Continual Learning for LiDAR Semantic Segmentation: Class-Incremental and Coarse-to-Fine strategies on Sparse Data

Elena Camuffo and Simone Milani
Department of Information Engineering, University of Padova, Italy
{elena.camuffo,simone.milani}@dei.unipd.it

Abstract

During the last few years, Continual Learning (CL) strategies for image classification and segmentation have been widely investigated designing innovative solutions to tackle catastrophic forgetting, like knowledge distillation and self-inpainting. However, the application of continual learning paradigms to point clouds is still unexplored and investigation is required, especially using architectures that capture the sparsity and uneven distribution of LiDAR data. The current paper analyzes the problem of class incremental learning applied to point cloud semantic segmentation, comparing approaches and state-of-the-art architectures. To the best of our knowledge, this is the first example of class-incremental continual learning for LiDAR point cloud semantic segmentation. Different CL strategies were adapted to LiDAR point clouds and tested, tackling both classic fine-tuning scenarios and the Coarse-to-Fine learning paradigm. The framework has been evaluated through two different architectures on SemanticKITTI [2, 16], obtaining results in line with state-of-the-art CL strategies and standard offline learning.

1. Introduction

The recent development of autonomous driving, equipped with automatic visual understanding systems, has fostered the research on object identification and segmentation towards more generalizable and portable strategies via inline learning behaviors in place of distinct offline training phases. Particularly, continual learning investigates the capability of incorporating new knowledge in well-established models, shifting to new tasks, labels, or data distributions. Indeed, most of the cases do not benefit from previous training data, which are unavailable due to portability, privacy issues, or costly labeling process. The process is thus usually accomplished by preserving previous knowledge, whilst avoiding catastrophic forgetting. Different CL strategies have been deeply investigated for classification [11, 24, 29, 43] and object detection [23, 40, 45] tasks, while less examples exist for semantic segmentation. The reason is that semantic segmentation is a harder task with respect to classification: classification is about providing one single class label for each image, while in segmentation already-known and totally-new classes can be present together in the same acquisition (image, point cloud, etc.). As a matter of fact, handling with such heterogeneity is one of the additional issues that is addressed considering multiple continual learning scenarios for semantic segmentation [6, 13, 25, 36, 48]. So far, continual learning has generally been applied to images in vision tasks. With respect to point cloud data, images are far simpler data to be retrieved, labeled, and processed thanks to their regular distribution in space, limited memory requirements, and computational cost. Instead, CL techniques on point cloud data are still quite unexplored, made an exception for a few attempts [5, 8, 12, 26, 30], and the investigation of class-incremental CL has not been considered yet. Even if it may seem a simple adaptation of image-based network architectures to point cloud, the sparsity of LiDAR point cloud data and the significant difference in the processing architectures makes their investigation anew. More precisely, the problem turns out to be more challenging in relation to point-based architectures [21, 41, 42], where raw points are directly processed with no discretization (see Sec. 2 for more details). In addition, the inherent nature of point cloud datasets, acquired with different methodologies and in different noise conditions, can vary the final accuracy.

This paper analyzes the challenges of class-incremental continual learning on point cloud semantic segmentation and adapts different CL solutions to different architectures and partitionings. The main novelties brought by this work can be listed as follows.

- It is the first application of class-incremental continual learning to LiDAR semantic segmentation, where point sparsity enhances the issues concerning the co-occurrence of already-known and totally-new classes.
- It provides a general evaluation of different techniques to mitigate catastrophic forgetting on LiDAR point
clouds (adapted from image semantic segmentation to a domain with new peculiarities).

- It considers both general Class-Incremental and Coarse-to-Fine scenarios [28, 34, 46, 50].

The paper is structured as follows. Section 2 overviews recent state-of-the-art works, Section 3 presents the main single-headed class-incremental continual learning scenarios. Section 4 analyzes different CL solutions overviewing different partitioning and strategies to mitigate catastrophic forgetting. Finally, in Section 5 we discuss the results obtained with different techniques with a comparison among dataset setups and CL strategies; conclusions are drawn in Section 6.

2. Related Work

LiDAR Semantic Segmentation is usually tackled via deep learning solutions that treat input points in several ways: discretizing into voxels, projecting on 2D images, or processing directly as they are [3, 15]. Voxel based methods [33, 47] are generally used for well-structured point clouds (e.g., terrestrial laser scanning acquisitions [1]) and can easily apply 3D convolutions. Projection based methods [10, 38, 51] are well suited for LiDAR point clouds, as they are acquired in concentric circles and can be easily projected onto cylindrical [51] or spherical [38] surfaces. Both categories of methods are efficient in terms of computational complexity, but approximate point distributions to regular structures. Methods that directly process data [21, 41, 42] avoid such loss of structural information, which is crucial when dealing with few data samples and limited resources. Recent works rely on 4D convolutions [14] or transformers [18] to accomplish the task but require huge computational power and storage capacity; more lightweight approaches process points in mixed strategies, providing voxel-wise predictions refined with point-wise labels [52, 53]. Such models have been widely adopted for various learning approaches including class-balancing regularization [4, 10] Knowledge distillation [20] and open-world 3D scene understanding [5].

Some works have also adopted incremental learning settings for 3D data in object detection [12], place recognition [26], remote sensing [30] and few-shot learning [8]. Nonetheless, no prior work has explored Class-Incremental Continual Learning for LiDAR Semantic Segmentation.

Class-Incremental Continual Learning (CL) has developed a growing research interest for image classification [11, 24, 29, 43], object detection [23, 40, 45] tasks. Many of these works observed the catastrophic interference problem when sequentially learning examples of different input patterns. Among the most popular strategies to reduce forgetting are replay methods [32, 43] that store exemplars to be replayed in future steps and non-exemplar methods; among these parameter isolation dedicates different model parameters to each task to prevent any possible forgetting and regularization methods [7, 24, 29, 48], which add extra regularization loss to consolidate knowledge on previous data. This line of work avoids storing raw inputs, prioritizing privacy, and alleviating memory requirements.

CL in Semantic Segmentation literature is more limited [6, 13, 25, 36, 48]. Major attention has been devoted to class incremental semantic segmentation to learn new categories from new data. The problem is formalized in [36] and tackled with regularization approaches such as parameters freezing and knowledge distillation. In [25] knowledge distillation is coupled with a class importance weighting scheme to emphasize gradients on difficult classes. In [37] the latent space is regularized to improve class-conditional features separation. Cermelli et al. study the distribution shift of the background class in [7] and tackle the problem in a weakly-supervised fashion in [6]. Maracani et al. [32] proposed replaying old classes using either generative networks or web crawled images.

Coarse-to-Fine Semantic Segmentation has been explored in few previous work for part-based regularization [28, 50] and approaches where coarse-level classes are refined into finer categories [34, 46]. Previous approaches developed also different ways to draw class hierarchies. In [44] they are estimated a prior based on semantics; in [4], they are drawn a posteriori based on network misclassifications. Some examples have shown also developing Coarse-to-Fine for continual learning [44, 49], but no previous literature has explored with point cloud data.

3. Problem Formulation

Given an input point cloud, i.e., a set of \( N \) 3D points \( X = \{x_1, x_2, \ldots, x_N\} \), and a set of candidate semantic labels \( Y = \{y_1, y_2, \ldots, y_N\} \), the objective of semantic segmentation is to associate each input point \( x_i \) with a semantic label \( y_i \). Each label belongs to a given subset of semantic classes \( C \) that contains also a special class \( b \) (void, which corresponds to the background for images). The void class typically identifies objects out of the range of LiDAR precision or belonging to unknown classes. For the sake of clarity, we call it background as for images to avoid confusion with empty voxels and masked points. Semantic segmentation task is usually achieved by using a suitable deep learning model \( M : X \rightarrow \hat{Y}_C \), commonly composed of a feature extractor \( \hat{E} \) followed by a decoding module \( D, M = D \circ \hat{E} \).

In standard supervised learning, the model is learned in a single shot with standard offline training over a fixed and complete set of data \( T \subset X \times Y \).

In class incremental learning, instead, we assume that the training is performed in multiple steps and only a subset of the training data is available for training at each step \( k = 0, \ldots, K \). More in detail, we start from an initial step \( k = 0 \) where only training data concerning a subset
of all the classes $C_0 \subset C$ is available (we assume that $b \in C_0$). We denote with $\mathcal{M}_1 : X \mapsto \hat{Y}_{C_0}$, $\mathcal{M}_0 = D_0 \circ \mathcal{E}_0$ the model trained after this initial step. At a generic step $k$, a new set of classes $C_k$ is added to the class collection $C_0 \rightarrow (k-1)$ learned up to that point, resulting in an expanded set of learnable classes $C_0 \rightarrow k = C_0 \rightarrow (k-1) \cup C_k$ (we assume $C_0 \rightarrow (k-1) \cap C_k = \emptyset$). The model after the $k$-th step of training is $\mathcal{M}_k : X \mapsto \hat{Y}_{C_0 \rightarrow k}$, where $\mathcal{M}_k = D_k \circ \mathcal{E}_k$.

Some approaches train the decoder alone after the initial step while keeping the encoder frozen. In our approach, both the encoder and decoder are updated, while only the last layer is changed. This is because the encoder is usually kept frozen when the training is performed on random objects datasets, and the proportion of data between the first step and the subsequent ones is unbalanced toward the first. Instead, in our partitioning (and in general, in autonomous driving datasets), dataset cardinality, number of samples, and frequency of classes are balanced across the different steps.

3.1. Scenarios

Multiple scenarios have been proposed for CL on image semantic segmentation [35,36]. In each scenario, the original dataset $T$ is divided into $K$ groups that correspond to learning steps; the overall set of labels is also generally divided into $K$ groups. At learning step $k$, the training procedure updates the model using only the tuple $(T_k, C_k)$.

**Sequential**: samples in the $k$-th experience are provided with the labels of all the groups $\leq k$; while points with labels $> k$ are not present at all. Each learning step contains a unique set of samples, whose points belong to classes seen either in the current or in the previous steps.

**Sequential masked**: samples in the $k$-th experience are provided with the $k$-th experience labels, while the points with labels of classes $< k$ are marked as unlabeled, and points with labels $> k$ are not present at all. In this setup, each learning step contains a unique set of samples, whose points belong either to novel classes or to the unlabeled class, which is not predicted by the model and is masked out from both the results and the training procedure.

**Disjoint**: samples in the $k$-th experience are provided with the $k$-th experience labels, while points with labels of classes $< k$ are marked as background, and points with labels $> k$ are not present at all. At each learning step, the unique set of training samples is identical to the sequential setup, but old classes are gathered into a common label (background, instead of being unlabelled as in the Sequential masked scenario) that changes its distribution at each step.

**Overlapped**: At each learning step $k$ all the dataset samples are available but only the current step classes $C_k$ are labeled with the correct label and the rest ($C_j \neq C_k$) with the background labels, either if they belong to experience $j < k$ or $j > k$. In this setup, each training step contains...
all the point clouds that have at least one point belonging to a class in set $C_k$, with only the classes of the set annotated and the rest set to background. In this paper, this scenario is reported for comparison assuming that data can never be shared among different experiences.

**Coarse-to-Fine:** At each learning step $k$, all points are labeled but their classes are refined with respect to the previous step. At step $k$, coarse labels $C_{k-1}$ are substituted with labels from fine classes $C_k, |C_k| > |C_{k-1}|$. Each coarse class corresponds to its unique set of fine classes.

### 4. Methodology

We considered RandLA-Net [21], one of the most famous point-based architectures as a reference, in order to frame the problem in a perspective different from images (i.e., using MLPs in place of convolutional setting). RandLA-Net is an efficient point-based lightweight network composed of an MLP based encoder-decoder structure that achieves remarkably high efficiency in terms of memory and computation. In addition, we evaluate on Cylinder3D [52, 53] voxel based architecture for comparison. Nonetheless, the framework can be applied on top of any architecture for point cloud semantic segmentation.

SemanticKITTI [2, 16] has been chosen as a reference dataset, since it is one of the most popular benchmarks for LiDAR semantic segmentation in autonomous driving. SemanticKITTI consists of 43,552 densely annotated LiDAR scans, 19,130 for training, and 4,071 for validating (that we used for testing, as done by all competing works being the test labels not publicly available).

### 4.1. Dataset Partitioning

We divided the dataset in order to define incremental experiences for continual learning. As no previous works can be found, we propose two subdivisions of SemanticKITTI: one based on class cardinality and frequency within different sequential acquisitions, and another based on a Coarse-to-Fine semantic partitioning. We denote the former as CIL, as it is based on the effective subdivision of [25], and the latter as C2F.

#### 4.1.1 Semantic Partition (CIL)

Tab. 1 shows the first partition of SemanticKITTI, based on class cardinality and frequency within different sequential acquisitions. This grouping is based on the effective subdivision of Cityscapes [9] proposed in [25]. Cityscapes is an autonomous driving dataset for image semantic segmentation with a class set similar to SemanticKITTI. Specifically, in $T_0$ class sky is substituted with parking and other-ground, in $T_1$ traffic-light and walls with trunk, and in $T_2$ train, bus and rider are replaced with motorcyclist, bicyclist and other-vehicle. Such classes are underlined in Tab. 1. Sequence subdivisions are chosen to maximize labeled classes in each group while keeping semantic consistency and balancing the total amount of points/acquisitions within each training subset. Fig. 1 shows the percentage of labeled points within each group normalized for each class $c$ on the cardinality of the specific class.

#### 4.1.2 Coarse-to-Fine Partition (C2F)

Fig. 2 shows the second partition of SemanticKITTI, where grouping is made by splitting labels into three learning steps, and labels are clustered to guarantee semantic consistency. The first coarse set resembles the semantics of CIL: classes of each $C^{C2F}_0$ form a single coarse class in $C^{C2F}_0$. $C^{C2F}_1$ is obtained by further partitioning coarse classes into 2 mid-level classes each; finally, the fine set $C^{C2F}_2 = C$, i.e., contains all the classes of SemanticKITTI. Note that this partition shares the dataset split of sequences with CIL setup reported in Tab 1.

### 4.2. CL Strategies

In standard training, background points are usually masked out from the learning procedure and the model weights are optimized using the cross-entropy objective

$$L_{ce} = -\frac{1}{|T_k|} \sum_{X_n \in T_k} \sum_{c \in C_{k-1}} Y_n[c] \cdot \log(M_k(X_n)[c])$$

where $Y_n[c]$ the one-hot encoded ground truth and $M_k(X_n)[c]$ is the prediction score for class $c$. In the incremental learning setup, we assume that at each incremental training step $k$ only samples from new classes $C_k$ are available. In order to include this new information in the model, weights are initialized from previous model $M_{k-1}$ ($k \geq 1$) and the new class set $C_{0\rightarrow k}$ is then learned by optimizing the standard objective $L_{ce}(M_k; C_{0\rightarrow k}, T_k)$ with data from the current training partition $T_k$. However, a naïve fine tuning on novel classes leads to catastrophic forgetting, and therefore, by including points from past classes (no labels) as background, we can recover previous knowledge by self-predicting old labels. To this extent, we consider two strategies: Knowledge Distillation (KD) [29] and Background
**4.2.1 Knowledge Distillation**

Knowledge Distillation (KD) is the process of transferring knowledge from a model to another (typically smaller) [19, 20]. In our case, we apply knowledge distillation to recall past classes while learning new ones and to deal with the background shift phenomenon. Old labels are recovered from the previous step model $M_{k-1}$. KD is modeled as an additional objective function $L_{kd}$ tuned by a hyperparameter $\lambda$, obtaining as a final objective for model $M_k$:

$$\mathcal{L} = \mathcal{L}_{ce}(y_k, \hat{y}_k) + \lambda \cdot L_{kd}(\hat{y}_{k-1}, \hat{y}_k)$$  \hspace{1cm} (2)

where $y_k$ are the ground truth labels of step $k$, $\hat{y}_k$ are the predictions of step $k$ and $\hat{y}_{k-1}$ are the prediction labels of step $k - 1$. At step $k = 0$ KD is not applied, as at that stage we lack any prior knowledge of past classes.

**Output Level Distillation ($L_{kd}$)** is typically modeled as an additional Cross-Entropy [17], where labels for past classes are obtained from previous model predictions:

$$L_{kd} = -\frac{1}{|T_k|} \sum_{x_n \in T_k} \sum_{c \in C_{k-1} \cup C_k} M_k(x_n)[c] \log(M_k(x_n)[c])$$  \hspace{1cm} (3)

where $M_k(x_n)[c]$ and $M_{k-1}(x_n)[c]$ are the prediction scores for class $c$, obtained with current and previous models, respectively. KD can be modeled either by masking predictions of each model $M_k$ on classes $C_k$ or joining all the predictions of unknown classes [7]. An alternative formulation updates also the previous model to refine predictions [48]. For Coarse-to-Fine CL, the predictions for classes within the same coarse class are summed before applying the Cross-Entropy loss.

**Feature Level Distillation ($L_{kd}^*$)** is typically modeled as $L_p$ (L2 or L1) norm between the features of current and previous models:

$$L_{kd}^* = \frac{||\mathcal{E}_{k-1}(X_n) - \mathcal{E}_k(X_n)||_p}{|T_k|}$$  \hspace{1cm} (4)

where $\mathcal{E}_k(X_n)[c]$ and $\mathcal{E}_{k-1}(X_n)[c]$ are the features of current and previous models, respectively. The choice of this design is motivated by the fact that this strategy provides learning of a feature map rather than a classification map and distance loss provides an alignment of such embeddings. KD at feature and output levels can also be combined to obtain joint effects from both.

**4.2.2 Background Self-Inpainting**

Background Self-Inpainting is a simple self-teaching mechanism that provides pseudo labeling of current background samples [32], reducing the background shift while bringing a regularization effect similar to knowledge distillation. At every step $k$ with training set $T_k$, we take the background points of each ground truth map $Y_k$ and we label it with the associated prediction from the previous model $M_{k-1}$ (Fig. 4). More formally, we replace each original label map $Y_k$ available at step $k > 0$ with its inpainted version $Y_k^{ip}$:

$$Y_k^{ip}[i] = \begin{cases} Y_k[i] & \text{if } Y[i] \in C_k \\ \rho \cdot \hat{Y}_{k-1}[i] & \text{otherwise} \end{cases}$$  \hspace{1cm} (5)

where $(X_k, Y_k) \in T_k$, $i$ denotes $i$-th sample in $T_k$ and $\rho$ defines the inpainting rule. $\rho$ is designed as:

$$\rho = \begin{cases} 1 & \text{if } \hat{Y}_{k}[i] - \hat{Y}_{k}^{*}[i] > \tau_1 \text{ and } \hat{Y}_{k}^{*}[i] > \tau_2 \\ 0 & \text{otherwise}. \end{cases}$$  \hspace{1cm} (6)

$\hat{Y}_{k}[i]$ and $\hat{Y}_{k}^{*}[i]$ denote the sets of first and second maximum in the softmax predictions $\hat{Y}_k$ at step $k$. Labels at step $k = 0$ are not inpainted, as at that stage we lack any prior
knowledge of past classes. When background inpainting is performed, each set $T_k^p \subset X \times Y_{C_{0-k}}$ ($k > 0$) contains all samples of $T_k$ after being inpainted.

### 5. Experimental Results

For the experimental evaluation, we use RandLA-Net [21] (a widely used lightweight point based architecture) and compare results with Cylinder3D [52, 53] (state-of-the-art model based on voxels and point-wise prediction refinements). Note that the approach is generalizable to any other segmentation architecture for point clouds. We used SemanticKITTI [2, 16] as a reference dataset, with classes and data partitioned into learning steps according to CIL and C2F splits, described in Sec. 4.1. The model performance is measured by the mean Intersection over Union (mIoU) [31]. From now on, we use the notation mIoU$_k$ to denote the mIoU computed on class set $C_k$.

We train both RandLA-Net and Cylinder3D using Adam optimizer with their standard learning rate policy, momentum, and weight decay. The initial learning rates are set relatively to 0.01 and 0.001 for the two models. The incremental training has been performed decreasing the learning rate according to a polynomial decay rule with power 0.95. Note that at step $k$ the learning rate is initialized to the last learning rate of step $k - 1$ to better preserve previous weights. We set batch size 3 and train each learning step $k$ for a number of epochs $2 \times |C_k|$, proportional to the number of new classes to be learned. We used PyTorch [39] for the implementation and train both models on NVIDIA GeForce RTX 3090Ti graphics processing unit with CUDA 11.6. The training of each learning step took around 5 hours with RandLA-Net and 30 hours with Cylinder3D. The code is available at: https://github.com/LTTM/CL-PCSS. In the following sections, we discuss the results obtained with CIL and C2F partitions. Baseline results are obtained by training the model from scratch on the whole dataset with all the classes, using the aforementioned configuration.

#### 5.1. CIL

We performed our first set of experiments following CIL configuration. First, we developed standard fine-tuning methods, knowledge distillation, and self-inpainting to compare Cylinder3D with RandLA-Net. Results are reported in Tab. 2. In general, the obtained performances reflect the behavior of

### Table 2. Results of Cylinder3D and RandLA-Net (CIL). Bold denotes the best result, underline the second best. †: baselines retrained with our configuration.

<table>
<thead>
<tr>
<th>Method</th>
<th>mIoU$_0$</th>
<th>mIoU$_1$</th>
<th>mIoU$_2$</th>
<th>mIoU$_{0,1}$</th>
<th>mIoU$_{0,2}$</th>
<th>mIoU$_{0,1,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline†</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>54.1</td>
</tr>
<tr>
<td>Sequential</td>
<td>55.4</td>
<td>52.7</td>
<td>46.3</td>
<td>50.1</td>
<td>52.8</td>
<td>50.6</td>
</tr>
<tr>
<td>Disjoint</td>
<td>55.4</td>
<td>0.0</td>
<td>21.4</td>
<td>20.4</td>
<td>0.0</td>
<td>25.7</td>
</tr>
<tr>
<td>Cylinder3D†</td>
<td>55.4</td>
<td>0.0</td>
<td>63.3</td>
<td>20.0</td>
<td>0.0</td>
<td>25.2</td>
</tr>
<tr>
<td>Output KD [29]</td>
<td>55.4</td>
<td>45.6</td>
<td>41.1</td>
<td>43.5</td>
<td>47.4</td>
<td>32.0</td>
</tr>
<tr>
<td>Self-Inpaint [32]</td>
<td>55.4</td>
<td>51.2</td>
<td>47.1</td>
<td>49.4</td>
<td>46.2</td>
<td>36.8</td>
</tr>
<tr>
<td>Baseline†</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>47.2</td>
</tr>
<tr>
<td>Sequential</td>
<td>49.0</td>
<td>57.0</td>
<td>39.0</td>
<td>48.8</td>
<td>58.0</td>
<td>48.4</td>
</tr>
<tr>
<td>Disjoint</td>
<td>49.0</td>
<td>0.0</td>
<td>48.5</td>
<td>22.0</td>
<td>0.0</td>
<td>31.4</td>
</tr>
<tr>
<td>Cylinder3D†</td>
<td>49.0</td>
<td>0.0</td>
<td>37.9</td>
<td>18.0</td>
<td>0.0</td>
<td>26.1</td>
</tr>
</tbody>
</table>

### Table 3. Per-class mIoU results of RandLA-Net (CIL). Bold denotes the best result, underline the second best. †: baseline retrained with our configuration.

| Method                  | car      | bicycle  | motorcycle | truck     | other-vehicle | person            | bicyclist | motorcyclist | road      | parking    | sidewalk  | building   | fence      | vegetation | traffic-sign | truck     | parking    | sidewalk  | building   | fence      | vegetation | traffic-sign | pole      | traffic-sign | parking    | sidewalk  | building   | fence      | vegetation | traffic-sign | pole      | traffic-sign |
|-------------------------|----------|----------|------------|-----------|---------------|-------------------|------------|----------------|-----------|------------|-----------|------------|-----------|------------|-------------|-----------|------------|-----------|------------|-----------|------------|-------------|-----------|-------------|-----------|------------|-----------|------------|-------------|-----------|-------------|-----------|-------------|-----------|------------|-------------|-----------|-------------|-----------|-------------|
of the corresponding image-based setups. As expected, the disjoint scenario performs the worst in all the learning steps, strongly suffering the background shift phenomenon. Sequential masked improves a little the results besides showing again the effect of catastrophic forgetting when accounting for new classes.

Overall, Cylinder3D shows worse performance with respect to RandLA-Net within a CL framework, despite its better performance in the non-incremental scenario. Its results on the baseline show a strong performance decrease losing 9.8% mIoU in the sequential setup (best case for fine-tuning). Besides, the size of the architecture and the processing of voxels make the training time 6 times longer. KD partially recovers knowledge from previous classes but keeps stability when learning new ones. Instead, Self-Inpainting is more performing and accounts better for new classes. Note that, mIoU₀ is efficiently recovered by both methods across the steps, while mIoU₁ is hardly restored. This is partially due to the fact that labels for step \( k = 0 \) are robust classes with many points and a general method for catastrophic forgetting mitigation can be sufficient for recovering knowledge. Contrarily, step \( k = 1 \) has a huge decrease in the number of labeled points that reflects the increasing difficulty in obtaining good recovery from previous training. Self-Inpainting hardly pseudo-labels old classes, not preserving the uncertainty of predictions, and penalizing the learning of new classes. The introduction of inpainting rules (Tab. 5) brings a slight improvement, avoiding strongly wrong predictions to be considered as true labels in the subsequent step. In this way, previous knowledge is hardly restored only in case of accurate predictions and blended into the current model in a stronger way with respect to KD. Finally, both methods perform worse with respect to the sequential setup. Qualitative results of the self-inpainting strategy are shown in Fig. 5.

On the other hand, RandLA-Net shows great suitability for continual learning settings, with a decrease of only 4.7% mIoU in the sequential setup, with respect to the baseline. Surprisingly, recovering knowledge via distillation obtains 44.4% mIoU, improving the sequential setup of 1.5% mIoU with a gap from the baseline of only 2.8% mIoU. This is again a consequence of the label frequency of each split:

![Figure 5. Some qualitative results on CIL with Cylinder3D.](image-url)
Table 7. Per-class mIoU results on C2F with RandLA-Net. **Bold** denotes the best result. †: baseline retrained with our configuration.

<table>
<thead>
<tr>
<th>step</th>
<th>car</th>
<th>bicycle</th>
<th>motorcycle</th>
<th>truck</th>
<th>other-vehicle</th>
<th>person</th>
<th>bicyclist</th>
<th>motorcyclist</th>
<th>road</th>
<th>parking</th>
<th>sidewalk</th>
<th>other-ground</th>
<th>building</th>
<th>fence</th>
<th>vegetation</th>
<th>trunk</th>
<th>terrain</th>
<th>pole</th>
<th>traffic-sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>89.7</td>
<td>6.2</td>
<td>17.4</td>
<td>47.0</td>
<td>24.8</td>
<td>28.3</td>
<td>65.5</td>
<td>0.0</td>
<td>88.8</td>
<td>30.7</td>
<td>73.0</td>
<td>0.6</td>
<td>83.6</td>
<td>45.8</td>
<td>87.2</td>
<td>55.4</td>
<td>75.1</td>
<td>50.8</td>
<td>24.4</td>
</tr>
<tr>
<td></td>
<td>90.3</td>
<td>0.2</td>
<td>16.6</td>
<td>53.2</td>
<td>30.0</td>
<td>14.8</td>
<td>42.1</td>
<td>0.0</td>
<td>90.7</td>
<td>26.9</td>
<td>74.1</td>
<td>0.1</td>
<td>84.5</td>
<td>43.9</td>
<td>86.5</td>
<td>56.8</td>
<td>73.9</td>
<td>52.0</td>
<td>21.1</td>
</tr>
<tr>
<td></td>
<td>89.2</td>
<td>0.0</td>
<td>12.5</td>
<td>65.7</td>
<td>29.6</td>
<td>3.6</td>
<td>41.4</td>
<td>0.0</td>
<td>91.2</td>
<td>37.9</td>
<td>74.3</td>
<td>0.8</td>
<td>84.8</td>
<td>45.4</td>
<td>86.2</td>
<td>50.1</td>
<td>72.9</td>
<td>48.7</td>
<td>18.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8. Per-class Coarse-to-Fine results, standard fine-tuning. Different colors are associated with different Coarse partitions.

in performance between frequent and infrequent classes. For example, class *car* obtains always good results around 90% mIoU, while class *bicycle* (belonging to the same partition) obtains 1.6% mIoU on the baseline but is never classified correctly in the incremental setup. Similarly, *traffic-sign* obtains 27.6% mIoU in the baseline but its performance decrease drastically when incremental learning is applied; in Self-inpainting it is never classified correctly. Indeed, the lowest standard deviation is achieved by the baseline model.

Finally, worth to be mentioned is the result of the Overlapped setup (reported in Tab. 4) in comparison with the Disjoint one: training on the whole dataset brings as expected an improvement (2.6% on the final mIoU) but still suffers for catastrophic forgetting.

### 5.2. Coarse-to-Fine

Results on the Coarse-to-Fine partition are reported in Tab. 7. Fine-tuning on coarse classes obtains results comparable with the standard baseline training (47.1% vs 47.2% mIoU), despite a higher standard deviation among classes. Note that this tuning is performed on the split sequences of Tab. 1, while general Coarse-to-Fine approaches rely on the Overlapped approach. Contrarily to CIL, a direct application of Knowledge Distillation leads here to poor performance, both when applied at the output level and intermediate feature space, losing 2.0% and 2.2% mIoU respectively. Hence, this setting requires the application of more sophisticated loss functions in order to obtain a substantial improvement. Tab. 8 reports the results of the C2F fine-tuning in terms of mIoU, Point Accuracy (PA) [22] and Point Precision (PP) [27]. Results in the first step achieve higher accuracy with a low standard deviation, which increases with learning steps (classes are fairly distributed up to step $k = 1$). This result suggests that additional intermediate steps could be beneficial to improve the final performance, which in general requires further refinement.

### 6. Conclusion

The paper tackled the problem of class incremental continual learning for LiDAR semantic segmentation. We formally partitioned SemanticKITTI into semantically-consistent groups, and we evaluated partitions on CL strategies addressing different techniques to prevent catastrophic forgetting. A comparison of RandLA-Net and Cylinder3D performance shows that the former (point based and lightweight) fits better into the class-incremental setup. Experiments on our proposed subdivision of SemanticKITTI prove the efficiency of CL strategies in alleviating catastrophic forgetting, even on sparse data. The overall problem still requires improvements as the performance is lower than the ones achieved after standard single-step training. Future work will expand the experimental framework by introducing class balancing constraints among previous and current experiences, and geometric constraints, designed ad hoc for LiDAR point clouds.

### Acknowledgements

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, a partnership on “Telecommunications of the Future” (PE0000001 - program “RESTART”).
References


[27] Alex Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. Proceedings of the IEEE


Marco Toldo, Umberto Michieli, and Pietro Zanuttigh. Learning with style: Continual semantic segmentation across tasks and domains. 2022. 1, 2, 5, 6, 7


Boyu Yang, Fang Wan, Chang Liu, Bohao Li, Xiangyang Ji, and Qixiang Ye. Part-based semantic transform for few-shot semantic segmentation. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. 2

