretization Representation allows for the voxelization of the point clouds as dense grids and then leverage the standard 3D convolution operator. The process of voxelization involves using a high voxel resolution, resulting in small and densely packed voxels, providing a higher level of detail, allowing for precise segmentation and identification of fine-grained object boundaries. The structured grid-like format preserves structural information, as the neighborhood, within the point cloud, making it amenable to 3D deep learning models, which lead to a steady performance improvements in this area. However, the voxelization step introduces discretization artifacts and information

[1]The distortion effect can manifest as errors in the spatial representation of objects, leading to deviations from the true geometry

[2]Permutohedron of order n is an $(n-1)$-dimensional polytope embedded in an n-dimensional space



(a) Dense Discretization Representation (b) Sparse Discretization Representation

Figure 2.3: Discretization-based Methods

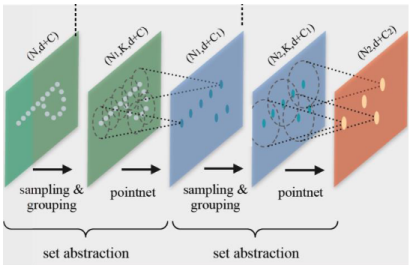
loss, depending on the voxel dimensions. A high resolution may results in empty or sparsely populated voxels while a lower resolution introduces loss of details. Furthermore, dense discretization representation can be computationally demanding, in particular when dealing with large and dense point cloud, also leading to high memory consumption. The grid resolution choice is a crucial step, but it is not trivial in practice.

Sparse Discretization Representation proposes to handle the naturally sparse volumetric representation, as the number of non-zero values only accounts for a small percentage. This approach transforms the continuous 3D point cloud into a discrete grid structure while managing the level of sparsity to optimize computational efficiency, since it is inefficient to apply dense convolutional neural networks on spatially-sparse data. Spatially Sparse Convolution operators [7] are applied to the voxelized point clouds optimizing computational efficiency by managing the level of sparsity, especially for resource-constrained applications. However finding the right balance between voxel resolution and computational efficiency can be a challenge, as overly sparse representation may lose important details. Therefore, this approach involves trade-offs between accuracy and computational performance.

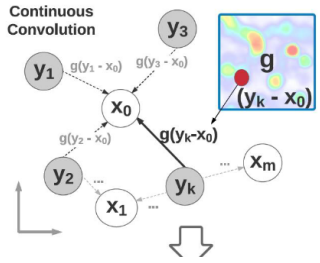
2.1.3 Point-based Methods

Point-based methods excel at handling irregular point clouds by directly processing individual points. They avoid the need for voxelization or discretization, making them well-suited for scenarios with unstructured or irregularly sampled 3D data. Features are extracted directly from each point, often analyzing the local context, considering neighboring points to improve accuracy. Point-based methods offer flexibility and adaptability, as they process directly the point cloud, without the overhead of creating structured grids, making them suitable for sparse point clouds. However the irregular distribution of points and varying point densities are a key challenge, as the memory management and computational efficiency when dealing with large point clouds. Based on the pioneering work PointNet [29], which employs shared MLPs and symmetrical pooling functions, many point-based methods have been proposed. These methods can be divided into point-wise MLP methods, point convolution methods, RNN-based methods and graph-based methods.

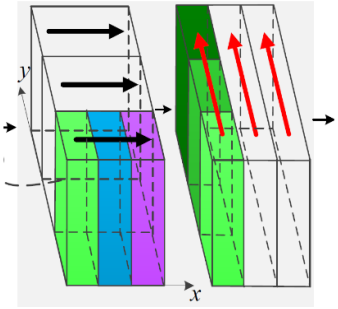
Pointwise MLP Methods use shared MLP as basic unit in their network for its high efficiency. However, point-wise features extracted by shared-MLP cannot capture local geometry in 3D point clouds and the interactions among points. To capture wider context for each point and learn richer local structures, several networks introduced mechanism based on neighboring feature pooling, attention-based aggregation and local-global feature



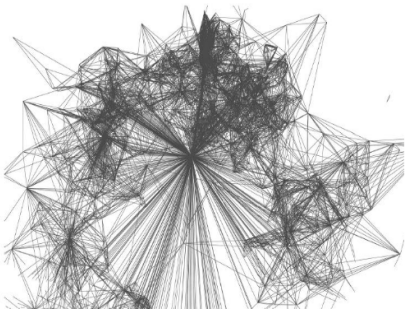
(a) Point-wise MLP Methods



(b) Point Convolutions Methods



(c) RNN-based Methods



(d) Graph-based Methods

Figure 2.4: Point-based Methods

concatenation.

To understand the local geometric patterns, neighboring feature pooling is involved to learn features for each point by aggregating the information from local neighboring points. PointNet++ [30] gather points hierarchically and gradually learns from larger local regions. To overcome non-uniformity and varying density of point clouds, multi-scale grouping and multi-resolution grouping are also proposed.

Attention-based aggregation approaches employs the Attention mechanism [39] to model the relation between points and better explore connections between group of points.

Local-global concatenation methods, propose to incorporate local structures and global context from point clouds, to capture both local information and scene-level global features.

Point Convolution Methods proposes effective convolution operators for 3D point clouds. Some approaches employed a point-wise convolution operator where neighboring points are binned into kernel cells and then convolved with kernel weights. KPConv [36] proposes a Kernel Point Fully Convolutional Network based on Kernel Point Convolution where the weights are determined by the Euclidean distances to kernel points and the number of kernel points is not fixed.

However, point convolution methods still suffer from the inherently irregular and unstructured properties of 3D point clouds.

RNN-based Methods exploits Recurrent Neural Networks (RNN) to better capture inherent context features from point clouds. Basic approaches convert the point cloud into grid blocks trough voxelization processes and then recreate a sequence structure to deal with recurrent networks. RSNet [18] employs a slice pooling layer to project features of unordered points onto an ordered sequence of feature vectors so that traditional RNNs can be applied.

Graph-based Methods represent a point cloud as a set of interconnected points, creating simple structures. These properties are later used to capture the structure and context information. Graph networks are also employed to learn embeddings for each 3D points, which can be used as features for other networks. [40] proposes Graph Attention

Convolution to selectively learn features from a local neighboring set, assigning attention weights to the points, based on their spatial position and feature differences.

However, graph-based methods may be computationally intensive and requires higher memory consumption, due to irregular graph structures, varying neighborhood connectivity, and the need for flexible message passing among nodes. Large parameter counts, and sparse operations contribute to higher memory consumption.

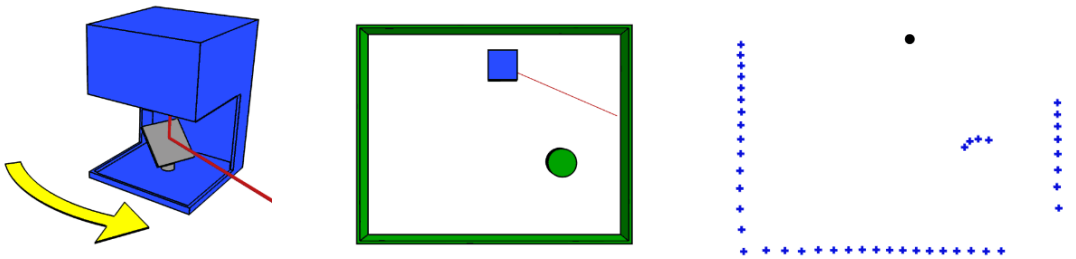
2.1.4 Hybrid Methods

Hybrid Methods learn multi-modal features from 3D scans or combine data from different sensors to further leverage all available information and enhance the understanding and classification of objects or scenes in three-dimensional space. Some methods propose to extract 2D and 3D features from two different networks, fuse together the learned feature and then use them to perform segmentation. Other methods incorporate data from camera or other sensors, such as RGB images or depth maps, to provide additional information that complements the point cloud data.

Combining data from multiple sensors or networks enhances the accuracy of semantic segmentation, as each source contributes to its unique strengths. The fusion of multi-modal data enables a more comprehensive understanding of the environment, improving the performance. However, combining data from different sources may requires additional computational resources, along with sophisticated fusion techniques and alignment of data, which can be challenging.

2.2 LiDAR Semantic Segmentation

LiDAR is a remote sensing technology that employs laser light to measure distances and create highly detailed, three-dimensional representation of the environment. LiDAR sensors emit laser pulses in the form of laser beams or laser light that are reflected back to the sensor when surfaces are encountered. They measure the time-of-flight, i.e. the round-trip-time of a pulsed light from the sensor to the object or surface and back, computing distances with high precision and accuracy. Many sensors have a 360-degree horizontal field of view, enabling them to capture the entire environment around them. However, they may differ in terms of their vertical field-of-view resolution, which refers to the number of beams, as well as the range at which they operate.



(a) Lidar sensor (b) Environment (c) Point Cloud

Figure 2.5: Lidar 2D Scan Sample

LiDAR Semantic Segmentation is a sub-task of Point Cloud Semantic Segmentation that involves point clouds generated by a LiDAR sensor, especially in the autonomous driving field, as shown in Figure 2.6. LiDAR point clouds usually include a set of points with four attributes (X, Y, Z, R) where X, Y, Z are the coordinates and R corresponds to the remission value, the reflectance or return strength. Point clouds generated by LiDAR

sensors, which consist of a large number of 3D points in unconstrained environment, results in some crucial challenges [22]. As previously introduced, point clouds are typically sparse, noisy, unstructured, irregular and exhibit varying density. Moreover LiDAR point clouds are commonly incomplete due to occlusion between objects, cluttered background and unsatisfactory material surface reflectivity. They may also be affected by changes in sensing and weather condition, having impacts on the appearances of objects.

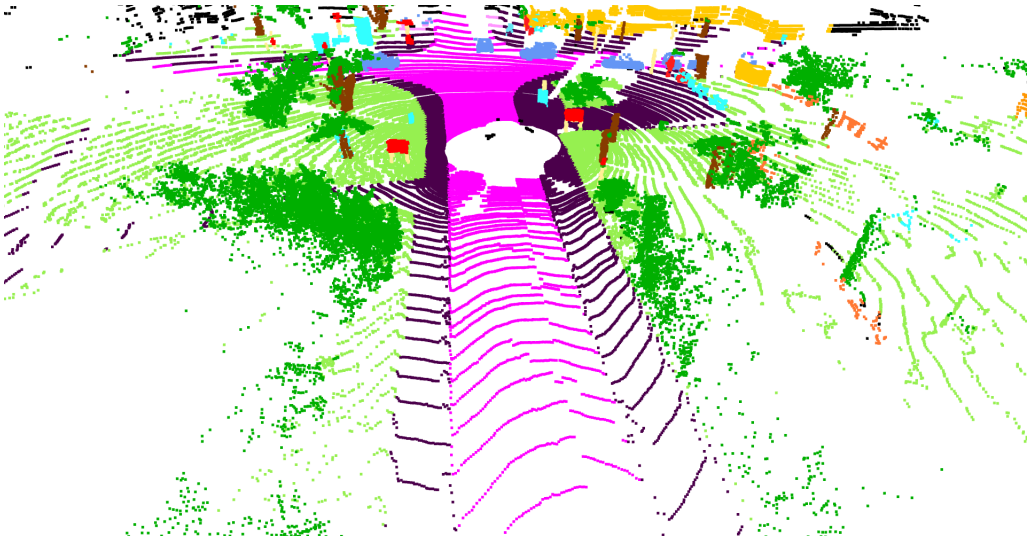


Figure 2.6: LiDAR Scan from SemanticKITTI [3]

These characteristics introduce numerous challenges, including data volume and computational demands, particularly in real-time applications. To cope with this issue, various approaches have been proposed, which can be divided in traditional and deep learning methods. Traditional models often uses handcrafted features to extract geometric information about point distribution and uses a classifier as Support Vector Machine (SVM) or Random Forest (RF) to output the labels. Deep learning based approaches can be divided in the same four groups presented before: point-based methods, projection-based models, voxel-based methods and other technique that combine more sensors of multi-modal feature learning.

Projection-based methods [19] project 3D LiDAR data onto a surface to generate 2D images. SqueezeSeg [42] introduces an encoder-decoder network that employs convolution, pooling, and de-convolution operators to make predictions on range images. The pipeline incorporates a Conditional Random Field (CRF) to refine the segmented points generated by the CNN. Subsequently, SqueezeSegV2 [43] enhances performance by introducing a Context Aggregation Module (CAM) to reduce sensitivity to dropout noise. This version also makes slight improvements to the model structure, training loss, batch normalization, and introduces an additional input channel.

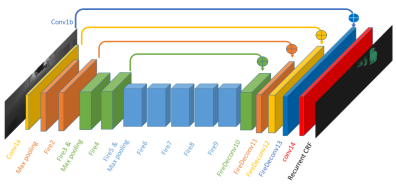


Figure 2.7: SqueezeSeg

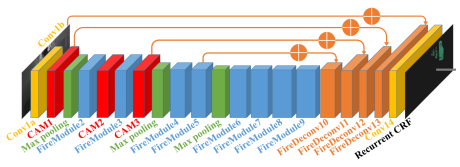


Figure 2.8: SqueezeSegV2

RangeNet++ [28] is one of the main works that uses spherical representation and proposes real-time semantic segmentation of LiDAR point clouds. It exploits range im-

ages as an intermediate representation in combination with a Convolutional Neural Network (CNN). To obtain accurate results, it proposes a novel post-processing GPU[3]-based k-Nearest-Neighbor (kNN) algorithm that deals with discretization errors and blurry CNN outputs, mitigating the reprojection error - an essential concern in these methods. RangeNet++ architecture, shown in Figure 2.9, employs a encoder-decoder structure with significant downsampling and residual connections.

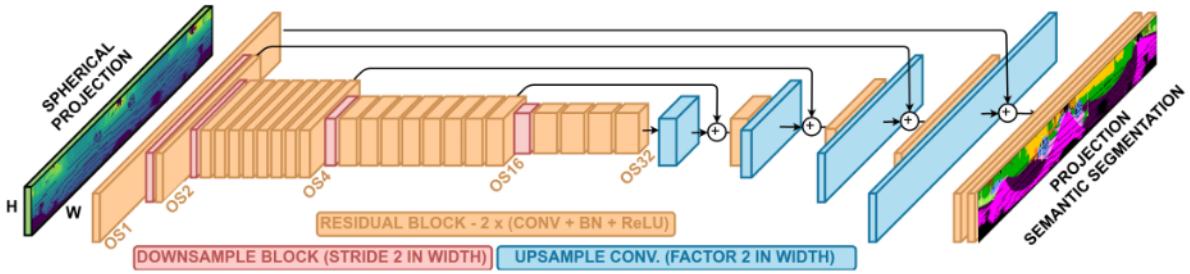


Figure 2.9: RangeNet++ Architecture

The point cloud reconstruction from range image is tackled in a simple, yet effective, loss-less way. RangeRet++ uses all the pixel pairs (u, v) for all the 3D points p_i obtained during the initial rendering process and indexes the range image with the image coordinates that correspond to each point.

CENet [6] presents a concise and efficient image-based network that integrates convolution with larger kernel size instead of MLPs used in PointNet for point feature learning. A backbone network is then employed for feature extraction along with auxiliary multiple semantic heads to obtain the final features, which will be later concatenated and fed to a classification head to produce the final predictions.

Voxel-based methods represent the whole point cloud as a grid of voxels to exploit the structured data representation.

Cylinder3D [50] proposes a dense discretization approach with sparse convolution operators, where cube voxel are substituted by cylindrical partition as shown in Figure 2.10.

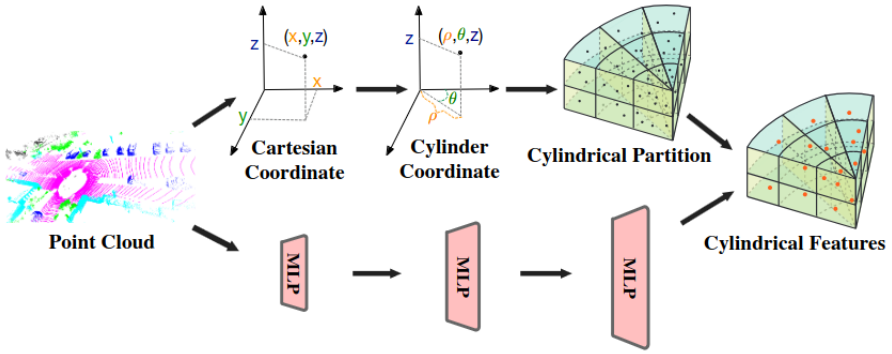


Figure 2.10: Cylindrical Partition

The LiDAR point cloud is divided into cylindrical partitions, transforming the points from Cartesian coordinate system to the Cylinder coordinate system, where a point is represented as (ρ, θ, z) with ρ radius and θ azimuth. Point-wise features are obtained from MLPs and assigned based on the partitions. The core of the architecture is an Asymmetrical 3D Convolution operator than strengthens the horizontal and vertical responses

[3]Graphics Processing Unit, a specialized electronic circuit designed to accelerate the processing of images and graphics

and better matches the distribution of object points. Since the predictions are assigned to each single cell, a point-wise refinement module is employed to exploit features before and after the 3D convolution networks, and fuse them together to refine the output.

Other methods propose multi-modal features from 3D scans or combine data from different sensors to further leverage all available information and enhance the semantic segmentation.

2DPASS [48] is a fusion-based approach that exploits camera and LiDAR sensor to conduct semantic segmentation through multi-modality data fusion. It employs two independently encoder-decoder networks to encode multi-scale features from 2D image and 3D point cloud (Fig. 2.11) and later process them to predict the class labels.

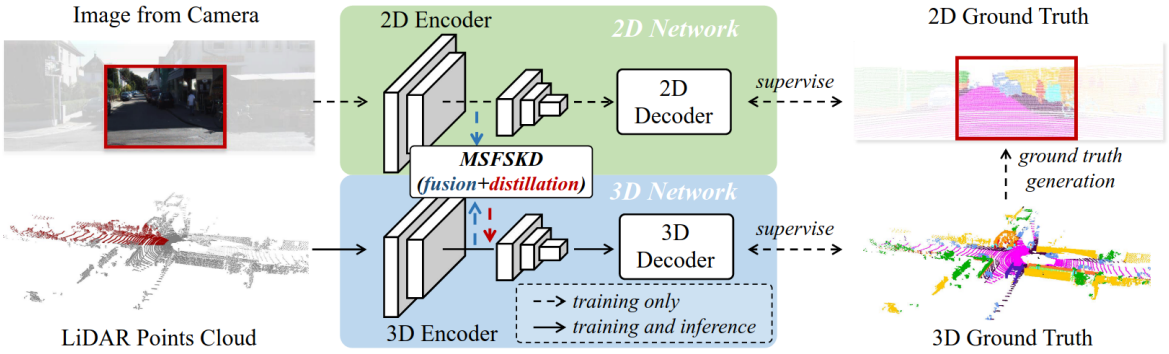


Figure 2.11: 2DPASS Architecture

In the training pipeline a Knowledge Distillation (MSFSKD) module is employed to leverage both 2D and 3D features to enhance the 3D network using texture and color-aware 2D priors as well as retaining the original 3D-specific knowledge. Simple MLPs are involved in the KD module to fuse features and learn both predictions. The point-to-pixel correspondence is performed using camera intrinsic and extrinsic matrices K, T, in order to associate each 3D point to its projection into the image plane. This approach demonstrates that the proposed knowledge distillation module is able to improve the performances of the 3D network without affecting execution time and memory consumption during inference.

PVKD (Point-to-Voxel Knowledge Distillation) [17] proposes a multi-modal feature approach with a teacher-student model to transfer the hidden knowledge from both point level and voxel level to the slim student network.

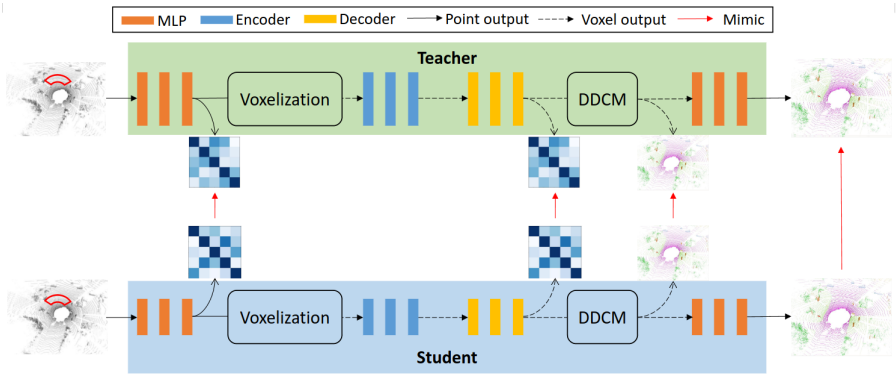


Figure 2.12: PVKD Architecture

PVKD first leverages both the point-wise and voxel-wise output distillation to complement the sparse supervision signals. Then, to exploit the structural information, the point cloud is divided into supervoxels along with a difficulty-aware sampling strategy to

better sample supervoxels containing less-frequent classes and furthest object. On these supervoxels, inter-point and inter-voxel affinity distillation is proposed to help the student model better capture structural information about the surrounding environment.

In conclusion, while LiDAR semantic segmentation holds great promise for enhancing autonomous driving systems, it faces significant challenges related to the data properties and limitations of the cited approaches. Furthermore, the data-hungry nature of these models, as discussed in [11], persists, as most of them require substantial amounts of data for training. The limited availability of large-scale LiDAR datasets for training and evaluation remains a critical hurdle that must be addressed to realize the full potential of this technology.

2.3 Transformers

Transformers [39] have become the standard in the Natural Language Processing (NLP) field, especially when dealing with Large Language Models (LLMs), due to their parallel computation and greater performance with long term dependencies. Transformers introduced a non sequential representation since sentences are processed as a whole, self-attention mechanism to compute similarity scores between words and positional embedding to replace recurrence.

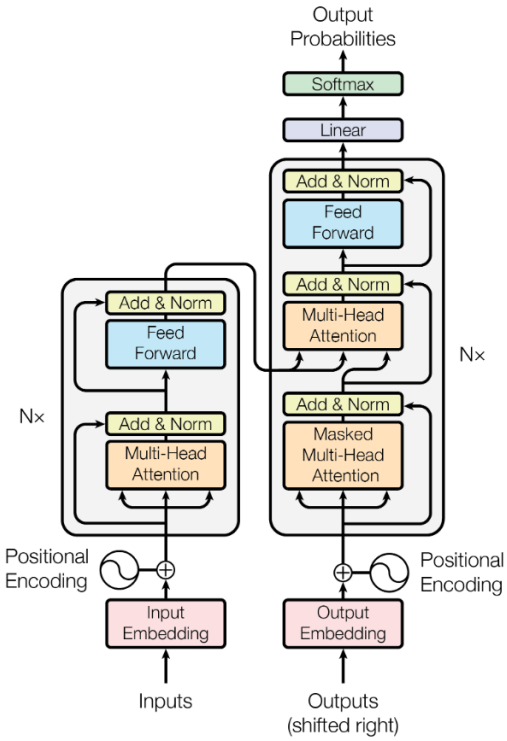


Figure 2.13: The Transformer architecture

The model architecture is composed of the Encoder and Decoder blocks where an input sequence $(x_1, ..., x_n)$ is mapped to a sequence of continuous representation $\mathbf{z} = (z_1, ..., z_n)$ by the Encoder and then to the output sequence $(y_1, ..., y_n)$ computed by the Decoder. Transformers are auto-regressive, implying that during the generation of each subsequent symbol, they incorporate information from previously generated symbols as additional input. In other words, the model's predictions for each step are influenced by the context of the symbols generated before it.

Each Encoder is composed by N identical layers and each layer is divided into a

Multi-Head Attention (MHA) mechanism and a Feed Forward Network (FFN), where the output of each sub-layer is enhanced by incorporating a residual connection [14] and Layer Normalization [2]. The output of each sub-layer is indeed $LayerNorm(x + Sublayer(x))$ where $Sublayer(s)$ is the function implemented by MHA or FFN. The Decoder is also composed of N identical layers, employing a third sub-layer which performs multi-head attention on the output of the Encoder stack. The self-attention sub-layer employs a masking that, in addition to the output embeddings shifted by one position, ensures that the output at position i depends only on known outputs at position less than i. The overall Transformers architecture is shown in Figure 2.13.

The Attention mechanism allows the model to understand connections between words in a sentence or other relevant words with respect to the current one. It maps a set of queries and key-values pairs to an output, where they are all vectors. Given d_k as query and key dimension and d_v as value dimension, they are all packed into matrices Q, K and V, to produce the output matrix as described in Equation 2.1.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

This Attention function is a dot-product (multiplicative) attention, except for the scaling factor $\frac{1}{\sqrt{d_k}}$ which avoids small gradients of the softmax function, for large values of d_k.

$$\begin{aligned} &MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \\ &\text{where } head_i = Attention(XW_i^Q, XW_i^K, XW_i^V) \end{aligned} \tag{2.2}$$

Transformers employ Multi-Head Attention mechanism which consists in a number h of heads, each with different, learned linear projections of queries, keys and values, that allow to perform the attention function in parallel, obtaining d_v-dimensional output values, where d_v is the value dimension. The outputs are then concatenated and projected into the final values. Projections are parameter/weight matrices W_i^Q, W_i^K, W_i^V, W_i^O.

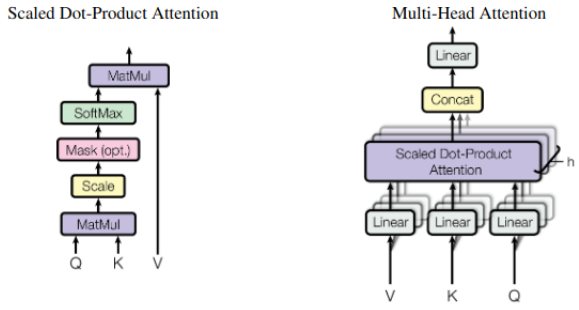


Figure 2.14: Attention Mechanism

The other sub-layer consists of a fully connected Feed-Forward Network with two linear transformations and a ReLU activation function.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.3}$$

The input tokens are converted into learned embeddings of dimension d_{model} and then *positional encodings* are added to inject some information about the relative or absolute

position of the token in the sequence, as shown in Figure 2.13. In particular, input and output tokens are converted into vectors using learned embeddings, while learned linear transformation and softmax are used to convert output to predicted next-token probability. Regarding the sequence order, since the model does not use recurrence or convolution, *positional encodings* of dimension d_{model} are added to the input embeddings at the bottom of the encoder and decoder stacks. In this work, sine and cosine functions of different frequencies are employed as follow:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.4}$$

where *pos* is the position and i the dimension. The choice of the sinusoidal version over learned position positional embeddings is due to being able to handle longer sequences than those used during training.

2.4 Vision Transformers

Vision Transformers (ViT) [9] are the first attempt to apply pure Transformers on computer vision tasks, showing that reliance on CNNs is not necessary. They achieved great results on image classification tasks requiring fewer computational resources to train.

ViT share the same Encoder architecture employed by Transformers, with a key change to make it suited for image processing, they represent an image as a sequence of patches, a small rectangular region of an image. In this way they obtain a parallelism with the NLP's text representation as sequence of words.

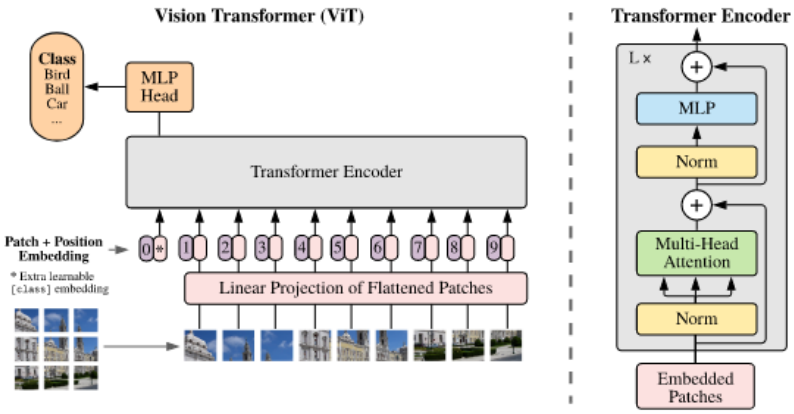


Figure 2.15: Vision Transformer architecture

Since the original Transformers accepts 1D input, images of dimension $(H \times W \times C)$ are reshaped into a sequence of flattened 2D patches of size $(Nx(P^2 \cdot C))$ where H, W and C are height, width and number of channels of original image, P is the patch size and $N = \frac{H \cdot W}{P^2}$ the number of resulting patches, corresponding to the input sequence length for Transformers. A trainable linear projection flattens each patch to a vector and maps them to length D. Finally standard learnable 1D position embeddings are added to the patch embeddings to produce the input for the encoder.

The Transformers is the same as presented in Section 2.3, composed of alternating layers of Multi-Head Self Attention (MSA) and Multi-Layer Perceptron (MLP) blocks. LayerNorm (LN) is applied before every block and residual connections after every block, as shown in Figure 2.15. MLPs contains two layers with a GeLU [15] activation function.

The output of the Transformer encoder is a sequence of vectors which represent the features of the image that can be used for vision applications such as image classification.

The operations performed by the Vision Transformers network are reported as follow:

$$
\begin{aligned}
z_0 &= [x_{class}; x_p^1 E; x_p^2 E; \cdots; x_p^N E] + E_{pos} \\
z_l' &= MSA(LN(z_{l-1})) + z_{l-1} \\
z_l &= MLP(LN(z_l')) + z_l' \\
y &= LN(z_l^0)
\end{aligned}
\tag{2.5}
$$

where E is the patch embedding projection, E_{pos} the positional embeddings, $LN(\cdot)$ the layer normalization and $l = 1...L$ correspond to the l layer.

Self-attention mechanism is useful in images, since it allows to learn long-range dependencies between the patches and how different regions of the image contributes to the overall label, in case of image classification or scene understanding. On the other hand, Vision Transformers are computationally expensive to train, due to the large number of parameters and they are not as interpretable as CNNs [49].

It is important to notice that Vision Transformers, when trained on mid-size datasets, obtain lower accuracy values with respect to SoTA methods such as ResNet [14]. However, when trained on larger datasets, and eventually pre-trained on other datasets, ViT approaches beat state of the art methods.

2.5 Pyramid Vision Transformers

Pyramid Vision Transformer (PVT) [41] is a powerful Transformer backbone, which achieves high output resolution for dense prediction tasks, employing a progressive shrinking pyramid to reduce computations and inherits advantages of both CNNs and Transfomers. PVTs introduce a useful architecture for dense/pixel-level prediction applications such as image segmentation, since the pyramid structure allows tho generate multi-scale feature maps.

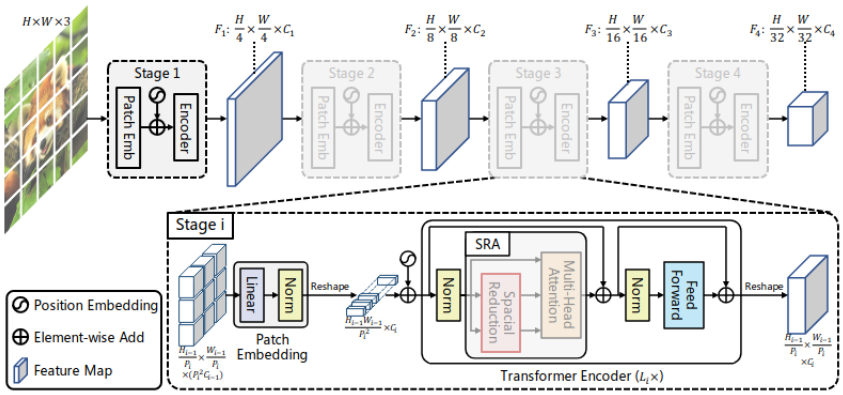


Figure 2.16: Pyramid Vision Transformer architecture

Pyramid Vision Transformer consists of four stages, all sharing a similar architecture of a patch embedding layer and L_i Transformer encoder layers. The output of each stage is reshaped to a feature map F_i of size $\frac{H}{s_i} \times \frac{W}{s_i} \times C_i$ where H and W are the image height and width, s_i is the stride of stage i and C_i the number of channels of stage i. Stride values for feature maps F_1, F_2, F_3, F_4 are respectively $[4, 8, 16, 32]$.

At the beginning of stage i, the input feature map $F_i \in \mathbb{R}^{H_{i-1} \times W_{i-1} \times C_{i-1}}$ is divided into $\frac{H_{i-1}W_{i-1}}{P_i^2}$ patches, where P_i denotes the patch size at i-th stage. Then each patch is flattened and projected to a C_i-dimensional embedding, producing embedded patches of size $\frac{H_{i-1}W_{i-1}}{P_i} \times \frac{W_{i-1}P_i}{\times} C_i$.

The Transformer Encoder at stage i has L_i encoder layer, each composed of an attention layer and a feed-forward layer, introducing a Spatial Reduction Attention (SRA) layer to replace the standard Multi-Head Attention (MHA) layer. SRA receives in input query Q, key K and value V as MHA, but it reduces the spatial space of K and V before attention function (Figure 2.16), saving computation and memory consumption. SRA formulation is reported in the equation below.

$$SRA(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.6}$$

$$head_j = Attention(XW_j^Q, SR(K)W_j^K, SR(V)W_j^V) \tag{2.7}$$

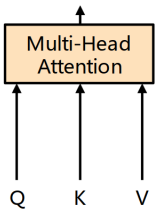
Where $SR(\cdot)$ is the operation for reducing the spatial dimension of K and V as:

$$SR(\boldsymbol{x}) = Norm(Reshape(\boldsymbol{x}, R_i)W^S) \tag{2.8}$$

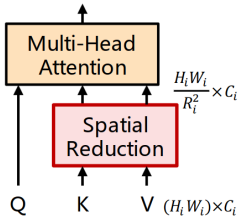
with x the input sequence and R_i the reduction ratio at stage i. The reshape operation transforms an input sequence x to size $\frac{H_iW_i}{R_i^2} \times (R_i^2 C_i)$, while the norm operation refers to Layer Normalization. The Attention function is calculated as in the original Transformer as:

$$Attention(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = softmax(\frac{\boldsymbol{q}\boldsymbol{k}^{\boldsymbol{t}}}{\sqrt{d_{head}}})\boldsymbol{v} \tag{2.9}$$

The computational and memory cost of attention operation is R_i^2 times lower than multi-head attention, so Spatial Reduction Attention can handle larger input feature maps.



(a) Multi-Head Attention



(b) Spatial Reduction Attention

With respect to Vision Transformer, whose output resolution is relatively low and makes it difficult to directly apply ViT to dense prediction tasks, Pyramid Vision Transformer is more flexible due to the progressive shrinking pyramid and more friendly to computation/memory consumption, while dealing with higher resolution feature maps or longer sequences.

2.6 RangeFormer

RangeFormer [21] is the range-view-based State of The Art (SoTA) approach for LiDAR Segmentation. It deals with range view images and Transformers blocks to overcome the issue with the projection of point cloud into 2D plane, while achieving great real-time performances.

RangeFormer converts the input point cloud into a range image of size $H \times W$, assigning to each pixel's features the coordinates, depth, intensity and existence (indicates

whether or not a grid is occupied by valid point) of the corresponding 3D point. The final range image is $R(u, v) \in \mathbb{R}^{(6,H,W)}$. RangeFormer's architecture is divided into three main blocks as shown in Figure 2.18: a Range Embedding Module (REM), a Pyramid structure Transformers blocks and a Semantic Head.

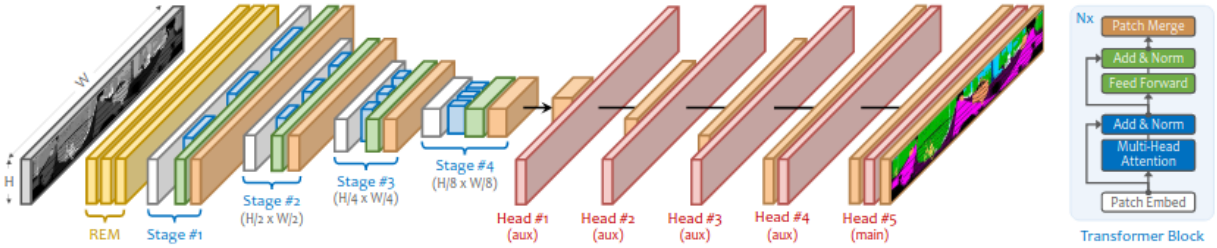


Figure 2.18: RangeFormer architecture

The Range Embedding Module (REM) consists of three MLP layers that map each point of the input range image into a higher-dimension embedding $F_0 \in \mathbb{R}^{(128,H,W)}$.

The core of RangeFormer is the 4-stage pyramid structure of Transformer blocks, similar to PVT [41], where at stage the corresponding down-sampling factors are 1, 2, 4 and 8. At the beginning of each stage, the input feature map is divided into 3×3 overlapping patches which are then fed into the Transformer blocks. Every stage incorporates a customized number of Transformer blocks where each block is divided into two modules as the original architecture: a Multi-Head Self-Attention (MHA) and a Feed-Forward Network (FFN). The MHA module serves as the main computing bottleneck and it is formulated as in the original implementation:

$$\begin{aligned} O = Mul(Q, K, V) &= Concat(head_1, ..., head_h)W^O \\ head_i &= Attention(XW_i^Q, XW_i^K, XW_i^V) \\ Attention &= \sigma(\frac{QK}{\sqrt{d^{head}}})V \end{aligned} \tag{2.10}$$

where $Attention(\cdot)$ denotes the self-attention operation, σ is the softmax function, d^{head} the dimension of each head and W^Q, W^K, W^V, W^O are the weight matrices for query Q, key K, values V and output O. As in PVT, a Spatial Reduction Layer (SRA) is implemented in order to reduce the sequence lengths of K and V by a factor R and save computation time. The feed-forward network is composed of two MLPs and GeLU activation function as follows:

$$F = FFN(O) = Linear(GeLU(Linear(O))) \oplus O \tag{2.11}$$

RangeFormer does not use explicit position embeddings but it incorporates them directly within the feature embeddings using a 3 by 3 convolution with zero padding in FFN, as introduced in SegFormer [47]. The previous equation becomes:

$$F = MLP(GeLU(Conv_{3\times3}(MLP(O)))) \oplus O \tag{2.12}$$

.

In particular, the depth-wise convolution is employed for reducing the number of parameters and improving efficiency.

The third block of the architecture involves a Semantic Head of MLPs that gradually map the output features from the four Transformer's stages to the semantic predictions.

Feature maps are retrieved from all the stages and their dimensions are unified through *Channel unification* and *Spatial unification*. Channel unification considers each feature map F_i with embedding size $d^{F_i}, i = 1, 2, 3, 4$ and via one MLP layer, it maps them to the same number of channels. Spatial unification consists of a resizing procedure of F_2, F_3, F_4 to the original size $H \times W$ by a simple bi-linear interpolation as follow:

$$H_i = Bi - Interpolate(Linear(F_i)) \tag{2.13}$$

Finally the four H_i are concatenated together and fed to two MLP layers, mapping the channel dimension to the number of classes d^cls, to create the probability distribution for the classes.

RangeFormer introduces four data augmentation techniques to help the model learn more general representation, increasing accuracy and robustness.

RangeMix mixes two scans along the inclination $\phi = \arctan(\frac{p_z}{\sqrt{p_x^2+p_y^2}})$ and azimuth θ directions, switching certain rows of two range images.

RangeUnion fills the empty grid of one range image with grids from another scan, using the pixel existence feature p^e that indicates void grid. Given a number of empty grids equals to $N_{union} = \sum_n p_n^e$, $k_{union} N_{union}$ candidate grids for point filling are randomly selected, with k_{union} set to 50%.

RangePaste copies rare classes from one scan to another scan at correspondent positions in the range view, to boost learning of classes with low distribution.

RangeShift slides a scan along the azimuth direction $\theta = \arctan(\frac{p_y}{p_x})$ to change the global position embedding. It corresponds to shifting the range image grids along the row direction with k_{shift}.

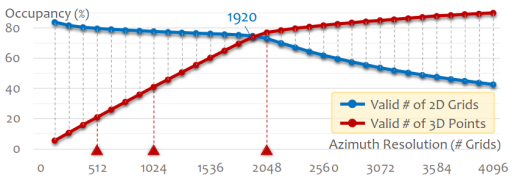


Figure 2.19: Occupancy trade-off

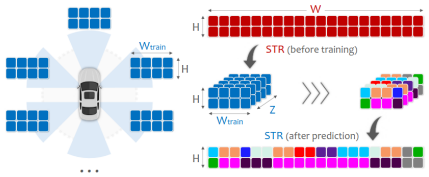


Figure 2.20: STR paradigm

RangePost is the new proposed approach for post-processing operation in order to tackle the re-projection from the range image to the point cloud, known as "many-to-one" conflict. Instead of using k-NN approach as in RangeNet++ [28], RangePost handle the task in a supervised manner. The point cloud is sub-sampled into equal-interval sub-clouds which are stacked together and fed to the network. The predictions are then stitched back to their original position. In this way the method is able to reduce the information loss caused by the "many-to-one" mappings.

RangeFormer introduces also a new training paradigm, due to the occupancy trade-off between (Fig. 2.19) the number of 3D points in the LiDAR scan and the number of non-empty grids in the range image (statistics computed on the SemanticKITTI dataset [3]). Scalable Training from Range View (STR) proposes a "divide-and-conquer" learning paradigm, where points are partitioned into multiple groups based on the azimuth angle of each point, $\theta_i = \arctan(\frac{p_i^y}{p_i^x})$. This will produce Z non-overlapping views of the complete point cloud, with Z hyper-parameter than determines the number of groups. Points from each group will be projected in the 2D plane separately to mitigate the "many-to-one" issue.

2.7 Retentive Networks

Since their appearance, Transformers became the de facto architecture in the Natural Language Processing field, especially for Large Language Models (LLMs), overcoming the training issues of recurrent models. However, their training parallelism causes an inefficient inference because of the $O(N)$ complexity per step, which does not allow Transformers to solve the so called *impossible triangle* in Figure 2.21.

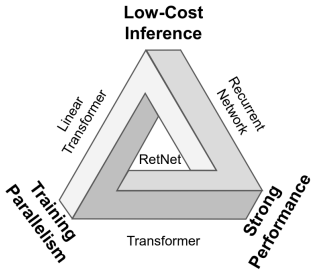


Figure 2.21: Impossible Triangle

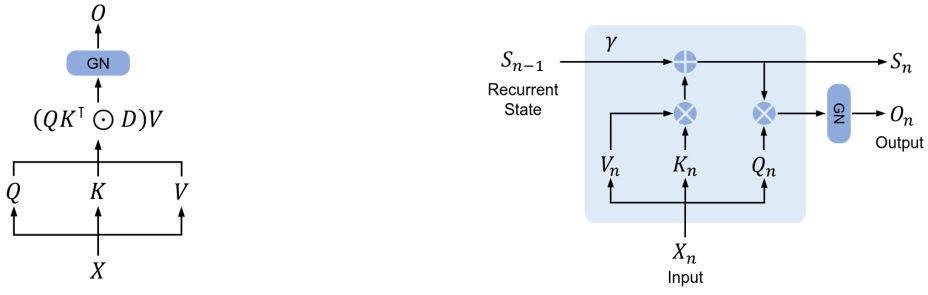
Retentive Network [35] is a novel architecture for LLMs which achieves training parallelism, low-cost inference and good performances, solving the *impossible triangle*. Table 2.1 shows the strengths of three major NLP models, with Retentive Network achieving best results in each field of the impossible triangle.

	Training Parallelism	Inference Cost	Memory Complexity	Performance
RNNs	✗	$O(1)$	$O(N)$	✗
Transformers	✓	$O(N)$	$O(N^2)$	✓
RetNet	✓	$O(1)$	$O(N)$	✓

Table 2.1: NLP architecture comparison

Retentive Network (RetNet) is composed of L identical blocks with a similar layout as Transformers, employing residual connections and layer normalization. Each block contains a multi-scale retention module (MSR) and a feed-forward network (FFN) module. The network's input is a sequence $x = x_1 \cdots x_{|x|}$, encoded and packed into $X^0 = [\boldsymbol{x_1} \cdots \boldsymbol{x_{|x|}}] \in \mathbb{R}^{|x| \times d_{model}}$, where d_{model} is the hidden dimension.

RetNet introduces the concept of **retention** in place of attention, which has a dual form of recurrence and parallelism as shown in Figure 2.22.



(a) Parallel representation

(b) Recurrent representation

Figure 2.22: Dual form of Retentive Network

The retention block is first introduced in a recurrent settings, i.e. processing each n-th input element individually. Given an input $X \in \mathbb{R}^{|x| \times d_{model}}$, it is projected to one-dimensional function $v_n = X_n \cdot w_V$. Considering a sequence modeling problem that maps $v_n \longrightarrow o_n$ through states s_n, we obtain the recurrent representation:

$$s_n = As_{n-1} + K_n^T v_n \qquad A \in \mathbb{R}^{d \times d}, K_n \in \mathbb{R}^{1 \times d} \tag{2.14}$$

$$o_n = Q_n s_n = \sum_{m=1}^{n} Q_n A^{n-m} K_m^T v_m, \qquad Q_n \in \mathbb{R}^{1 \times d} \tag{2.15}$$

where v_n is mapped to state vector s_n and then a linear transformation encodes sequence information recurrently. Equation 2.15 looks very similar to the original Transformers formulation, where the softmax has been replaced with the position embedding term A^{n-m} which we also refer to as *pos* matrix. The equation can be expanded into:

$$o_n = \sum_{m=1}^{n} (Q_n(\gamma e^{i\theta})^n)(K_m(\gamma e^{i\theta})^{-m})^T v_m \tag{2.16}$$

where $Q_n(\gamma e^{i\theta})^n$, $K_m(\gamma e^{i\theta})^{-m}$ is known as xPos [34], a relative position embedding proposed for Transformers. Simplifying γ as a scalar, we obtain a parallelizable formulation:

$$o_n = \sum_{m=1}^{n} \gamma^{n-m} (Q_n e^{in\theta})(K_m e^{im\theta})^{-T} v_m \tag{2.17}$$

From the above equation, the parallel representation of retention is constructed as shown in Figure 2.22a, using queries Q, keys K and values V matrices. The retention layer is defined as:

$$\begin{gathered} Q = (XW_Q) \odot \Theta, \quad K = (XW_K) \odot \bar{\Theta}, \quad V = XW_V \\ \Theta_n = e^{in\theta}, \quad D_{nm} = \begin{cases} \gamma^{n-m}, & n \geq m \\ 0, & n < m \end{cases} \\ Retention(X) = (QK^T \odot D)V \end{gathered} \tag{2.18}$$

where $\bar{\Theta}$ is the complex conjugate of Θ and $D \in \mathbb{R}^{|x| \times |x|}$ combines casual masking and exponential decay along relative distance as one matrix. The retention formulation in Equation 2.18 presents some small but crucial differences with respect to the attention operation (Eq 2.1). The first step of computing Q, K and V is the same except for the *pos* embedding which is multiplied element-wise to Q and K. The softmax function is replaced with a D matrix of relative positional embedding and casual mask representation, which can be pre-computed.

Regarding positional encoding, Θ in Eq. 2.18 encodes relative positional information into vectors of Q and K matrices via vector rotation, making them *position aware* through an Hadamard product. The rotation vectors are extracted from Euler's formula (Fig. 2.23) $e^{i\phi} = \cos\phi + i\sin\phi$ and Q_n and K_m vectors are rotated at each position by the rotation vectors. The dot product between vectors with the same rotation will produce 1 while as we move further, the dot products tend to 0 as the vectors are almost orthogonal to the others.

The D matrix in Eq. 2.18 represents casual mask and exponential decay weighting scheme, replacing Transformer's masked attention and softmax. When the ordered vectors

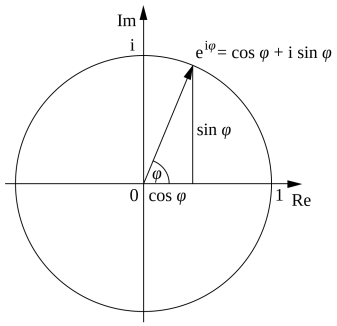


Figure 2.23: Euler's Formula

are in the past $n < m$, an exponential decay factor is applied using γ, while for future vectors $n > m$ the weight is set to 0. The diagonal of the matrix D is filled with only ones since, for the same value of n and m, we have $\gamma^{n-m} = \gamma^0 = 1$, while the upper triangle contains only zeros.

$$\begin{bmatrix} 1 & 0 & 0 & \cdot & 0 & 0 \\ \gamma & 1 & 0 & \cdot & 0 & 0 \\ \gamma^2 & \gamma & 1 & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \gamma^{n-2} & \gamma^{n-3} & \gamma^{n-4} & \cdot & 1 & 0 \\ \gamma^{n-1} & \gamma^{n-2} & \gamma^{n-3} & \cdot & \gamma & 1 \end{bmatrix} \tag{2.19}$$

Regarding the recurrent representation in Figure 2.22b, RetNet mechanism can be written as recurrent neural networks (RNNs), which is suitable for inference. Considering the n-th timestamp, the output is obtained as:

$$\begin{aligned} S_n &= \gamma S_{n-1} + K_n^T V_n \\ Retention(X_n) &= Q_n S_n, \qquad n = 1, \cdots, |x| \end{aligned} \tag{2.20}$$

where Q, K, V, γ are the same of Equation 2.18.

RetNet introduces also a third form, the chunkwise recurrent representation, a hybrid form of parallel representation and recurrent representation, to accelerate training for long sequences dividing the input into chunks.

As introduced at the beginning of the section, Retentive Network employs a Multi-Scale Retention, similar to the Multi-Head Attention of Transformers, using $h = \frac{d_{model}}{d}$ retention heads in each layers, with d as head dimension. Each head uses different parameters matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$ and different γ. Given input X, the MSR layer is defined as:

$$\begin{aligned} \gamma &= 1 - 2^{-5-arange(0,h)} \in \mathbb{R}^h \\ head_i &= Retention(X, \gamma_i) \\ Y &= GroupNorm_h(Concat(head_1, \cdots, head_h)) \\ MSR(X) &= (swish(XW_G) \odot Y)W_O \end{aligned} \tag{2.21}$$

where $W_G, W_O \in \mathbb{R}^{d_{model} \times d_{model}}$ are learnable parameters, GroupNorm [44] normalizes the heads' outputs and swish-gate [31], which introduces non-linearity, is defined as:

$$f(x) = x \cdot \sigma(x) \tag{2.22}$$

where $\sigma(x) = (1 + exp(-x))^{-1}$ is the sigmoid function.

Three normalization factors are implemented in Equation 2.18, as QK^T is normalized to $\frac{QK^T}{\sqrt{d}}$, D is replaced with $\tilde{D}_{nm} = \frac{D_{nm}}{\sqrt{\sum_{i=1}^{n} D_{ni}}}$, and finally the retention score $R = QK^T \odot D$ is normalized as $\tilde{R}_{nm} = \frac{R_{nm}}{max(|\sum_{i=1}^{n} R_{ni}|, 1)}$ so the output becomes $Retention(X) = \tilde{R}V$. These normalization techniques do not affect the final results while improving the numerical flow stability of forward and backward phases.

The overall architecture of Retentive Network is composed of L layers, each with a multi-scale retention (MSR) and a feed-forward network module. Given an input sequence transformed into embedded vectors and packed into a matrix, the output of a layer l is the following:

$$\begin{aligned} Y^l &= MSR(LN(X^l)) + X^l \\ X^{l+1} &= FFN(LN(Y^l)) + Y^l \end{aligned} \tag{2.23}$$

where $LN(\cdot)$ is LayerNorm and the feed-forward network is computed as $FFN(X) = GeLU(XW_1)W_2$ with W_1, W_2 parameter matrices.

Chapter 3

Method

In this Chapter, the overall architecture of RangeRet will be presented and extensively described. Starting from the conversion of the raw point cloud into a range view representation, moving on to the key parts of the model, as the application of Retentive Networks in computer vision tasks, ending on post-processing techniques to deal with re-projection errors and information loss.

3.1 Pre-processing

RangeRet tackles the 3D Semantic Segmentation task, not exploiting directly the point cloud, but working on its range view representation in 2-dimensions to achieve fast computing time at the slight expense of accuracy.

For a given LiDAR scan, i.e. a point cloud, the points are rasterized into a **range image**, a 2D cylindrical projection $R(u, v)$ of size $H \times W$ where H is the height and W the width. In Equation 3.1 is described the process to project a 3D point $p_n = (x_n, y_n, z_n)$ into the correspondent grid coordinate (u_n, v_n), where $d = \sqrt{x_n^2 + y_n^2 + z_n^2}$ is the distance, or depth, between the point and the center of LiDAR sensor, F is the vertical filed-of-views computed as $F = |f^{up}| + |f^{down}|$.

$$\begin{pmatrix} u_n \\ v_n \end{pmatrix} = \begin{pmatrix} \frac{1}{2}[1 - \arctan(y_n, x_n)\pi^{-1}]W \\ [1 - (\arcsin(z_n, d^{-1}) + f^{down})F^{-1}]H \end{pmatrix} \tag{3.1}$$

The **range image** $R(u, v) \in \mathbb{R}^{(5,H,W)}$ that will be fed to the network is composed of five rasterized features, coordinates (x, y, z), depth d and remission or intensity r. The same process is applied to the semantic labels, obtaining $L(u, v) \in R^{(H,W)}$.

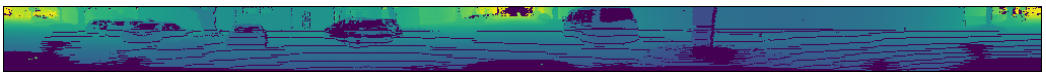


Figure 3.1: Range View Image

Figure 3.2: Range View Labels Image

Usually the height H in Eq. 3.1 is determined by the sensor typology, in particular the beam number while the width W can be chosen as desired. Most used width values are $[512, 1024, 2048]$, even though RangeFormer [21] shown that 1920 corresponds to the best trade-off value in term of grid occupancy, valid number of 2D grids and valid number

of 3D points. Consider a point cloud from the SemanticKITTI [3] dataset which contains about 120k points, choosing a range image of size $(H, W) = (64, 512)$ will results in $1 - \frac{64 \times 512}{120000} \approx 0.73\%$ information loss. Increasing the width W to 1024 would reduce the information loss to 45%, while a width of 1920 would result in a pixel count similar to the number of points in the point cloud.

Therefore, a good choice of the horizontal angular resolution may reduce issues with the model training such as empty grids, shape deformation and semantic incoherence. In the three range images below, it is possible to notice the differences with respect to the width value.

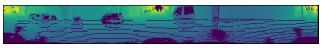


Figure 3.3: Range Image (64×512)

Figure 3.4: Range Image (64×1024)

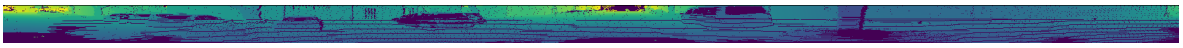


Figure 3.5: Range Image (64×2048)

3.2 Framework

RangeRet proposes an Encoder-Decoder architecture, which can be divided in three main parts:

- **Range View Embedding Module**, divided in Range Embedding Module (REM) and Vision Embedding Module (VEM), where each pixel of the input range view representation is mapped to a higher feature dimension. Patches are then extracted and flattened to be fed into the Retentive Network architecture.
- **Retentive Network**, a set of custom RetNet [35] blocks serves as the core component of the model architecture. These blocks have been adapted with specific modifications to address vision tasks, particularly for image processing.
- **Semantic Head**, reshape the features learned by Retentive Network into a two-dimensional representation and then a set of Multi-Layer Perceptrons (MLPs) map each range image pixels to the class predicitons.

A main residual connection [14] is employed between the output layer of the Range Embedding Module and the input layer of Semantic Head module. The whole RangeRet architecture is shown in Figure 3.6.

3.2.1 Range View Embedding

The Range View Embedding module contains two different sub-modules, a Range Embedding Module (REM) as in RangeFormer [21] and a Vision Embedding Module (VEM).

Range Embedding Module is responsible for mapping each point in the input range image with a number C of features $R(u, v) \in \mathbb{R}^{(C,H,W)}$ to a higher-dimension embedding

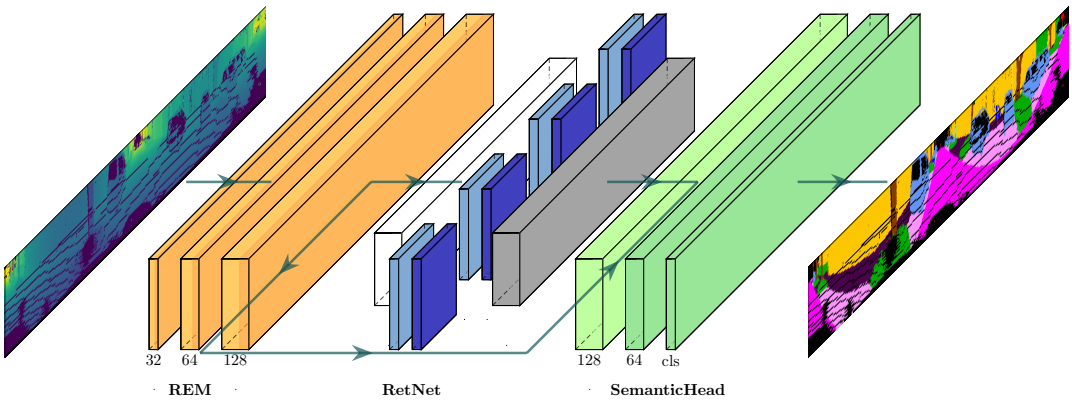


Figure 3.6: RangeRet Architecture

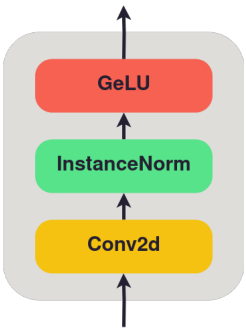


Figure 3.7: Single BasicConv2d layer

$emb \in \mathbb{R}^{(F,H,W)}$ through three basic convolution layers (Fig. 3.7), as suggested in [46], with F the final number of feature in the channel dimension.

Each basic convolution layer consists of a 2D convolution operator with kernel size 3×3, stride 1, padding 1 and dilation 1, InstanceNorm [37] and GeLU [15] activation function. The formulation of a single basic convolution layer is the following:

$$O = GeLU(InstanceNorm(Conv2D(X))) \tag{3.2}$$

where $X \in \mathbb{R}^{(C,H,W)}$ and $O \in \mathbb{R}^{(F,H,W)}$ are respectively the input and output of the layer. Range view representation provided as input has a channel dimension $C = 5$ that will be mapped to $F = 128$, with intermediate features as follows: $(5 - 32), (32 - 64), (64 - 128)$. However, hidden and final channel dimensions can be set as desired at different values. We found 128 a valid value to generate a valuable feature map as input to the Retentive Network architecture. Both implementation choices are based on the experiments conducted.

GeLU activation function has the following formula:

$$\begin{aligned} GeLU(x) &= x \cdot \sigma(x) \\ \text{where} \quad \sigma(x) &= \frac{1}{1 + \exp(-x)} \end{aligned} \tag{3.3}$$

where x is the input of the activation function and $\sigma(x)$ represents the sigmoid function.

InstanceNorm is a special normalization operator, similar to LayerNorm [2], except that it is applied to each channel of the channeled data as RGB images while LayerNorm

is usually applied on the entire sample and in NLP tasks. Additionally, InstanceNorm usually does not apply affine transform. It is described as:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta \tag{3.4}$$

where mean $E[x]$ and standard-deviation $Var[x]$ are calculated per-dimension separately for each object and γ, β are learnable parameter vectors of size C as the channels.

The whole architecture of the Range Embedding Module is shown below, in Figure 3.8.

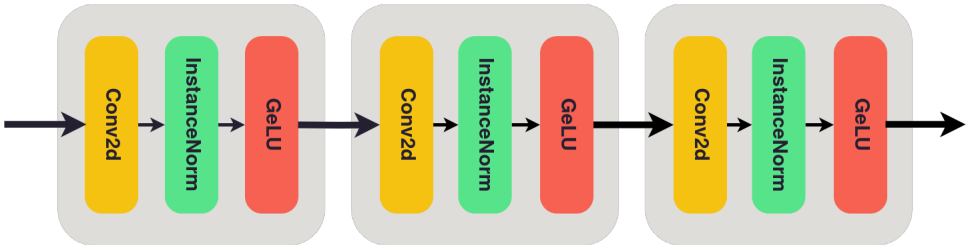


Figure 3.8: Range Embedding Module

Vision Embedding Module operates on the output of the Range Embedding Module, it extracts a number of patches of given dimension (P, P) and then flattens them to generate the inputs for Retentive Network, core of the architecture. Given an image of size (B, C, H, W) with B batch size, C number of channels, H and W the height and width of the image, VEM produces a number N of flattened patches with F features, i.e. the output of the module has size (B, N, F) where B is still the batch size. RangeRet operates on one image at a time, so the batch size dimension B will be always 1, which, according to experimental findings, contributes to mitigating the overfitting issue.

The patches are extracted using a simple convolution layer with kernel size equals to the patch size (P, P) and stride s. A flattening operator is then employed to reshape 2-dimensional patches into a one-dimensional tensor and finally LayerNorm [2] is applied to each one. The Vision Embedding module's work can be resumed as follows:

$$O = LayerNorm(Flatten(Conv2d(X))) \tag{3.5}$$

where X is the output of the Range Embedding module and O the output of the Vision Embedding module.

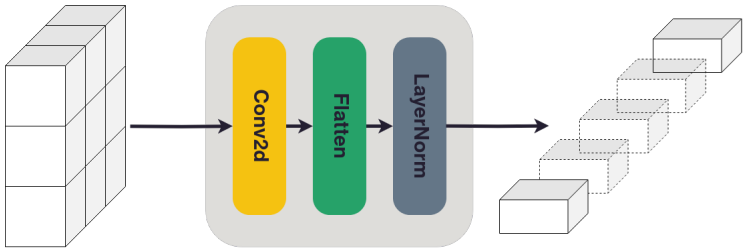


Figure 3.9: Vision Embedding Module

The number of patches generated depends on the input image size (H, W) and the parameters of convolution operator such as kernel size and stride, which correspond respectively to patch size and overlapping factor between patches. To compute the number

of patches that can be obtained on both image dimensions, the following formula can be used:

$$\begin{aligned} H_p &= \lfloor (H - P)/s \rfloor + 1 \\ W_p &= \lfloor (W - P)/s \rfloor + 1 \end{aligned} \tag{3.6}$$

where H_p and W_p are respectively the number of patches along the image vertical and horizontal dimension, P the size of a patch and s the stride value. The final number of patches is: $N = H_p \times W_p$.

It is important to compute and store the number of patches extracted along both image dimensions, as this information will be employed later, in order to reshape the one-dimensional output from RetNet to a two-dimensional object since we are dealing with range images.

3.2.2 RetNet

The core of the architecture is represented by Retentive Network [35], a novel LLM introduced in Chapter 2, which has been modified to be exploited in computer vision tasks. RetNet is composed of L identical blocks (Fig. 3.10), each divided in two sub-layers. The first sub-layer employs LayerNorm and a Multi-Scale Retention (MSR) mechanism, while the second sub-layer presents a LayerNorm and a Feed-Forward Network (FFN). Both leverage residual connections between the input and the output of each sub-layer.

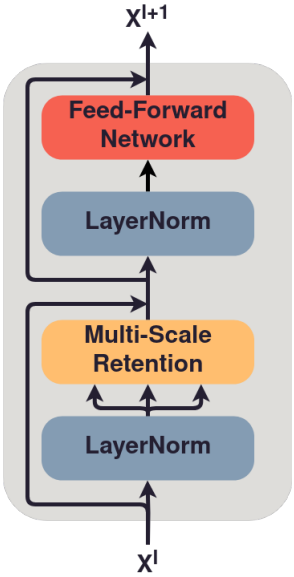


Figure 3.10: RetNet Block

The input of the first block corresponds to the features extracted by the Range Embedding Module (REM) and Vision Embedding Module (VEM) of size (B, N, F) where B is the batch size, N the sequence length, i.e. the number of flattened patches and F is the number of features for each patch. The output of a block l (Eq. 3.7) which corresponds to an intermediate representation, will be fed as input to the subsequent block $l + 1$, while the output of the last block L will be used in the Decoder part to perform class labels predictions. RangeRet employs 4 RetNet blocks, with a single block l defined by the following equations:

$$\begin{aligned} Y^l &= MSR(LayerNorm(X^l)) + X^l \\ X^{l+1} &= FFN(LayerNorm(Y^l)) + Y^l \end{aligned} \tag{3.7}$$

where X^l is the input, Y^l the multi-scale retention value along with a residual connection and X^{l+1} the output of the current block or input for the next block.

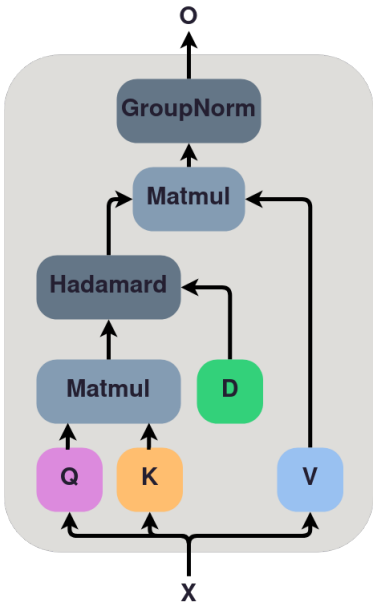


Figure 3.11: Parallel Retention Mechanism

The **Multi-Scale Retention** (MSR) module exploits the retention mechanism using h retention heads, similar to the Multi-Head Attention of Transformers [39], computing different retention scores and then combining them to obtain the final features. Consider X the input flattened patches of size (B, N, F) where the number of features F correspond to the model dimension d_{model}, a single retention in parallel form, represented in Figure 3.11 can be split in two main parts.

First the input X is separately multiplied by the learnable weight matrices $W_Q \in \mathbb{R}^{d_{model} \times d_{head}}$, $W_K \in \mathbb{R}^{d_{model} \times d_{head}}$, $W_V \in \mathbb{R}^{d_{model} \times d_{vhead}}$ to obtain the query $Q \in \mathbb{R}^{N \times d_{head}}$, key $K \in \mathbb{R}^{N \times d_{head}}$ and value $V \in \mathbb{R}^{N \times d_{vhead}}$ matrices. Since we are employing h heads, the second dimension of both weights and Q, K, V matrices depends on the number of heads and corresponds to $d_{head} = d_{model}/h$ for Q and K, d_{vhead} for V, which is usually doubled with respect to the single head dimension.

$$Q = (XW_Q) \odot \Theta \qquad K = (XW_K) \odot \bar{\Theta} \qquad V = XW_V \tag{3.8}$$

Then xPos [34], a position aware embedding denoted as Θ and $\bar{\Theta}$ in Eq. 3.8, is combined into the matrices Q, K to infer information regarding the patch position in the input sequence.

The second part corresponds to the retention score computation using matrices Q, K, V and D which represent causal masking and exponential decay along distances, substituting the Softmax work in Transformers. The matrix D has size $\mathbb{R}^{N \times N}$ since it has to match the product QK^T.

$$Retention(X) = (QK^T \odot D)V \tag{3.9}$$

The output of a single retention mechanism enlarges its feature dimension due to the value matrix V, obtaining that $Retention(X) \in \mathbb{R}^{N \times d_{vhead}}$.

The network employs a decay matrix D inspired by [10], which differs from the original one proposed in RetNet. It leverages the two-dimensional property of images, as well as the geometrical and spatial information among patches. The decay factors are modeled on the Manhattan distance between patches, as presented in [10], combined with an

exponential mapping to better handle the influences between two patches and match the weights range of the original matrix. The Manhattan Distance between two patches $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ is defined as:

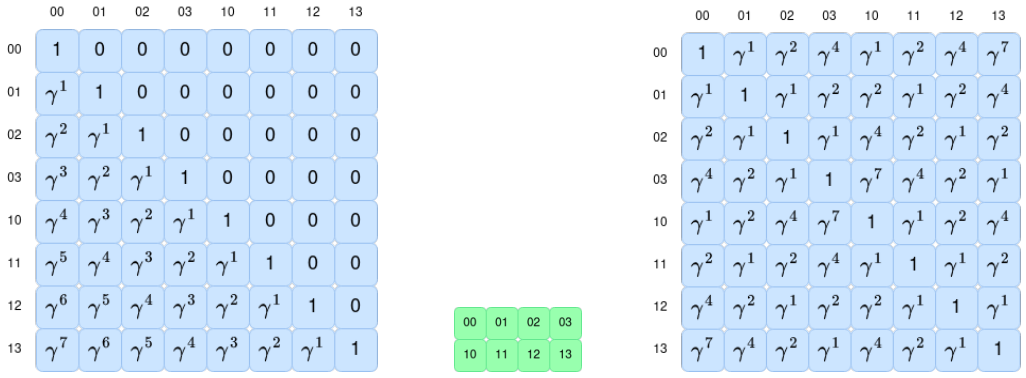
$$ManhattanDistance(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2| \tag{3.10}$$

The distance is then mapped as follows:

$$Mapping(x) = \left(\frac{x-a}{b-a}\right)^2 \cdot (d-c) + c \tag{3.11}$$

where x represents the original Manhattan distance, which is mapped from a value in the input domain interval $[a, b]$ into a range of output values $[c, d]$. Specifically the input range of values corresponds to $[0, max\_dist]$ where $max\_dist = H_p + W_p - 2$ is the maximum Manhattan distance between two patches in the image, with H_p, W_p that represent the number of patches along the vertical and horizontal range image dimension. The output range corresponds to the total number of patches extracted, i.e. it is $[0, N-1]$ with $N = H_p \times W_p$. This mapping ensures a similar range for what concern the decay values, with respect to the original D while improving the representation of two-dimensional linkages among patches.

Since the model deals with image patches, future information are allowed so the upper triangle in the matrix contains the same values of the bottom triangle in a symmetric way with respect to the diagonal. The overall decay matrix D can be depicted as in Figure 3.12c.



(a) Uni-directional D (b) Image (c) Exp Manhattan D

Figure 3.12: Decay Matrices

Dealing with Multi-Scale Retention, involves a number $h = d_{model}/d_{head}$ of heads, thus h retention values calculation, where d_{model} is the model hidden dimension of both input and output feature, while d_{head} corresponds to the dimension of a single head. For every head $i = 1, \cdots, h$ we employ different decay values γ thus different D matrices and different retention scores. However, the decay values are fixed along the blocks.

$$\begin{aligned} \gamma &= 1 - 2^{-5-arange(0,h)} \in \mathbb{R}^h \\ D_i &= \text{mask}(\gamma_i) \\ head_i &= Retention(X, D_i) \end{aligned} \tag{3.12}$$

Then the features obtained from each head are concatenated, normalized and transformed to the final output features through learnable weight matrices W_G, W_O for projection operations and swish-gate [31] function to introduce non-linearity.

$$
\begin{aligned}
Y &= GroupNorm_h(Concat(head_i, \cdots, head_h)) \\
MSR(X) &= (swish(XW_G) \odot Y)W_O
\end{aligned}
\tag{3.13}
$$

The projection matrices W_G and W_O have respectively size $\mathbb{R}^{d_{model} \times d_v}$ and $\mathbb{R}^{d_v \times d_{model}}$, where $d_v = d_{vhead} \cdot h$.

As suggested in [35], the retention score is normalized with the following operations: QK^T is divided by the square root of $d = d_{model}/h$ to obtain ${}^{QK^T}/_{\sqrt{d}}$, the matrix D becomes $\tilde{D}_{nm} = {}^{D_{nm}}/_{\sqrt{\sum_{i=1}^{n} D_{ni}}}$ and the retention score is replaced with $\tilde{R}_{nm} = {}^{R_{nm}}/_{max(|\sum_{i=1}^{n} R_{ni}, 1|)}$.

The second sub-layer corresponds to a **Feed-Forward Network** (FFN) composed of two Linear units (MLPs) and a GeLU activation function as:

$$
O = MLP(GeLU(MLP(X))) \tag{3.14}
$$

where $X, O \in \mathbb{R}^{N \times d_{model}}$ are respectively the input and output of the network. The FFN transforms the input sequence features from a dimension of d_{model} to an intermediate hidden dimension that is twice the size and then maps it back to the original dimension.

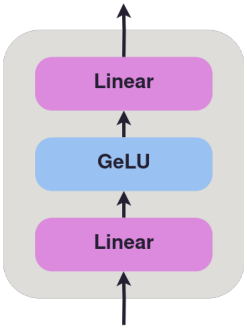


Figure 3.13: Feed-Forward Network

RetNet output is one-dimensional since it deals with a sequence of patches, thus it must be reshaped to a two-dimensional representation, i.e. an image. Introduced in the Vision Embedding Module (VEM), the number of patches extracted along both vertical and horizontal dimensions are H_p and W_p. The output of the Retentive Network, with a shape of (B, N, F), is consequently reshaped to (B, H_p, W_p, F) while preserving the number of features for each patch. This transformation represents the initial range image as a patched image representation with smaller size, that will be later exploited in the Semantic Head module.

3.2.3 Semantic Head

The Semantic Head module corresponds to the Decoder part of RangeRet's encoder-decoder structure. It is in charge of reshaping the output of Retentive Network back to the original range image dimensions and then gradually mapping its features to the number of classes d^{cls} through Multi-Layer Perceptron layers.

From a 2D tensor of size (H_p, W_p) based on the number of patches and overlapping factor computed in the Vision Embedding module (VEM), we use bi-linear interpolation to resize it back to the original range image size (H, W).

The resized feature map, with dimensions (H, W, C) where C represents the channel dimension, is integrated with the output of the Range Embedding Module (REM) via a residual connection using simple addition. This integration aids the Semantic Head in enhancing the segmentation process by leveraging the features generated by the Retentive

Network. This is particularly beneficial given the limited number of patches extracted and utilized during the learning procedure with respect to the total number of pixels in the range image. These fused feature maps are then gradually mapped to a number of features equal to d^{cls} as the classes. This is achieved with two MLP layers as follows:

$$O = MLP(GeLU(LayerNorm(MLP(X)))) \tag{3.15}$$

with $GeLU(\cdot)$ activation function and $LayerNorm(\cdot)$ between the two layers, as suggested in [24]. RetNet's output has a default channel dimension of 128, as its input, so the two MLPs in Figure 3.14 will map features as follow: $(128 - 64)$ and $(64 - d^{cls})$ with 64 as default semantic head hidden dimension. The final features correspond to the number of classes in order to produce the probability distribution for each class, instead of simply predicting the single class label. This gives more information to the training procedure and the loss function, while providing better model performance.

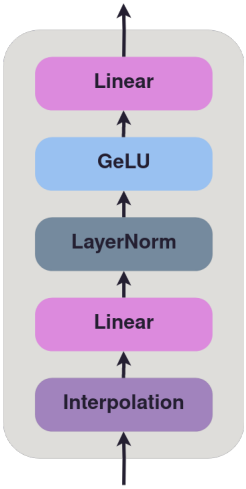


Figure 3.14: Semantic Head Module

3.3 Post-processing

The predictions in the final range image of shape (B, C, H, W), where the channel dimension C corresponds to the number of classes d^{cls}, are projected back to the original point cloud in a simple but efficient manner. Using both the range information and pixel coordinates for point cloud reconstruction would result in a notable reduction in the number of points, as a range image of size $(64, 1024)$ contains only $65,536$ points compared to a scan of 120,000 points. Instead, as suggested in [28], we employ all the pixel pairs (u, v) for all the 3D points p_i obtained during the rasterization process and simply index the range image with the coordinates that correspond to each point. This is a fast and loss-less technique that ensure a semantic label for each point in the original point cloud.

To enhance the predictions and fix the problems after the re-projection from a 2D plane to three-dimensional space, a post-processing technique is employed. A fast, GPU-based k-nearest neighbor (kNN) approach [28] is applied to find, for each point in the 3D semantic point cloud, a vote of the k points closest to it. A defined threshold, referred to as the cut-off, sets the maximum allowed distance for a point to be considered a near neighbor.

The range image representation is utilized to achieve a parallelized nearest neighbor search. A window $[S, S]$ around each point in the $[H, W]$ range image is obtained, producing an $[S^2, HW]$ matrix through the "im2col" primitive. This matrix unwraps the

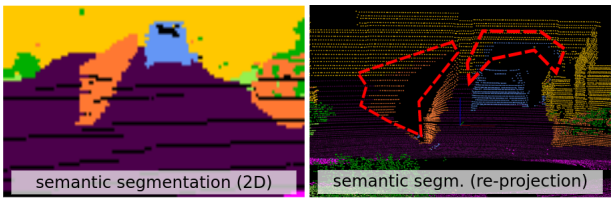


Figure 3.15: Re-Projection Problem from [28]

$[S, S]$ neighborhood for each column, providing the actual pixel's range. Using the (u, v) tuples obtained during the rasterization process, the representation is extended to a matrix of dimension $[S^2, N]$, capturing the range neighborhoods of all scan points. A key step involves replacing the center row of the matrix with actual range readings, resulting in a $[S^2, N]$ matrix containing range readings and unwrapped $[S, S]$ neighborhoods in its columns. The algorithm efficiently calculates the absolute difference in range between the query point and surrounding points within the $[S, S]$ neighborhood. This is done by subtracting the $[1, N]$ range representation of the LiDAR scan from each row of the $[S^2, N]$ neighbor matrix, producing an $[S^2, N]$ matrix. The next step involves weighting the calculated distances using an inverse Gaussian kernel, which penalizes larger differences in range between points that are distant in (u, v) coordinates. An $argmin$ operation is applied to find the k closest points for each column containing the S^2 candidates. This operation yields the indexes for the k points with the least weighted distance within the $[S, S]$ neighborhood. The algorithm checks which of the k points fit within the allowed threshold (cut-off) and accumulates votes from all the labels of the points within that radius. This is achieved through a gather add operation, resulting in a $[C, N]$ matrix, where C is the number of classes, and each row contains the number of votes in its index class. Finally, a simple $argmax$ operation over the columns of the matrix returns a $[1, N]$ vector containing the clean labels for each point in the input LiDAR point cloud.

3.4 Data Augmentation

Data augmentation plays a crucial role in enhancing the versatility and generalization of deep learning models. In this project, straightforward yet effective strategies were implemented to augment point cloud data, reinforcing the model against overfitting and enhancing its performance in diverse scenarios. These augmentation techniques are applied to each point cloud of the training set, without generating new data, but slightly modifying the input ones, preserving the overall geometrical and spatial information among points. The configuration of data augmentation include scaling, rotation, jittering, flipping and is described as follows.

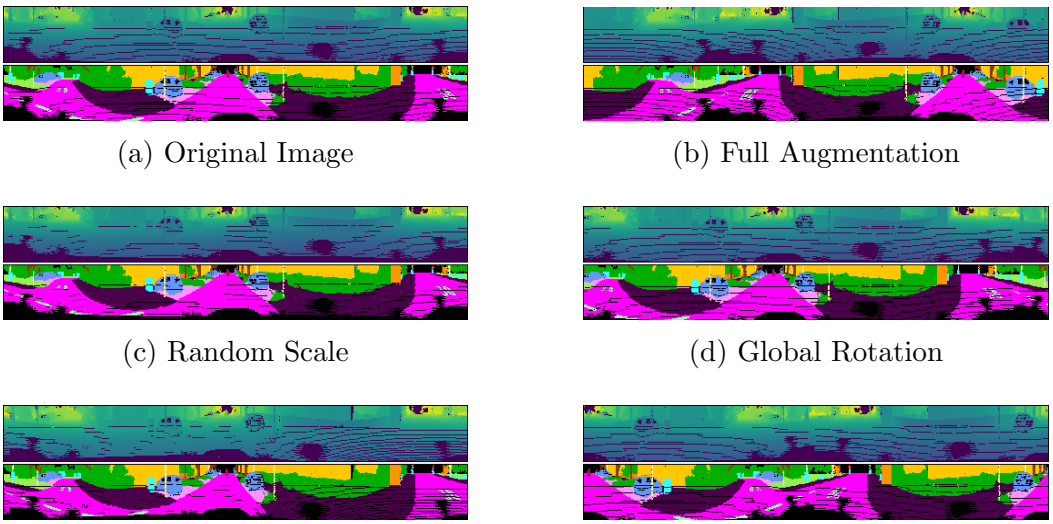
- **Random Scale**: this augmentation randomly scales each 3D point $p = (x, y, z)$ in the point cloud. Specifically, the coordinates x and y are multiplied by a random value chosen from the range $[0.95, 1.05]$ while the coordinate z is not altered

- **Global Rotation**: the whole point cloud is rotated in the XY-plane by a random angle α in the range $[0, 360]$. Explicitly the rotation matrix $R = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$ is multiplied by the x, y coordinates of the 3D points.

- **Random Jitter**: coordinates x, y of the 3D points are randomly jittered within a range $[-0.3m, +0.3m]$, without affecting the coordinate z

- **Random Flip**: performs a global transformation of point coordinates (x, y) flipping along the X axis, Y axis or both

These common point cloud augmentation strategies are applied to each input of the network, during the training procedure, employing the transformations in the same order they are presented above. These techniques introduce diversity into the training set, helping the model to generalize better to unseen data. It exposes the model to a wider range of variations and scenarios that it might encounter during testing. Moreover data augmentation help prevent overfitting, where the model becomes too specific to the training data and performs poorly on new, unseen data. Figure 3.16 shows how each augmentation techniques on point cloud affect the resulting range image.



(a) Original Image

(b) Full Augmentation

(c) Random Scale

(d) Global Rotation

(e) Random Jitter

(f) Random Flip

Figure 3.16: Augmentation strategies effect on range images

Chapter 4

Experiments

This Chapter provides an overview of the tests and experiments conducted to develop the architecture of the proposed method. It begins with an introduction to the datasets and metrics employed to evaluate RangeRet, along with the obtained results. Subsequently, a detailed analysis of the experiments is presented to assess the method's validity, along with justifications for the implementation choices made. Finally, an ablation study section is included to demonstrate how modifications to individual modules impact the network's performance.

4.1 Dataset

The main LiDAR dataset used for testing purposes is SemanticKITTI [3], based on the odometry dataset of KITTI Vision Benchmark [12], which represents inner city traffic, residential areas, but also highway scenes and countryside roads in Karlsruhe, Germany. The 3D point clouds are generated with a LiDAR sensor, specifically the Velodyne HDL-64E which has 360° as horizontal field of view and 64 laser beams. The dataset consists of 22 sequences, splitting sequences 00 to 10 (exc. 08) for training set, 11 to 21 as test set and 08 as validation set. Overall, it contains 19130 full 3D scans for training, 4071 for validation and 20351 for testing.

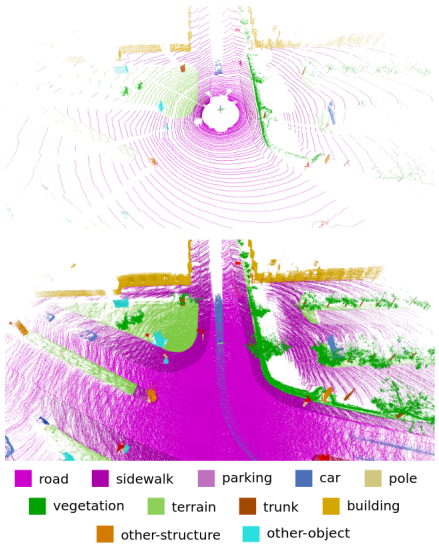


Figure 4.1: Single scan (top) and multi-scan (bottom) with labels

SemanticKITTI provides point-wise labels for sequential point clouds, which allows investigation on individual scans of the whole sequence or consecutive scan aggregation

to assess how the semantic segmentation performances are affected. Each scan includes a different number of 3D point with attributes (x, y, z, r) where x, y, z are the coordinates in 3D space and r is the remission value, i.e. the strength of the reflected laser beam which depends on the properties of the surface that was hit.

The dataset is annotated with 28 classes where 6 classes are assigned to the attribute moving or non-moving as for vehicles or persons and 1 class corresponds to outliers, for erroneous laser measurements caused by reflections or other effects. As common for datasets captured in natural environments some classes are under-represented, since they do not occur often. The label distribution is shown in Figure 4.2, where solid bars represent non-moving objects and hatched bars moving objects.

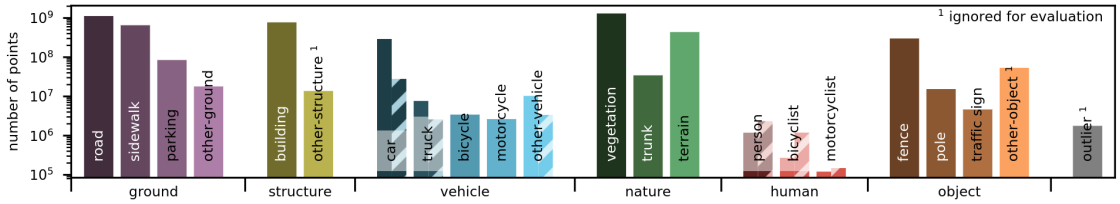


Figure 4.2: SemanticKITTI label distribution

The ground classes as road, sidewalk, building, vegetation and terrain are the most frequent classes. Motorcyclist occurs rarely but still more than 100000 points are annotated. Classes as other-structure and other-object have either few points and are too diverse with a high intra-class variation, so usually they are not included in the evaluation, same as for the outlier class. Furthermore, in a single scan, classes with moving and non-moving attributes are combined together, resulting in a total number of 19 classes for training and evaluation.

4.2 Metrics

To evaluate the labeling performance, it is used the commonly applied mean Jaccard Index or mean intersection-over-union (mIoU) over all classes, described as:

$$mIoU = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FP_c + FN_c} \tag{4.1}$$

where TP_c, FP_c, FN_c correspond to the number of true positive, false positive and false negative for class c, and C is the total number of classes.

4.3 Implementation Details

The experiments are conducted on SemanticKITTI dataset, using 64×1024 as range image size. We use **AdamW** optimizer [25] with $lr = 1e-3$ and weight decay of 0.05. AdamW is a stochastic optimization method that modifies the typical implementation of weight decay in Adam [20], by decoupling weight decay from the gradient update. The original Adam uses L_2 regularization as follow:

$$g_t = \nabla f(\theta_{t-1}) + \lambda \theta_{t-1} \tag{4.2}$$

where λ is the rate of the weight decay. AdamW instead adjusts the weight decay term to appear in the gradient update as:

$$\theta_t = \theta_{t-1} - \eta(\frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t + \lambda\theta_{t-1}) \tag{4.3}$$

Algorithm 1 AdamW Optimizer

Given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
Initialize time step $t \leftarrow 0$, parameter vector $\theta_{t=0} \in \mathbb{R}^n$, first moment vector $m_{t=0} \leftarrow 0$, second moment vector $v_{t=0} \leftarrow 0$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
    $t \leftarrow t + 1$
    $\nabla f_t(\theta_{t-1}) \leftarrow \text{SelectBatch}(\theta_{t-1})$ ▷ Select batch and return corresponding gradient
    $g_t \leftarrow \nabla f_t(\theta_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ ▷ Element-wise operations
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ β_1 is raised to the power of t
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ β_2 is raised to the power of t
    $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ Can be fixed, decay, or used for warm restarts
    $\theta_t \leftarrow \theta_{t-1} - \eta_t(\alpha\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda\theta_{t-1})$
until stopping criterion is met
return optimized parameters θ_t

We use **OneCycle** scheduler, introduced in [32], to dynamically adjust the learning rate during the training process. Its main idea is to vary the learning rate in a single cycle which typically goes through two phases. In the first part, the learning rate starts at a small value and gradually increases to help the model escape from local minima and speed up the initial convergence. In the second part, the learning rate is gradually decreased in order to help the model fine-tuning its parameters with a small learning rate, hopefully converging to a better minimum. The OneCycle policy [33] changes the learning rate after every batch by a step factor where the total number of steps corresponds to $epochs \times steps\_per\_epochs$. We use maximum learning rate as $1e-2$ and steps per epochs equals to the training set size, since batch size is 1, i.e. a single image is processed at time. The percentage of the cycle (in terms of the number of steps) dedicated to increasing the learning rate is 0.3. As a result, the learning rate will rapidly increase in the initial epochs and then gradually decrease for the rest of the training procedure.

The model training is supervised by a combination of loss functions, in particular the Cross-Entropy loss, Lovasz-Softmax loss [4] and Focal loss [23].

The **Cross-Entropy loss** is useful when training a classification problem with C classes. It expects an input of size $(B, C, d_1, ..., d_k)$ with $K \geq 1$ for k-dimensional case, containing unnormalized logits for each class and can be provided with weights for each class in case of unbalanced training set. The loss is described as:

$$l(x, y) = L = l_1, ..., l_N{}^T, \qquad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \tag{4.4}$$

where x is the input, y the target, w the weight, C the number of classes and N spans the minibatch dimension. This case is equivalent to applying the log operator to the $Softmax(\cdot)$ function, followed by Negative Log Likelihood loss.

Lovasz-Softmax loss [4] is a surrogate for the optimization of the intersection-over-union (IoU) in the context of semantic segmentation. Regarding multi-class semantic segmentation, the loss employs a $Softmax(\cdot)$ function to map the output of the model to probability distribution and uses the class probability $f_i(c) \in [0, 1]$ to construct a vector of pixel errors $m(c)$ for class $c \in C$ as:

$$m_i(c) = \begin{cases} 1 - f_i(c) & \text{if} \quad c = y_i^* \\ f_i(c) & \text{otherwise} \end{cases} \tag{4.5}$$

The vector of errors $m(c) \in [0,1]^p$ is then used to construct the loss surrogate to Δ_{J_c}, the Jaccard index for class c:

$$loss(f(c)) = \bar{\Delta}_{J_c}(m(c)) \tag{4.6}$$

Thus, Lovasz-Softmax loss is defined as:

$$loss(f) = \frac{1}{|C|} \sum_{c \in C} \bar{\Delta}_{J_c}(m(c)) \tag{4.7}$$

which is piece-wise linear in f, the normalized network output.

Focal loss [23] addresses class imbalance during training, applying a modulating term to the cross entropy loss in order to focus learning on hard misclassified examples. It proposes to add a factor $(1 - p_t)^\gamma$ to the cross entropy loss, with tunable *focusing* parameters $\gamma \geq 0$. It is defined as:

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t) \tag{4.8}$$

With $\gamma = 0$, Focal loss is equivalent to Cross Entropy and as γ is increased the effect of the modulating factor is likewise increased. When an example is misclassified and p_t is small, the modulating factor is near to 1 and the loss is unaffected, while as p_t tends to 1, the factor goes to 0 and the loss for well-classified examples is down-weighted. For better performance, the adopted Focal loss is α-balanced as:

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t) \tag{4.9}$$

with $\alpha_t \in [0,1]$, yielding sightly improved accuracy over the non-α-balanced form. The effect of the parameter γ on the loss function, compared to the Cross-Entropy loss, is represented in Figure .

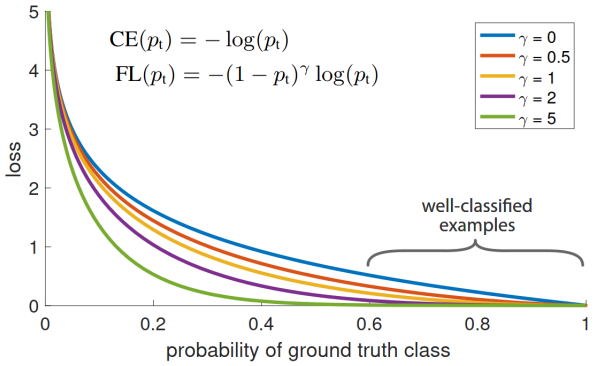


Figure 4.3: Effect of γ on Focal Loss

4.4 Experiments

In this section, all the conducted experiments will be presented, starting from the first tests with the implementation of Retentive Network and the subsequent changes to improve the network performance. The experiments were performed on a single NVIDIA GeForce RTX 2080 GPU with 8 GB of memory, which required greater commitment regarding

the memory usage. In order to systematically develop the network structure and enhance its performance, a series of tests were conducted using progressively larger subsets of the SemanticKITTI dataset. In the initial phase, a small subset comprising 100 samples for training and 10 for validation was employed to establish the foundational architecture of RangeRet. Subsequently, larger subsets containing 1000 and 2000 scans were utilized to refine implementation choices and assess network performance with increased input data. Finally, a subset representing 25% of SemanticKITTI, i.e. with 5000 scans, was employed to fine-tune hyperparameters and derive the optimal network configuration within the constraints of the available hardware resources. This optimization preceded the comprehensive testing of the network on the complete SemanticKITTI dataset.

Initial Setup The initial setup, inspired by [21] comprises a Range Embedding Module (REM) with 3 MLP layers to map each pixel in the range image into a higher dimension, a Vision Embedding module (VEM) based on the Microsoft Torchscale [27] implementation which consists of a convolution layer and a flatten operator and a Semantic Head module to map the feature learned by RetNet into the number of classes for the predictions. A basic implementation of RetNet[1] is employed with 4 layers with 4 heads each as default. For training, the Cross Entropy loss function between range images and the Stochastic Gradient Descent (SGD) optimizer with $lr = 1e - 2$ were employed, leveraging the strengths of SGD to assess the network learning capabilities. The choice of patch size and stride was influenced by the limited GPU memory, allowing for a maximum patch size of 4×4 with a stride of 4. The GPU constraints prevent the handling of 4×4 patches with a stride less than 4 or 3×3 patches with a stride less than 3. In the Semantic Head module, several tests were conducted to compare bi-linear interpolation and 2D transpose convolution for recreating the input range image size from the reshaped RetNet output. As there were no clear differences in the results obtained between using one approach over the other, we opted to maintain bi-linear interpolation, aligning with the SoTA method RangeFormer.

Patch Configuration Considering the constraints of limited GPU memory, our first focus was on determining the optimal patch size value to configure the network and obtain better results. This is because the number of patches directly affects RetNet memory consumption and computation time. Observing the use of 4×4 patches with stride 4 on a 64×1024 image, it is noted that this configuration produces exactly 16 and 256 patches along the vertical and horizontal dimensions, resulting in a total number of 4069 generated patches. We discovered that using patches of size 7×7 with stride 4, for a total number of 3825 patches, slightly improve the network performance, probably due to the overlapping configuration with respect to the non-overlapping one, even though the number of patches generated is a bit lower. However, a range image of size 64×1024 contains 65536 pixels.

Size	**Stride**	**#patches**	**Ratio (%)**
(3, 3)	1	63364	96.6
(4, 4)	1	62281	95.0
(4, 4)	2	15841	24.2
(4, 4)	3	7161	10.9
(4, 4)	4	4096	6.25
(7, 7)	4	3825	5.84

Table 4.1: Patch size configuration on 64×1024 image

Using 7×7 patches with a stride of 4 results in 3825 patches, which represents 5.84%

[1] A basic Retentive Network architecture inspired by https://github.com/Jamie-Stirling/RetNet

of the image features if we consider a point cloud with 120000 points. Table 4.1 shows that as the size and stride decrease, more features are retained and utilized in the learning process, increasing significantly the memory usage as side effect. Therefore, the choice of patch size is a key decision in the trade-off among performance, memory consumption and inference time.

It is important to notice that this patch configuration 7×7, stride 4, does not leverage a row/column of the range image since kernel size and stride do not perfectly match the image size as patches 4×4 with stride 4 would do. Regardless, the overlapping configuration seems to learn better features, compared to the non-overlapping one, as shown in Table 4.2. Patches 2×8 suggested in [1] seem also a valid option to exploit the range image structure and properties.

Size	**Stride**	**mIoU (%)**
(4, 4)	4	40.9
(2, 8)	(2, 8)	42.0
(7, 7)	4	**42.3**

Table 4.2: Comparison between patch configuration

We compare the training process using patches size 7×7 and 4×4 with stride 4 in Figure 4.4 using the standard decay matrix D, without employing data augmentation, on the SemanticKitti subset with 5000 scans. The first configuration yields slightly better results in terms of validation mIoU [2] and also demonstrates lower loss values. This is attributed to the second setting being more susceptible to overfitting issues in the last epochs.

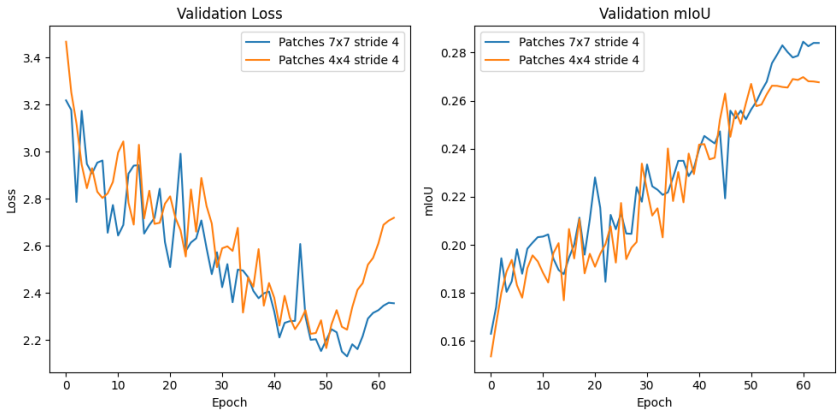


Figure 4.4: Training comparison between patch configuration

We obtain the same scenario when using data augmentation on the point clouds and the exponential Manhattan matrix D, as shown in Figure 4.5. The configuration with patches of size 7×7 and stride 4 generates better outcomes with respect to patches 4×4 with stride 4. Moreover, the first configuration shows less overfitting in the last training epochs, as in the previous setup.

Optimizer The initial setup employs SGD as optimizer which is able to asses the network performance, finding the optimal minimum point even though it may not be global, thus ensuring that the network will learn. We then tested AdamW [25] optimizer with $lr = 1e-4$ and $weight\_decay = 0.05$ obtaining greater results on both training and validation set. The validation mIoU on the subset with 100 scans increased from

[2]The mIoU is computed between range images on the complete sequence 08 of SemanticKitti

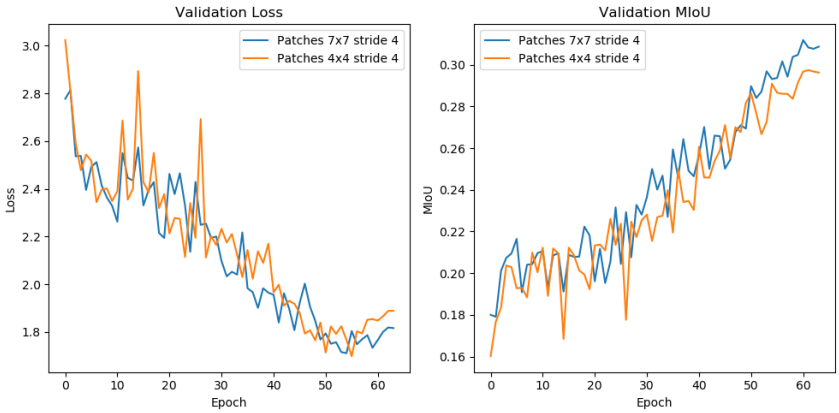


Figure 4.5: Training comparison between patch configuration with augmentation

15.9% to 18.8%, providing better generalization performance. As later described, with the introduction of a scheduler, especially the 1cycle policy [33], for a better management of the learning rate, we modified the AdamW learning rate as $1e-3$ which produced greater results.

Range Embedding Module In order to generate good features for each pixel in the input range image, we analyzed the structure of the Range Embedding Module (REM). The basic setup provides 3 Linear layer which gradually map features from 5 to 128. Thus we enhanced the module with a better structure as three blocks of Linear-LayerNorm-GeLU which produces greater results and it also able to slightly reduce the overfitting with respect to three simple Linear layers. Moreover we tested the hidden dimension of the linear layers, finding out that the mapping $(5-64)-(64-128)-(128-128)$ has slightly better results compared to the mapping $(5-32)-(32-64)-(64-128)$ on the validation set. Later, as mentioned in [46], we introduced a REM using convolution-based layers to compare its performance with respect to the MLP implementation. In parallel with the structure of the Linear layers, we designed three blocks of convolutional layers composed of Conv2D-InstanceNorm-GeLU, each with a kernel size of 3×3, stride 1 and padding 1, to better leverage the 2D input features. We then tested this approach with the same feature mapping in the MLP structure, and discovered that, in this case, the mapping $(5-32)-(32-64)-(64-128)$ produces slightly better results compared to $(5-64)-(64-128)-(128-128)$, which is the opposite of the results observed in the MLP case. The convolution-based module improved the results on the validation set with respect to the MLP implementation, so we decided to use this approach for the Range Embedding Module. Usage of InstanceNorm on the network input slightly improved the results and helped with the overfit issue. It is preferred in place of LayerNorm, since it is suitable for channeled data as RGB images, even though the results obtained employing the two norms are almost the same.

Semantic Head As done for the Range Embedding Module, we explored different implementations also for the Semantic Head, using linear layers, convolution layers or a combination between them. The first implementation consists of two linear layer than map the features from 128 to 64 and finally to d^{cls}, the number of classes. Between the two layers, a LayerNorm and GeLU activation function are employed. We tested a convolution-based approach as for REM, using a structure Conv2D-InstanceNorm-GeLU-Conv2D, obtaining no improvements with respect to the MLP implementation. Furthermore, we found that increasing the number of convolution layers led to more pronounced overfitting, particularly when employing three or more convolution layers, without yielding significant improvements to the results. We also combined a layer of MLP and a convolution operator,

observing that it did not produce better results. The best implementation seems to be the MLP structure with 2 linear layers and a LayerNorm and GeLU activation function between them.

To better analyze how the different implementations of the Range Embedding Module and Semantic Head affects the results, we tested all the combinations between the previously described approaches. Thus, we exploited the initial MLP architecture for both modules, a convolution-based approach and a combination between them. We evaluated the performance using the subset with 100 scans and implemented the approach with the highest mIoU [3], i.e. generalization ability. Results are reported in Table 4.3.

REM	**Head**	**mIoU (%)**
Linear	Linear	17.49
Linear	Conv	16.64
Conv	Linear	**20.88**
Conv	Conv	19.43

Table 4.3: Comparison among modules architecture

The worst approach is represented by the REM composed of linear layers and head of convolution layers since the MLP struggles more to learn features from 2D input with respect to the convolution operator, especially at the beginning of the architecture. In fact the best approach results to be a convolution-based REM and a Semantic Head of Linear layers, better learning the input features for RetNet and exploiting simple but efficient predictions with MLPs. The other two combination provides intermediate results which are not close enough to the ones obtained by the best architecture configuration.

Scheduler We investigated the impact of different learning rate scheduling strategies on the training dynamics and performance of the model since learning rate plays a crucial role in optimizing the model parameters during training. We tested the following strategies:

- Constant: an initial learning rate is set and does not change during training;
- Cosine Annealing [26]: starts with a large learning rate and then rapidly decrease it to minimum value before increasing rapidly again as a simulated restart of the learning process;
- One Cycle [33]: anneals the learning rate from an initial value to a maximum value and then from that maximum to a learning rate much lower than the initial.

The initial tests were performed with a constant learning rate, but as the network main architecture was designed we explored different learning rates approaches. We employed Cosine Annealing with two configurations, considering AdamW with $lr = 1e-4$ and a minimum learning rate of $1e-5$, but also AdamW with $lr = 1e-5$ and $min\_lr = 1e-6$. We obtained better results compared to the constant policy but, unfortunately, both approaches have disadvantages. The first approach leads to more overfitting after a few epochs, as it exploits the learning procedure at the beginning. On the other hand, the second approach, which is less affected by overfitting, produces lower results in the same number of epochs, possibly requiring more time. Regarding the OneCycle policy, we tested different combinations of the optimizer learning rate lr in the range of $[1e-3, 1e-4, 1e-5]$ and the maximum learning rate max_lr in the range of $[1e-2, 1e-3]$. The configuration $lr = 1e-3, max\_lr = 1e-2$ was able to improve the results compared to the previous

[3] mIoU on the validation set with 10 scans is considered and reported in the table

approaches, better handling overfitting issues. Therefore, we implemented this learning setup with AdamW $lr = 1e-3$ and the OneCycle scheduler with $max\_lr = 1e-2$. The other combinations produced lower results and were more influenced by the overfitting phenomenon, leading us to discard them.

Learning Rate To evaluate the effectiveness of the selected scheduler (OneCycle) and learning rate values, we conducted a series of tests on the main RangeRet model with the standard decay matrix D, excluding the residual connection. These experiments aimed to analyze the impact of these training hyper-parameters on both the training and validation sets, with a particular focus on their role in mitigating the overfitting issue. We employed the SemanticKITTI subset containing 5000 scans and no data augmentation. The initial configuration employs $lr = 1e-3, max\_lr = 1e-2$ with weight decay equals to 0.05, and presents a mild overfitting effect in the last epochs.

Decreasing the learning rate to $lr = 6e-4$ and $max\_lr = 6e-3$ results almost in the same validation mIoU [4] as shown in Figure 4.6 but the overfitting problem is more evident than before. This may be due to the initial lower learning rate that drives the model weights in one of the first encountered local minimum area and then leverage the information in a small neighborhood, fitting more on the training data with respect to learning a general representation. Indeed, the mIoU obtained on the training set is one of the highest encountered among all the experiments with training hyper-parameters.

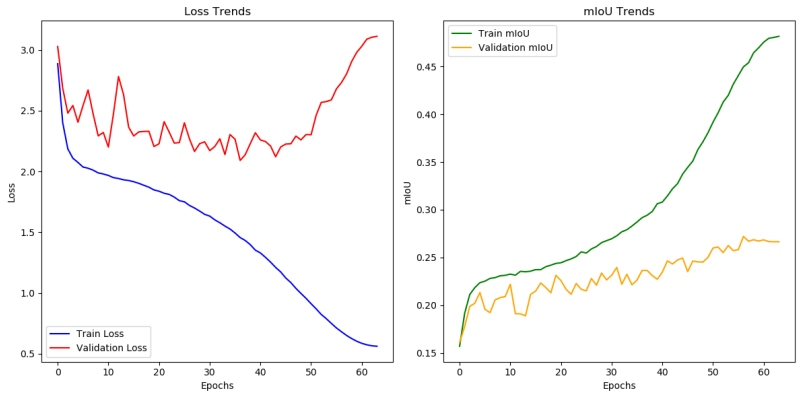


Figure 4.6: Training with $lr = 6e-4, max\_lr = 6e-3, wd = 0.05$

With the information obtained by the previous test, we tried to increment the learning rate to $lr = 3e-3$ and $max\_lr = 3e-2$ to find out if this setting is able to better prevent overfitting. As shown in Figure 4.7, the loss trends are better, with the validation loss curve decreasing as the train loss. The higher learning rate drives the network among different local minimum, preventing the risk to stuck the learning in the first encountered minimum point. However, since the final learning rate is also higher, the learning procedure may not be able to completely leverage the information of the neighborhood, oscillating around the local minimum without ever reaching it.

These settings do not present overfitting issue but, as drawback, the validation mIoU is lower in both sets, especially in the validation set.

Discovering that with higher learning rate values, the overfitting issue may be avoided most of the time, but the obtained results are lower, we tested $lr = 2e-3$ and $max\_lr = 2e-2$, expecting a bit of overfitting, but slightly better results.

Indeed, as shown in Figure 4.8, the mIoU metric is slightly higher on both train and validation sets, with the losses trend almost similar to the previous run. This training configuration is able to prevent overfitting, while generating higher outcomes than before

[4] The mIoU reported in the plots is computed between range images

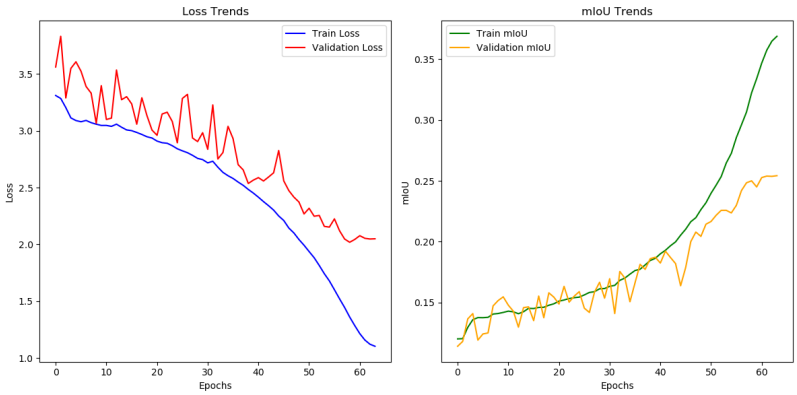


Figure 4.7: Training with $lr = 3e-3, max\_lr = 3e-2, wd = 0.05$

but still not at the level of the one with $lr = 1e-3, max\_lr = 1e-2$. Thus, we choose to keep the initial configuration even though it may suffer from overfitting in the last epochs but it ensures better validation results.

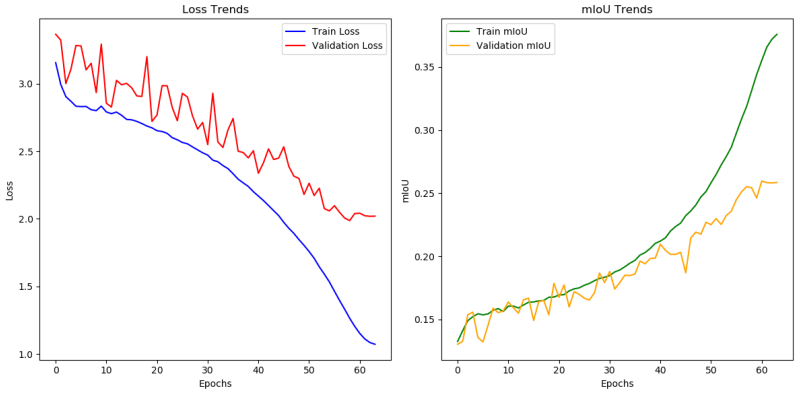


Figure 4.8: Training with $lr = 2e-3, max\_lr = 2e-2, wd = 0.05$

As illustrated in Figure 4.8, the mIoU metric exhibits a slight improvement on both the training and validation sets, with the loss trends remaining comparable to the previous run. This training configuration effectively mitigates overfitting while yielding superior results compared to the previous setup. However, it is not able to reach the performance achieved with $lr = 1e-3, max\_lr = 1e-2$. Despite the possibility of overfitting in the final epochs, we opt to retain the initial configuration as it consistently delivers better validation results. Future application of regularization techniques is expected to enhance the model generalization ability.

Weight Decay We then decided to observe how the training setup is influenced by the weight decay hyper-parameter. The default value employed is 0.05, so we decided to test lower and higher values as $0.01, 0.06$. The test were performed on the RangeRet model with standard matrix D, without residual connection, using the SemanticKITTI subset with 5000 scans and no data augmentation. The initial configuration employs $lr = 1e-3, max\_lr = 1e-2, weight\_decay = 0.05$.

A lower weight decay value is expected to penalize large network weights less, thereby mitigating the impact of L2 regularization. Therefore, we experimented with a weight decay value of 0.01 and observed the anticipated outcome. The training process exhibited a notable overfitting issue, manifesting itself around the midpoint of the procedure. This resulted in the model fitting more closely to the training data, evident from achieving the highest train mIoU among all the other experiments with different learning rates.

Although the metric on the validation set showes similar values to previous experiments, the substantial overfitting makes this model not reliable.

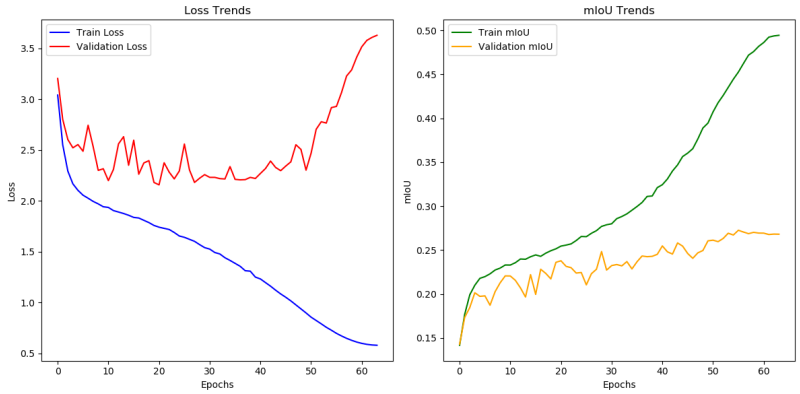


Figure 4.9: Training with $lr = 1e-3, max\_lr = 1e-2, wd = 0.01$

Therefore, to prevent the overfitting issue, we experimented with an slightly higher weight decay value as 0.06, expecting to obtain almost the same results of the best model, and a better generalization. Figure 4.10 shows that an higher L2 regularization mitigate the overfitting issue even though it is slightly present in the last epochs. The validation mIoU is higher with respect to using high learning rates but still not as good as the one with weight decay equals to 0.05.

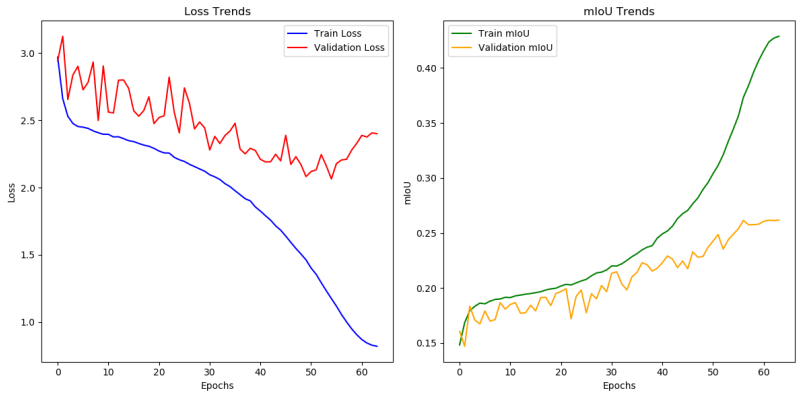


Figure 4.10: Training with $lr = 1e-3, max\_lr = 1e-2, wd = 0.06$

At the end, we opt to keep the weight decay value to 0.05, combined with $lr = 1e-3, max\_lr = 1e-2$, since later used data augmentation techniques helped the training procedure to mitigate the overfitting issue and produced better results on the validation data.

Regularization In order to enhance the generalization capabilities of the model and prevent overfitting, a critical issue to deal with during the model design and hyper-parameters tuning, two widely used regularization techniques are employed as data normalization and dropout [16]. As previously introduced, the Range Embedding and Semantic Head modules utilize InstanceNorm [37] and LayerNorm [2], respectively. These normalization techniques ensure that the inputs to each layer have a mean activation output of zero and a unit standard deviation. Such normalization contributes to improved speed, performance, and stability in neural networks. Moreover, the normalization of retention score as suggested in [35] improves the overall numerical stability and does not affect outputs and backward gradients due to the GroupNorm [44] presence. As shown in Figure 4.11, InstanceNorm is suitable when dealing with images since it normalizes across

each channel in each training sample instead of LayerNorm which normalizes each example over its channels and pixels. Finally GroupNorm, employed in Retentive Network, normalizes each sample considering a group of few channels.

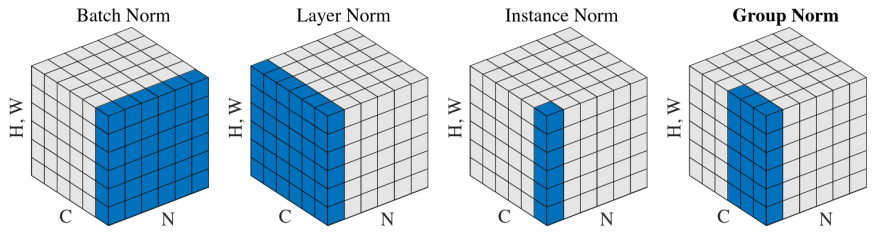


Figure 4.11: Normalization methods from [44]

To address overfitting, we experimented with the Dropout technique on the Range Embedding Module, Semantic Head, and the Feed-Forward Network of RetNet. With a dropout probability (p) set at 0.1, the learning process demonstrated resilience against overfitting without significant hindrance. Increasing the dropout probability to $p = 0.2$ mitigated overfitting to a certain extent but at the cost of a slightly slowed learning procedure. Ultimately, we opted not to employ dropout techniques, as the 1cycle scheduler and the later incorporation of more training data and augmentation techniques proved effective in yielding satisfactory results without substantial overfitting concerns.

Loss Functions The choice of an appropriate loss function significantly influences the model training and performance. We explored three distinct loss functions: Cross Entropy Loss, Lovasz Softmax Loss [4], Focal Loss [23] and combinations among them. Cross Entropy is an effective loss for multi-class segmentation so we started the experiments with it, assigning also the class weights to improve its performance. Lovasz Softmax Loss is designed to optimize for IoU (Intersection over Union) directly. It addresses the issue of class imbalance and provides a more nuanced optimization criterion for tasks like Semantic Segmentation. Indeed, the model tested with only Lovasz loss produces greater results w.r.t. Cross-Entropy, even though, as specified in [4], it is preferred for fine-tuning approaches and it may produce better results combined with other losses. Focal Loss is tailored to address the problem of class imbalance by assigning higher weights to hard-to-classify examples, thus it is particularly useful when dealing with datasets where certain classes are underrepresented as in our settings. This loss produces encouraging results also when used alone.

With the aim of analyzing how each loss can influence the model performance and results, we evaluated combinations among them using the SemanticKITTI subset with 1000 scans for training and the complete sequence 08 for evaluation [5] (Table 4.4). When employing a single loss, the Lovasz loss obtains greater results due to its optimization for the intersection-over-union metric, which is crucial in the proposed method. Focal loss produces better outcomes compared to Cross-Entropy since it is an improvement of the latter with a focus on low-represented classes. It is noteworthy to mention that the Lovasz loss requires much more time in the training process (8.9 it/s), while the other two losses process samples at almost the same speed (10.0-10.3 it/s). Then we experimented combinations among them, observing improvements in all the cases. Cross-Entropy benefits from both Focal loss and Lovasz loss, improving the results compared when employed alone, but the best setting seems to be Focal loss and Lovasz, combining the properties of optimizing the IoU metric while helping the predictions of rare classes. We also tested the model employing all the three losses, obtaining the second best results and good loss trend.

[5]Mean Intersection over Union is computed between point clouds

Cross-Entropy	Focal	Lovasz	Training Time (it/s)	mIoU (%)
✓			10.3	33.7
	✓		10.0	35.1
		✓	8.9	36.8
✓	✓		9.9	37.2
✓		✓	8.9	36.8
	✓	✓	8.8	**38.1**
✓	✓	✓	8.8	<u>37.3</u>

Table 4.4: Loss comparison with 1000 samples

Therefore, we tested the configuration with all three losses and the one with Focal and Lovasz on the SemanticKITTI subset containing 5000 scans to facilitate a more thorough comparison and determine the appropriate configuration. We excluded the setting with Cross-Entropy and Focal loss, as the Lovasz loss is specifically designed for the semantic segmentation task that the model is employed for, and we aim to leverage this unique property. As illustrated in Table 4.5, the results with more training data exhibit variations, with the configuration utilizing all three losses yielding superior outcomes, likely capitalizing on the strengths and characteristics of each loss function. Considering the outcomes, the three losses were implemented in the final network setup, using an average sum approach where each loss is equally weighted.

Loss	mIoU (%)
Focal + Lovasz	41.2
Cross-Entropy + Focal + Lovasz	**42.3**

Table 4.5: Loss comparison with 5000 samples

Some tests were conducted to compare the loss computation between range images and point clouds during the training procedure. As expected, computing the loss between images reduces the processing time in both training and validation steps, while marginally decreasing memory consumption since it is not necessary to re-project the predictions from the image onto the point cloud. Additionally, we observed that using the loss between range images resulted in a slight improvement in the validation metric, approximately of 2 percentage points.

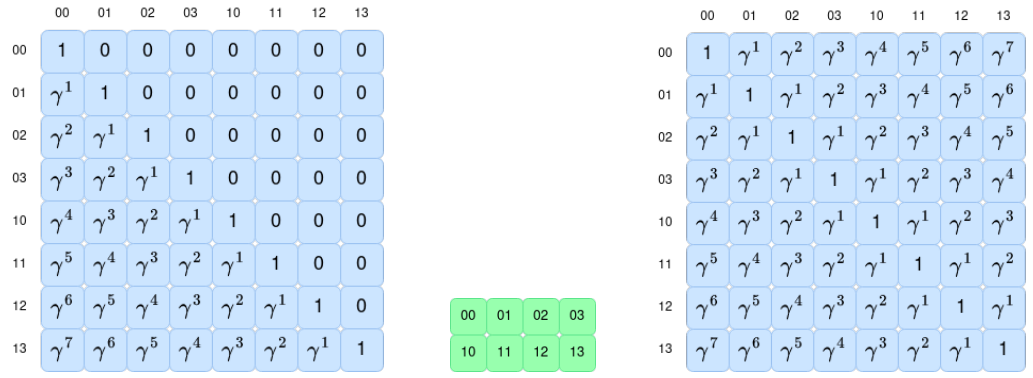
Decay Matrix The Retentive Network architecture proposes a uni-directional decay matrix (Figure 4.12a) that weights the influence among tokens in a sequence. This means that the further a token is in the past, the less important it is for the current time step. Then, the mask property ensure that information from future time-steps is not leaked so the upper triangle is filled with zeros while the bottom triangle contains the decay values.

Matrix D	mIoU (%)
Unidirectional	36.6
Bidirectional	37.7
Manhattan	37.5
Exp Manhattan (Ours)	**39.6**

Table 4.6: Decay Matrix D comparison

This configuration is suitable for Natural Language Processing tasks, but it does not fully exploit the information in Computer Vision tasks. When dealing with images, con-

sider pixels or patches, one can influence all the others with no concern about past of future reference. With these assumptions, as employed in [10], we designed a bi-directional matrix D (Figure 4.12b) with the upper triangle filled with decay values in a symmetric way with respect to the bottom triangle. This way we exploit all the retention scores, unlocking the upper part of the matrix, aiming to obtain better network performance. Indeed, the bi-directional matrix produces better results as shown in Table 4.6 without requiring additional memory or extra computational time.



(a) Uni-directional D (b) Bi-directional D

Figure 4.12: Basic decay matrices

Additionally, as proposed in [10], we conducted tests with another version of the matrix D (Figure 4.14a) aimed at better capturing the influence among patches. This involves considering the Manhattan distance between two patches and modeling the matrix weights accordingly, using both the bottom and upper triangle of the matrix. This layout perfectly match the patch organization, considering the two-dimensional structure of images, instead of representing them as a uni-dimensional sequence. Indeed, each patch is more influenced by the four neighbors patches and this is not correctly represented in the original matrix. According to the experiments, we discovered no improvements in using this matrix layout with respect to the bi-directional matrix.

Therefore, we analyzed the Manhattan-based matrix, finding that the decay values lay on a range which differs from the one of the original triangle matrix. Specifically, the original matrix considers values based on the distance between patches, that, in the uni-dimensional representation is $[0, N-1]$ where $N = h \cdot w$ is the number of patches, with h, w height and the width. In a two-dimensional domain, the maximum Manhattan distance is proportional to the image dimension and can be computed as $d_{max} = (h-1) + (w-1)$. This is because the maximum horizontal distance is $(w-1)$, and the maximum vertical distance is $(h-1)$. Thus the distance values are in the range $[0, d_{max}]$, which for large values of height and width results significantly smaller than the original range.

In order to match the original range, we apply an exponential mapping of the Manhattan distances as follows, between the input range $[a, b]$ and target range $[c, d]$:

$$Mapping(x) = \left(\frac{x-a}{b-a}\right)^2 \cdot (d-c) + c \tag{4.10}$$

Consider an image 64×1024 and patches 7×7 with stride 4, the patches extracted are 15 along the vertical dimension and 255 along the horizontal. Thus, the maximum Manhattan distance is 268 while the total number of patches would be 3825. Figure 4.13 shows the mapping from an input range $[0, 268]$ to the target range $[0, 3824]$.

Experiments with this exponential Manhattan-distance-based decay matrix (Figure 4.14b) show significant improvements with respect to the basic Manhattan distance matrix

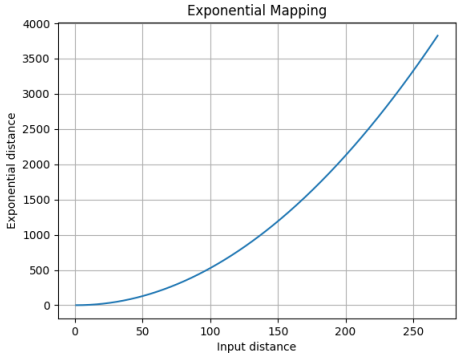
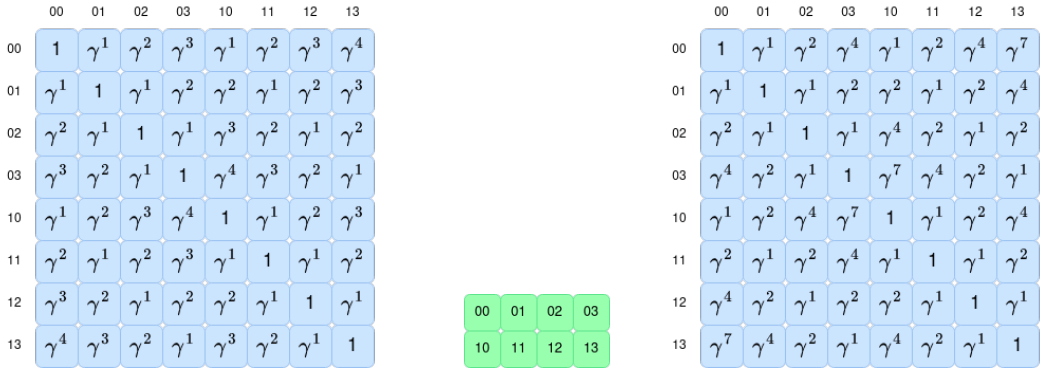


Figure 4.13: Exponential Mapping of Manhattan distance

and the bi-directional matrix. Indeed, the proposed decay matrix inherits the structure of the Manhattan-based that better exploit the geometric layout of patches in an image, while sharing the same values range with the bi-directional matrix.



(a) Manhattan-based D (b) Exp Manhattan-based D

Figure 4.14: Manhattan-distance-based Matrices

During the experimental phase, we identified a challenge with the Retentive Network architecture, particularly in the matrix D which involves causal masking and exponential decay that is different for each head but shared among the RetNet layers. Since it is a matrix of size $N \times N$ where N is the sequence length, its dimension directly depends on the input fed to the model. When dealing with inputs of varying sequence lengths, the matrix D needs to be computed in every forward pass for each input, resulting in increased computation time. However, since we are working with 3D point clouds that are projected into fixed-size range images, all inputs generate the same sequence length. This allows us to pre-compute the decay matrix as a model attribute, leading to time savings in both the training procedure and the inference step.

Patch	**Stride**	**Size** D	**Memory (MB)**
(7, 7)	4	(3825, 3825)	55
(4, 4)	3	(7161, 7161)	195
(4, 4)	2	(15841, 15841)	957

Table 4.7: Decay matrix D memory consumption

Due to its size, as the sequence length increases, the matrix D grows exponentially (Table 4.7), resulting in significant memory consumption [6]. Since it is unique for each

[6]Memory usage in MegaByte assuming a matrix where each element is a 32 bit float

head but shared among the layers, we opted to pre-compute it when creating the RetNet model and then access it during a forward pass, rather than assigning it to each Multi-Scale Retention block across the layers. This approach reduces memory usage by a factor of l, corresponding to the number of layers, providing memory savings, especially when dealing with longer sequences, i.e. a larger number of patches.

Network Configurations The hyper-parameters tuning includes also the hidden size dimension employed in the network modules as well as the structure of Retentive Network in terms of number of layers and heads. In the Range Embedding Module (REM) we kept the main configuration of RangeFormer [21], where the pixels in the input range image are mapped into 128 features, representing a good trade-off between amount of information, time and memory consumption. As previously explained in the REM experiment section, we compared two hidden size configurations: $(5-32)-(32-64)-(64-128)$ and $(5-64)-(64-128)-(128-128)$ with the first slightly outperforming the second. The Semantic Head module was modeled in a symmetric way w.r.t. the REM, even though MLPs are employed. The feature mapping is implemented as $(128-64)-(64-d^{cls})$ with d^{cls} the number of classes, since we found that an hidden dimension of 64 produced better results than other values as 32 or 128. For what concern RetNet, we set as default values a number of layers and heads equals to 4. A lower number of layers results in a simpler model, with less capabilities of learning, while a greater number of layers increase the complexity of the network as well as the training procedure and learning issues. It is worth to notice that less layers would allow a greater number of patches to be processed in a limited memory settings, but at the same time, due to the less complexity of the model, we did not find improvements in this approach. We set as default values $d_{model} = 128$ as the features learned by the REM and VEM modules, with the dimension of matrix V doubled to 256, which led to small improvements. The same dimension was set as the Feed-Forward Network hidden size, which contributed to small improvements. We tested the full RangeRet network with hyper-parameters dimensions doubled with respect to the default one, obtaining almost the same results but as counter effect, more overfitting issues, due to the increased complexity of the model. Therefore, we kept the default hyper-parameters as the initial network.

Moreover, we tested the Retentive Network architecture with different numbers of layers and heads to observe how these parameters affect memory consumption and inference time. As expected and reported in Table 4.8, layers and heads directly influence the model's complexity and hardware requirements. In particular, we observed a more pronounced influence of the head parameter compared to the number of layers.

Layers	**Heads**	**Training Memory (GB)**	**Inference (ms)**
2	4	2.4	31
4	4	3.1	45
4	8	4.4	62
8	4	4.2	66
8	8	6.7	100

Table 4.8: Comparison among RetNet layers and heads values

Residual Connection We evaluated the model performance by introducing a residual connection between two network modules. Specifically, we tested two different residual connections: one between the Range Embedding Module and Semantic Head, and the other between the input and output of the Retentive Network architecture.

The first residual connection significantly improved the results on the validation set,

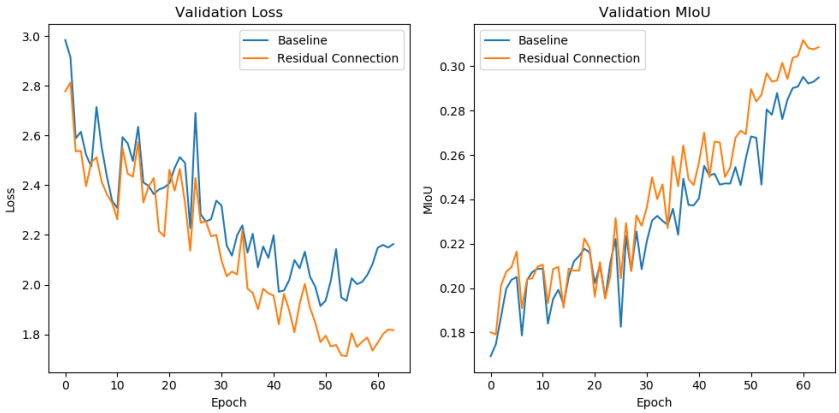


Figure 4.15: Residual connection effect on validation metrics

while the second implementation did not seem to affect the performance, producing almost identical outcomes. The combination of the REM output with the resized output of RetNet, seems to help the network learning procedure, facilitating the flow of information through the network. In addition, the features learned by REM may help the bi-linear interpolation which reconstruct the range image size from a small amount of features due to the patch extraction process, especially when a large kernel or stride value is employed. Therefore, this may be a reason why residual connection between the input and output of RetNet does not improve the results, obtaining almost the same outcome of the baseline approach. Given the obtained results, we opted not to apply both residual connections, as only the first one significantly improves the model performance.

It is interesting to note that the introduction of the residual connection between REM and Head slightly reduces the overfitting, while promoting better convergence and improving the mIoU metric on the validation set, as shown in Figure 4.15.

Data Augmentation Data augmentation is a crucial aspect of the proposed method, applied directly to each input rather than generating new training data. This technique involves making slight modifications to the existing point cloud inputs during training, enhancing the model robustness and generalization capabilities. We exploited four different data augmentation techniques involving scaling, rotation, jittering and flip, described in Pseudocode 2. RangeRet works on range image, thus we decided to employ data augmentation on point clouds, affecting only the x, y coordinates, without altering the height coordinate z which is crucial in the rasterization process.

Model	Augmentation	Train mIoU	Val mIoU	Point Cloud mIoU
RangeRet		**46.96**	**29.54**	39.6
RangeRet	✓	42.27	29.50	**40.3**

Table 4.9: Data Augmentation training

The four techniques are applied to each input in the specified order, altering the point cloud, and, more importantly, influencing the appearance of the corresponding range image. Consequently, we investigated the impact of data augmentation during the training process to understand its effect on loss trends and model performance. Additionally, we utilized data augmentation as a regularization technique to expose the model to various scenarios and enhance generalization performance. Given the observed results indicating that the residual connection helps mitigate the overfitting issue, we used the model without residual connection to asses the usefulness of data augmentation. Figure 4.16 shows the

training process of the final model, without data augmentation. As can be seen, it is significantly influenced by overfitting in the last 10 epochs even though this does not seems to highly affect the validation mIoU. The evaluation on sequence 08 of SemanticKITTI gives 39.6% of mIoU.

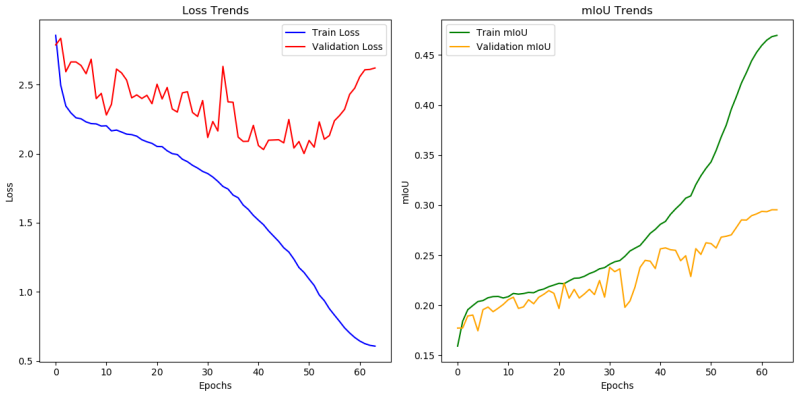


Figure 4.16: Training without data augmentation

Applying data augmentation, the training process benefits from the loss point of view, mitigating the overfitting issue as represented in Figure 4.17. Even though the training mIoU [7] is lower, the validation metric is almost the same of the previous tests, while the overall mIoU between point cloud on sequence 08 of SemanticKITTI is slightly higher (40.3%). This shows how data augmentation helps the model in improving the generalization ability, slightly reducing the performance on the training set but maintaining or even increasing the outcomes on the validation set.

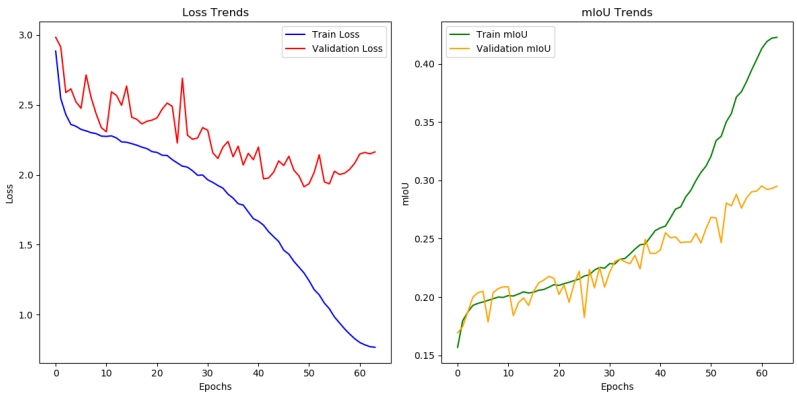


Figure 4.17: Training with data augmentation

However, data augmentation has its own drawback since it requires more computational time to apply transformations on the input point clouds. Indeed, the number of training items processed decrease from $9.5it/s$ to almost $8.9it/s$, increasing the duration of the whole training process.

[7] Train mIoU and Val mIoU are computed on range images, Point Cloud mIoU is computed between point clouds on the whole sequence 08 of SemanticKITTI

Algorithm 2 Data Augmentation

$SCALE\_RATE = 0.05$
$JITTER\_RATE = 0.3$
$DROP\_RATE = 0.1$

function RandomScale(scan, rate)
    $scale \leftarrow \text{uniform}(1 - \text{rate}, 1 + \text{rate})$
    $\text{scan}[:, 0] \leftarrow \text{scan}[:, 0] \times \text{scale}$
    $\text{scan}[:, 1] \leftarrow \text{scan}[:, 1] \times \text{scale}$
    return scan
end function

function GlobalRotation(scan)
    $\text{angle} \leftarrow \text{deg2rad}(\text{random}() \times 360)$
    $\cos, \sin \leftarrow \cos(\text{angle}), \sin(\text{angle})$
    $\text{R} \leftarrow \begin{bmatrix} \cos & \sin \\ -\sin & \cos \end{bmatrix}$
    $\text{scan}[:, :2] \leftarrow \text{dot}(\text{scan}[:, :2], \text{R})$
    return scan
end function

function RandomJitter(scan, rate)
    $\text{jitter} \leftarrow \text{clip}(\text{normal}(0, \text{rate}, 2), -\text{rate}, \text{rate})$
    $\text{scan}[:, :2] \leftarrow \text{scan}[:, :2] + \text{jitter}$
    return scan
end function

function RandomFlip(scan)
    $\text{flip} \leftarrow \text{choice}(4, 1)$
    if flip == 1 **then**
        $\text{scan}[:, 0] \leftarrow -\text{scan}[:, 0]$
    else if flip == 2 **then**
        $\text{scan}[:, 1] \leftarrow -\text{scan}[:, 1]$
    else if flip == 3 **then**
        $\text{scan}[:, :2] \leftarrow -\text{scan}[:, :2]$
    end if
    return scan
end function

Post Processing To improve and refine the results on the point cloud while addressing errors arising from the re-projection procedure, we tested the commonly used k-Nearest Neighbors (kNN) post-processing technique introduced in [28]. By comparing the raw and refined results of various approaches, we consistently observed improvements (Table 4.10). Additionally, we identified a correlation between the raw mIoU and the enhanced mIoU with kNN post-processing, indicating that higher basic results correspond to a greater probability of enhancing the results. Figure 4.18 shows how the k-NN post-processing technique significantly improves the evaluation results when applied to good starting predictions. Thus, with an initial mIoU value greater than 40%, the increment brought by k-NN is 3 percentage points, while with basic results around 30%, it enhances the results by not even 1 percentage point.

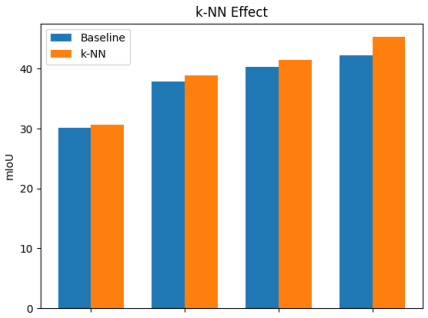


Figure 4.18: Post-processing improvements

Next, we compared the results obtained with different k values, examining the impact of this parameter on the IoU of individual classes and the overall mIoU. The values reported in Table 4.10 show that as we enlarge the neighborhood, the results slightly improve with respect to the basic predictions. In particular, $k = 7$ seems to achieve better performance, increasing the IoU of almost all classes and obtaining greater results than the standard value $k = 5$. Higher values such as $k = 9$ do not seem to further improve the overall metric.

kNN	car	bicycle	motorcycle	truck	other vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	**mIoU (%)**
-	82.3	14.9	13.7	47.0	14.4	24.3	39.9	0.0	**90.2**	31.8	73.1	0.3	73.0	38.1	79.4	43.9	71.0	32.9	32.7	42.3
3	86.2	15.8	14.8	47.2	14.7	25.9	44.3	0.0	90.2	**31.9**	**73.3**	**0.4**	77.9	39.6	81.6	47.8	71.7	43.2	35.7	44.3
5	88.3	16.9	15.6	47.8	**14.9**	27.6	46.6	0.0	90.1	31.8	73.2	**0.4**	79.3	40.2	82.4	49.7	**72.0**	46.5	36.8	45.3
7	**88.6**	17.6	15.9	48.0	**14.9**	28.3	47.3	0.0	89.9	31.6	73.0	**0.4**	79.9	40.5	82.7	50.7	**72.0**	**47.2**	**37.2**	**45.6**
9	88.4	**18.1**	**16.0**	**48.2**	**14.9**	**28.7**	**47.5**	0.0	89.7	31.4	72.7	**0.4**	**80.2**	**40.7**	**82.8**	**51.3**	71.9	**47.2**	37.1	**45.6**

Table 4.10: Results with kNN post-processing

Additionally, we investigated how the inference time is influenced by varying k. As expected and reported in Table 4.11, an increase in the k value corresponds to a longer inference time, necessitating more computational resources for the post-processing technique.

k-NN	Baseline	3	5	7	9
Inference (ms)	45	47	48	49	52

Table 4.11: Post-porcessing effect on inference time

Regarding the mIoU values of individual classes, it is evident that almost all classes experience improvements with the application of k-NN post-processing. This is particularly notable for large objects closer to the sensor, such as cars and buildings, or objects more affected by re-projection issues, such as poles, cyclists, and trunks, owing to their distinct shapes. However, highly correct predictions, such as roads, may be slightly lowered in favor of rarer classes in order to improve the overall accuracy and mIoU.

Training Paradigms Inspired by [21], we tested a training paradigm to address memory constraints while utilizing more patches to enhance network performance. The paradigm requires to divide the range image along the horizontal dimension obtaining a number of sub-images of size $H \times W_{train}$. Each sub-image is fed to the network for the learning process and at the end they are stacked together to compare the predictions with the true labels. In this way the model learns from more examples without increasing

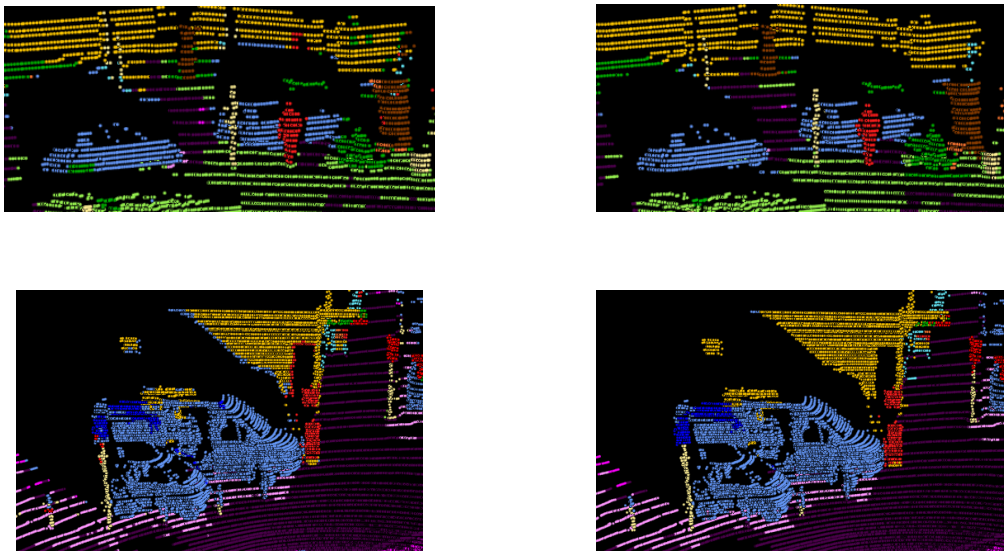


Figure 4.19: Comparison between raw predictions (left) and refined with k-NN (right)

the memory consumption. However, both the training and inference time are affected by slower run-time. We tested this paradigm with two settings, employing 4 sub-images:

- Setting 1: range images of size 64×2048 with patches 7×7 and stride 4;
- Setting 2: range images of size 64×1024 with patches 4×4 and stride 3.

We observed slightly better performance from both approaches compared to the basic training procedure. However, it resulted in higher computational time requirements during both training and inference phases. As these approaches did not yield substantial improvements while increasing the time to infer a sample, we decided not to further investigate them. Nevertheless, these approaches were instrumental in understanding that the number of patches extracted from the range image can impact network performance due to the larger number of samples fed to the Retentive Network architecture.

4-Stage Approach Pyramid Vision Transformer [41] and RangeFormer [21] propose a pyramidal structure to enhance the performance on dense prediction tasks. In contrast, our approach involves utilizing a single stage of the Retentive Network block. This choice is influenced by the memory constraints encountered during the network development process, as well as considerations regarding training and inference times. A more complex model composed of a larger number of stages was avoided to mitigate these constraints. Moreover, as demonstrated in [1], it is possible to achieve great results even with a single stage, favoring faster inference times and a more lightweight training process.

Specifically, we experimented with a 4-stage approach in the early development of the network, observing a reduction in training time to 4.5 iterations per second, an inference time of 125 ms for processing a single sample and no significant improvements on the results. Consequently, we abandoned this approach in favor of the single-stage method.

Data Hungry During the experimental phase, we worked with subsets of various sizes to analyze the network structure, determine which modules to apply, and compare different implementations. The primary tests were conducted using the SemanticKITTI subset of 5000 scans, and we evaluated the results on the entire sequence 08 of SemanticKITTI [8], which comprises 4071 point clouds. Consequently, we examined the influence of the amount of training data on the training procedure, assessing the model ability to generalize on the complete validation set of SemanticKITTI.

[8]The mIoU reported in Table 4.12 does not consider k-NN post-processing technique

Subset	Ratio (%)	mIoU (%)
100	0.5	27.5
1000	5.2	37.3
5000	26.1	42.3
Full	100.0	43.5

Table 4.12: Training data amount influence over generalization ability

As expected, employing more data enhances the model generalization ability, leading to higher results on the validation set of SemanticKITTI. Furthermore, it is interesting to note that using only 1000 scans, which correspond to 5.2% of the entire dataset, produces commendable results, not significantly distant from those obtained with 5000 input point clouds or the entire dataset. This indicates that the model can effectively learn key elements in the range image even when trained on a small amount of data, resulting in robust generalization performance. However, when utilizing a small subset, such as 100 point clouds, the model exhibits limitations, particularly in predicting rare classes that are further under-represented in the subset.

4.5 Results

We evaluated RangeRet on the validation and test sets of SemanticKITTI [3] to analyze the performance compared to different scenarios and other methods. The predictions on the test set were submitted to the competition website [9] for the evaluation since labels for the test set sequences are not public.

In Table 4.13, various range-view-based approaches that achieve real-time performance on the validation set of SemanticKITTI are reported, considering two different range image sizes: 64×2048 and 64×1024. In the first scenario, RangeNet++ [28] achieves the best results, while our proposed method demonstrates favorable performance on a few classes. The other listed approaches do not achieve comparable results, likely due to their smaller network size. However, with images of size 64×1024, the situation changes significantly, with RangeRet obtaining significantly better results compared to RangeNet++ in almost all classes.

Approach	Size	car	bicycle	motorcycle	truck	other vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	**mIoU (%)**	Acc (%)
SqueezeSeg-CRF [42]	64×2048	72.2	13.7	9.8	2.8	9.1	18.2	25.9	**0.1**	84.6	17.6	62.8	**0.8**	68.1	19.5	69.5	24.7	66.0	12.2	22.9	31.6	76.4
SqueezeSegV2-CRF [43]		83.4	17.5	35.2	24.5	**26.0**	26.6	43.5	0.0	90.6	30.9	73.6	0.7	74.3	41.8	74.2	34.7	68.8	13.4	24.4	41.3	83.8
RangeNet++ [28]		**91.0**	**25.0**	**47.1**	40.7	25.5	**45.2**	**62.9**	0.0	**93.8**	**46.5**	**81.9**	0.2	**85.8**	**54.2**	84.2	**52.9**	72.7	**53.2**	**40.0**	**52.8**	**90.1**
RangeRet (Our)		89.6	24.2	14.1	**46.1**	15.2	30.7	38.6	0.0	91.6	35.7	76.2	0.3	82.0	40.1	**84.4**	52.7	**72.8**	50.9	36.9	46.4	88.6
RangeNet++ [28]	64×1024	87.1	8.3	4.1	18.3	9.7	8.6	36.2	0.0	85.0	14.7	69.5	0.0	**81.1**	**50.1**	81.6	30.4	**71.9**	41.5	30.7	38.4	86.5
RangeRet (Our)		**88.7**	**19.4**	**22.1**	**58.7**	**19.2**	**29.4**	**47.4**	**0.0**	**89.4**	**31.9**	**72.6**	**0.3**	80.0	42.8	**82.9**	**50.4**	71.4	**46.2**	**37.6**	**46.9**	**87.5**
RangeFormer [21]	64×1920	95.4	58.5	73.7	91.3	73.1	76.5	88.7	0.0	95.0	56.5	82.0	10.0	88.7	65.8	86.8	67.2	73.7	64.3	52.2	**67.9**	-

Table 4.13: Comparison among range view approaches on SemanticKITTI val set

In order to analyze the network behavior in various 3D scenarios, we compared the results on the test set of SemanticKITTI. In this case, we struggled to achieve results comparable to RangeNet++, narrowing the gap in the 64×2048 configuration but producing slightly worse results with 64×1024 range images. This discrepancy may be attributed to the diverse point cloud scenarios present in the test set compared to the validation set. Moreover, it is important to note that RangeNet++ has significantly more parameters (50.3M) compared to our approach, RangeRet (1.7M), as reported in Table 4.15.

[9] SemanticKITTI competition website: https://codalab.lisn.upsaclay.fr/competitions/6280

Approach	Size	car	bicycle	motorcycle	truck	other vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	**mIoU (%)**
SqueezeSeg [42]	64×2048	68.8	16.0	4.1	3.3	3.6	12.9	13.1	0.9	85.4	26.9	54.3	4.5	57.4	29.0	60.0	24.3	53.7	17.5	24.5	29.5
SqueezeSeg-CRF [42]		68.3	18.1	5.1	4.1	4.8	16.5	17.3	1.2	84.9	28.4	54.7	4.6	61.5	29.2	59.6	25.5	54.7	11.2	36.3	30.8
SqueezeSegV2 [43]		81.8	18.5	17.9	13.4	14.0	20.1	25.1	3.9	88.6	45.8	67.6	17.7	73.7	41.1	71.8	35.8	60.2	20.2	36.3	39.7
SqueezeSegV2-CRF [43]		82.7	21.0	22.6	14.5	15.9	20.2	24.3	2.9	88.5	42.4	65.5	18.7	73.8	41.0	68.5	36.9	58.9	12.9	41.0	39.6
RangeNet++ [28]		**91.4**	25.7	**34.4**	**25.7**	**23.0**	**38.3**	**38.8**	**4.8**	**91.8**	**65.0**	**75.2**	**27.8**	**87.4**	**58.6**	**80.5**	55.1	**64.6**	**47.9**	**55.9**	**52.2**
RangeRet (Our)		89.4	**27.5**	14.7	20.0	14.0	31.1	31.4	2.1	89.3	57.2	69.8	11.9	87.1	54.0	78.8	**55.5**	62.5	45.0	50.9	47.0
RangeNet++ [28]	64×1024	**90.3**	20.6	**27.1**	**25.2**	**17.6**	**29.6**	**34.2**	**7.1**	**90.4**	**52.3**	**72.7**	**22.8**	83.9	**53.3**	**77.7**	**52.5**	**63.7**	**43.8**	47.2	**48.0**
RangeRet (Our)		88.0	**26.9**	19.0	22.8	13.6	28.0	32.2	3.6	87.8	49.8	67.1	10.9	**85.2**	51.0	76.5	52.2	57.0	40.5	**47.8**	45.2
RangeFormer [21]	64×1920	96.7	69.4	73.7	59.9	66.2	78.1	75.9	58.1	92.4	73.0	78.8	42.4	92.3	70.1	86.6	73.3	72.8	66.4	66.6	**73.3**

Table 4.14: Evaluation on SemanticKITTI test set

In Table 4.15, we present a comparison between RangeRet and other methods across different modalities [10]. The comparison underscores the significant advantages of RangeRet over voxel-based approaches. These voxel-based methods exhibit higher memory usage during inference and require more computation time to process a single scan, rendering them unsuitable for real-time scenarios. In contrast, RangeRet, with its lightweight architecture, operates efficiently with minimal memory requirements, achieving real-time performance by processing nearly 20 scans per second. Notably, approaches like Cylinder3D, PVKD, and 2DPASS have parameter counts approximately 30 times greater than RangeRet, while their inference speeds are 6 and 15 times slower, respectively. However, it is essential to acknowledge that RangeRet's main drawback lies in its performance, as the results are comparatively lower than those achieved by other state-of-the-art approaches. Regardless, the obtained results are nearly similar to those produced by RangeNet++, despite utilizing a significantly lower number of parameters and being faster.

Approach	**Size**	**Inference (ms)**	**Memory (GB)**	**mIoU (%)**
SqueezeSeg-CRF [42]	0.91M	**23**	**1.2**	30.8
SqueezeSegV2-CRF [43]	0.93M	29	**1.2**	39.6
RangeNet++ [28]	50.3M	84	1.4	52.2
Cylinder3D [50]	55.8M	330	3.1	68.9
PVKD [17]	55.8M	746	7.2	71.2
2DPASS [48]	45.6M	285	5.4	72.9
RangeFormer [21]	24.3M	-	-	**73.3**
RangeRet (Our)	1.7M	49	2.0	45.2

Table 4.15: Trade-off comparisons of efficiency and accuracy on SemanticKITTI test set

[10] Trade-off tests were conducted on an NVIDIA GeForce RTX 2080

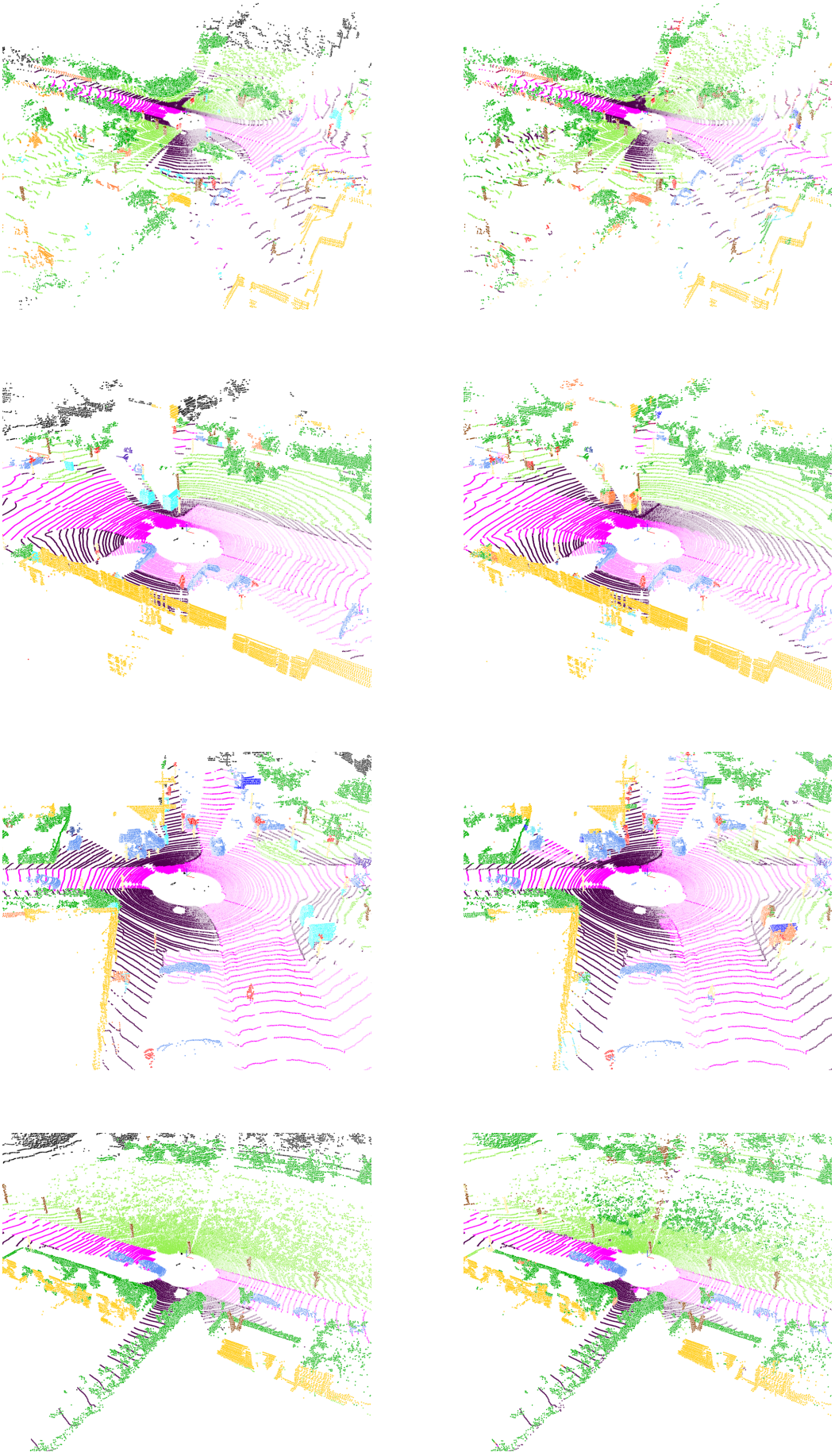


Figure 4.20: Qualitative comparisons between ground truth (left) and predictions (right)

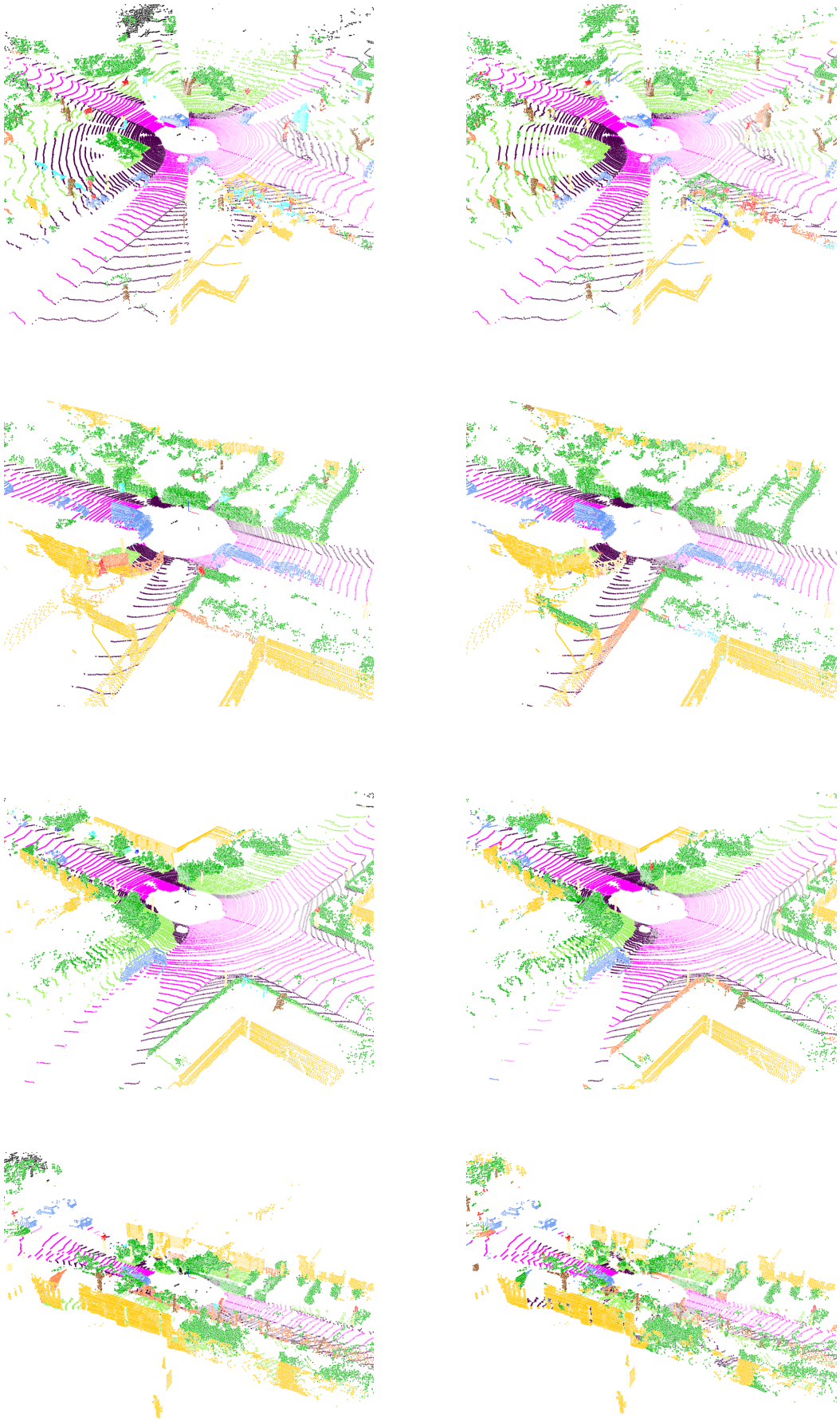


Figure 4.21: Qualitative comparisons between ground truth (left) and predictions (right)

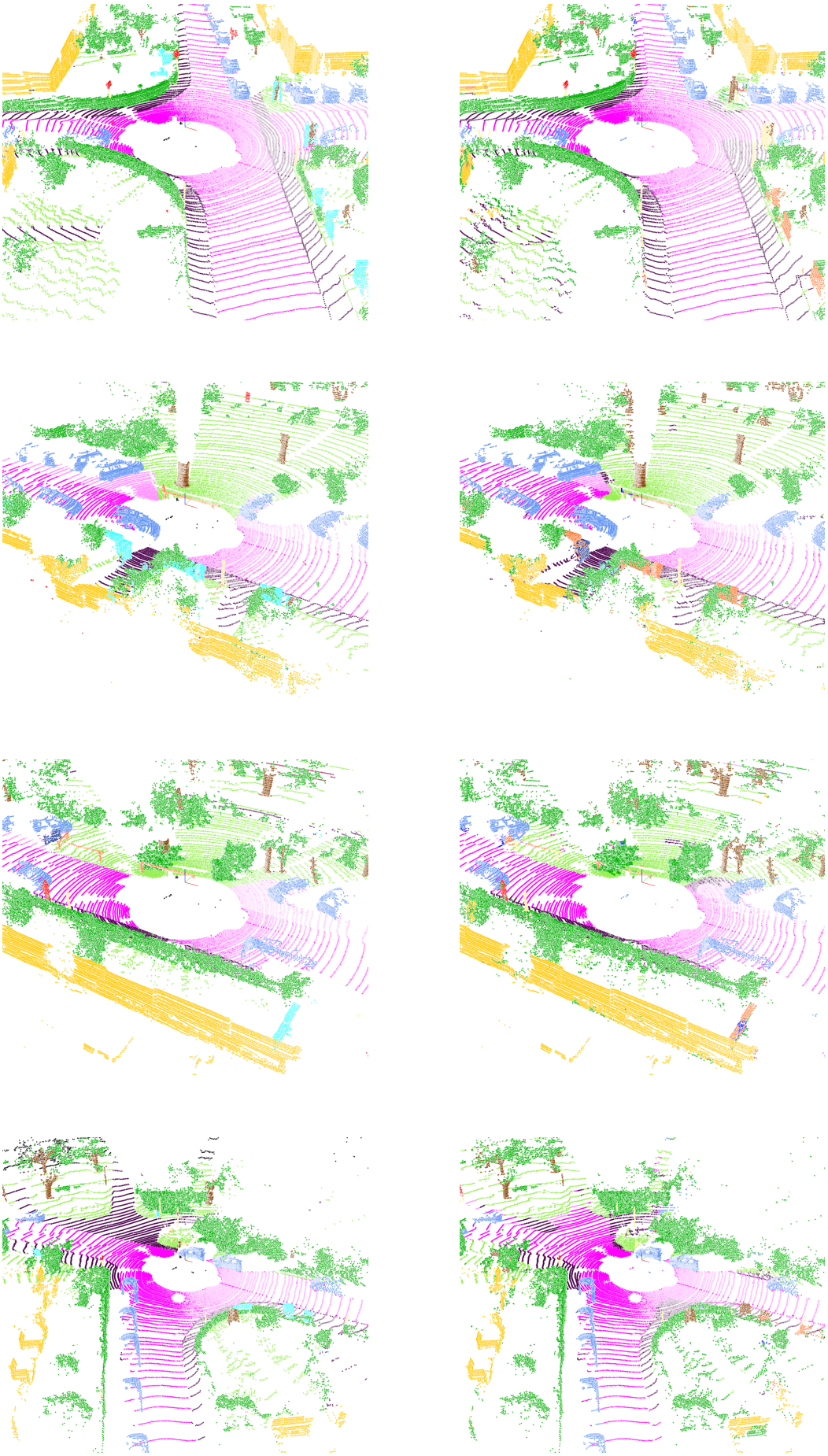


Figure 4.22: Qualitative comparisons between ground truth (left) and predictions (right)

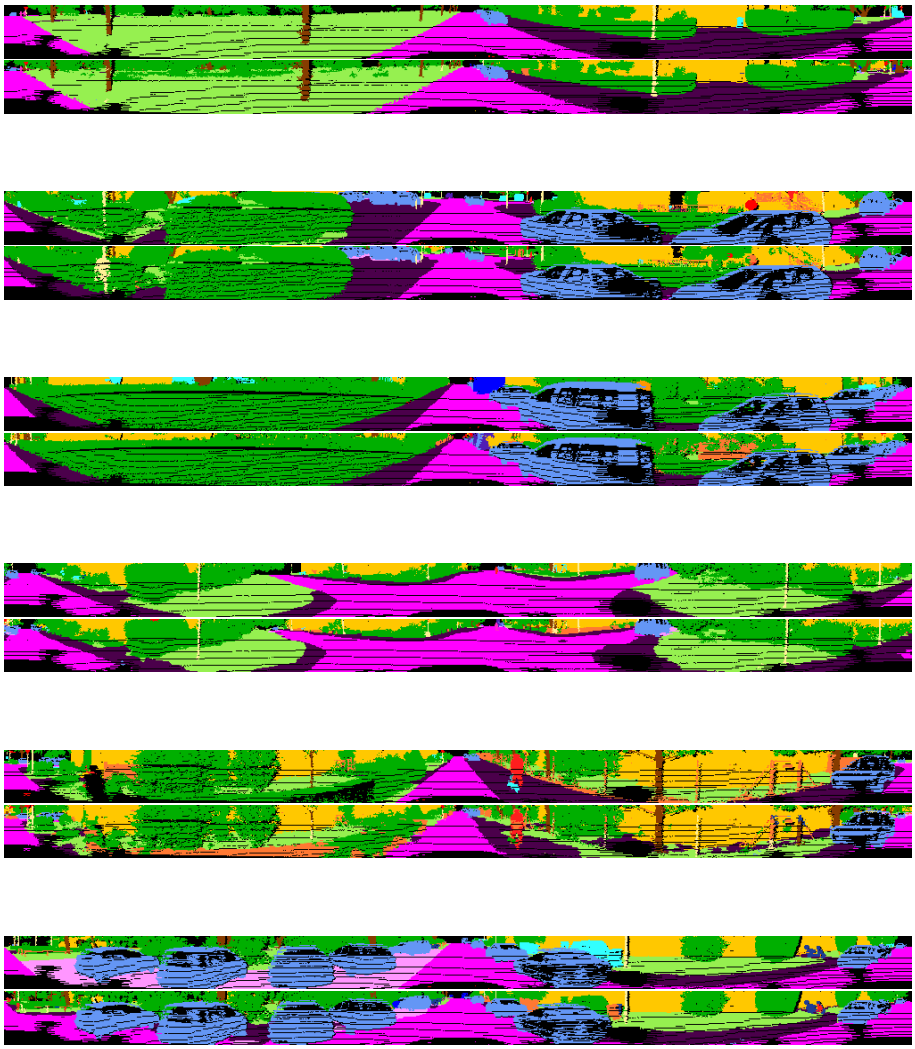


Figure 4.23: Qualitative comparisons between range images ground truth (top) and predictions (bottom)

Chapter 5

Ablation Study

In this chapter, we conduct a comprehensive ablation study to deconstruct the proposed method **RangeRet**, isolating and analyzing key components. Ablation studies are crucial for understanding the contribution of specific elements to the model performance. By selectively disabling or altering specific modules and techniques, we aim to discern their individual impact on the model performance and improvements. Starting from the basic model with MLP-based Range Embedding Module (REM) and Vision Transformer (ViT), we discuss the results obtained introducing a convolution-based REM, the Retentive Network architecture, the new decay matrix and a residual connections which leads to the final creation of RangeRet.

The ablation study experiments are performed on the SemanticKITTI subset containing about 5000 scans and evaluated on the complete sequence 08 of SemanticKITTI which consists of 4071. All the network setups and further changes share the same training configuration as follows:

- Loss Functions: Cross-Entropy, Lovasz Loss and Focal Loss
- Optimizer: AdamW with $lr = 1e-3$
- Scheduler: OneCycle with $max\_lr = 1e-2$ and cycle percentage of 0.02
- Epochs: 64

Baseline Implementation Inspired by [1] and [21] which perform LiDAR Semantic Segmentation on range images employing Vision Transformers, we set a baseline approach for our ablation studies. This consists of a Range Embedding Module composed of three Linear layers, i.e. Multi-Layer Perceptrons as in [21], a group of Vision Transformers blocks and a Semantic Head which uses Linear layers to produce the predictions. The pyramidal structure is abandoned in favor of a single stage of Transformers blocks as in [1], due to the GPU chip-set employed in the experiments and its constrained amount of memory. The Range Embedding Module maps the 5 input features into $64, 128$ and then again to 128, while the Semantic Head module presents a mapping from 128 to 64 and finally the number of classes d^{cls}. Four Transformers block complete the network, each with 4 heads. For fair comparison with the Retentive Network architecture that will be later introduced in the network, we set xPos [34] as relative position embedding and we double the value V matrix dimension with respect to the query Q and key K matrices. In addition, the Feed-Forward Network hidden size in the Transformers blocks is twice as the hidden dimension of the tokens.

This model has a number of parameters equals to $1.49M$ parameters. We trained this baseline network configuration with the settings listed above and observed the results.

Range Embedding Module As previously introduced in the method section and as suggested in [46], we observe the results introducing convolutional layers in the Range Embedding Module, in place of Linear layers. Specifically, these layers maps the 5 input features into 32, 64 and 128 using convolutional kernels of size 3×3 with stride 1 and padding 1 to preserve the original image size. This change introduces more network parameters for a total number of $1.56M$. Both training and inference processes benefit from this modification, since convolutional layers are suitable in the two-dimensional domain of images. In particular the inference time increases of a notable value from $21.6it/s$ to $23.0it/s$ while the memory consumption slightly decreases in both operations.

	Training		Inference	
REM	Iterations/s	Memory (GB)	Iterations/s	Memory (GB)
Linear	7.7	4.1	21.6	1.7
Conv	**8.2**	**4.0**	**23.0**	**1.6**

Table 5.1: Range Embedding Module time performance

Regarding the model performance, we also have a boost in the mIoU metric using convolutional layers in the REM as shown in Table 5.2.

REM	**Architecture**	**Params**	**mIoU (%)**
Linear	Transformers	1.49M	30.1
Conv	Transformers	1.56M	**37.3**

Table 5.2: Ablation study on Range Embedding Module

Retentive Network The introduction of Retentive Network [35] in the network corresponds to the key change in the network architecture. Even though the two models have almost the same overall structure, they differ in the Attention/Retention mechanism, where RetNet remove the Softmax operator and replace it with a pre-computed weight decay and casual masking matrix. The new model with Retentive Network has a slightly higher number of parameters ($1.69M$) due to the introduction of weight matrices for the computation of G in each block. The parameters size configuration of Retentive Network is the same as Transformers, as previously introduced, to perform a fair comparison between the two models. With the introduction of Retentive Network, the new architecture consumes significantly less memory in the training procedure, while using some additional memory in the inference phase. In addition, also the training process benefits from this improvement, increasing the number of sample processed as shown in Table 5.3. As drawback, both the time and memory usage in inference are slightly worse with respect to the model composed of Transformers blocks.

	Training		Inference	
Architecture	Iterations/s	Memory (GB)	Iterations/s	Memory (GB)
Transformers	8.2	4.0	**23.0**	**1.6**
Retentive Network	**8.9**	**3.1**	22.2	1.9

Table 5.3: Comparison between Transformers and Retentive Network time performance

For what concerns the evaluation of the two architecture on the validation set of SemanticKITTI, the result are very similar as shown in Table 5.4. The Retentive Network model employs the basic decay matrix D presented in [35] for the Natural Language Process field, only considering influence from the past.

REM	Architecture	Augmentation	Params	mIoU (%)
Conv	Transformers		1.56M	**37.0**
Conv	Retentive Network		1.69M	36.6
Conv	Transformers	✓	1.56M	37.3
Conv	Retentive Network	✓	1.69M	**37.8**

Table 5.4: Ablation study on Core Architecture

As introduced in this section, the ablation study employs as default data augmentation on point cloud. However, in Table 5.4 we also include the evaluation results without data augmentation, since we discovered different outcomes, mainly influenced by augmentation techniques. Observing the results we may state that the two networks produce almost the same metric values, since there is no evident winner. Retentive Network performs better than Transformers when data augmentation is employed, while it obtains lower results without augmentation. However the mIoU differences between them is in a strict range and the overall gain is not remarkable that suggest the usage of one over the other.

The main improvements remain the training memory consumption (-22.5%) and process time ($+8.5\%$), which allow the training procedure to be handle by cheaper GPU hardware and requiring smaller amount of memory.

Decay Matrix With the observations derived by introducing Retentive Network in the architecture, we focused on improving the performance of the model with respect to the Transformers. Therefore we analyzed the decay matrix, and inspired by the alternatives introduced in [10], we tested various configuration of the matrix. The basic matrix D contains only zeros in the upper triangle in order to ensure future that information from future tokens influence the current token. However this is an issue relative to the NLP field and do not apply in Computer Vision task where each patch can leverage the information from all other patches. Thus we introduce a new decay matrix D which uses an exponential Manhattan-distance-based weight distribution, inspired by the one proposed in [10]. This configuration does not require more memory, while unlocking the upper triangle to improve the learning procedure as shown in Table 5.5 .

Decay Matrix	Model Params	mIoU (%)
Standard	1.69M	37.8
Our	1.69M	**40.3**

Table 5.5: Ablation study on decay matrix D

As can be seen, using all the values in the decay matrix, removing any mask operation, and assigning the correct weights among patches, allows the model to improve the results without side effects on the memory consumption or inference time. The gain in terms of performance is evident, since the model goes from a mIoU of 37.8% to 40.3%, which can even be improved with post-processing techniques.

Residual Connections The last contributions to the model architecture is the introduction of a single residual connection between the last layer of the Range Embedding Module and the input of Semantic head, subsequently to the bi-linear interpolation. The number of parameters is not altered since the residual connection consists of a simple addition between tensors and it does not employ any learnable parameters. However, the computational time may be slightly affected by this operation as we observed during the experiments, with a smaller number of iterations per second performed in the training procedure (Table 5.6).

The residual connection helps the network learning procedure, especially when a low

	Training		Inference	
Residual Connection	Iterations/s	Memory (GB)	Iterations/s	Memory (GB)
	8.9	3.1	22.2	1.9
✓	8.8	3.1	22.2	1.9

Table 5.6: Comparison on the effect of a residual connection in the time performance

number of patches is fed to the Retentive Network architecture. The addition of features learned from the Range Embedding Module to the features produced by the Retentive Network, interpolated to the original range image size, improves the predictions produced by the Semantic Head module by an evident factor. The mIoU increase from 40.4% to 42.3% as reported in Table 5.7.

Architecture	**Residual Connection**	**Params**	**mIoU (%)**
RangeRet		1.69M	40.3
RangeRet	✓	1.69M	**42.3**

Table 5.7: Ablation study on the residual connection

Furthermore, it's noteworthy that the residual connection not only leads to superior evaluation results but also produces significant improvement when a post-processing refinement technique is applied.

Comparisons We now compare the previous improvements in a complete overview on the different changes in the modules and how they affect both the training procedure in terms of memory and computation time, and the results on the validation set.

REM	**Architecture**	**Decay Matrix**	**Residual Connection**	**Params**	**mIoU (%)**
Linear	Transformers			1.49M	30.1
Conv	Transformers			1.56M	37.3
Conv	RetNet	Standard		1.69M	37.8
Conv	RetNet	Our		1.69M	40.3
Conv	RetNet	Our	✓	1.69M	**42.3**

Table 5.8: Ablation Study

Considering the results presented in Table 5.8, it is evident that the introduced modifications and modules contribute to enhancing the generalization ability of the network. The total number of parameters undergoes a note-worthily insignificant increase, progressing from the baseline model (1.49M) to the final architecture (1.69M). However, the incorporation of the new decay matrix and the residual connection, which do not add learnable parameters, significantly aids the network in achieving great results that can be even improved using a post-processing technique.

Among all the changes, the Range Embedding Module (REM) seems to contribute significantly to the results, enhancing both the generalization ability and the time complexity. Indeed, it is responsible for learning pixel features from the input range image and subsequently passing them to the Vision Embedding Module—a crucial step to provide features for the Retentive Network architecture, which forms the core of the model. As expected, convolutional layers are more suitable when dealing with images than Linear layers, better leveraging the geometrical information of 2D inputs.

In addition, the introduction of the Retentive Network with the new configuration of the decay matrix appears to better leverage the two-dimensional input, replacing the Softmax operation employed in the standard Attention and resulting in another improvement. However, this enhancement comes at the cost of a slightly increased memory requirement, albeit with a reduction in inference time. Further experiments with different data and datasets may be needed to better assess, in general, the strengths or superiority of a Retention operation—almost similar to a linear Attention with a pre-computed decay matrix—compared to the well-established Softmax operator when dealing with image inputs or other two-dimensional inputs.

Chapter 6

Conclusion

This Master's Thesis introduced **RangeRet**, a novel approach for 3D LiDAR Semantic Segmentation in the field of autonomous driving, leveraging range images and Retentive Networks to achieve real-time performance. The results highlighted the efficacy of this lightweight network in effectively managing limited memory resources while attaining commendable results and high inference throughput, even on cost-effective hardware chipsets.

The integration of Retentive Networks, replacing the well-established Transformers, demonstrated substantial equivalence between the two architectures. Notably, the Retentive Network exhibited superior memory usage management during the training process. However, the novel contributions to the Retention score computation revealed an enhanced ability to capture geometrical and spatial information in two-dimensional inputs, leading to significant improvements.

The comprehensive ablation study, encompassing the evaluation and discussion of each module, outlined the development of the network and its key architectural elements. These elements contribute to notable enhancements in terms of accuracy, memory consumption, and inference efficiency.

While Retentive Networks represent a novel architecture, additional studies, especially within the computer vision field, are imperative. Future exploration aims to extend the application of RangeRet to other 3D LiDAR tasks, such as object detection, classification, and panoptic segmentation. Adaptations to handle diverse datasets, including those from different LiDAR sensors like nuScenes [5] and PandaSet [45], or in weakly annotated scenarios as seen in ScribbleKITTI [38], will be a crucial avenue of investigation. Furthermore, the network will be applied to tasks such as 2D image classification or segmentation, using datasets like Cityscapes [8].

In addition, we would like to further explore the main contribution brought by the Retentive Network, i.e., the recurrent paradigm, in the vision field. The aim is to significantly improve the inference time and facilitate real-time applications in real-world scenarios.

In addressing these new challenges, the ongoing objective is to enhance network performance, striving for improved results while maintaining real-time usage capabilities.

Bibliography

[1] Angelika Ando et al. *RangeViT: Towards Vision Transformers for 3D Semantic Segmentation in Autonomous Driving*. 2023. arXiv: 2301.10222 [cs.CV].

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[3] Jens Behley et al. *SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences*. 2019. arXiv: 1904.01416 [cs.CV].

[4] Maxim Berman, Amal Rannen Triki, and Matthew B. Blaschko. *The Lovász-Softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks*. 2018. arXiv: 1705.08790 [cs.CV].

[5] Holger Caesar et al. *nuScenes: A multimodal dataset for autonomous driving*. 2020. arXiv: 1903.11027 [cs.LG].

[6] Hui-Xian Cheng, Xian-Feng Han, and Guo-Qiang Xiao. *CENet: Toward Concise and Efficient LiDAR Semantic Segmentation for Autonomous Driving*. 2022. arXiv: 2207.12691 [cs.CV].

[7] Spconv Contributors. *Spconv: Spatially Sparse Convolution Library*. https://github.com/traveller59/spconv. 2022.

[8] Marius Cordts et al. *The Cityscapes Dataset for Semantic Urban Scene Understanding*. 2016. arXiv: 1604.01685 [cs.CV].

[9] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[10] Qihang Fan et al. *RMT: Retentive Networks Meet Vision Transformers*. 2023. arXiv: 2309.11523 [cs.CV].

[11] Biao Gao et al. *Are We Hungry for 3D LiDAR Data for Semantic Segmentation? A Survey and Experimental Study*. 2020. arXiv: 2006.04307 [cs.CV].

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[13] Yulan Guo et al. *Deep Learning for 3D Point Clouds: A Survey*. 2020. arXiv: 1912.12033 [cs.CV].

[14] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[15] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG].

[16] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580 [cs.NE].

[17] Yuenan Hou et al. *Point-to-Voxel Knowledge Distillation for LiDAR Semantic Segmentation*. 2022. arXiv: 2206.02099 [cs.CV].

[18] Qiangui Huang, Weiyue Wang, and Ulrich Neumann. *Recurrent Slice Networks for 3D Segmentation of Point Clouds*. 2018. arXiv: 1802.04402 [cs.CV].

[19] A. Jhaldiyal and N. Chaudhary. "Semantic segmentation of 3D LiDAR data using deep learning: a review of projection-based methods". In: *Applied Intelligence* 53 (2023), pp. 6844–6855. DOI: 10.1007/s10489-022-03930-5.

[20] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[21] Lingdong Kong et al. *Rethinking Range View Representation for LiDAR Segmentation*. 2023. arXiv: 2303.05367 [cs.CV].

[22] Ying Li et al. *Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review*. 2020. arXiv: 2005.09830 [cs.CV].

[23] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[24] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. arXiv: 2201.03545 [cs.CV].

[25] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[26] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983 [cs.LG].

[27] Shuming Ma et al. "TorchScale: Transformers at Scale". In: *CoRR* abs/2211.13184 (2022).

[28] Andres Milioto et al. "Rangenet++: Fast and accurate lidar semantic segmentation". In: *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2019, pp. 4213–4220.

[29] Charles R. Qi et al. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: 1612.00593 [cs.CV].

[30] Charles R. Qi et al. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV].

[31] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

[32] Leslie N. Smith. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. 2018. arXiv: 1803.09820 [cs.LG].

[33] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018. arXiv: 1708.07120 [cs.LG].

[34] Yutao Sun et al. *A Length-Extrapolatable Transformer*. 2022. arXiv: 2212.10554 [cs.CL].

[35] Yutao Sun et al. *Retentive Network: A Successor to Transformer for Large Language Models*. 2023. arXiv: 2307.08621 [cs.CL].

[36] Hugues Thomas et al. *KPConv: Flexible and Deformable Convolution for Point Clouds*. 2019. arXiv: 1904.08889 [cs.CV].

[37] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. *Instance Normalization: The Missing Ingredient for Fast Stylization*. 2017. arXiv: 1607.08022 [cs.CV].

[38] Ozan Unal, Dengxin Dai, and Luc Van Gool. *Scribble-Supervised LiDAR Semantic Segmentation*. 2022. arXiv: 2203.08537 [cs.CV].

[39] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[40] Lei Wang et al. "Graph Attention Convolution for Point Cloud Semantic Segmentation". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 10288–10297. DOI: 10.1109/CVPR.2019.01054.

[41] Wenhai Wang et al. *Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions*. 2021. arXiv: 2102.12122 [cs.CV].

[42] Bichen Wu et al. *SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud*. 2017. arXiv: 1710.07368 [cs.CV].

[43] Bichen Wu et al. *SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud*. 2018. arXiv: 1809.08495 [cs.CV].

[44] Yuxin Wu and Kaiming He. *Group Normalization*. 2018. arXiv: 1803.08494 [cs.CV].

[45] Pengchuan Xiao et al. *PandaSet: Advanced Sensor Suite Dataset for Autonomous Driving*. 2021. arXiv: 2112.12610 [cs.CV].

[46] Tete Xiao et al. *Early Convolutions Help Transformers See Better*. 2021. arXiv: 2106.14881 [cs.CV].

[47] Enze Xie et al. *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers*. 2021. arXiv: 2105.15203 [cs.CV].

[48] Xu Yan et al. *2DPASS: 2D Priors Assisted Semantic Segmentation on LiDAR Point Clouds*. 2022. arXiv: 2207.04397 [cs.CV].

[49] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: 1311.2901 [cs.CV].

[50] Xinge Zhu et al. *Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation*. 2020. arXiv: 2011.10033 [cs.CV].