ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Master Thesis

# Continual Learning of Semantic Segmentation for mobile Robotics

**Spring Term 2021**

**Supervised by:**
Hermann Blum
Francesco Milano
Dr. Cadena Cesar

**Author:**
Jonas Frey

# Declaration of Originality

I hereby declare that the written work I have submitted entitled

**Continual Learning of Semantic Segmentation for mobile Robotics**

is original work which I alone have authored and which is written in my own words.[1]

**Author**

| | |
|---|---|
| Jonas | Frey |

**Supervising students**

| | |
|---|---|
| Hermann | Blum |
| Francesco | Milano |

**Supervising lecturers**

| | |
|---|---|
| Cesar | Cadena |
| Roland | Siegwart |

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Zurich, 06.08.21
Place and date

Frey
Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

# Contents

# Acknowledgements

Throughout working on my master's thesis I have received a great deal of support and assistance. First and foremost, I would like to thank Hermann Blum, Franceso Milano and Dr. Cesar Cadena for the weekly guidance within the past 6 months. Your theoretical and practical advice eased the progress of working on this thesis significantly.

In addition, I would like to thank everybody at the ASL for all their advice and help. Thanks to Yash Vyas, Carter Fang, Eric Vollenweider and Richard Fehler for your assistance. Lastly, I would like to thank my roommates Dr. Peter Coburger, Moritz Scharnhölz and my girlfriend for all the support during this time.

# Abstract

Neural networks are an essential part of modern perception pipelines for mobile robots. It has been shown that up-scaling the size of the network and training data corpus leads to better performance. This comes with the cost of increased energy consumption and longer training time. When new samples or training data are acquired, expensive retraining or fine-tuning of the network has to be performed. We study continual learning (CL) of semantic segmentation for mobile robots, where the goal is to integrate new knowledge into a neural network on the fly, while preserving previously obtained knowledge.

Firstly, an investigation and performance evaluation of CL strategies focusing on experience replay methods is conducted. Quantitative results are given for real and synthetic indoor datasets. We demonstrate an overall accuracy increase of 50.9 % for a random replay buffer strategy compared to standard optimization. Secondly, we compare different methods of fusing information with prior knowledge to create an online supervision signal, so-called pseudo labels. These labels are generated without human annotations. We evaluate optical flow, super-pixel and 3D voxel-based volumetric mapping approaches for pseudo-label generation. Next, we show iterative 3D mapping and re-training can further increase the supervision signal. The retrained neural network exceeds the computationally more expensive pseudo labels by 17.4% in relation to the pseudo label performance increase. Continual adaptation to multiple scenes can improve the accuracy by up to 12.1% for individual scenes. Lastly, we demonstrate the effectiveness of our method on data recorded with a handheld RGB-D sensor. Our dataset includes various scenes recorded in ETH Zurich buildings simulating a robotic mission.

# Symbols

## Symbols

| | |
|---|---|
| $y$ | Target |
| $\hat{y}$ | Prediction |
| $d$ | Depth measurement |
| $x$ | Input |
| $f_\theta$ | Network function parameterized by $\theta$ |
| $F$ | Optical Flow |
| $\mu$ | Learning Rate |
| $H$ | SE3 Pose |
| $K$ | Camera intrinsics |
| $\mathcal{D}_S$ | Source domain |
| $\mathcal{D}_T$ | Target domain |
| $\mathcal{X}$ | Input domain |
| $\mathcal{Y}$ | Output domain |

## Indices

| | |
|---|---|
| $u$, $v$ | Image coordinates |

## Acronyms and Abbreviations

| | |
|---|---|
| CL | Continual Learning |
| MT | Multi-Task |
| MIT | Multi-Incremental-Task |
| SIT | Single-Incremental-Task |
| | |
| IMU | Inertial Measurment Unit |
| EKF | Extended Kalman Filter |
| SLIC | Simple Linear Iterative Clustering |
| CRF | Conditional Random Field |
| TSDF | Truncated Signed Distance Function |
| | |
| SLAM | Simultaneous Localization and Mapping |

| | |
|---|---|
| SfM | Structure from Motion |
| VO | Visual Odometry |
| | |
| ACC | Average Accuracy |
| FWT | Forward Transfer |
| BWT | Backward Transfer |
| $\Omega$ | Overall Performance |
| $\Omega_F$ | Final Performance |
| MSE | Mean Squared Error |
| | |
| CNN | Convolutional Neural Network |
| SVM | Support Vector Machine |
| VAE | Variational Autoencoder |
| GAN | Generative Adversarial Network |
| DANN | Domain-Adversarial Neural Network |
| SCNN | Fast Semantic Segmentation Network |
| RAFT | Recurrent All-Pairs Field Transforms |
| | |
| SGD | Stochastic Gradient Descent |
| Adam | Adam |

# Chapter 1

# Introduction

Deployment of modern robotic systems outside industrial halls requires more advanced perception, navigation, and reasoning capabilities. In the future, we expect robots to handle complex tasks, including autonomous driving, construction, elderly care and assist, search and rescue operations, and space exploration [1]. To achieve these tasks, modern systems have a suite of sensors including cameras, ranging sensors, force, localization (GPS), Inertial Measurement Units, and many more that allow them to acquire data on their internal state and environment [1]. To extract useful information from these sensors, neural networks, trained on a large corpus of data, are used. Using a learning based approach we avoid creating hand-crafted features but instead rely on the quality of the training data to achieve good performance. A mismatch between the training data distribution and the data perceived by the robot can cause system failures. This can harm humans or result in significant damage to equipment in safety-critical applications. To avoid data distribution mismatch and integrate novel information, major companies constantly gather new data and iteratively retrain neural networks to increase performance [2].

These datasets can reach up to 300 million images for classification tasks [3], 1.5-petabytes for autonomous driving data or include a total of 500 billion tokens for natural language processing [4]. Despite these major efforts, those datasets fail to adequately represent all possible scenarios and corner cases in the wild. Intrinsically for various tasks, new data is continually generated making it infeasible to represent all future data with a dataset captured at a fixed point in time.

This motivates the need for a more flexible system capable of adaptation. The field of Continual Learning (CL) tackles integrating new knowledge into a network while preserving previously learned information. This paradigm can reduce the training cost, integrate new knowledge, handle changing data distributions, reduce memory consumption and ultimately enable adaptation to new environments [5].

Historically the field of lifelong learning, the predecessor of CL, was established in 1995 by [6, 7]. The first work on CL was published later in 2003 [8] with a focus on robotic applications in reinforcement environments. The capability to integrate new knowledge and remember previously learned knowledge is referred to as plasticity and stability respectively. These terms originate from biology [8] to describe the ability of the human brain to establish new synaptic connections or generate new memories while being stable enough to retain important previous information.

When training neural networks, the objectives of maximizing plasticity and stability are adversarial. Perfect stability is achieved by fixing the network's parameters,

but no new knowledge is learned. On the other hand the network can be optimized to fit the novel training without any constraint, to maximize plasticity. This results in the forgetting of previous experience, which is referred to as catastrophic forgetting [9]. This adversarial relationship is therefore referred to as the plasticity-stability dilemma [10].

The field of CL tackles the challenge of mitigating catastrophic forgetting while allowing the integration of new knowledge into neural networks under various constraints. Depending on the exact CL scenario (sec. 2), constraints typically include computational capacity and time, dataset memory, model size, different forms of supervision signals, and data sample correlation.

Current research in CL focuses mainly on small-scale problems in image classification [11, 12, 13, 14], object detection, segmentation [15] and reinforcement learning scenarios of arcade video games [16, 6, 8]. These fields are chosen based on the superior results of deep learning for vision-based tasks and the availability of datasets with clear performance indicators, which allow benchmarking of different methods. These tasks are often not chosen based on real-world problems or use cases [12].
Besides the previously mentioned constraints, CL is challenging for multiple reasons. In the field of deep learning, no rigorous theoretical analysis exists which can explain the outstanding practical results achieved. The exact interplay of highly non-linear models with first-order optimization procedures is not well understood and is still an open research topic. Formulation of the CL problem as constrained optimization is intractable to compute with growing numbers of model parameters and previous experience. Additionally, the credit assignment problem in neural networks and lack of interpretability introduce further challenges.

Semantic segmentation is an essential part of the vision pipeline of modern robotics systems. It is the task of assigning a label to a location from a predefined set according to its semantic association, which depends on the use case. This enables various further downstream tasks such as navigation, grasping, planning, and high-level reasoning [1]. It is especially suitable for CL given the rich literature of semi-supervised and unsupervised learning approaches [17, 18, 19].
The contribution of this master thesis is threefold.

**(1)** Firstly, we expand the scope of CL to the field of multi-class semantic segmentation. This enables robotic systems to acquire new data and learn without the need for human annotation during a mission. Additionally, semantic segmentation is highly dependent on the geographic region, and the underlying data distribution is under constant change. This change is induced by rearrangement of objects, novel scenarios, novel objects, dynamic scenes, season, illumination, viewpoint changes, or occlusions. Training a network on fixed data cannot generalize to every possible scenario [19]. Time and cost of expensive retraining for a neural network deployed on a mobile robot is impractical to achieve adaptation. While deployment times of robots to a specific scene are often on the order of minutes, the retraining process may take significantly longer. On the other hand, improving and adapting a network to the current environment is promising and has already shown great success [20, 15].
For this, we introduce a systematic benchmark for learning semantic segmentation using the ScanNet [21] and Hypersim [22] datasets. ScanNet features indoor RGB-D video sequences of 460 different scenes, while Hypersim provides photo-realistic renderings of RGB-D images of 599 different buildings. On both benchmarks, we employ an experience replay CL strategy (sec. 2) as a baseline model. We additionally perform an exhaustive ablation study to measure the performance impact of pa-

rameter settings in the CL strategy. We set our results in the context of prior work performed on other CL datasets.

**(2)** Secondly, we evaluate different methods to generate a semantic segmentation supervision signal relying solely on an RGB-D sensor. For this we utilize Kimera-Semantics [23] to accumulate predicted labels into a voxel-based semantic reconstruction map. The semantic map is ray traced with the known camera extrinsics to generate pseudo-labels for each image. This allows for temporally and spatially consistent label generation of a scene that improves over the single-view image network prediction. We compare this with other methods using clustering approaches [24, 25] and optical flow label propagation [26].

**(3)** Lastly, we perform CL experiments with the pseudo-labels using the previously analyzed experience replay method for the ScanNet dataset. We show that we can rapidly adapt to the current scene without further information and improve semantic segmentation performance. Generating a supervision signal with the adapted version of the neural network is repeated iteratively, resulting in a further performance increase. We show the effectiveness of our method based on data captured in various ETH Zurich buildings with a handheld RGB-D sensor.

# Chapter 2

# Background

In the following, we provide background information for CL, semantic segmentation, and semantic supervision signal generation. In the section 2.1, we introduce basic terminology, a systematic framework, current challenges, strategies and benchmarks for CL.
For semantic segmentation (sec. 2.2), we provide a task description and review currently used network architectures. Additionally, we give a brief overview of domain adaptation strategies used today for semantic segmentation.
Lastly in section 2.3, we focus on methods to generate semantic supervision signals without the need for human annotation. We elaborate on underlying problem constraints and different applicable information fusion techniques.

## 2.1 Continual Learning

### 2.1.1 Context and Terminology

Humans have the remarkable capability to acquire new knowledge continually. To handle our limited memory capacity, we unconsciously selectively forget past experiences. Our brain is capable of finding a good trade-off under the dilemma of stability and plasticity [27]. This problem also arises in artificial neural networks and is tackled by the field of CL. It focuses on mitigating catastrophic forgetting, while not impeding the learning of new information.

To understand catastrophic forgetting we review the training procedure of modern neural networks. Deep learning relies on 1st order gradient-based optimization techniques such as SGD or ADAM [28]. A network $\hat{y} = f_\theta(x)$ parameterized by $\theta$ maps from the input domain $X$ to $Y$. The goal during optimization is to minimize a loss function. The network is highly nonlinear and differentiable. During training, stochastic optimizers shuffle the training data randomly at each epoch. This ensures that samples in the training data are drawn in a way that is independently and identically distributed (i.i.d). Multiple samples are pooled to a mini batch. When the mini batch is propagated though the network $f_\theta$, the average loss of all samples is calculated. Given the differentiability of the network $f_\theta$, we can compute the backpropagation graph offline. It allows for efficient computation of the loss gradient with respect to the network weights $\theta$. The gradient is used to update the parameters of all weights in the network. This procedure is performed for multiple epochs until the learning procedure converges or is stopped.

When changing the data distribution or optimization objective, all weights are updated to fit the new data or objective as good as possible, without any constraints.

This unconstrained updating results in a strong decrease in accuracy on the previous objective or data distribution. Each objective or distinct data distribution, that we want the network to learn, is called a **task**.

Recent research in neural network pruning shows that smaller sub networks exist within larger neural networks, and that they achieve similar performance while utilizing fewer connections and neurons [29]. This leads to the hypothesis that new information can be integrated into a neural network under slight performance degeneration on the previous task in our CL application. CL, apart from catastrophic forgetting and the stability-plasticity dilemma, has to take memory management, handling and detection of domain shifts efficiently into account. A wide variety of different scenarios and settings exist.

To provide a clear and shared understanding of the terminology we follow [8] by categorizing the related learning fields. This allows us to highlight important differences and common denominators.

| Paradigm | Available | Previous | Memory | Explore | Objective |
|---|---|---|---|---|---|
| **Online** | single | - | ✓ | × | Learn on stream data |
| **Few-shot** | few | - | × | × | Fast concept learning |
| **Curriculum** | - | × | × | × | Last task |
| **Meta** | - | - | × | × | Learning to learn |
| **Transfer** | - | × | × | × | Knowledge transfer |
| **Active** | - | - | × | ✓ | Explicit data collection |
| **Continual** | limited | ✓ | ✓ | × | All tasks |

Table 2.1: Comparison of different learning settings: ✓ → positive, × → negative, - → non distinguishable. **Available** refers to how often a data-point is available for training. **Previous** declares if the performance on a previous task is from explicit importance. **Memory** describe if the memory constraint is an explicit feature of the learning paradigm. **Explore** reefers to whether choosing how to sample from the underlying data distribution is essential part of the problem and feasible.

Online learning is a sub-field of CL where each data point is only available once [8]. Curriculum learning explores the creation of a learning curriculum with intermediate tasks, interpreted as stepping stones, to achieve a final objective [30]. It is closely linked to Reinforcement Learning. Transfer learning focuses on leveraging the knowledge acquired on one task to solve another target task [31]. In most cases, the performance on the previous task is not of importance. Active learning allows for an explicit notation of sampling of data [8]. Therefore the learning strategy has access to a sampling procedure and can choose where to generate new data. This field is also related to Reinforcement Learning and is often of particular importance for robotics applications given that robots can gather new data in the real world.

### 2.1.2 Definition and Frameworks

In the following, we lay out the formal mathematical notation of the CL problem. Each **learning task** $T_t$ (subscript $t$ denotes a unique **task label**) describes a learning object. Multiple learning tasks aggregate to a CL problem.
Each task can be associated with a **training dataset** $D_t^{train} = \{(x_t^i, y_t^i)\}_{1..N_t}$, where $x_t$ is the input data point and $y_t$ is the associated label with a total of $N_t$ data points. A **test dataset** for each task $D_t^{test}$ is used to evaluate individual task performance. Tasks might vary in size and difficulty. When individual task boundaries are known, the information of task label $t$ is accessible by the CL strategy. A **CL strategy**

refers to an algorithm or procedure that mitigates the problems accompanying CL and updates a neural network.
The availability of task labels $y_t$, allows us to distinguish between **supervised**, **semi-supervised**, and **unsupervised** CL scenarios. In the supervised scenario, for each data point a label is available. In the semi-supervised scenario, for some data points labels are available. In the unsupervised setting, no labels are provided at all. It is also important to consider if data is accessible, i.i.d., or correlated (online learning scenario).

CL problems are further categorized according to the tasks involved. Firstly the **Single-Incremental-Task (SIT)** scenario describes a CL scenario with a single objective for all tasks but changing data distributions. The deviation between different data distributions may induce catastrophic forgetting. The **Multi-Task (MT)** setting enforces different objectives for each task e.g. introduction of novel labels with each task. Less restrictive is the **Multi-Incremental-Task (MIT)** scenario. Here tasks can be revisited multiple times and at least two different tasks exist.

When designing a CL strategy, technical limitations including **training time**, **compute** and **storage capacity** have to be taken into account. Without these constraints, simple retraining from scratch for each newly available data point can be considered a viable solution. However, this fails the objective of CL research. Additionally, encoding of a priori known information into the learning strategy or network structure is desirable. It allows to limit the search space and can ease the optimization problem [32].

When selecting a CL strategy, the objective is of significant importance and highly use case dependent. For many applications, forgetting is not tolerable, or different tasks have varying relative importance, which should be reflected in the chosen learning strategy. Two terms often used in context with CL are **concepts** and **instances** - here the notion of a concept is inspired by a Bayesian perspective. To capture and therefore learn a new concept, a random latent variable $L_C$ has to be available. When a new instance of the previously encountered concepts is required, the learned latent variable $L_C$ can be utilized to reason about the data point containing the new instance. This Bayesian viewpoint has limited practical application when working with deep neural networks and will not be further used.

### 2.1.3 Strategies

CL strategies can be subdivided into four categories. These categories are not mutually exclusive and often can be applied orthogonally to one another. Therefore a variety of approaches exist, which combine elements of multiple categories.

**Architectural** approaches operate by modifying the architecture of the neural network. This includes methods introducing task-wise new layers [33]. Shared layers at the input level allow for a global feature extraction. Task independent and task specific layers in the form of prediction heads are used [34]. Other methods reorganize layer structures based on the data [8].

**Regularization** approaches regularize the training process of the neural network by limiting the optimization procedure [8]. The limitation of trainable parameters is studied task specifically [35] and with data dependence [36]. Other methods based on the Fisher information [37] or L2-norm weight decay minimize discrepancies in parameter space between the current network weights and a reference network.

**Rehearsal** based approaches or experience replay approaches store samples in a experience or memory buffer for past tasks. These experiences can be reconsolidated. Most approaches are based on random sampling of the data to a fixed size memory [38, 39, 40]. New task samples and stored samples are used to update the network parameters. Optimization on new samples motivates acquisition of new knowledge. The rehearsed/replayed samples motivate preservation of previous learned knowledge. These approaches show good performance if the memory buffer is capable to represent the previous tasks well [39]. Research has been carried out into the population of the buffer [38, 41], reconsolidation strategies of the buffer, and novel optimization algorithms which take the memory buffer into account in a more sophisticated manner than classic optimization procedures [42, 43]. Additional work in formulating CL as constrained optimization based on constraints induced by memory samples have been proposed [44].

**Generative Replay** relies on fitting a generative model based on experienced data. Generative models including Variational Autoencoders (VAEs) [45] or Generative Adversarial Networks (GANs) [46] have been utilized to tackle image classification tasks. Approaches using a single generative models to capture all past experience or task specific generative model approaches have been explored in [47, 48, 49]. During training of a novel task the generative models supply an effective method of sampling from the previous task distribution. These approaches are similar to the rehearsal based approaches constraining the optimization procedure. Fitting high quality generative models is not an easy objective and when failing to capture the relevant details in the generative model catastrophic forgetting may occur [8]. The generative model can be interpreted as a compression algorithm that allows us to efficiently extract and compress the current training data and store it to the generative models weights that can be reconsolidated later for sampling.

### 2.1.4 Metrics

Strict evaluation criterion and benchmarking procedures are an essential part of measuring real progress in any quantitative scientific field. Given the large number of constraints and degrees of freedom in CL, rigorous evaluation is key. To achieve this, one needs to be aware of the underlying assumptions and limitations of different benchmarking strategies and metrics.

Multiple metrics exist for measuring learning and forgetting. For each task the test dataset $D_t^{test}$ with suitable performance metric $m(y, \hat{y})$ is given with $m(y, \hat{y}) \in [0, 1]$. In the sequential task setting where one task after the other is encountered we can evaluate the model after training on task $t$ for all test datasets $j$. We denote the resulting performance evaluated with the metric $m$ as $R_{i,j}$.

The accuracy matrix provides an overview of the performance achieved with the rows indexing the number of trained tasks, and columns the individual test dataset. $R_{i,i}$ denotes the accuracy achieved on the test dataset of task $i$, after being training on the corresponding training dataset $i$. An example of the accuracy metric for a three task scenario is shown in figure 2.1 on the left, with color coded performance and numeric values in percentage.

**Average Accuracy (ACC)** allows to measure the overall accuracy achieved during the learning progress. It is given by the sum of the low left diagonal plus the trace of the accuracy matrix normalized by the number of tasks. The elements are highlighted (green) in the middle plot of figure 2.1. This metric weights the performance over the full training procedure on task 1, N times compared to the

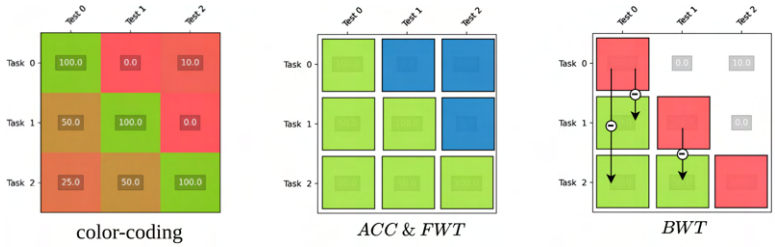



Figure 2.1: Example accuracy matrix. More details in section 2.1.4.

final task performance which is only evaluated once.

$$ACC = \frac{\sum_{i=1}^{N} \sum_{j=1}^{i} R_{i,j}}{\frac{N(N+1)}{2}} \tag{2.1}$$

**Forward Transfer (FWT)** is the normalized sum of all elements in the upper right triangle of the accuracy matrix (Fig. 2.1, middle blue). A positive forward transfer is given if the current task helps to learn concepts useful for future tasks. Therefore the accuracy of the future tasks is greater than 0.

$$FWT = \frac{\sum_{i=1}^{j-1} \sum_{j=1}^{N} R_{i,j}}{\frac{N(N-1)}{2}} \tag{2.2}$$

**Backward Transfer (BWT)** "measures the influence that learning a task has on the performance on previous tasks" [8, P. 16]. It is defined as the difference between the performance achieved on a task after training on another task (lower left diagonal elements) and the performance directly achieved after training on the original task (diagonal elements). This is illustrated in the right plot of figure 2.1. A positive backward transfer implies that in average the accuracy of previous tasks increased while training on further sequential tasks.

$$BWT = \frac{\sum_{i=2}^{N} \sum_{j=1}^{i-1} R_{i,j} - R_{j,j}}{\frac{N(N-1)}{2}} \tag{2.3}$$

**Overall performance ($\Omega$)** is different to the other metrics based on the fact that we can reformulate a CL classification setting to a classical learning problem. For this we provide all data available at the same time without any memory and compute restrictions. This is should result in the optimal performance if trained correctly. We denote the performance of the network trained on all data till task $i$ evaluated on task i as $R^C_{i,i}$, which is an upper bound for the achievable network performance. The overall performance is the ratio of the CL performance normalized by the best possible performance in average over all tasks. This metric is designed to disentangle the difficulty of a task from the measurement of catastrophic forgetting.

$$\Omega = \frac{1}{N} \sum_{i=1}^{N} \frac{R_{i,i}}{R^C_{i,i}} \tag{2.4}$$

In practice this metric might be misleading given that $R^C_{i,i}$ is often not known or a specific scenario is created where favorable hyperparameters for the CL strategy are chosen. This general problem of creating a fair comparison is reconsidered later when performing CL experiments.

**Other metrics** include learning speed, model size(MS), and sample storage size(SSS). These metrics focus on measuring the complexity in relation to growing task numbers [8].

## 2.2 Semantic Segmentation

### 2.2.1 Task Definition

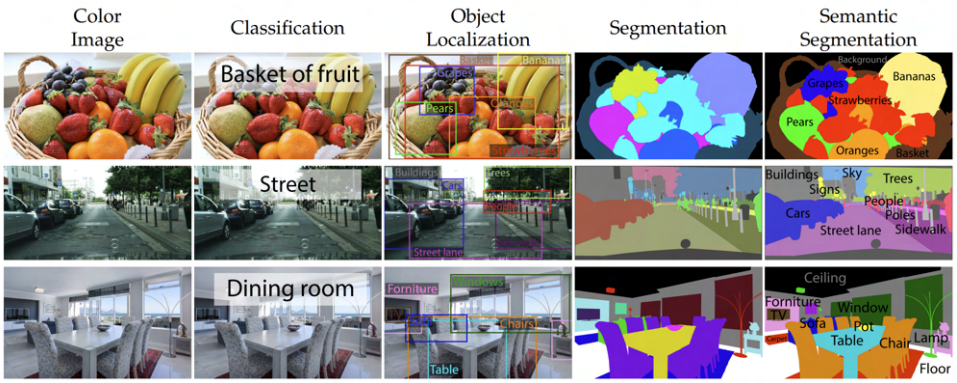



Figure 2.2: Examples of image understanding tasks. Source [31, p. 2].

**General:** Semantic segmentation assigns a label to a location from a predefined set according to its semantic association. It is an essential part of image understanding and modern vision pipelines used for autonomous navigation [26], medical imaging [50], social media [51], graphic design, robotics, and many other applications [31]. Different modalities of data, including point clouds, images, or radar data are used. In this thesis, we limit the scope to learned RGB image segmentation. In figure 2.2 the differences between classification, localization and segmentation tasks is illustrated. Classification and localization are sparse tasks given that labels are not assigned for every pixel. In the dense task of segmentation, pixels are labeled into distinguished categories. The color-coding indicates different categories. In semantic segmentation, a class (color in the image) is assigned to semantic label.

**Notation:** We will follow the mathematical notation used in [52]. The task of semantic segmentation is defined as finding a suitable function $h : \mathcal{X} \rightarrow \mathcal{Y}$ projecting from an input domain $\mathcal{X}$ (RGB-Image) to the label space $\mathcal{Y}$. One data sample is denoted as $(x, y) \in (\mathcal{X}, \mathcal{Y})$ and drawn from an unknown probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. The chosen network architecture limits the search space of possible candidate function $h$ to subset of functions $\mathcal{H}$. The source training dataset $D_s$ consists of $n \in [0, N)$ image-label pairs $(x_n, y_n)$. The objective is to find the function $h$ fitting the training data optimally according to some criterion $L$.

**Loss functions:** The two most common loss functions used in semantic segmentaion are the cross entropy and focal loss [52]. We denote $q_c$ as the probability that the target label belongs to class $c \in [1, .., C]$ and $p_c$ as the predicted probability. We obtain $p_c$ by applying a softmax function to the segmentation prediction head, resulting in a valid probability distribution.
The cross entropy is given by:

$$CE(p_t) = -\sum_{c=1}^{C} q_c \log p_c \tag{2.5}$$

The focal loss reshapes the cross entropy loss function and down weights easy examples. This helps when training on class imbalanced datasets [53].

The focal loss is given by:

$$FL(p_t) = -\sum_{c=1}^{C} q_c(1-p_c)^{\lambda} \log p_c, \quad with\ \lambda \geq 0 \tag{2.6}$$

**Evaluation Metrics:** Semantic segmentation performance is measured using the accuracy ($Acc$) and mean intersection over union ($mIoU$) metric. $Acc$ is obtained by counting the total number of correctly classified pixels divided by the total amount of pixels:

$$Acc = \frac{TP + FP}{TP + TN + FP + FN} \tag{2.7}$$

Here we denote the number of true positive classifications (TP), true negative (TN), false positive (FP) and false negative (FN).
The intersection of union is computed as:

$$IoU = \frac{TP}{TP + TN + FN} \tag{2.8}$$

In a multi-class setting the $mIoU$ is computed as the mean over each class [54]. All TP, TN, FP, FN have to be accumulated class-wise over all images in the test set before calculating the mean over the classes. There have been different $IoU$ metrics referred to as $iIoU_{class}$ and $iIoU_{category}$ proposed but they are less frequently reported [55].

### 2.2.2 Network Architectures

Convolutional Neural Networks (CNNs) currently achieve top performance on all semantic segmentation benchmarks [31]. Most research done so far has focused on Encoder-Decoder structures, which compress an input image to a lower dimensional latent space. The decoder predicts the semantic segmentation [56, 50]. In the encoder, multiple levels of convolution and pooling layers extract latent features. With increasing depth, the receptive field increases and allows for information accumulation of a larger input area of the original image. Subsequently, in the decoder upsampling and deconvolution layers, we decompress the lower dimensional latent space to the original input image resolution.

U-Net [50] introduces dense skip connections between the encoder and decoder feature maps similar to ResNets [57]. Global context pooling modules such as pyramid pooling module (PSPNet [58]) and atrous convolutional layers (DeepLab [25]) achieve impressive performance on multiple benchmarks. In DeepLab [25] additional Conditional Random Field (CRF) postprocessing further increases the performance. Later, novel information aggregation layers [59] or attention mechanisms [60] pushed the state-of-the-art without the need for postprocessing.

Fast-SCNN [61] operates similarly to [59] and introduces the 'learning to downsample' module. This significantly decreases the inference time while achieving competitive results. In figure 2.4 the architecture of Fast-SCNN is illustrated. First, features are extracted using 2D convolutions and consecutive fast depth-wise separable filters. The encoder consists of a 2-branch architecture. The two feature branches are fused using standard 2D convolutions. Two depth-wise separable convolutions (DSConv) along with one normal 2D convolution operation upsample the fused features. The network in total consists of $1.1M$ parameters. Fast-SCNN achieves a throughput of 285.5 fps at an input resolution of $512 \times 1024$ using a Nvidia Titan X. The small memory footprint and fast inference time led us to choose Fast-SCNN as

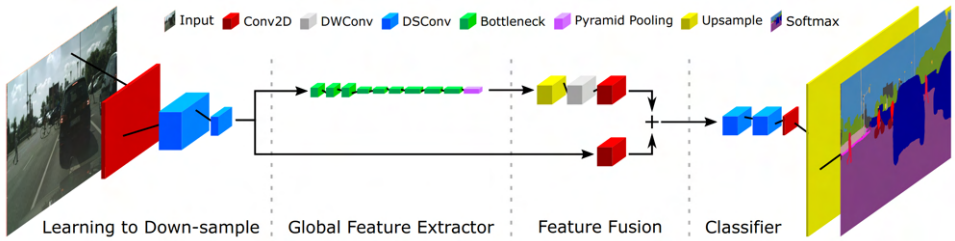



Figure 2.3: Fast-SCNN model architecture. Source [61].

our standard semantic segmentation network architecture.

Of note is the recent progress in attention-based models using transformer architectures [51]. We refer the reader to further information on modern deep learning techniques for semantic segmentation in the exhaustive survey [31].

### 2.2.3 Domain Adaptation

The underlying assumption for the generalization capability of a neural network is the similarity between the source $\mathcal{D}_S$ and target domain $\mathcal{D}_T$. A sample drawn from $\mathcal{D}_S$ is also part of $\mathcal{D}_T$ and vice-versa. If $D_S$ is too dissimilar to $D_T$, because a domain shift or change is present, we observe poor performance [52]. We refer to the domain difference as the "domain gap". Overcoming this gap is explicitly researched in the field of domain adaptation.

One major limitation in the training of semantic segmentation networks is the tedious labor and cost associated with generating high-quality densely annotated frames. For this reason, most datasets only contain a small subset of images with high-quality per-pixel label annotations [62, 54, 55].

This further complicates the problem of generating a high-quality source dataset covering the entire target domain. Domain adaptation has the potential to reduce the need for expensive real-world human-annotated datasets significantly.
Popular benchmarks for domain adaptation quality include adapting from comparably cheap synthetic datasets to expensive real-world data, primarily focusing on autonomous driving [63, 64].

Domain adaptation strategies can be categorized according to the level of supervision and overlap in terms of semantic categories between the source and target domain. For our application in mind, we focus on unsupervised methods where the same semantic categories appear in both domains (unsupervised closed set domain adaptation).

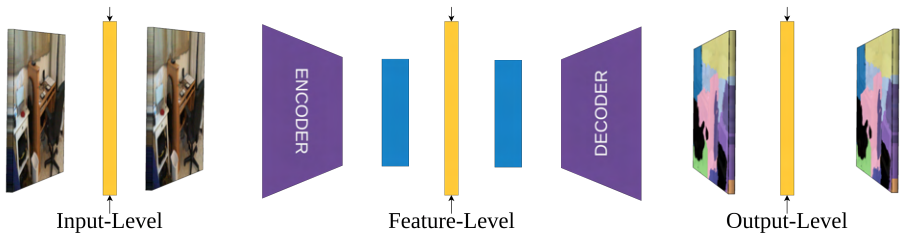



Figure 2.4: Domain adaptation level.

Additionally, we can distinguish methods based on their mode of operation into three categories. Input level approaches: solve the problem of domain adaptation by modifying the input to the neural network such that the image is more similar to the source domain. For this, a variety of input level style transfer methods have been proposed [65, 66], to map a target distribution sample meaningful to the source domain. These approaches often produce transformed images with a lack of semantic discriminativeness [52].

Feature level approaches tackle the problem by modifying or regularizing the latent feature representation of the network - the objective of class-wise and global alignment of source and target features motivates learning of discriminative domain-invariant features. Ganin et al. [18] propose Domain-Adversarial Neural Network (DANN). Different from standard feed-forward networks DANN features a domain classifier network. The domain classifier is provided with latent features and induces a loss on the feed-forward network when the class is correctly identified. The induced loss motivates embedding of the target domain and source domain samples to indistinguishable domain invariant features. CyCADA [67] introduces adversarial domain adaptation by leveraging cycle, feature, and image-level consistency. Other work includes self-training approaches where iteratively, according to a design curriculum, pseudo labels are generated, and the network is retrained [17].

Lastly, output level approaches solve the domain adaptation problem based on the output prediction of the network. For this, properties such as label statistics are used. The output level approach is primarily used if the source and target domain are known. With knowledge about both domains, it is feasible to compute a low-dimensional mapping of the output source to the target domain [19].

## 2.3 Semantic Supervision Signal Generation

To adapt a neural network a learning signal must be available that can be used to update the network. When predicting semantic segmentation on a mobile robot we are not limited to extracting all information from a single view image. Firstly a robot is equipped with multiple sensors. Secondly, a robot navigating within its environment observes the scene from different viewpoints. Lastly, a robot can actively decide where to collect more data.

Focusing on a passive restriction-less system, not interfering with the actual mission of the robot, we do not further investigate active data collection. In section 2.3.1 single image-based *clustering and regularization* strategies are reviewed. They assume that neighboring pixels, texture, and shape of the scene provide rich information for clustering and segmentation.

Section 2.3.2 discusses *optical flow* estimation techniques allowing for label propagation over time, when a continuous data stream is available. A brief overview of optical flow estimation techniques and work related to semantic segmentation label propagation is given.

In section 2.3.3 we provide an overview of *3D scene reconstruction* tools. These allow for global and local information accumulation. For this, data from various viewpoints is fused into a map.

### 2.3.1 Clustering and Regularization

Superpixel approaches reduce the number of primitives by means of clustering. Most superpixel approaches solely rely on RGB images and trade-off boundary adherence for compactness. Figure 2.5 (far left image) illustrates an exemplary superpixel segmentation. Methods include graph-based, clustering-based, energy optimization approaches and other [68]. Simple linear iterative clustering (SLIC [24]), iteratively associates pixels to uniformly initialized clusters across the image and recomputes the cluster center. The 5 dimensional cost function consisting of 3D CIELAB color space and pixel coordinates is minimized until the algorithm converges according to the following loss function:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}\ , \tag{2.9}$$

$$d_{uv} = \sqrt{(u_k - u_i)^2 + (v_k - v_i)^2}\ , \tag{2.10}$$

$$D_s = d_{lab} + \frac{m}{S} d_{xy}\ , \tag{2.11}$$

where $d_{lab}$ denotes the euclidean distance in the CIELAB color space and $d_{uv}$ distance in the $uv$ image plane in pixels. $S$ is a normalization factor given by the chosen grid distance proportional to the number of superpixels, and $m$ is the compactness hyperparameter.

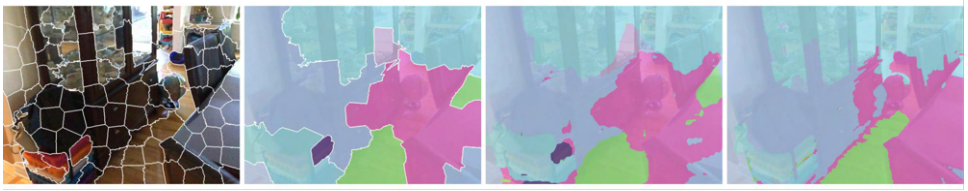



Figure 2.5: Examples of SLIC and CRF post-processing: Far left SLIC segmentation, left SLIC clustering, right original network prediction, far right CRF post-processing output.

CRFs are a probabilistic graphical model originally introduced for the segmentation and labeling of sequence data [69]. Neighborhood relations of pixels are explicitly encoded into the model in the form of conditional independence assumptions. First CRF approaches in the field of segmentation clustered regions and modelled them as individual random variables [70]. The clustering reduces the computational complexity compared to performing full inference on pixel level.

Krähenbühl and Koltun [71] introduce a highly efficient inference algorithm by limiting pairwise edge potential to a linear combination of Gaussian kernels, allowing for dense pixel-wise CRF modeling. Each pixel in the image $I(u, v)$, with $N$ total pixels is modeled by a random variable $\mathbf{X}_n$. Each random variable takes a value in the set of $\{1, c\}$ labels. We want to maximize the probability of $P(\mathbf{X}|\mathbf{I})$ with respect to all $\mathbf{X}_n$ random variables. For this, the Gibes distribution is used to model the conditional distribution.

$$P(\mathbf{X} = \hat{x}|\bar{I} = I) = \frac{1}{Z(I)} exp(-E(\hat{x}|I)) \tag{2.12}$$

with a suitable energy function:

$$E(\hat{x}, I) = \sum_{i \leq N} \psi_u(\hat{x}_i|I) + \sum_{i \neq j \leq N} \psi_p(\hat{x}_i, \hat{x}_j|I)\text{,} \tag{2.13}$$

Here $\psi_u(\hat{x}_i|I)$ measures the cost of assigning a label to a pixel with regards to some initial segmentation algorithm. It motivates agreement with the initial estimate and penalizes disagreement. The function $\psi_p(\hat{x}_i, \hat{x}_j|I)$ models the pairwise potential between pixels. Similar to SLIC it allows the introduction of a similarity metric based on the position and color encoding. We can solve for $P(\mathbf{X}|\mathbf{I})$ using the Mean-Field Approximation and an effective message passing strategy to obtain a solution for the dense-CRF problem within milliseconds.

### 2.3.2 Optical Flow Estimation

Optical flow is defined as the disparity vector field between two consecutive images. Each pixel in image $I_1(u_1, v_1)$ can be mapped to the consecutive image plane of the next frame $I_2(u_2, v_2)$ as follows:

$$\begin{pmatrix} u_2 \\ v_2 \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} + F(u_1, v_1) \tag{2.14}$$

Optical flow is induced by the relative motion between the observer (camera) and the scene. Before deep learning, optical flow estimation was dominated by variational methods [72].

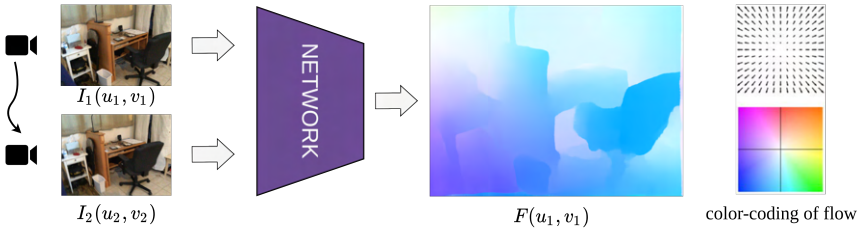



Figure 2.6: Optical flow estimation example, right color coded disparity field.

FlowNet [73] is one of the first deep learning architectures that pioneered end-to-end trained optical flow estimation networks. A CNN extracts meaningful features from two consecutive images, and a decoder network generates a per-pixel flow prediction. Further developments in the field such as FlowNet 2.0 [74] and PWC-Net [75] achieve superior results, reducing memory requirements while fully real-time capable. Recently the authors of [76] achieve state-of-the-art performance on the optical flow KITTI [77] and Sintel [78] benchmarks. They employ a recurrent refinement network to improve an initial optical flow estimate iteratively. Classical optimization procedures inspired the design of the recurrent architecture. It intrinsically features weight sharing and is fully trainable end-to-end.

The authors of [26] use a learned optical flow network for tracking segmented object instances over time. Nilsson and Sminchisescu [79] efficiently semantically segment video sequence by forward propagation of past predictions using optical flow. A Spatial Transformer Network [80] fuses the propagated and new label prediction. The authors of [81] effectively tackle the task of moving object segmentation by predicting optical flow and labels jointly. Reversely, Jain et al. [82] predicts semantic segmentation from an optical flow and RGB image. The model learns to fuse motion and appearance. In [83] optical flow is used to instantiate super-trajectories, representing tracks of semantic segmentation over time.

### 2.3.3 3D Semantic Reconstruction

Using optical flow to propagate labels over time accumulates propagation noise. Even with precise modern optical flow estimation methods [76], noise can significantly degenerate the propagated label over long horizons.

A more powerful tool is using 3D semantic reconstruction. It allows fusing semantic information into a map. The map allows the integration of information from various viewpoints over long trajectories. The fundamentals for 3D semantic reconstruction are well researched in the fields of simulations localization and mapping (SLAM), structure from motion (SfM), visual odometry (VO), and scene reconstruction work. To efficiently limit the scope, we focus on reviewing work capable of:

1. Mapping room size scenes
2. Integration of semantic information
3. Real-time deployment on limited hardware
4. Operating on RGB-(D) data (optionally IMU)

These four criteria allow deployment on a robotic system. Limiting the data sources to RGB-(D) sensors allows scaling to multiple robotic systems independent of the used locomotion (wheeled, legged, flying).

All reviewed open-source libraries achieve the task of semantic reconstruction by initial estimation of locally consistent camera trajectories. Additionally, global bundle adjustment and loop closure detection are integrated into multiple systems, allowing for refined pose estimates. Some tools do not support adequate online adaptation of the reconstruction given the computational burden associated [84, 23]. A more precise reconstruction of the map can be constructed in hindsight with refined camera pose estimates. The reviewed approaches diverge in terms of the geometry used to associate structure with semantics.

In table 2.2 we provide a summary of 3D semantic reconstruction tools.

| Method | Sensors | Geometry | Real-Time | Integration |
|---|---|---|---|---|
| **SLAM++ [85]** | RGB-D | objects | 20fps | Bayesian |
| **SemanticFusion [86]** | RGB-D | surfels | 25fps | Bayesian |
| **XIVO [87]** | mono/IMU | objects | ✓ | Bayesian |
| **Voxblox++ [84]** | RGB-D | TSDF | × | NaN |
| **Kimera [23]** | mono/stereo/IMU | mesh/TSDF | 30 fps | Bayesian |

Table 2.2: Comparison of 3D reconstruction tools.

SLAM++ [85] creates a semantic segmentation at an object level. A 6D object detection algorithm extracts object poses in the scene. The objects are tracked in an object graph that is iteratively refined using pose graph optimization and utilized for re-detection of objects. SemanticFusion [86] utilizes a CNN to predicted dense semantic segmentation from individual RGB frames. ElasticFusion [88] is used to provide reliable camera pose tracking and creates a surfel reconstruction of the scene. Each surfel present in the scene stores a label probability distribution. At each semantic segmentation estimate, the associated surfels are recursively updated using Bayesian filtering. Additional CRF map regularisation increases map consistency, exploiting geometrical properties such as convexity of structures comprising

a strong signal about the semantic relationship.

In XIVO [87] an object detector is used to detect semantics at an object level. The pose of an objects is model probabilistically. A set of Extended Kalman Filters (EKFs) track the objects pose by estimating the posterior.

VoxBlox++ [84] is based on Voxblox [89] and implements volumetric aware semantic mapping and 3D object discovery. Voxblox provides the scene reconstruction backend by generating a voxel-based volumetric map using a Truncated Signed Distance Function (TSDF). Voxblox is CPU-based only and suitable to reconstruct room size scenes in real-time at a resolution $\leq 3cm$. Depth measurements by an RGB-D sensor are integrated into the TSDF by means of ray tracing.

Kimera [23] also uses Voxblox as a backbone. It provides a robust SLAM system capable of mono and stereo operation with IMU preintegration. Each voxel in the map stores a label probability distribution. Similar to SemanticFusion on a surfel level, Kimera stores and tracks label probability distributions at a voxel scale. The predicted semantic segmentation estimate is associated with the corresponding depth measurement and used to update the voxel probability distribution using Bayesian filtering.

This paradigm bears great potential while being the most complex and compute-intensive compared to previously mentioned methods.

## 2.4 Tools and Frameworks used

In table 2.3 we listed the most important tools, frameworks, and implementations used to conduct all experiments.

| Tool | Description | License | Link / GitHub |
|---|---|---|---|
| **PyTorch** | Deep Learning Framework | BSD | `pytorch.org` |
| **OpenCV** | Computer Vision Library | Apache2 | `opencv.org` |
| **Scikit-image** | Image processing in Python | BSD | `scikit-image.org` |
| **NeptuneAI** | Logging | - | `neptune.ai` |
| **Trimesh** | Triangular mesh toolbox | MIT | `trimsh.org` |
| **Avalanche** | Library for Continual Learning | MIT | `avalanche.continualai.org` |
| **ROS** | Robot Operating System | ROS | `ros.org` |
| **SimpleCRF** | CRFs for image processing | BSD3 | iLab-git/SimpleCRF |
| **Kimera** | Real-Time 3D Semantic Reconstruction from 2D data | BSD2 | MIT-SPARK/Kimera-Semantics |
| **Orbslam2** | SLAM | GPLv3 | raulmur/ORB_SLAM2 |
| **RAFT** | Implementation of RAFT | BSD3 | princeton-vl/RAFT |
| **Fast-SCNN** | Implementation of Fast-SCNN | Apache2 | Tramac/Fast-SCNN-pytorch |

Table 2.3: Overview of used tools and frameworks.

# Chapter 3

# Continual Learning of Semantic Segmentation

## 3.1 Problem Description

Semantic segmentation is an essential task for various use cases with constant changes of data distribution in the wild. To operate a mobile robot in an unknown environment without known semantics, we explore the possibility of applying CL for semantic segmentation, allowing for adaptation and learning during the mission. We focus on indoor scenes for two main reasons. Firstly, unlike autonomous driving, indoor scenes feature a wide variety of classes. Depending on the scene, different labels are necessary to explain the semantics (kitchen $\leftrightarrow$ bathroom), compared to the relatively homogeneous and structured environment of self-driving. For self-driving applications, the complete set of labels appear within different cities or countries. Additionally, operating robots indoors has significant practical applications, including elderly care, service robotics, or search & rescue missions e.g. inside burning buildings. With the sparse literature in CL for dense semantic segmentation [20, 90, 91], no standard benchmarking procedure exists. We firstly evaluate different datasets and propose a sophisticated CL setup for semantic segmentation. Secondly, we provide a strong baseline on the provided benchmark for both introduced challenges applying experience replay methods.

## 3.2 Benchmark

### 3.2.1 General

During the design phase of the benchmarking procedure, we focus on incorporating the most common CL best practices and strictly designing a benchmark with real-world applications in mind. We formulate the CL problem in the MIT scenario, which is the most generic. The MIT formulation covers scene-dependent labels and revisiting scenes multiple times. Typical robotic indoor tasks require navigation between different rooms that strongly deviate in their semantics multiple times. We have chosen to create two tracks based on the availability of supervision.

**Track supervised** provides full ground truth annotations for all tasks. The provided labels disentangle the problem of generating a supervision signal from the CL strategy itself.

**Track semi-supervised** exclusively provides ground truth annotations for a pre-training task. The actual target tasks do not have semantic annotations available. This scenario precisely reflects real-world robotic use cases. Large annotated datasets are available before the mission for pre-training, but the robot needs to adapt during the mission in a self-supervised manner.

Each task consists of a scene/environment which is mutually exclusive from all other tasks. We assume well-defined task boundaries. Existing work in pose retrieval based on visual similarity [92], mission scheduling, and uncertainty detection [93] allow us to reason about whether a robot has entered a novel environment.

### 3.2.2 Data

We evaluate different datasets. In 3.1 we state the minimal and optimal properties of a suitable datasets.

| **Minimal** | **Optimal** |
|---|---|
| Indoor scenes | Camera trajectories |
| Pixel-wise dense semantic segmentation | Depth data |
| Multi-class | Create mew scenes |
| Multiple scenes | |
| Real or photorealistic rendered textures | |
| High-quality labels | |
| Open-source licensed | |

Table 3.1: Overview requirements dataset.

| | **Real** | | | | **Render** | |
|---|---|---|---|---|---|---|
| | **NYUv2** [62] | **SUN** RGB D [94] | **sceneNN** [95] | **ScanNet** [21] | **SceneNet** RGB D [96] | **Hypersim** [22] |
| **Ann.** | KF | KF | Videos | Videos | Video | KF |
| **Scenes** | 464 | - | 100 | 1513 | 57 | 461 |
| **Ann. I.** | 1449 | 10K | - | 2.5M | 5M | 77.4K |
| **Type** | real | real | real | real | p. | p. |
| **Traj.** | × | × | × | yes | yes | yes |
| **Depth** | yes | yes | × | yes | yes | yes |

Table 3.2: Evaluation of existing datasets: Annotations (Ann.) are either provided for key frames (KF) or the full video. Scenes is the total number of scenes. Ann. I. denotes the total number of annotated images. Type of data is either real data (real) or photorealistic renderings (p.). Traj. the availability of camera trajectories. Depth denotes if depth measurements are provided.

ScanNet features all the requirements set. It contains trajectories recorded with a Kinect sensor. BundleFusion [97] is used to create a precise 3D reconstruction of each scene offline. An annotation tool allows for human annotation of the created mesh map. Individual frames are labelled by re-projecting the annotated mesh. ScanNet consists of 2.5M densely per-pixel labelled images. The scenes feature a variety of objects encountered indoors. Additionally, ScanNet is fully open-source and provides an easy way of capturing and annotating new scenes.

For synthetic data, we reduced the choice down to two candidates: SceneNet RGB-D and Hypersim dataset. We favored the Hypersim dataset based on the high level of

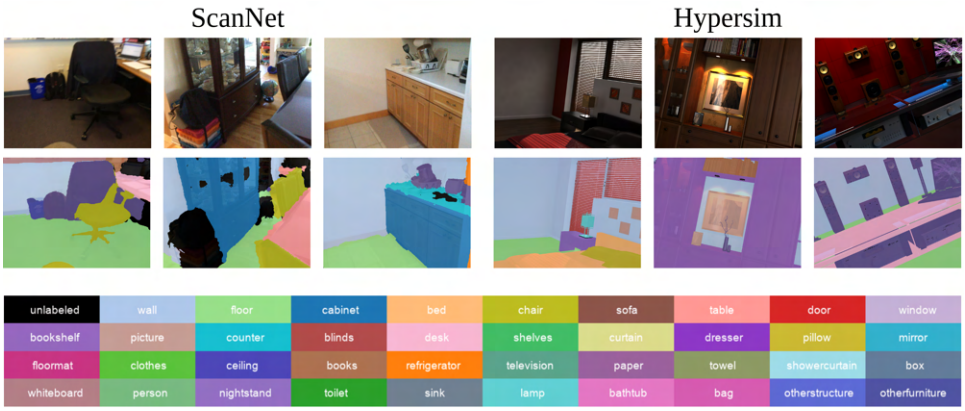



Figure 3.1: Example illustration of the ScanNet and Hypersim datasets. Illustration of three images with ground truth corresponding annotation. NYU-40 color code.

detail and photorealistic rendering. All scenes are created by professional artists. It consists of 461 individual scenes compared to SceneNet RGB-D with 57. The key-frame-based data is a drawback compared to the video data available for SceneNet RGB-D regarding self-supervised learning. We choose to provide a synthetic and real dataset for the fully supervised task. The self-supervised track is implemented using the ScanNet dataset to most realistically represent a robotic mission. For both datasets, NYU40 classes are available. Examples of both datasets are provided in figure 3.1 with the corresponding color-coding for NYU40 labels.

### Hypersim

Hypersim consists of a set of trajectories for each scene. The total trajectory length of all trajectories in a scene is denoted as $N_s$. The first 80% of frames per scene are used for training and the last 20% for validation. The trajectories are randomly generated. The training and test datasets might overlap in terms of visible parts of the scene. It is improbable that both contain the same viewpoints and lighting conditions. On the other hand, the test dataset is a good performance indicator. Visual features such as floor material, wall painting, or interior styles are consistent within a scene but deviate between scenes. The difficulty level is scene-dependent, given semantic distinguishability, differences in the overlap of the test and train sequence, lighting conditions, and other factors. We denote **HypersimXX-PY** as a specific implementation where XX indicates the total number of scenes/tasks sequentially experienced and Y the number of order permutations used. For experiments, we use **Hypersim04-P1** allowing a short turn-over time and rapid development. We use the Hypersim dataset to verify all results achieved on the ScanNet dataset in the supervised scenario.

### Scannet

For ScanNet we follow the same procedure as for Hypersim. We use the same 80%-20% train test split trajectory-set-wise. The original video sequence is recorded at 30fps. To reduce the size of the trajectories, we subsample every 10th frame. We denote these experiments with **ScanNetXX-PY**. Additionally we introduce **ScanNetXX-PY+**. The + denotes a ScanNet specific pre-training task. The pre-training task consists of 25k frames over the entire 1513 scenes, containing roughly every 100th frame. We exclude all frames from the first ten scenes used for the CL tasks. The ScanNet dataset is used for the **supervised** and **semi-supervised**

track. In the semi-supervised scenario each datapoint in task $t$ consists of a tuple containing the image, depth map, camera intrinsics, camera extrinsics, trajectory frame ID and task count respectively $(I_i, D_i, K, P_i, i, t)$. In the supervised track, additionally the ground truth label $y_{gt}$ is available.

### 3.2.3 Evaluation Metrics

The CL performance is evaluated using the following standard metrics introduced in 2.1.4: **Average Accuracy (ACC)**, **Backward Transfer (BWT)**, **Forward Transfer (FWT)**. Instead of reporting the **Overall performance ($\Omega$)** we introduce the **Final Performance ($\Omega_F$)**. It is given by the performance achieved after training on all tasks normalized by a network trained on all data simultaneously. This metric is sensitive to forgetting and does not explicitly measure forward transfer. It indicates the final performance achieved at the end of the CL problem. $\Omega_F$ greater one implies that the incremental CL setup outperforms training with all data available at the same time.

$$\Omega_F = \frac{1}{|T|} \sum_{i=1}^{|T|} \frac{R_{|T|,i}}{R^C_{|T|,i}} \tag{3.1}$$

When evaluating $\Omega_F$ the same pitfalls apply as for evaluating the overall performance $\Omega$. Given that retraining the network on all data has to be performed only once, $\Omega_F$ is easier to evaluate. Changing dataset sizes impacts the optimal hyperparameter choices for the optimizer and learning rate schedule.

## 3.3 Rehearsal / Memory Buffer

Rehearsal approaches have shown great success on CL problems. They have been used to tackle multiple different continual learning scenarios [5]. The random replay buffer is a specific instance of a rehearsal-based CL learning strategy. During training on task $t$, samples of task $t$ are stored in a memory buffer $M$. When training on further tasks, the stored samples can be reconsolidated or replayed for training. Because of simplicity and effective catastrophic forgetting avoidance, random replay buffers are used as hard baselines for multiple CL problems [12].

### 3.3.1 Constrained optimization

Reformulation of CL as a constrained optimization motivates replay-based methods [98]. In this setting, each previous sample establishes a constraint. Provided a new sample, the objective is to find the optimal network parameters $\theta^*$ such that the loss function is minimized, without increasing the loss on any previously seen sample. We refer to this as learning without forgetting.

$$\theta^*_k = argmin_\theta \;\; l(f(x_k;\theta), y_k) \tag{3.2}$$
$$s.t. \quad l(f(x_i;\theta_k), y_i) \;\leq\; l(f(x_i;\theta_{k-1}), y_i); \quad \forall i \in [0, .., k-1]$$

The number of constraints grows linearly with the number of samples. In theory, given the large number of parameters in a neural network, optimizing the current network parameters $\theta_k$ is possible while applying to $k$ constraints, established by previous samples. Generally, when theoretically solving the sequential optimization problem 3.2, the solution is different from the one retrieved when all data is available simultaneously. This mathematical objective would motivate sequential overfitting

to every training sample. At some point, the constraints prohibit every parameter updated of the network, effectively freezing the network. The linear growth of constraint and high degrees of freedom make solving this optimization problem intractable. Noticeably, with an increasing number of constraints, redundancy is likely. This directly motivates the design of memory-based approaches, which store a subset of the previous data.

### 3.3.2 Random Memory Buffer

Most neural networks are optimized using 1st order gradient-based optimizer. Stochastic gradient decent (SGD) optimizes the parameters $\theta$ of a neural network $f_\theta$ over a mini-batch, consisting of $n$ samples, according to a differentiable loss function $l(\hat{y}, y)$. The differentiable loss function $l(\hat{y}, y)$ measures the discrepancy between the predicted label $\hat{y}$ and ground truth label $y$. The learning rate parameter $\mu$ governs how fast the weights are changed.

$$\theta_{t+1} = \theta_t - \frac{\mu}{n} \sum_{i=1}^{n} \frac{d}{d\theta} l(f_\theta(x_i), y_i) \tag{3.3}$$

When utilizing a replay buffer (stored previous samples) we can directly apply SGD to the same problem. We mix samples from the previous tasks and current task and apply the same weight update 3.3. We can further distinguish between the gradient induced by replayed and new samples.

$$\theta_{t+1} = \theta_t - \frac{\mu}{n_{new} + n_{replay}} \frac{d}{d\theta} \left( \sum_{i=1}^{n_{new}} l(f_\theta(x_i), y_i) + \sum_{j=1}^{n_{replay}} l(f_\theta(x_j), y_j) \right) \tag{3.4}$$

The gradient induced by the $n_{replay}$ samples motivates a decrease in loss on the stored samples and therefore impedes stability interpreted as "good performance" on old samples. Indifferent to regularization approaches, rehearsal approaches do not explicitly penalize deviation of the model parameters. The gradients induced by the $n_{new}$ samples motivate reducing the loss on the new task and therefore the integration of new knowledge, which directly tackles the stability-plasticity dilemma.

### 3.3.3 Specifications

In the design of a memory buffer CL strategy multiple design decisions have to be made. In the following, we introduce suitable notation to describe these decisions.

#### Task dependency

Storing data of the previous task can be either achieved in a task dependent or task independent manner. A task-dependent memory buffer $M_t$ stores samples from the task $t$. A new buffer is populated for each task. Task dependent memory buffers can only be used when task boundaries are known. A task-independent memory buffer $M$ stores samples from all previous tasks available. For our experiments, we only consider task-dependent memory buffers, given that task boundaries are known.

**Choice:** Task-Dependent, Task-Independent **Default:** Task-Dependent

### Buffer Size

The buffer size determines the total number of samples, which can be stored. The buffer size is proportional to the memory requirements of the CL strategy. With unlimited buffer size, the CL problem reduces to classical optimization procedure with all data accessible, if other constraints are not considered. The buffer size is a typical engineering trade-off parameter and depends on the amount of data and storage available. The buffer size can be chosen as a fixed size or dependent on the total amount of samples per task.

**Choice:** Fixed (F-XXX), Percentage(P-XX) **Default:** P-10

### Filling Strategy

The random memory buffer is denoted as random because the samples are stored in the memory based on random sampling. In equation 3.2 we observe that not all samples imply useful constraints given the expected constraint redundancy. We investigate different methods of populating the memory buffer, namely: uncertainty/confidence, loss, label statistics, trajectory coverage, and latent feature-based. We provide further details on the exact implementation in the corresponding experiments section.

### Sampling Strategy

The sampling strategy determines how many and which samples are rehearsed. A sample is either selected by random or according to some curriculum or metric. In our experiments, we only evaluate random sampling. We propose different strategies further elaborated in the experiments section 3.4 for the number of samples replayed per batch.

**Choice:** Random-Sampling (RS) **Default:** RS

### Latent Replay

Latent replay allows us to reduce storage size and increase training speed. Instead of storing the full input $(x_i, y_i)$ where for our task $x_i$ is a high-dimensional RGB image, the authors of [99] showed that only replaying samples at the feature level storing $(z_i, y_i)$ can achieve similar performance. In this setting, only the lower layers of the network are task-dependently fine-tuned.

**Choice:** Latent Replay (LR-BOOL) **Default:** LR-False

### Data Augmentation

Data augmentation is an essential part of training large neural networks on image data [100]. With access to the original image (no latent replay), we can apply standard augmentation methods, including color jitter, random cropping, rotations, and flips at the input level. These human-designed data augmentation methods do not change the label of the underlying image. They force the network to learn a solution invariant to the applied transformations [100]. We investigate the performance difference of augmenting samples in the memory buffer. As our experiments show, this is a performance critical parameter and often neglected in the CL community.

**Choice:** Data Augmentation (DA-BOOL) **Default:** DA-True

**Label Type**

In the student-teacher, curriculum learning, and knowledge distillation literature, a common practice is to train on soft labels. A soft label provides a probability distribution over the label classes. It is usually inferred by a teacher network. The knowledge encoded in the soft labels generated by a teacher network is shown to be a good training signal. This might be due to information about the similarity between classes or, in general, a easier optimization objective [101]. Multiple works [102, 101] show the effectiveness of soft labels for training.

$$l(\hat{y}, y) = \frac{1}{n_{soft} + n_{hard}} \left( \sum_{i=1}^{n_{hard}} l_{ENT}(\hat{y}_i, y_i) + \sum_{j=1}^{n_{soft}} l_{MSE}(\hat{y}_j, y_j) \right) \tag{3.5}$$

We use a separate loss function for soft and hard labels (equ. 3.5). As introduced in section 2.2 we minimize the cross-entropy and mean squared error (MSE) for hard and soft labels respectively. A hyperparameter $\lambda$ weights the new and replayed loss terms. In our experiments in the supervised setting the soft labels are used for the replayed samples and the current task labels are the ground truth one-hot-encoded labels.

**Choice:** GT-Label, Soft-Label, Hard-Label **Default:** GT-Label

## 3.4 Experiment

By default we use the ScanNet4-P1+ setting. For the hyperparameter evaluation we report numbers for the Hypersim4-P1 setup. Experimentally a single permutation is sufficient to reliably reason about relative performance impacts of hyperparameters. We report Hypersim results if they deviate from the results achieved on the ScanNet experiments. If the Hypersim results align with the ScanNet experiments, we only report the results in the appendix for completeness. Additionally, a task size of 4 limits the computational burden and at the same time allows for clear observation of catastrophic forgetting. The statistics for both datasets are given in table 3.3.

| | **Hypersim** | | | | **ScanNet** | | | |
|---|---|---|---|---|---|---|---|---|
| Task | Train | Test | Scenes | Traj. | Train | Test | Scenes | Traj. |
| **Pre-training** | - | - | - | - | 19492 | 4873 | all | - |
| **Task 1** | 6660 | 1995 | Ai_001 | 10 | 1408 | 352 | scene0000 | 3 |
| **Task 2** | 6180 | 965 | Ai_002 | 10 | 228 | 57 | scene0001 | 2 |
| **Task 3** | 4065 | 1790 | Ai_003 | 9 | 996 | 249 | scene0002 | 2 |
| **Task 4** | 3460 | 1910 | Ai_004 | 10 | 384 | 96 | scene0003 | 3 |

Table 3.3: Task Statistics Overview for ScanNet4-P1+ and Hypersim4-P1. Sample count per task for training (Train) and test (Test) dataset size. Total number of individual camera trajectories per task (Traj.).

**Training Procedure**: SGD is used to train the semantic segmentation network. We perform a large hyperparameter search to design a fair and balanced training

schedule. When switching between tasks, we evaluate different methods to continue training. Only restoring the weights leads to ambiguous results introducing undesirable randomness. The randomness is caused by the initial update steps of SGD, which might significantly disturb the model's parameters given the wrong momentum used. We resolve this problem by fully restoring the optimizer state, including momentum parameters, or alternatively starting with a burn-in optimization phase. At the beginning of the training, a burn-in phase linearly increases the learning rate to the target rate. For more information about the importance of momentum and learning rate schedule, we refer the reader to the two papers proposing the OneCycleLearning [103] and the importance of momentum [104]. We use OneCycleLearningRate with an initial learning rate of 1e-5 ramping up to 0.05 linearly.

We have evaluated using different batch-sizes [4, 8, 16, 32], learning rates [0.08, 0.05, 0.03, 0.01, 0.001, 0.0001], and optimizers [SGD, ADAM]. Using SGD with a batch-size of 16 and a learning rate of 0.05 leads to fast convergence within 5 hours training on all 4 tasks.

**Stop Criterion**: For stopping the training, we evaluate three different procedures. The first is stopping after a fixed amount of epochs per task. Each sample in a task is equally often used to extract information. A downside of this method is that depending on how often samples are drawn from the memory buffer, different amounts of compute/forward passes are performed per task. The second procedure is limiting the number of forward passes or time spend on each task. This criterion is insensitive to the individual task length. Task lengths deviate in up to a factor of 4 between the smallest and largest task (tab. 3.3). The third option explored is early stopping based on the progress achieved on the training or test loss. Criterion-based stopping introduces a new set of hyperparameters and can significantly impact the performance. When performing initial experiments, some tasks stopped after a few minutes while other optimized for 10h+. The 10h+ task led to complete catastrophic forgetting of all previous knowledge. Given the burden and additional hyperparameters of designing a good progress criterion-based stopping, we choose to use a fixed number of 25 epochs. We performed experiments with [5, 10, 25, 50, 100]. The 25 epochs are a good trade-off between time and convergence reached.

### 3.4.1 Experiment - Catastrophic Forgetting

To illustrate the significance of catastrophic forgetting, we trained without any CL strategy to fit the data sequentially. We compare this to a random replay buffer in the default setting: [Task-Dependent, F-50, RS, LR-False, DA-True, GT-Labels] In other words, we utilize a task-dependent buffer capable of storing 50 samples. Samples are replayed and stored into the buffer by random. No latent replay is performed. Replayed samples are augmented and the labels are given by the ground truth annotations. This setting is the default configuration. For further experiments and we only denote changing parameters. In the learning curve plot (fig. 3.2), the background color indicates the currently trained task. We plot the test accuracy for all tasks including the pre-training task (fig. 3.2, labeled Test 0). After pre-training the network classifies the correct label with a 50% chance. As reference, random label assignment, given the 40 target classes, leads to an accuracy of 2.5%. At each transition of the training task (fig. 3.2, change in background color), corresponds to a row in the accuracy matrix in figure 3.4. Without any measurements taken to avoid catastrophic forgetting the performance of each individual task sharply increases during its training. The resulting good performance on the specific trained task results in high values on the diagonal of the accuracy matrix (fig. 3.4, left matrix).

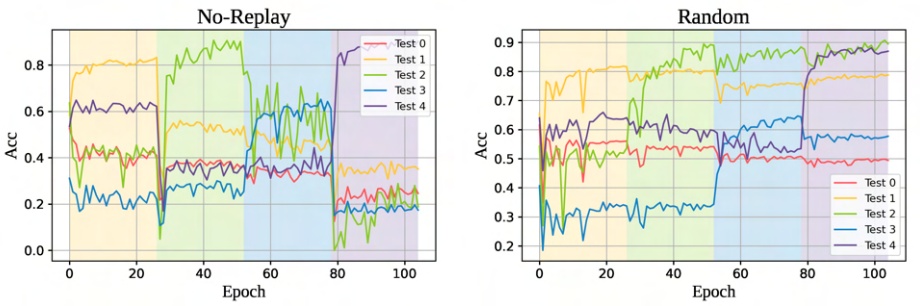



Figure 3.2: Learning Curve: No replay strategy (left), random memory buffer (right). Background color indicates training task.

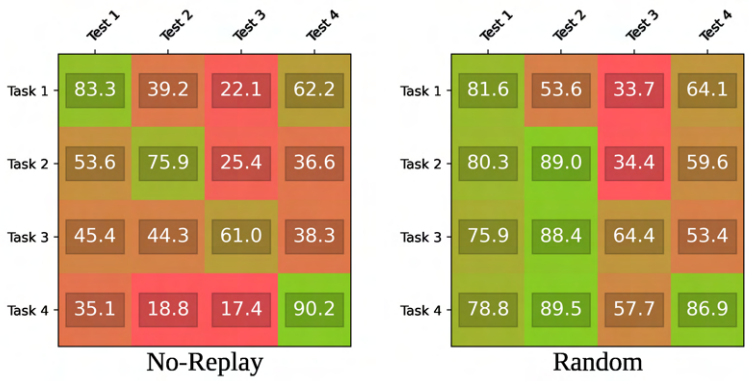



Figure 3.3: Accuracy Matrix: No replay strategy (left), random memory buffer (right).

With no CL strategy the performance on previous task severely decays. This results in the lower left diagonal with column-wise strictly decreasing values (fig. 3.4, left). The standard memory buffer successfully mitigates catastrophic forgetting.
After training on all tasks (fig. 3.4 lowest row), only the performance on the last task is acceptably good without a CL strategy.

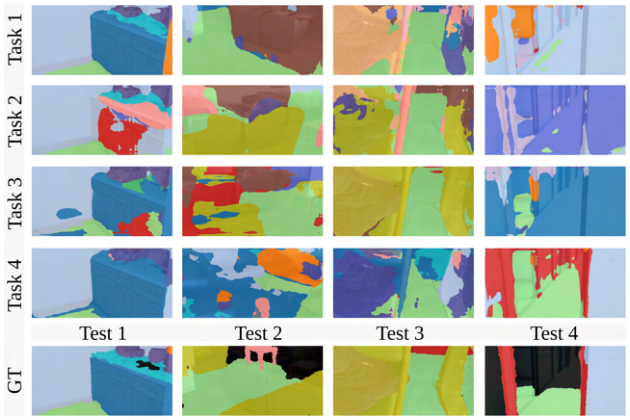



Figure 3.4: Catastrophic forgetting illustrated. Network predicted images same arrangement as accuracy matrix. Bottom row illustrates ground truth label. Good performance on diagonal elements. Lower left diagonal elements illustrate forgetting.

We evaluate the standard metrics in table 3.4

| MODEL | **ACC** | **FWT** | **BWT** |
|---|---|---|---|
| No-Replay | 52.5 | 37.3 | -41.3 |
| Random | 79.2 | 49.8 | -2.8 |

Table 3.4: ScanNet CL metrics: Catastrophic forgetting.

The Average Accuracy of the random replay strategy is significantly better (52.5% → 79.2%). The positive forward transfer indicates that, on average, before training on a task, an accuracy of 37.3% or respectively 49.8% can be achieved just by the previously acquired knowledge. This is due to the carry-over between tasks and the pre-training. To illustrate the forgetting visually, in figure 3.1 for each task a sample in the same arrangement as the accuracy matrix is plotted for the non-replay model. The diagonal elements look significantly better compared to non-diagonal elements.

### 3.4.2 Experiment - Memory Size

At first, we perform experiments on the buffer size. We compare different sizes for fixed-size buffers and buffer sizes related to the training task size. We evaluate fixed buffer sizes of F-5, F-50, and F-500 and percentage-wise buffer sizes of P-5, P-10, and P-20.

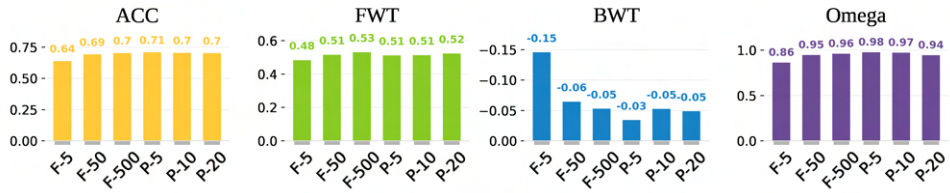



Figure 3.5: ScanNet CL performance for different memory size. Evaluation of average accuracy (ACC), forward transfer (FWT), backward transfer (BTW), and total performance (Omega).

| MODEL | **ACC** | **FWT** | **BWT** | $\Omega_F$ | **Total** |
|---|---|---|---|---|---|
| F-5 | 63.9 | 48.3 | -14.6 | 86.4 | 20 |
| F-50 | 69.2 | 51.5 | -6.5 | 94.6 | 200 |
| F-500 | 70.3 | 53.1 | -5.3 | 96.0 | 1612 |
| P-5 | 70.9 | 51.1 | -3.4 | 97.7 | 151 |
| P-10 | 70.5 | 51.3 | -5.3 | 97.3 | 302 |
| P-20 | 70.3 | 52.3 | -4.9 | 94.4 | 604 |

Table 3.5: ScanNet CL performance for different memory sizes tabular. Total denotes amount of stored samples in memory buffer.

With an increasing number of samples, the accuracy over all tasks improves. This aligns with research results achieved on other CL benchmarks [39]. With 500 samples Task 1 and Task 3 can be fully stored, resulting in an effective memory size of 1612. Counter intuitively with decreasing task specific memory size the *Acc* is

increasing for the ScanNet dataset.

We repeated the experiment on the Hypersim dataset given the disagreement with our intuition. The results are reported in table 3.6 and align with our intuition. The results of the ScanNet dataset might be due to the strongly deviating task sizes.

| MODEL | **ACC** | **FWT** | **BWT** | **Total** |
|---|---|---|---|---|
| F-50 | 79.4 | 50.0 | -7.2 | 200 |
| F-500 | 81.6 | 50.6 | -3.4 | 2000 |
| F-2000 | 81.7 | 50.8 | -3.4 | 8000 |
| P-10 | 81.2 | 49.9 | -4.8 | 2037 |
| P-20 | 81.6 | 50.5 | -3.2 | 4073 |

Table 3.6: Hypersim CL performance for different memory sizes tabular.

Overall the task size dependent buffers seem to be more efficient. We use the task dependent setting and store 10% of the data. This decreases the memory consumption significantly while still observing catastrophic forgetting to test further hyperparameters.

### 3.4.3 Experiment - Memory Sampling

We compare different random sampling strategies. The ratio of sampling from a previous task to sampling new task data is an essential lever in trading off knowledge acquisition and forgetting. In the **Adaptive** setting, the underlying assumption is the equal importance of each task for training. Samples from the current as well as the previous tasks are chosen with the same probability. In the **Fixed Task** setting, the assumption is that a fixed amount of samples are sufficient for each task to not forget it. Therefore all previous tasks are reconciled with a fixed ratio. With an increasing number of tasks, the total rehearsal ratio increases. In the **Fixed Total** setting, the total rehearsal probability is fixed.

The task-wise rehearsal probability is illustrated in figure 3.6, where each row corresponds to the current training task and the $i$-th column to the probability of sampling from task $M_i/T_i$. The upper diagonal is always 0, given that no samples from future tasks are available in the incremental task setting. We train the de-


Figure 3.6: Rehearsal probability illustrated. More details in text.

fault model with the different memory sampling strategies. The results achieved on ScanNet4-P1+ are plotted in figure 3.12. The full learning results are provided in the appendix A.1.1.

We observe that the choice of hyperparameter for the **fixed task** and **fixed total** impacts the performance of the learning procedure. Given the good average

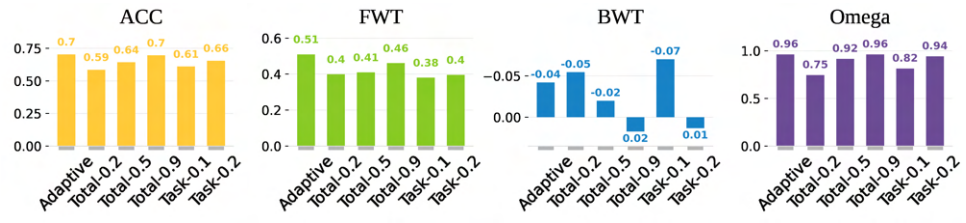



Figure 3.7: ScanNet CL performance for different memory sampling strategy. Evaluation of average accuracy (ACC), forward transfer (FWT), backward transfer (BTW), and final performance (Omega).

accuracy of the **adaptive** strategy without any hyperparameter tuning we use the adaptive setting for all future experiments.

### 3.4.4 Experiment - Augmentation

A memory buffer can be interpreted as a small dataset. The total iterations trained on the stored samples is higher than for non-stored samples. The small buffer size and repeated sampling potentially causes overfitting to the buffer. Overfitting is especially problematic in scenarios where mismatch between the expressiveness of the model (high parameter count) and dataset size is present [105]. The small dataset size of the memory buffer is by design a part of the CL strategy and cannot be avoided.
We, therefore, argue that we face two problems when training with a memory buffer.

- Stored samples are non representative for the previous tasks
- Overfitting to the buffer

To investigate the second, we train the network with and without augmentation on the replayed samples. We randomly apply standard augmentations including color-jitter, random cropping, random rotation, and horizontal flipping to the replayed image (fig. 3.8).

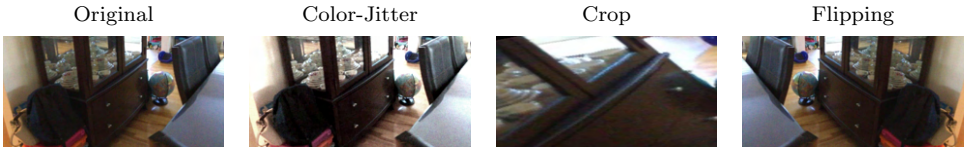



Figure 3.8: Data augmentations illustrated.

In figure 3.9 we monitor the training accuracy achieved on the training buffer compared to the accuracy achieved over all samples during training. The accuracy on replayed samples is for both runs higher than for new ones. The discrepancy without augmentation is significantly larger between all training samples and the replayed ones. The accuracy on the training data is higher without augmentation, but not the validation performance 3.7. This indicates overfitting.

In additional experiments we observed that with reduced memory size the effect of not augmenting the buffer even induces a larger performance decrease. Based on this experiment we always recommend applying data augmentation to the replayed samples.

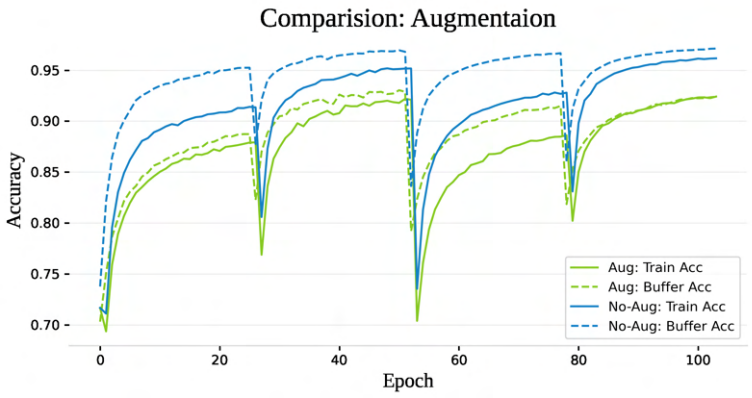



Figure 3.9: Training with and without buffer augmentation.

| MODEL | **ACC** | **FWT** | **BWT** |
|---|---|---|---|
| Aug | 78.7 | 50.1 | -4.1 |
| No-Aug | 77.0 | 48.8 | -7.5 |

Table 3.7: ScanNet CL performance with and without data augmentation tabular.

### 3.4.5 Experiment - Labels

In this experiment we compare if using a teacher network to predict target labels is beneficial. At each task switch we overwrite the teacher network with the continually trained network weights. The teacher network is only used to create labels for replayed samples.

We experimented with using hard-teacher labels and soft-teacher labels. We modify the loss function according to equation 3.5 for soft labels and perform a hyperparameter search for $\lambda \in [0.1, 1, 10, 20]$. We select the run with the highest average accuracy for illustration.

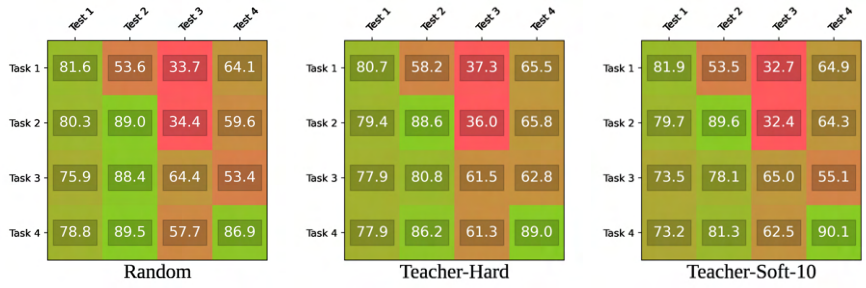



Figure 3.10: Accuracy Matrix: Random memory buffer (left), hard teacher labels (center), soft teacher labels (right).

Overall, using the ground truth labels resulted in the best performance. For scenarios under tight memory constraints using one-hot encoded teacher labels is a valid method. It significantly reduces the memory consumption given that the fixed size network weights are smaller than storing all labels of samples in the buffer. We hypothesize that integration of new knowledge is easier when regularizing using soft labels (fig. 3.10, right: high diagonal elements). Additionally we do not rec-

| MODEL | **ACC** | **FWT** | **BWT** | $\Omega_F$ |
|---|---|---|---|---|
| GT | 79.2 | 49.8 | -2.8 | 96.9 |
| Hard | 78.3 | 54.3 | -2.9 | 97.6 |
| Soft-10 | 77.5 | 50.5 | -6.9 | 95.6 |

Table 3.8: ScanNet CL performance for label setting tabular.

ommend to use soft labels given the additional hyperparameters and lack of clear performance benefit.

### 3.4.6 Experiment - Latent Replay

Instead of storing the samples at in input image level $I(u, v)$ we can stores samples at the feature level $F_{level}(x)$. This allows to decrease memory consumption and compute time needed for replayed samples. The subscript *level* denotes at which point of the architecture we extract the features. We have identified the **Pre-Fusion** and **Fused-Features** to be suitable for reducing the memory and at the same time the network has enough parameter at its disposal in the following layer to integrate new knowledge.

We freeze all weights for layer prior to the extraction level. When not freezing the weights, the stored feature maps of past samples become disentangle to the feature representation the original image produces. In table 3.9 we present the features size, the total parameter count and reduction of storage size compared to the original image.

| | **Input** | **Pre-Fusion** | **Fusion** |
|---|---|---|---|
| **Size** | 3×320×640 | 64×40×80+128×10×20 | 128×40×80 |
| **Reduction** | 1 | 0.375 | 0.66 |
| **Parameter** | 614400 | 230400 | 409600 |

Table 3.9: Fast-SCNN overview feature maps. Size denotes feature map dimension. Reduction is the storage reduction factor in reference to the input image.

Given the large storage reduction we chose to use the Pre-Fusion feature maps. We evaluate storing latent feature maps and the original image. Storing at the input levels allows us to apply input data augmentations. Data augmentation increases the average accuracy by 2%. Given that neither less forgetting on th previous task nor increased performance compared to the full training of the network can be observed, we conclude that freezing the network layers for this network architecture is not beneficial. We can explicitly not make any generalization assumption for generic CL based on these findings. The results may vary strongly depending on the task or used network architecture. For example in larger networks with 100M+ parameters latent replay may be more suitable.

### 3.4.7 Experiment - Buffer Filling Strategy

As previously elaborated in section 3.4.4, we hypothesize that the selection of samples stored in the memory buffer is critical to avoid catastrophic forgetting. Motivated by Support Vector Machines (SVMs), we investigate which samples are the

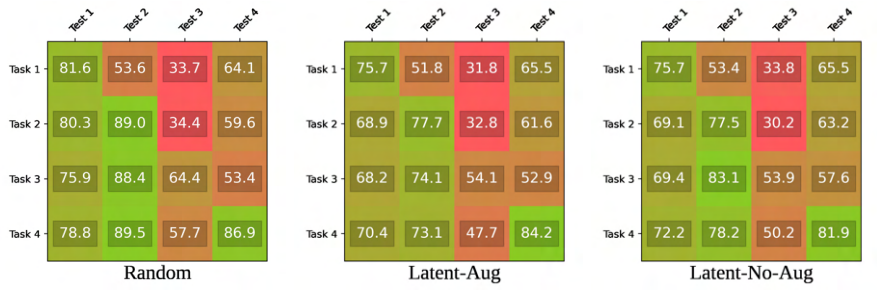



Figure 3.11: Accuracy Matrix: Random memory buffer (left), latent replay with augmentation (center), latent replay without augmentation (right).

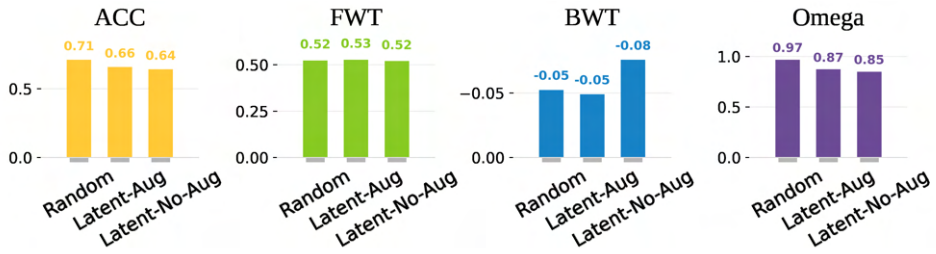



Figure 3.12: ScanNet CL performance for different latent replay settings.

| MODEL | **ACC** | **FWT** | **BWT** | $\Omega_F$ |
|---|---|---|---|---|
| **Random** | 71.0 | 52.3 | -5.2 | 96.9 |
| **Latent Aug** | 65.6 | 52.7 | -4.9 | 87.4 |
| **Latent No Aug** | 64.1 | 52.0 | -7.6 | 85.0 |

Table 3.10: ScanNet CL performance for different latent replay settings tabular.

best to store. In a SVM data points close to the decision boundaries are identified as the support vectors. They are the only data points defining the model. The concept of a decision boundary in neural networks is vague and adversarial attacks have shown that slight perturbations in the input image can lead to misclassification [106].

We populate the memory buffer according to the following metrics: **highest loss**, **lowest loss**, **highest uncertainty**, **lowest uncertainty**. These metrics are evaluated for each sample in the current training dataset $D_t^{train}$ when populating the buffer at the end of the current task $t$. In figure 3.13 we report the CL performance achieved was comparable to our default random sampling approach. We proposed to measure the uncertainty based on the entropy of the soft max output. This metric is commonly used for uncertainty detection of neural networks.

The performance for all metric-based filling strategies significantly decreases, especially when comparing the final performance $\Omega_F$. Additionally, we performed the experiments using the highest soft max output and the distance between the highest and second highest confident prediction as a certainty metric. Both metrics resulted in the same result. We observed a strong visual similarity between samples stored in the memory by inspection. An example of the buffer selection according to the

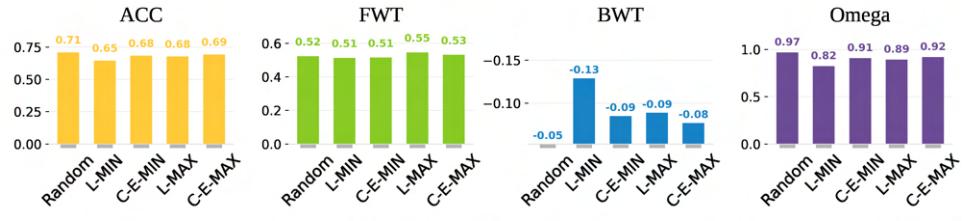



Figure 3.13: ScanNet CL performance for different buffer filling strategies.

highest loss metric is given in figure 3.14.

Figure 3.14: Examples of samples stored in the buffer according to the highest loss criterion.

To reason about the classes represented we count the label statistics of the buffer. For this, we report the histogram of label frequencies over the entire buffer. We compare the random selection, highest loss metric, and the corresponding training data distribution of the first task in figure 3.15.

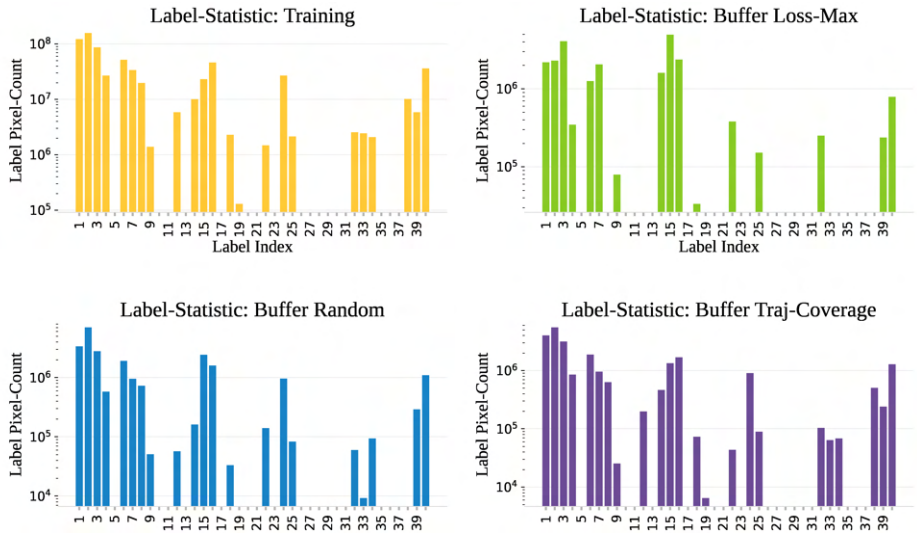



Figure 3.15: Pixel-wise label count in log-scale for training dataset (top left), max loss criterion (top right), random sampling (bottom left) and trajectory coverage (bottom right).

We can observe that the label statistics for choosing a sample based on the loss metric significantly deviates from the training data. The same holds for using the

uncertainty, loss, and accuracy metric sorted according to highest and lowest values. We conclude that sampling according to a metric is not suitable for domain coverage and therefore decreases performance. By random sampling, full domain coverage is likely.

We further investigate if improving over the random sampling is feasible. For this we evaluate storing samples equally over the time horizon of the trajectory (coverage), fitting the training label statistics (label-stat-matching), and feature-map-based clustering (feature). In the coverage setting every n-th frame is stored, where $n$ is given by the total task length divided by the memory buffer size. For label-stat-matching, we fit memory label statistics to the training data statistics. We minimize the MSE between the buffer and training data distribution using rejection sampling. The most computationally and memory expensive method is feature-based selection. We hypothesize that the feature maps contain valuable information. Feature maps can be viewed as learned latent embedding. We might not want to sample according to the input or output statistics but the internal learned representation of the neural network. If we can cover the latent space well these might be the most useful samples for future training.

| MODEL | **ACC** | **FWT** | **BWT** | $\Omega_F$ |
|---|---|---|---|---|
| **Random** | 71.0 | 52.3 | -5.2 | 96.9 |
| **Coverage** | 70.0 | 51.7 | -5.1 | 95.8 |
| **Label-Stat-Matching** | 71.4 | 52.0 | -3.7 | 97.7 |
| **Feature** | 70.6 | 52.1 | -5.1 | 96.3 |

Table 3.11: ScanNet CL performance for different buffer filling strategies.

We concluded that non of the domain coverage strategies result in a significant increase compared to the random buffer selection. Therefore we will utilize the random buffer filling strategy.

### 3.4.8 Experiment - Optimization

So far we treated replayed and new task samples exactly the same in the optimization procedure. Only for soft labels we introduced a individual term in the loss function to deal with replayed samples, otherwise plain SGD optimization is performed. The authors of [107] proposed to update the network parameters based on the gradient induced by the current sample with respect to the network weights $\theta_k$ and all previously seen samples. To integrate a single new example the gradient for each stored sample is calculated. Additionally, to solve the learning without forgetting equation 3.2, a quadratic program is solved to find the optimal parameter update. The others of A-GEM [42] simplified this procedure by calculating the gradient over all previous examples in a batch. This requires to calculate two gradients, one for the new sample and one for the memory buffer, compared to $k+1$ gradients where k is the number of previously stored examples in GEM. Instead of solving a quadratic program the new parameter update is given by projection of the both computed gradients, which can be evaluated significantly faster. Guo et al. [43] introduce a MEGA1, a generalization of A-GEM. We evaluated the CL performance with using A-GEM and MEGA1 on our supervised CL scenario. We did not achieve a performance increase compared to a random memory buffer with the exact same amount of compute provided. Additionally to verify our implementation we preformed experiments on the standard evaluated benchmarks, reproduced exactly

the authors results and showed that the memory buffer outperforms the proposed A-GEM and MEGA1 algorithm on P-MNIST10 [12] and CIFAR100.

These approaches may be more applicable in the field of stream learning or lifelong learning, where it is only possible to train on a samples once. We did not further take gradient based optimization into account. These experiments illustrate a dire need of the CL community to evaluate algorithms with scientific rigor to hard baselines that match the computations of methods fairly.

## 3.5 Discussion and Conclusion

Already a tiny **memory size** results in a significant performance increase. The increase of the average accuracy from 52.5% to 79.2% by storing 10% of the data of the previous data is remarkable.

We can observe that the **memory sampling** strategy is important to trade-off the acquisition of new knowledge and forgetting. With the adaptive setting where we assume that each task should be representative, we achieved a good trade-off on average.

**Data augmentation** is crucial for the performance of the CL strategy. The performance importance of data augmentation increases with decreasing memory buffer size. The Hypersim scenario is more sensitive to data augmentation than ScanNet. This may be because it is easier to overfit synthetic data compared to real-world data. We conclude that heavy data augmentation avoids overfitting the memory buffer. We can interpret the data augmentation as forcing the network to learn to be invariant to the applied transformations or synthetically increasing the dataset.

A **latent replay** strategy can be beneficial if extremely fast inference and a low memory footprint are critical. On larger networks, it is common to fix the feature extraction backbone, and only task-specific fine-tuning of the lower layers is performed. For these applications, latent replay is extremely promising. In recent literature [108] on latent replay, we found that the comparison to the input level replay methods is often performed without data augmentation on the input samples. This leads to misleading performance measurements that favor the latent replay methods.

A **teacher network** to predict labels did not lead to better CL performance. Under tight memory constraints, using a teacher network avoids storing the labels for each sample in the memory buffer at the cost of inference time. This may be efficiently combined with generative replay approaches.

When **populating the memory buffer**, the selected samples have a crucial impact on the training success. Populating the buffer by random sampling outperformed other methods. We credit this to the good domain coverage achieved by random sampling. These results align with [38], where selective sampling for coverage maximization was proposed.

The only major experience replay part under active research that we did not investigate is the sample selection during training [109]. In general, our experiments nicely align with previous research performed on memory buffers [108]. We can conclude that applying experience replay CL-strategy on multi-class semantic segmentation can be achieved efficiently. We propose whenever it is possible to store

samples from the current task in a CL scenario to employ an experience replay strategy. Orthogonal to experience replay other methods may further increase the performance.

For our following experiments, we will use the following settings:

- Task-dependent memory size of 10
- Random sampling with adaptive probability.
- Random buffer filling.
- No latent replay
- No teacher network.
- Full data augmentation.

# Chapter 4

# Supervision

## 4.1 Problem

Efficient adaptation can be accomplished if a good learning signal is present. The generation of a learning signal for semantic segmentation during a robotic mission without human annotation is a challenging problem. The underlying physical constraints limit the logical plausible semantics. Same objects or scene regions observed from different viewpoints and at different timestamps must be semantically consistent. We denote this as spatial (multi-view) and temporal consistency of a scene, respectively. Additionally to texture, the shape of objects and the scene carry helpful information for clustering. No hard real-time requirements are present for estimating the supervision signal, allowing utilization of more computation, past observations, and advanced information fusion algorithms.

For each scene we obtain an input image sequence $\mathbf{X} = \{x_i\}_{1..N}$, estimated semantic segmentations $\hat{\mathbf{Y}} = \{\hat{y}_i\}_{1..N}$, depth measurements $\mathbf{D} = \{d_i\}_{1..N}$ and the full extrinsics $\mathbf{H} = \{H_i\}_{1..N}$ and intrinsics $K$ of the camera. The objective is to generate the best possible $\hat{\mathbf{Y}}^{\circ} = \{\hat{y^{\circ}}_i\}_{1..N}$ that improves over $\hat{\mathbf{Y}}$ with respect to the ground truth labels $\mathbf{Y} = \{y_i\}_{1..N}$. Additionally, we constraint the computational power to maximum top-tier ARM chips or modern x86-64 desktop CPUs. The time to generate the supervision signal should be on the same order of magnitude as recording the trajectory.

## 4.2 Clustering and Regularization

### 4.2.1 Method

Clustering and regularization methods make use of two properties. Firstly, neighboring pixels are likely to have the same semantic label. Secondly, similarity in color-space indicates that pixels belong to the same class. As introduced in the background section 2.3, we use dense CRFs and SLIC to refine each estimate $\hat{y}_i$ individually. Both methods are easy to implement with open-source tools available. With SLIC we obtain superpixels regions that have no semantic label. For assigning a label to each region, we assign the label that is most often predicted by the network within the region in the **hard** voting strategy. Alternatively, we can accumulate the predicted probability distribution over all labels and set the label for the region with the highest total probability in the **soft** voting strategy. In the simplest form of SLIC we can tune the number of superpixels and compactness factor. We empirically found that 100-200 segments are sufficient to obtain enough level of

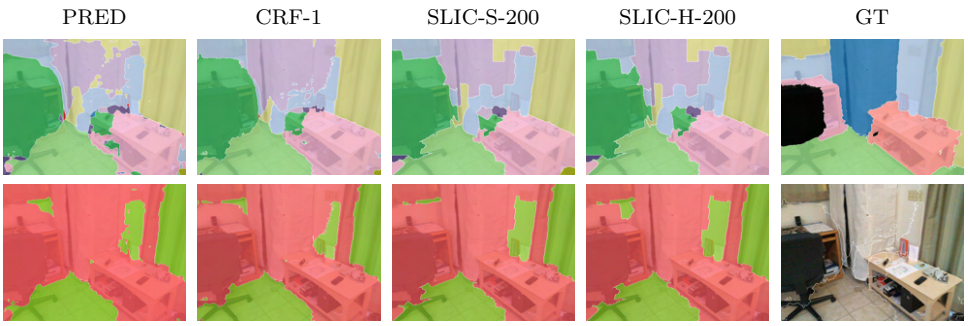



Figure 4.1: ScanNet example images for CRF and SLIC post-processing. Top row indicates label. Bottom row illustrates correctly (green) and false (red) classified pixels.

detail in the scene. Similarly, we tuned the compactness factor such that individual segments nicely align with boundary regions of the image.

For the CRF setup, we follow the method proposed in [25]. A pairwise Gaussian potential with a standard deviation of 1 pixel is used to model the neighboring relationship. For regularization based on color, a pairwise bilateral Gaussian potential function with a standard deviation of 3 and 67 pixels is used for the position and color encoding, respectively. We additionally performed experiments with different hyperparameter settings but could not improve on the default settings used by the authors of [25] for the MS-COCO dataset.

### 4.2.2 Experiments and Results

To compare both methods we apply them to Scene0000 in the ScanNet dataset. The pre-trained Fast-SCNN network predicts class-wise probabilities $\hat{\mathbf{Y}}$ for the full trajectory.

In figure 4.1 we visually compare the predicted result for the 1st frame. We color code correct (green) and wrong (red) classified pixels in the bottom row. The refined regions follow contours in the image better for both CRF and SLIC refinement. Both methods remove high-frequency label noise. Nearly no difference is visible between the hard and soft voting strategy used to estimate the superpixel label. When compared to the ground truth image, we can see that regions are correctly clustered but assigned to the wrong semantic label. The network did not successfully distinguish the cabinet and desk. There might be multiple reasons why the labels are assigned incorrectly including viewpoint, context, missing training data or inconsistent annotations in the training dataset.

| **Method** | **PRED** | **CRF-1** | **CRF-2** | **SLIC-H-100** | **SLIC-H-200** | **SLIC-S-200** |
|---|---|---|---|---|---|---|
| **Acc** | 57.1 | **57.8** | 57.6 | 57.0 | 57.0 | 57.1 |
| **Time** | - | 1021ms | 1021ms | 48.ms | 65ms | 65ms |
| **std** | - | ± 48.ms | ± 48.ms | ± 7.16ms | ± 7.16ms | ± 7.16ms |

Table 4.1: ScanNet evaluation accuracy and inference time post-processing.

In table 4.1 we report the accuracy and inference time for each method. Dense CRFs lead to the best overall performance, but only a slight increase in accuracy

is achieved compared to the raw network prediction. We measure inference time over 10 runs on an AMD Ryzen 5900X. For the superpixels, we use the original resolution of $944 \times 1272 \times 3$. Approximating the posterior of the dense CRF problem is computational more demanding and scales linearly with the number of pixels in the image. We downsample the input image to a third of the original resolution resulting in a 10x speedup with diminishing accuracy loss. We ran the experiments on a more extensive set of scenes leading to similar results. We conclude that post-processing using CRF only results in a slight accuracy increase. It is not a suitable method to generate a supervision signal, which should be significantly better than the network. Our experiments nicely align with the current trend of the latest machine learning modules [59] getting rid of CRF-post-processing in favor of inference time.

## 4.3 Temporal Consistency

### 4.3.1 Method

To leverage the temporal consistency and the provided sequence data we use optical flow to propagate predicted labels. To recap the optical flow $F(u_1, v_1)$ between two images is defined as follows:

$$\begin{pmatrix} u_2 \\ v_2 \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} + F_1^2(u_1, v_1) \tag{4.1}$$

Optical flow defines the mapping of a pixel in $(u_1, v_1)$ in $I_1$ onto the image plane of $I_2$ with coordinates $(u_2, v_2)$. To obtain the optical flow different methods including Gunnar Farneback's algorithm, depth based approaches, and learned flow estimation networks are used. We choose to use the last method, given the fast inference speed and no need for depth information. As elaborated in the background section 2, we utilize the Recurrent All- Pairs Field Transforms (RAFT) architecture [76]. The deep neural network architecture predicts an initial flow estimate and refines it using a recurrent unit. We use the pre-trained network on the Sintel [110] dataset to infer optical flow between consecutive images for each ScanNet scene with a subsampling factor of 10. We denote the forward propagation of a label as function $g_i^{i+1}(y_i^i; F_i^{i+1})$ taking the predicted optical flow and label for the image as an input. Here the label $y$ has a subscript denoting at which time step it was recorded and a superscript denoting the reference image-plane. We use function composition to notate propagation of a label over multiple frames as follow:

$$\begin{aligned} y_i^{i+2} = g_i^{i+2}(y_i^i) &= g_i^{i+1}(\, g_{i+1}^{i+2}(y_i^i; F_{i+1}^{i+2})\, ;\, F_i^{i+1}) \\ &= (\, g_i^{i+1} \circ g_{i+1}^{i+2}\,)(\, y_i^i\,) \end{aligned} \tag{4.2}$$

### 4.3.2 Experiments and Results

In the first experiment, we use the ground truth labels at each time step and assign a random label to 50% of the pixels. We forward propagate the last three frames to the current frame (fig. 4.2 ). In figure 4.2 we indicate correct and false classified pixels in the bottom row. The top row illustrates the forward propagated noisy labels. By inspecting the top row we can see that the estimated flow is sufficiently precise to propagate labels. The labels overlay nicely with the target frame object boundaries. Here it is important to mention that the saturation of the predicted labels and correctly classified labels in the first 4 frames is low. This is due to the reason that 50% of the pixels are randomly assigned. The last label indicates the aggregated label for the current frame. The label for the k-th frame is given by:

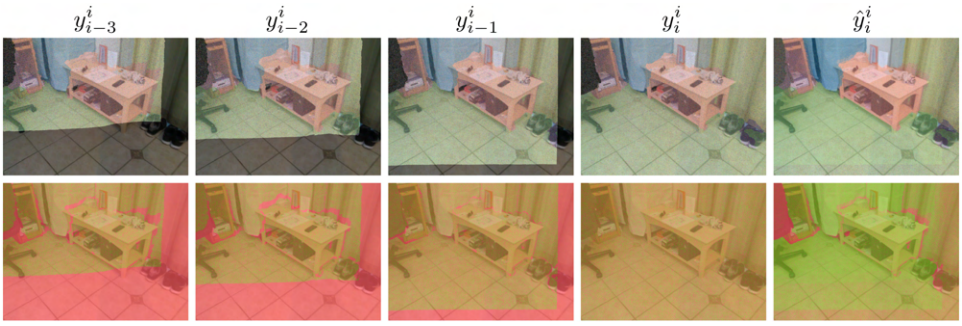



Figure 4.2: ScanNet example images optical flow noisy label propagation. Top row illustrates forward propagated label. Bottom row indicates false and correctly classified pixels.

$$\hat{y}^i_i(u,v) = argmax \sum_{i=1}^{Horizion} y^k_{k-i}(u,v) \tag{4.3}$$

Here $y$ defines the output probability distribution, and the argmax operation retrieves the index with the most probability mass summed over all forward propagated predictions. This is the same soft-voting strategy on a per-pixel level used for the superpixel label assignment. The last column (fig. 4.2, far-right $y^i_i$) illustrates the aggregated label. The top-left area of the image aggregates information from multiple frames and results in more correctly classified pixels.

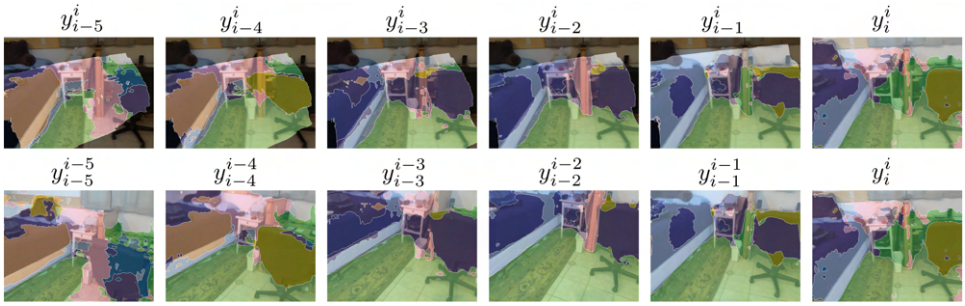



Figure 4.3: Optical flow quality label forward propagation. Bottom row original predicted label in reference frame. Top row forward propagated label to target frame.

In the second experiment, we used the predicted labels by the network to propagate over multiple timeframes. In figure 4.3 we set the forward propagated label in comparison to the prediction in the original image plane. The chair and bed in the image are classified correctly or incorrectly depending on the frame. Mistaking objects based on missing global context is a repeated failure mode in multiple scenes. Predicting semantic labels from a single image is significantly more demanding than from multiple viewpoints given that the correct label may not be deducible without context. This is referred to as aleatoric uncertainty, which is an inherent property of the task itself and cannot be resolved. To find the optimal horizon length we perform experiments with varying horizon lengths, evaluating all frames of the first scene in the ScanNet dataset.
The results in terms of accuracy are presented in table 4.2. A horizon length of 1

| **Horizon** | 1 | 2 | 4 | 6 | 12 |
|---|---|---|---|---|---|
| **Accuracy** | 57.1 | **58.7** | 58.5 | 58.3 | 57.7 |

Table 4.2: Comparison accuracy for different horizon lengths for label propagation.

is equivalent to the default network prediction.

Increasing the horizon length decreases the performance. We suspect that this is due to noise in the label propagation. Two view label fusion results in an accuracy increase of 1.6%. The additional memory required for storing the forward propagated label is neglectable. RAFT operates at 10 fps on a $944 \times 1272 \times 3$ resolution. Significantly faster flow estimation methods with a slight decrease in accuracy exist that allow real-time operation. The only downside of RAFT for our application is the 5GB GPU memory requirement.

Given the simplicity and computational efficiency of fusing two frames, this method is a good candidate for supervision signal generation. The apparent drawback of the two-view approach is that we do not fully exploit the spatial and temporal consistency globally but just in a local two-frame window. One option that we did not further investigate is to predict the optical flow for each frame to all other previous frames individually, instead of compositing the flow predictions. This can decrease the effect of label noise propagating but scales computational-wise linearly with the window size. Additionally estimating optical flow under larger viewpoint changes in general is a more difficult problem, therefore we expect more noise in the individual flow predictions.

## 4.4 3D Mapping

### 4.4.1 Method

The promising results achieved in the prior section by fusing two frames motivates fusion over a longer horizon with a more robust method. A longer time horizon allows observation of the scene from various perspectives. The network can predict the semantic with different contexts apparent at each frame. Under the assumption that, on average, objects are more often classified correctly than incorrectly, this can improve accuracy. The goal is to build a 3D reconstruction of the scene and associate the predicted semantics with the map.

In the background section, we concluded that Kimera Semantics is a suitable tool to accomplish this. An overview of the final working 3D mapping pipeline is given in figure 4.4. At each frame we create a point cloud $P_p(x_p, y_p, z_p)$ from the depth map $D(u, v)$ and the known camera intrinsics $K$. Each point in the point cloud can be associated with a semantic class $c_s$ based on the network prediction for the corresponding pixel. When creating the semantic point cloud we use a sub-sample hyperparameter $r_{sub}$ that limits the number of back-projected points to every $r_{sub}$-th pixel along the $u$ and $v$ axis independently. We obtain a sparse semantic point cloud with $N/r_{sub}^2$ points. We can mask the semantic point cloud according to confidence threshold $th_c^{net}$ of the network predictions. This allows us to solely integrate certain labels into the map. By default we integrate all points ($th_c^{net} = 0$) into the map.

The final masked semantic point cloud $P_s = (x_s, y_s, z_s, c_s)$ is integrated into the TSDF volume using the camera extrinsics by Kimera Semantics. BundleFusion

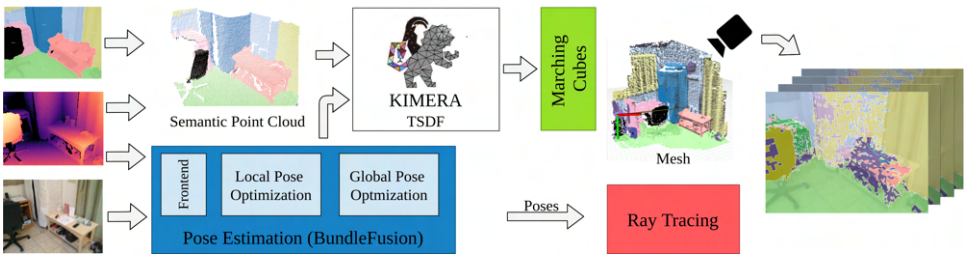



Figure 4.4: Overview 3D mapping supervision signal generation. Left input RGB-D image, predicted semantic segmentation. Pose estimation using BundleFusion. Kimera-Semantics integrates semantic point cloud into TSDF volume. Creating of mesh using marching cubes algorithm. Ray tracing of mesh using predicted camera poses to estimate pseudo labels for scene.

estimates the camera extrinsics for all ScanNet experiments offline. Important parameters are the resolution of the TSDF volume, which is a trade-off between memory/compute and precision. After the entire trajectory, or at regular intervals, the marching cubes algorithm generates a high-resolution mesh from the TSDF volume. Finally, we implement a fast CPU-based ray tracing to create a pseudo label for each frame. At each frame, the ray tracing algorithm determines the first intersection of the ray with the mesh.

The resulting 3D intersection point for each pixel can be used as the voxel volume index to obtain the stored label probability. This procedure is performed in parallel for all pixels in a frame. We denote the resulting label as $y_i^{pseudo}$ with $H \times W \times C$ where C denotes the number of classes. Instead of saving the entire probability distribution, we store the top 3 probabilities and their corresponding indices, allowing for fast PNG encoding and compression. Additionally, storing the probability distribution has a clear advantage over storing a one-hot encoded label. E.g. in future experiments, we will guide the learning process based on the certainty of each label.

### 4.4.2 Experiments and Results

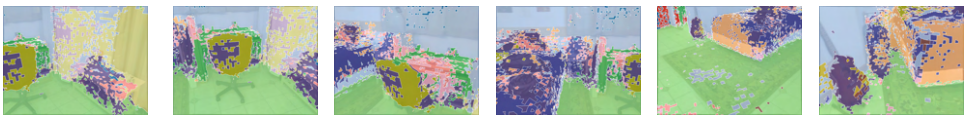


Figure 4.5: Ray traced pseudo labels examples pre-trained network map.

**Functionality:** To at first investigate the functionality of the pipeline, we perform the full mapping and label re-projection step with the ground truth labels. We expect to get a high accuracy with only small artifacts. These artifacts are induced by the discrete voxel volume integration and consecutive ray tracing process decreasing the performance. It is important to mention that the labels for ScanNet were generated in a similar manner by annotation in 3D and re-projection to individual frames. We set the voxel grid size $d_{voxel} = 5cm$ and use $r_{sub} = 5$. Over the full sequence we obtain an accuracy of 94%.

In figure 4.6 we can see the birds-eye view of the ground truth annotation and predicted annotation. The floor is estimated well over the full scene. Objects like the sofa or bed are not estimated correctly. In figure 4.5 we provide six example-frames

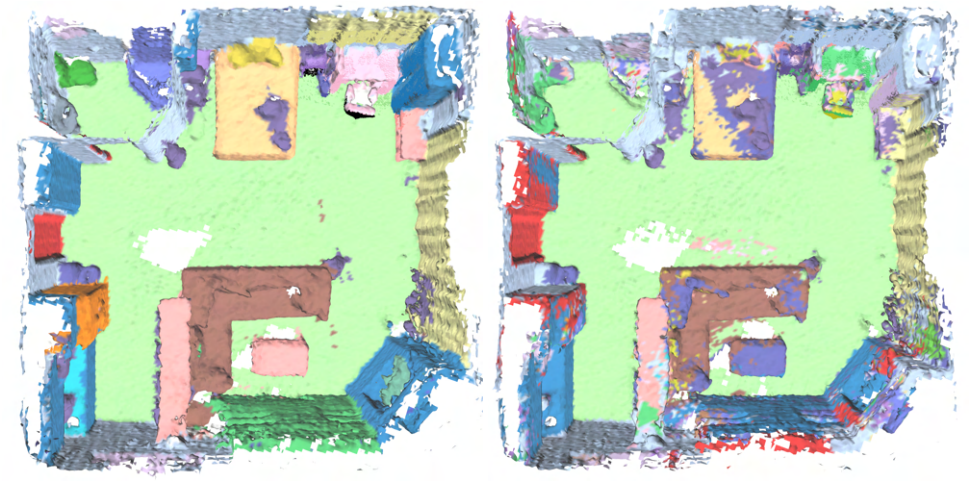


Figure 4.6: Generated 3D map bird's-eye view. Ground truth labels (left). Pre-trained network (right).

illustrating the quality of the estimated labels.

**Parameters and Runtime:** After we observed that our pipeline functions, we investigate the influence of the voxel grid size $d_{voxel}$ and the sub sampling ratio $r_{sub}$. In figure 4.7 we illustrate the effect of different voxel sizes to the scene reconstruction quality. With a smaller grid size more details can be captured adequately.

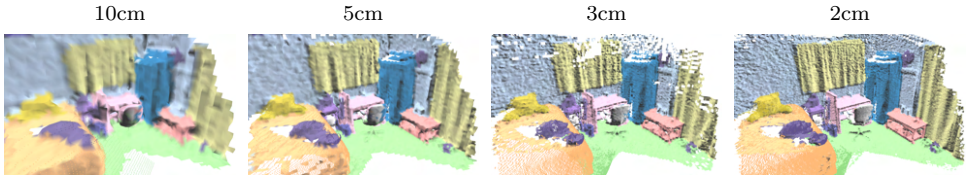
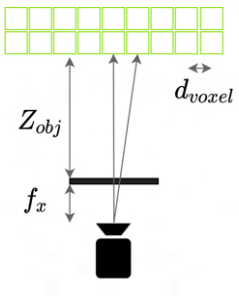



Figure 4.7: Comparison different map resolutions.

To allow efficient integration of observed frames we have to find correct balance between $r_{sub}$ and the $d_{voxel}$.




$$\frac{d_{voxel}}{Z_{obj}} = \frac{r_{sub}}{f_x}$$
$$r_{sub} = \frac{d_{voxel} f_x}{Z_{obj}} = \frac{0.05m}{2m} * 577px \tag{4.4}$$

We want to relate $r_{sub}$ to $d_{voxel}$. For this we illustrate a simplified 2d scenario. We enforce, that the $r_{sub}$-th neighboring pixels project to 3D points with a maximal grid distance $d_{voxel}$ apart at a fixed distance $Z_{obj}$. We set $Z_{obj}$ to 5 meters which is pessimistic estimate for room size scenes given that most measured depths are below 5 meters. The used camera has a focal length of 577 pixels. When we assume a voxel grid size of $5cm$ we can use every 5th frame following equation 4.4.

Experimentally this holds and results in a dense estimate of the scene.

| $\mathbf{d}_{voxel}$ | $\mathbf{r}_{sub}$ | Time | Mesh | Semantics | Acc |
|---|---|---|---|---|---|
| 10cm | 11 | 30ms | 3.0MB | 29.9MB | × |
| 10cm | 5 | 52ms | 3.0MB | 29.9MB | 62.5 |
| 10cm | 1 | 430ms | 13.0MB | 179.1MB | × |
| 5cm | 5 | 130ms | 13.0MB | 179.1MB | × |
| 5cm | 2 | 240ms | 36.0MB | 336.4MB | 61.6 |
| 3cm | 4 | 330ms | 36.0MB | 336.4MB | 63.5 |
| 3cm | 3 | 390ms | 36.0MB | 336.4MB | × |
| 2cm | 2 | 980ms | × | × | × |

Table 4.3: Inference speed and memory demand for different $\mathbf{d}_{voxel}$ and $\mathbf{r}_{sub}$ settings.

In table 4.3 we evaluate different settings of $d_{voxel}$ and $r_{sub}$ the time required to integrate a single frame, the total memory for the generated mesh, and the full voxel volume with the stored semantic probabilities. We also generate pseudo labels by ray tracing and evaluate the accuracy compared to the ground truth labels, for candidate settings of $d_{voxel}$ and $r_{sub}$ meeting the inference time of $\geq$ 3fps. The accuracy for the 10cm voxel resolution increased over the 5cm resolution. We explain this finding by the fact that the 10cm resolution filters out small inconsistent regions in the map. This effect improves more than the loss of scene details associated with the lower resolution. When further increasing the resolution the accuracy significantly degrades as expected.

With increasing $d_{voxel}$ the inference time and storage needed increases cubicly. We therefore limit the resolution to $\geq 3cm$. This allows operating the CL pipeline close to real-time. The small computational difference between using $r_{sub} = 2$ and $r_{sub} = 5$ for $d_{voxel} = 5cm$ can be explained by the fast-integration method used to build the TSDF. For more information we refer the reader to the implementation of VoxBlox [89].

**Uncertainty Aware** As discussed previously we can make use of the confidence estimate of the network given by the soft-max output. In figure 4.8 we illustrate the histogram of predicted confidence binned in 2% intervals. The histogram is evaluated for the full first scene of ScanNet for each pixel. The class probability with the highest value is the confidence score for each pixel.

The re-projected labels are similarly distributed with more pixels being fully confident compared to the network prediction. Only when observations with different labels are integrated for the same voxel the probability decreases. In other words, the re-projected certainty is a proxy for estimating multi-view coherence. This is due to the fact that before integrating labels into the map they are one-hot encoded.

An outlier cohort can be seen in the histogram of the re-projected labels in the first bin. Some rays do not intersect the mesh and therefore the correspondings pixels probability distribution is initialized uniformly. This results in a confidence of 2.5% for the 40 NYU labels.

To evaluate if the certainty measurement is a good proxy for the prediction performance, we calculate the *Acc* in relation to the confidence threshold $th_c$. Firstly, we evaluate the network output labels for different settings of $th_c^{net}$ in table 4.4

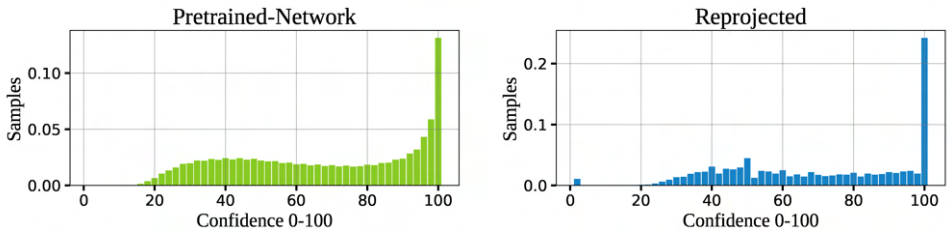



Figure 4.8: Comparison confidence histogram of pre-trained network output and re-projected labels.

Additionally, we evaluate the re-projected label confidence setting $th_c^{repro}$, when the map is generated with all network predictions available $th_c^{net} = 0$. For both confidence scores we evaluate [0, 0.5, 0.7, 0.9]. In tables 4.4 and 4.5 we report the ratio of valid pixels above the set certainty threshold. The *Acc* score is measured in a dense and sparse setting. In the sparse setting a pixel is evaluated if the predicted and ground truth labels are provided. Int the dense setting all pixels are evaluated. If no label is provided for a pixel, it is evaluated to wrongly classified.

| **Certainty Th.** | **Ratio Valid** | **Acc. Dense** | **Acc. Sparse** |
|---|---|---|---|
| 0 | 0.969 | 57.0 | 57.0 |
| 0.5 | 0.666 ¨ | 49.1 | 71.4 |
| 0.7 | 0.48 | 40.8 | 82.3 |
| 0.9 | 0.292 | 27.9 | 92.6 |

Table 4.4: Input level accuracy confidence correlation pre-trained network.

| **Certainty Th.** | **Ratio Valid** | **Acc. Dense** | **Acc. Sparse** |
|---|---|---|---|
| 0 | 96.9 | 61.6 | 61.6 |
| 0.5 | 68.8 | 53.2 | 74.9 |
| 0.7 | 50.5 | 44.3 | 85.0 |
| 0.9 | 32.7 | 30.4 | 90.2 |

Table 4.5: Output level accuracy confidence correlation re-projected labels pre-trained network map.

We can clearly see that with increasing certainty for both thresholds the accuracy increases (tab. 4.5, 4.4). Figure 4.9 illustrates the created map if $th_c^{net}$ is set to 0.9, or respectively 29.2% of all pixels are integrated into the 3D scene. The white areas have no label assigned.

Using Kimera-Semantics and performing re-projection of the generated map into the 2D image coordinates results in good performance compared to the other previously investigated methods. The capability to perform CPU based ray tracing allows to execute the supervision signal generation without hardware accelerators on mobile robots. The full 3D mapping runs on a single core at an adequate resolution. With enough RAM available even full buildings can be mapped. The multi-core CPU ray tracing allows to use the full computational power available on the robot. Also the capability to have a proxy measurement $th_c^{repro}$ for multi-view consistency is a clear advantage over previous methods.

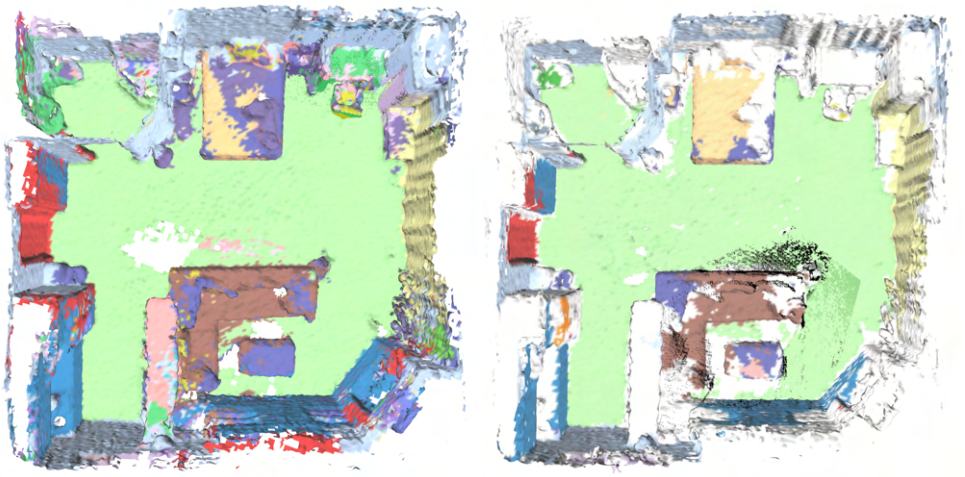


Figure 4.9: Generated 3D map bird's-eye view. Pre-trained network $th_c^{net}$=0(left). Pre-trained network $th_c^{net}$=0.9 (right).

## 4.5 External Knowledge

3D mapping is limited to generate a supervision signal based on the multi-view consistency assumption. Solely relying on our network predict we are not able to acquire new semantic information. This is desirable to integrate new classes and helps to resolve complete miss-classifications of objects over multiple frames. E.g., a bed may be consistently classified over multiple frames as a sofa. With solely 3D mapping, we are not capable of detecting this misclassification. This problem can only be resolved if we are provided with the missing semantic information.

A human or another algorithm may provide this information. Given that we want full autonomous operation, we explore the possibilities to use an auxiliary network for this task. We explore to use large dense semantic segmentation network (DeepLabV3, ResNet101, 58.6M), bounding box detector (YoloV5X, 87.7M) and instance segmentation network (Mask R-CNN, X101-FPN, 107M). All of these models achieve State-of-the-Art performance with the selected large backbone networks but are not real-time capable. The networks are trained on the COCO dataset which covers a large domain of objects. DeepLabV3 is trained on the COCO-Stuff datasets and the others are trained on COCO object detection. We can use the external knowledge to adapt our fast real time capable network. This procedure is refereed to as knowledge distillation.

Given that for the task of semantic segmentation we do not have a notion of an object instance, it is hard to usefully integrate a bounding box estimate provided by the bounding box detection network. Therefore, we decided to focus on the DeepLabV3 and Mask R-CNN. For both networks we achieved good performance, but decided to not further investigate approaches that rely on external knowledge. This allows to apply our method to wider range of mobile robots, which are not equipped with high-end GPUs with a large memory.

## 4.6 Conclusion

Regularization and clustering approaches perform significantly worse compared to methods, which leverage multi-view consistency. We have demonstrated that two-

view information aggregation using optical flow is a valid method to create pseudo labels outperforming single view predictions. This method is capable of running in real-time on robotic systems. When propagating labels over multiple frames, the performance decreases based on cumulative noise induced by the label propagation.

To reduce cumulative noise over a longer horizon, we investigated more sophisticated 3D semantic scene reconstruction approaches. We showed that with good bundle adjusted poses, multi-view information fusion significantly outperforms the previous methods. An improvement from 57.1% to 63.5% is achieved. The voxel resolution of 3cm to 5cm is precise enough to capture semantic details in the scene. Additionally, the semantic scene reconstructed in 3D is helpful for further downstream tasks such as collision-free planning, detection of dynamic objects, and many more. We also showcased that the confidence output of the network and re-projected labels directly correlates with the expected label accuracy. The created map is smaller in terms of memory compared to storing each label in 2D per frame.

# Chapter 5

# Self Supervised Continual Learning

To re-consolidate, in section 2.1 we investigated the different impact factors of the memory buffer based CL strategy and in section 4 we evaluated different methods for supervision signal generation. In this section, we combine those two approaches to tackle the semi-supervised CL-task.

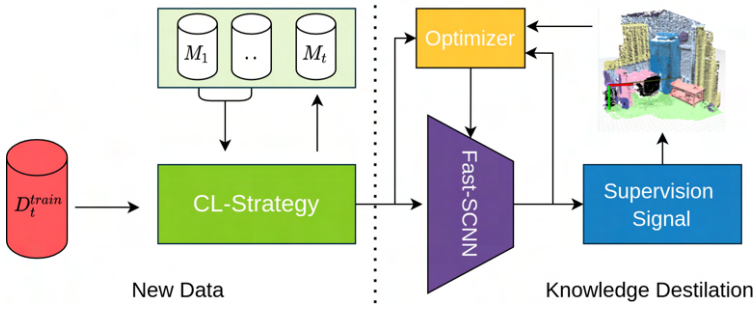



Figure 5.1: Self supervised CL information flow. More details in text.

Figure 5.1 illustrates a high-level overview and information passing between the components involved in our semi-supervised learning strategy. The CL strategy, in our case a memory replay buffer, provides samples from the new training dataset $D_t^{train}$ and previous tasks stored in $M_1 ... M_{t-1}$. While training on the current task, the CL learning strategy populates the current task buffer $M_t$. This procedure was elaborated in section 2.1. The supervision signal is generated based on the network predictions, which are accumulated in a semantic map. From the semantic map, pseudo labels are ray traced, functioning as a supervision signal.

If the network performance increases, the supervision signal generation can improve as well. Firstly, we perform an ablation study to evaluate the best pseudo label training hyperparameters 5.1.1. Secondly, in section 5.1.2, we investigate the possible performance increase by iteratively refining the supervision signal and training the network in-the-loop for multiple iterations. Lastly, for the ScanNet dataset we perform the full learning on the semi-supervised benchmark (sec. 5.1.3).

To show our pipeline in a real-world use case, we record data with a handheld Azure Kinect RGB-D sensor in various buildings of ETH Zurich. We will provide an overview of the full working pipeline and show qualitative results (sec. 5.2).

## 5.1 Semi-Supervised: ScanNet

### 5.1.1 Training on Pseudo Labels

We have concluded in section 4.6 that semantic scene reconstruction leads to the best pseudo label performance. The most confident network predictions have the highest accuracy. To investigate if we can directly make use of this correlation (tab. 4.4), we investigate training on sparse labels. In the default dense supervision setting, for every pixel in the image, a corresponding label is assigned. In contrast, in the sparse setting, only a few pixels have a corresponding label. Therefore uncertain pixels do not induce a loss. We refer to this training on sparse labels as distillation.

For distillation, we do not need to create a semantic reconstruction of the scene. The hyperparameter $th_c^{net}$ denoting the confidence threshold (sec. 4.4) selects the sparsity of the supervision signal generated. We will compare this to the full pseudo label generation via mapping and ray tracing. This setting is parameterized by $th_c^{net}$ and $th_c^{repro}$. The first parameter is the same as for the distillation setting but denotes the certainty threshold for integrating labels into the map. The second parameter refers to the minimal confidence of the re-projected label necessary to be marked as valid for training. Both parameters were introduced in section 4.4. E.g., the setting of $th_c^{net} = 0$ and $th_c^{repro} = 0$ describe fusing all predictions of the network into the 3D map and using all re-projected pixels as a training signal for retraining the network.

**Distillation:** In table 5.1 we report the distillation performance of the network on the validation dataset of the first task. Here it is important to denote that we previously provided the numbers for the full first trajectory of the first task; now, we solely report the numbers for the test set of the first task. Here it is essential that all experiments are evaluated using the same procedure.

| **Config** | Pre-Trained | $\mathbf{th}_c^{net}$: 50 | $\mathbf{th}_c^{net}$: 70 | $\mathbf{th}_c^{net}$: 90 |
|---|---|---|---|---|
| **Acc** | 65.3 | 62.5 | 63.3 | 64.6 |

Table 5.1: Distillation training for different $\mathbf{th}_c^{net}$ parameters.

The performance significantly degrades when retraining the network in the distillation setting. It shows that the sparse label accuracy does not directly correlate to the quality of the supervision signal. Before retraining, the network already predicts most pixels correctly for which a learning signal is provided. Therefore by retraining on these pixels, we motivate the network to be overconfident on them. Additionally, it is commonly known that even though neural networks are highly nonlinear, the predicted output lies on a smooth manifold [111]. If no label is assigned for a pixel, the closest valid label influences the prediction for the unlabeled pixel. These are two possible explanations in the curriculum learning community [101], why training on a high accuracy sparse signal reduces the overall performance.

**3D Mapping:** We compare this to the 3D mapping pseudo labels. We have chosen to use the labels generated with $d_{voxel} = 5cm$. We experiment with $th_c^{net} = [\,0, 0.5, 0.9\,]$ and $th_c^{repro} = [\,0.03, 0.5, 0.7, 0.9\,]$. We use a minimal threshold of 3% to avoid using labels initialized uniformly by random from $th_c^{repro}$. This is the case for pixels, that do not have a corresponding point on the generated 3D mesh (fig. 4.8).

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{th}_c^{net}$ | 0 | 0 | 0 | 0 | 50 | 50 | 50 | 50 | 90 | 90 | 90 | 90 |
| $\mathbf{th}_c^{repro}$ | 3 | 50 | 70 | 90 | 3 | 50 | 70 | 90 | 3 | 50 | 70 | 90 |
| **ACC** | 66.1 | **67** | 66.3 | 66.7 | 65.2 | 65.3 | 65.4 | 66.2 | 65.4 | 65.2 | 65.3 | 65.2 |

Table 5.2: Pseudo label training hyperparameter comparison.

Table 5.2 presents the accuracy for different hyperparameter settings. The best performance is achieved by integrating all observations into the map. Additionally, setting the confidence value to 50% further increases the accuracy. All hyperparameter settings improve over the distillation training (tab. 5.1). The best setting leads to an accuracy of 67.0% on the test data, outperforming the pre-trained model (65.3%) by 1.7%.

Additionally, to allow a comparison to the results reported in the prior section 4.4, we evaluate the accuracy of the first trajectory. The pre-trained network achieves an accuracy of 57.0%, 3D mapping and ray tracing increased the accuracy to 61.6% with a $5cm$ voxel grid size. The retrained network now achieves an accuracy of 62.4% This is a significant improvement of 5.4%. The retrained network is even capable of outperforming the expensively generated supervision signal after retraining. We primarily credit this to the fact that the network is capable of filtering artifacts induced by the supervision signal generation.

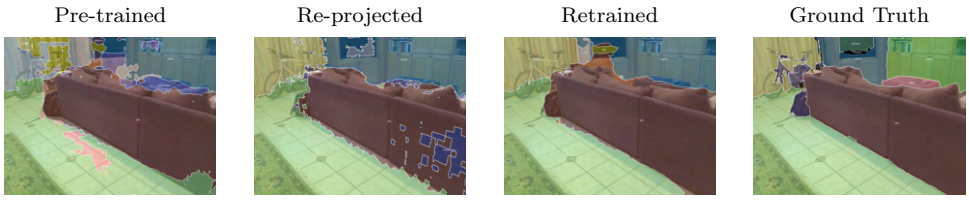



Figure 5.2: Label comparison for pre-trained, re-projected and retrained.

This is illustrated in figure 5.2. Here we compare the prediction by the pre-trained model (far left) with the generated pseudo label (left) and the retrained network prediction (right). We can clearly observe the artifacts induced by voxelization, ray tracing, and onehot-encoding the labels. For instance, when we retrain the network a smooth output covering the full sofa is generated. From these results, we can conclude that accumulating all predictions into the map is especially useful. Additionally, retraining can outperform the generated supervision signal.

### 5.1.2 Iterative Supervision Signal Improvement

The retrained network is now capable of predicting more accurate labels for the scene. Therefore we can rerun our supervision signal generation pipeline and retrain the models on the new pseudo labels. In table 5.3 we report the accuracy achieved on the first trajectory. The performance increases by each remapping step. When retraining on the pseudo labels provided by the 2nd iteration we experience an accuracy decrease of -0.9 percent. We stopped the iterative refinement of the label based on the fact that both re-projected labels maps (Re-pro. 2 and Re-pro. 3) achieved the same performance. A total accuracy increase of 6.8% is achieved.

| | Network 0 | Re-pro. 1 | Network 1 | Re-pro. 2 | Network 2 | Re-pro. 3 |
|---|---|---|---|---|---|---|
| *ACC* | 57.0 | 61.6 | 62.4 | 63.8 | 62.9 | 63.8 |
| *ΔACC* | – | 4.6 | 0.8 | 1.4 | -0.9 | 0.9 |

Table 5.3: Accuracy iterative mapping and retraining.

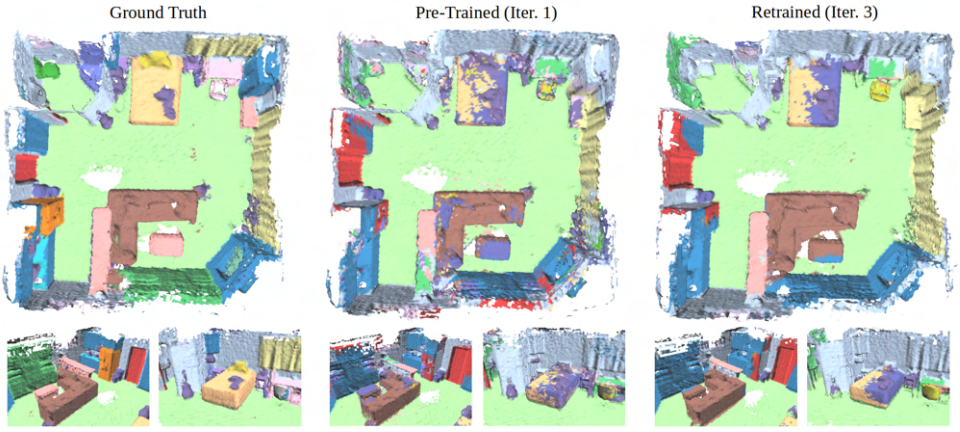



Figure 5.3: Visualization mapping improvement iterative retraining.

In figure 5.3 we illustrate a comparison between the ground truth, first iteration map, and final map. Before adaptation to the scene, a high level of noise is present for objects including the sofa, bed, and chair. When retraining this noise is efficiently filtered out. For the sofa, the correct label is now retrieved from all viewpoints. Given the visual similarity between the small table made out of the same material as the sofa, the network learned to falsely classify it as a sofa. For the chair in the ground truth labeling, no label is given therefore it is assigned by artifacts to the table which is behind the chair. Another object that experienced significant improvement is the bar table. On the other hand, the network learned to classify the toilet and guitar wrong, associating these objects with the wall class.

### 5.1.3 Semi-Supervised CL Benchmark

We at first generate the pseudo labels for all tasks with a resolution of 3cm based on the pre-trained network predictions. All pseudo labels are generated with the pre-trained network given that the performance during training degrades for future tasks. Experiments are performed on the Scannet4-P1+ scenario. The full accuracy matrix, including the pre-training task, is provided in figure 5.4.

| | **Task 0** | **Task 1** | **Task 2** | **Task 3** | **Task 4** |
|---|---|---|---|---|---|
| **Test Pre-train** | 61.7 | 65.9 | 54.5 | 40.3 | 62.8 |
| **Test Trained** | 61.7 | 66.1 | 51.6 | 39.3 | 74.9 |
| **Test Final** | 52.6 | 66.1 | 46.3 | 39.3 | 74.9 |
| **Train Pseudo** | - | 90.4 | 85.1 | 82.1 | 83.4 |

Table 5.4: Comparison CL benchmark unsupervised training.

In table 5.4, we denote the accuracy achieved after pre-training (fig. 5.4, first row), task specific training (fig. 5.4, diagonal), at the end of the CL scenario (fig. 5.4, last row) and the pseudo label accuracy on the test dataset.

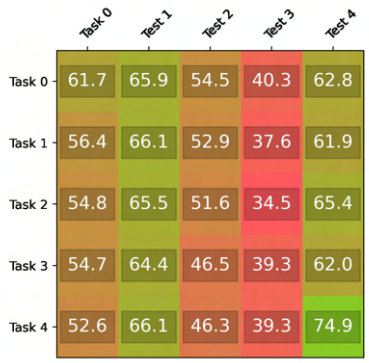



Figure 5.4: Accuracy matrix self supervised CL benchmark.

For two out of four tasks the final performance improves after the continual learning process. Only for task 2 does the performance decrease during task specific training. For task 3, the test accuracy increases during training but does not increase over the pretrained model. Significantly, for the last task the accuracy increases by over 12% compared to the pre-trained model.

| ACC | FWT | BWT |
|---|---|---|
| 56.0 | 52.4 | -2.1 |

Table 5.5: Semi-supervised CL performance.

In comparison after pre-training, we achieve an average accuracy of 55.8%. When completing the full continual learning we achieve an average accuracy of 56.6%. This is a slight performance increase of 0.8%. When comparing the accuracy achieved directly after adaptation to a scene we achieve an accuracy increase of 2.1%. The continually learned scene adaptation strongly depends on the quality of the supervision signal. When performing a visual inspection of the generated video sequences, we can clearly see that the predictions are more coherent from different viewpoints for the same object.

We investigate the failure cases of our proposed pseudo label generation. We mainly found three modes of failure.

**1.)** Degeneration of object boundaries due to integrating measurements from multiple viewpoints. This severely affects small objects in the foreground. Figure 5.5 (left plot) illustrates this with the kitchen plate being misclassified.

**2.)** Disagreeing observations lead to unreliable predictions. These unreliable predictions can be filtered out using the uncertainty of the map estimate. Despite this we do not include a method that is explicitly aware of uncertain regions during training.

**3.)** The prediction network might confidently predict the wrong class consistently over multiple frames for objects. We do not have an effective way in our pipeline

for recognizing or resolving this failure mode.

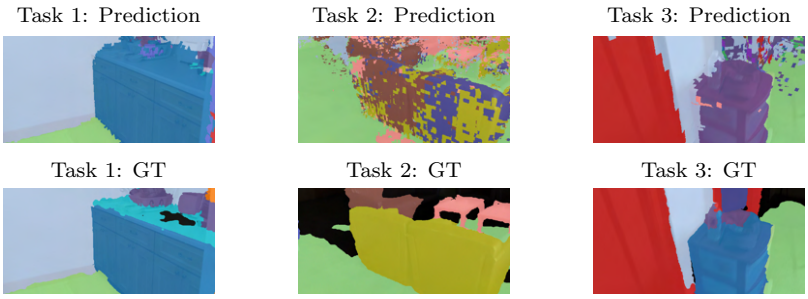



Figure 5.5: Failure modes of supervision signal generation.

In the accuracy matrix we observe that the task performance strongly deviates between tasks. This is due to the fact that the provided supervision signal's quality strongly deviates between scenes. Enforcing multi view consistency does not directly guarantee a perfect semantic supervision signal but more likely a temporally coherent one.

We have conducted experiments with generating the semantic segmentation labels after each task for the upcoming task based on the current network. This resulted in significantly worse performance given that when training on the current task, the performance on the future task suffers.
We have conducted further experiments with changing the buffer size, learning rate and including samples from the pre-training task with ground truth annotations in the memory buffer. Without ground truth annotations of the pre-trained task in the memory buffer, the performance decayed significantly faster. As we have seen when observing the pseudo labels, the predictions get more consistent over time.

### 5.1.4 Discussion

We have showcased that leveraging 2D and 3D data representations in a complementary manner improves semantic segmentation effectively. The performance increase achieved by iterative re-training is significant. This highly motivates further research in scene adaptation and unsupervised semantic segmentation using the proposed methods for 3D mapping and pseudo label generation. In the unsupervised CL experiments we have observed that the CL performance is highly dependent on the quality of the provided supervision signal. Even though our 3D mapping enforces multi-view consistency, which helps to increase the performance, multiple failure modes cannot be resolved. We could clearly show that the available supervision signal is the bottleneck when performing unsupervised continual learning. Fur future improvements we propose three research directions:
Firstly, exploring methods to integrate external knowledge for supervision. We propose to fuse multiple observations, predicted by different networks, into the 3D voxel map. Secondly, using the measurement of multi-view consistency and uncertainty of individual network predicting bears great potential. The uncertainty measurement may be integrated into the pseudo label generation or training procedure itself. Lastly, we propose to leverage scene understanding methods for generating the supervision signal. Identification of objects in 3D can effectively help to avoid mistaking small objects with the background. Additionally, physical constraints and object relations can help to reason about the semantic class.

## 5.2 Deployment

In this section we deploy our continual learning pipeline to real-world data.
We choose to use a handheld Azure Kinect RGB-D sensor to record data. It uses time-of-fight technology to estimate depth. It can capture color and depth data with a resolution of $3840 \times 2160$ and $640 \times 576$ respectively at 30fps. Depth can be measured between 0.5 - 5.46 meters depending on the setting. This sensor fails to retrieve depth measurements for light-absorbing surfaces or reflective surfaces.

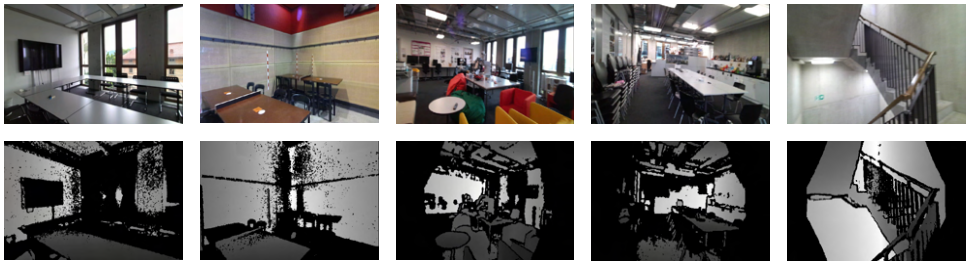


Figure 5.6: Example scenes recorded in the ETH Zurich.

In figure 5.6, we provide examples of 5 scenes we recorded at ETH Zurich. Each trajectory takes between 1-5 minutes. The scenes contain loop closures, motion blur, changing lighting conditions and vertical movement. Typical objects located in a cafeterias or office spaces are present. Also in a few scenes people are captured walking or sitting. The environment, with the exception of humans, is fully static.

For the floor in to conference room (fig. 5.6, far left) no depth measurements are available for the floor given the light absorbing carpet. We can choose between two settings for the field of view (FOV) of the depth camera. The WFOV allows to capture depths in the range of 0.25 - 2.88m with a FOV of $120° \times 120°$. The NFOV allows us to capture object in a distance of 0.5 - 5.45m with a FOV of $75° \times 65°$. In the WFOV a dense depth map fully covering the camera image is recorded (fig. 5.6, bottom row left) compared to the NFOV setting which only partially covers the RGB image (fig. 5.6, bottom row right). We observed that the best setting for our use case is the NFOV given the range of 5.45m which significantly helped to capture objects that are far away. All scenes are recorded at 30 fps.

### 5.2.1 Pose Estimation

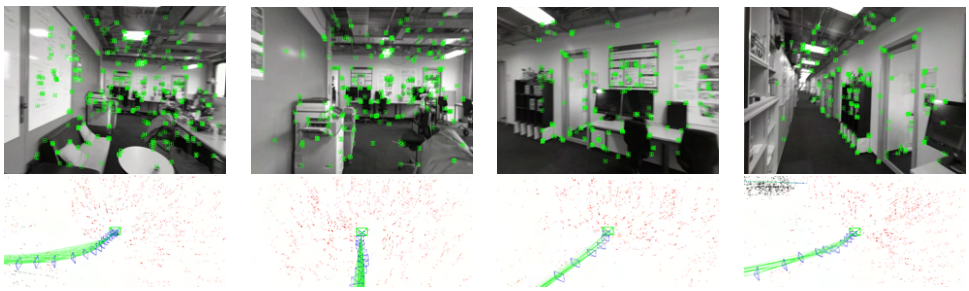


Figure 5.7: Example local trajectory.

To retrieve camera poses we investigated using an RGB-D based SLAM system. The depth values can be directly used to retrieve the absolute scale of the scene

which eases the SLAM problem. RGB-D based SLAM systems are significantly more robust than visual odometry (VO) and visual-inertial odometry (VIO) systems, when good depth data is available. We choose to use ORBSLAM2 based on fast inference time, loop closing and re-localization capabilities in a generated map. It runs up to four parallel threads executing tracking, local mapping, loop closing and bundle adjustment [112]. In figure 5.7, we provide an example of the retrieved local trajectory with illustrated landmarks and detected ORBSLAM features.

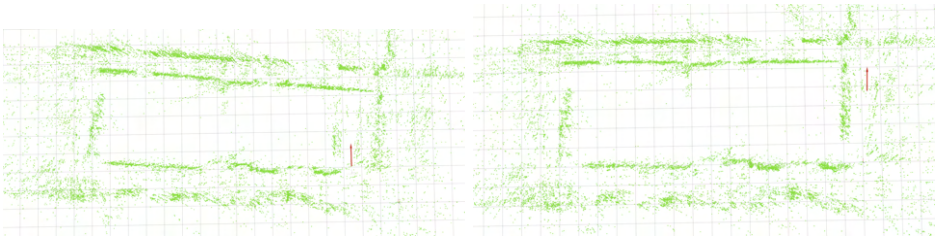


Figure 5.8: Loop closure detection and bundle adjusted landmarks: no loop closure (left), bundle adjusted landmarks (right).

The loop closing and local bundle adjustment capabilities of ORBSLAM2 can be seen in figure 5.8. On the left, the map is illustrated before the loop closure is detected. The red arrow marks the estimated camera pose. On the right, the bundle adjusted trajectory after loop closure detection is illustrated.

### 5.2.2 Mapping

Figure 5.9: Conference room 3D map.

We illustrate the mapping results for the *conference room* (fig. 5.9) and *hallway* (fig. 5.11) scenes. In the conference room no depth measurements are present for the television, windows and floor. This leads to the sparse 3D map. The table and walls are correctly classified in the map. The chairs are only partially observed and include a high level of noise. We can now compare the re-projected pseudo labels with the original network predictions in figure 5.10.

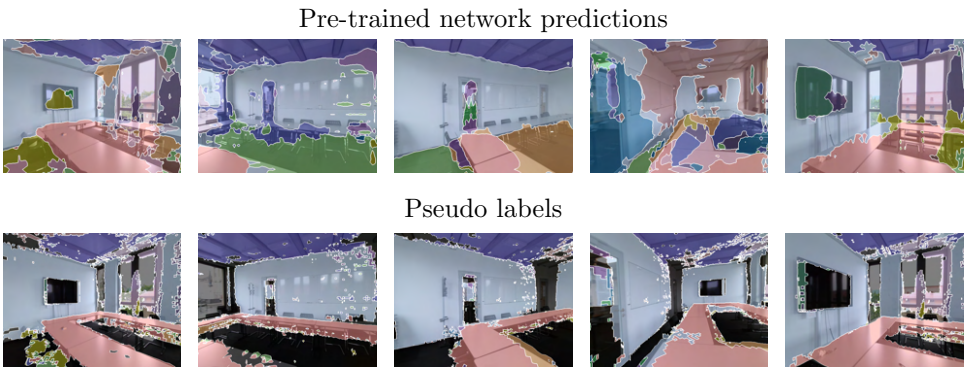



Figure 5.10: Comparison pre-trained prediction to pseudo label of conference room.

All illustrated frames are picked by random. We can observe in various frames that the network predictions fail to recognize the table. The pseudo labels on the other hand look significantly more accurate. This clearly illustrates the advantage of fusing information into a map. Similar results can be observed for the *hallway* scene. A consistent map is created with only small artifacts compared to the original network predictions. The bad performance of the pre-trained network can be induced by multiple factors. The network is pre-trained using data captured with a different sensor and on different scenes. In general the domain gap between the target and source domain is significant and therefore the bad performance on individual frames expected. Only when accumulating data in a map a reliable estimate is achieved, that can be used for further retraining and adaptation to the current scene.

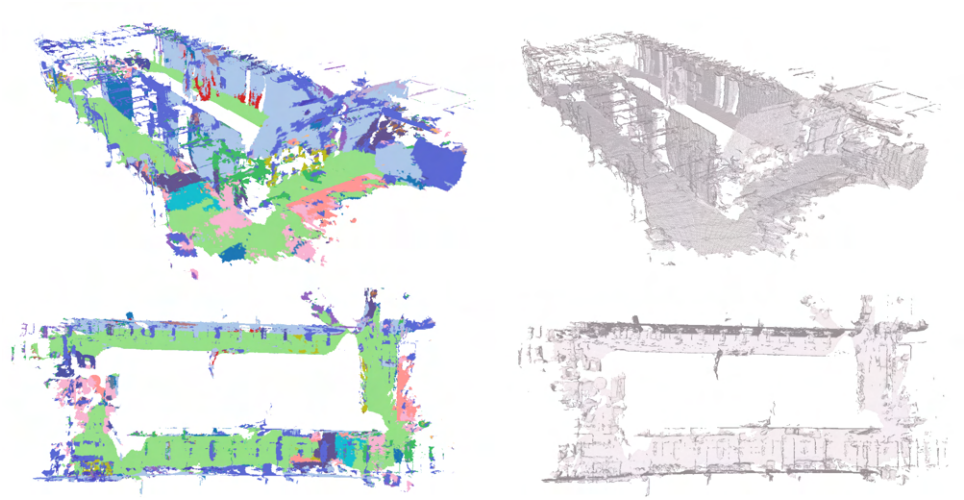


Figure 5.11: Hallway 3D map.


Figure 5.12: Comparison pre-trained prediction to pseudo label of hallway scene.

### 5.2.3 Discussion

We have shown that ORBSLAM2 is capable to provide good pose estimates for 3D mapping in real-time. The generated pseudo labels outperform single view predictions. The bad performance of the pre-trained model illustrates the need for a continual learning approach to integrate new knowledge acquired during the mission. Our current method is highly reliant on good depth estimates for pose estimation and mapping. If only sparse depth measurements are available less

information can be accumulated into the map. This problem may be mitigated by using a neural network to predict a dense depth map. For outdoor scenes, standard RGB-D cameras fail to estimate depth due to sunlight and larger distances outside of the sensor's specifications. Voxel-based approaches severely suffer for very large scenes given the cubic complexity. For indoor scenes, a 5cm resolution is sufficient to capture most semantic details of interest. Currently, our system is capable to distinguish into 40 different semantic classes. For most downstream tasks in robotics, not all classes are needed. Reducing the number of classes speeds up the mapping linearly. Our current implementation of CPU-based ray tracing takes 100ms per label. We also implemented a non-headless version using Open3D, which is mainly IO limited. Despite the fast pseudo label generation, we generate the 3D map in hindsight given the significantly better pose estimates obtained after global bundle adjustment. To increase the mapping accuracy a global bundle adjustment capable to modify the TSDF would be desirable.

# Chapter 6

# Conclusion

In conclusion, we have shown that CL can be accomplished for multi-class semantic segmentation. We have proposed two suitable benchmarks that allow for quantitative analysis of continual learning strategies. We used a memory replay buffer to efficiently mitigate catastrophic forgetting. In the continual learning experiments (sec. 3.4), we have concluded that even a small memory size can significantly increase performance. Additionally, we identified crucial design parameters governing the CL performance. This includes data augmentation, memory size, and buffer filling strategy.

For the semi-supervised benchmark (sec. 4.4), 3D mapping resulted in the best performance compared to regularization and optical flow-based approaches. The global multi-view consistency constraints allowed us to efficiently integrate observations from multiple viewpoints, resulting in a coherent map. The generated pseudo labels outperformed the network predictions. Iterative retraining and mapping (sec. 5.1.1) allowed us to further increase the performance. We additionally concluded that training on sparse labels with high accuracy is not a good supervision signal given that no training signal is provided for uncertain pixels.

When using a self-supervised learning signal to train a network over multiple scenes, we increase the performance for some scenes, achieving higher test accuracy overall. Despite this, in the semi-supervised setting, we were strongly limited by the generated supervision signal. Without external knowledge, a variety of failure modes could not be resolved. The created map after retraining was more coherent and the retraining reduced temporally inconsistent semantic predictions. When working with data recorded in the lab, we conclude that having access to precise poses is essential for precise mapping. The current implementation of our 3D mapping approach does not allow us to refine the map after detecting a loop closure. Therefore, we limited our approach to create the map in hindsight after observing the full trajectory. With the refined pose estimates, a high-quality 3D semantic map can be created. We achieved a more coherent network estimate after the integration of the newly acquired data. The generated pseudo labels outperformed single view predictions significantly.

With our work, we showcased unsupervised adaptation and deployment of a continual learning strategy for a robotic use case. We propose future research on two ends.

**1.)** CL is still an open research domain. We have shown that experience replay performs well but still suffers from catastrophic forgetting. Here all explained CL strategies seem to be viable candidates for further improvements. We motivate to

not limit the scope and explore the full spectrum of possible solutions.

**2.)** Unsupervised continual learning on pseudo labels has shown promising results. Utilizing multiple data modalities to generate a supervision signal can effectively reduce the need for human annotation. Leveraging complementary data structures (2D - 3D) is especially promising for semantic segmentation. We propose to further investigate using the semantic map as a memory buffer to store past experiences. This allows the creation of synthetic viewpoints training data if a textured 3D mesh of the scene is generated together with the semantic map. Effective utilization of uncertainty metrics to modify the training procedure may lead to additional improvements. Identifying objects within the semantic map can help to improve the compactness of the labels. Further research on leveraging scene understanding methods to reason about scene geometry, object modalities, and physical constraints is also worthwhile. Additionally, integration of external knowledge can improve the pseudo labels.

With future improvements in continual learning, we can hopefully achieve a fully autonomous robotic system that is capable to adapt to its environment online. For this, we need to improve on the currently available continual learning strategies and effectively acquire new training data unsupervised. With our work, we hoped to set the first stepping stone towards this goal.

# Bibliography


[1] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. USA: Bradford Company, 2004.

[2] J. Barnes, "Azure machine learning," *Microsoft Azure Essentials. 1st ed, Microsoft*, 2015.

[3] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," *CoRR*, vol. abs/1707.02968, 2017.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020.

[5] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. D. Rodríguez, "Continual learning for robotics," *CoRR*, vol. abs/1907.00182, 2019.

[6] M. B. Ring, "Continual learning in reinforcement environments," in *GMD-Bericht*, 1994.

[7] A. ROBINS, "Catastrophic Forgetting, Rehearsal and Pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[8] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini, "Developmental robotics: a survey," *Connection Science*, vol. 15, no. 4, pp. 151–190, 2003.

[9] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," ser. Psychology of Learning and Motivation, G. H. Bower, Ed. Academic Press, 1989, vol. 24, pp. 109–165.

[10] W. C. Abraham and A. Robins, "Memory retention – the synaptic stability versus plasticity dilemma," *Trends in Neurosciences*, vol. 28, no. 2, pp. 73–78, 2005.

[11] Y. Hsu, Y. Liu, and Z. Kira, "Re-evaluating continual learning scenarios: A categorization and case for strong baselines," *CoRR*, vol. abs/1810.12488, 2018.

[12] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," 2015.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[14] V. Lomonaco, L. Pellegrini, P. Rodríguez, M. Caccia, Q. She, Y. Chen, Q. Jodelet, R. Wang, Z. Mai, D. Vázquez, G. I. Parisi, N. Churamani, M. Pickett, I. H. Laradji, and D. Maltoni, "CVPR 2020 continual learning in computer vision competition: Approaches, results, current challenges and future directions," *CoRR*, vol. abs/2009.09929, 2020.

[15] A. Douillard, Y. Chen, A. Dapogny, and M. Cord, "PLOP: learning without forgetting for continual semantic segmentation," *CoRR*, vol. abs/2011.11390, 2020.

[16] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, "Towards continual reinforcement learning: A review and perspectives," *CoRR*, vol. abs/2012.13490, 2020.

[17] Y. Zou, Z. Yu, B. V. K. V. Kumar, and J. Wang, "Adaptation for semantic segmentation via class-balanced self-training," *CoRR*, vol. abs/1810.07911, 2018.

[18] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," 2016.

[19] Y. Tsai, W. Hung, S. Schulter, K. Sohn, M. Yang, and M. Chandraker, "Learning to adapt structured output space for semantic segmentation," *CoRR*, vol. abs/1802.10349, 2018.

[20] H. Blum, F. Milano, R. Zurbrügg, R. Siegwart, C. Cadena, and A. Gawel, "Self-improving semantic perception on a construction robot," *CoRR*, vol. abs/2105.01595, 2021.

[21] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[22] M. Roberts and N. Paczan, "Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding," *CoRR*, vol. abs/2011.02523, 2020.

[23] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020.

[24] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels," p. 15, 2010.

[25] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016.

[26] L. Porzi, M. Hofinger, I. Ruiz, J. Serrat, S. R. Bulò, and P. Kontschieder, "Learning multi-object tracking and segmentation from automatic annotations," *CoRR*, vol. abs/1912.02096, 2019.

[27] A. E. Takesian and T. K. Hensch, "Chapter 1 - balancing plasticity/stability across brain development," in *Changing Brains*, ser. Progress in Brain Research, M. M. Merzenich, M. Nahum, and T. M. Van Vleet, Eds. Elsevier, 2013, vol. 207, pp. 3–34.

[28] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[29] D. W. Blalock, J. J. G. Ortiz, J. Frankle, and J. V. Guttag, "What is the state of neural network pruning?" *CoRR*, vol. abs/2003.03033, 2020.

[30] G. Hacohen and D. Weinshall, "On the power of curriculum learning in training deep networks," *CoRR*, vol. abs/1904.03626, 2019.

[31] F. Lateef and Y. Ruichek, "Survey on semantic segmentation using deep learning techniques," *Neurocomputing*, vol. 338, pp. 321–348, 2019.

[32] F. Chollet, "On the measure of intelligence," *CoRR*, vol. abs/1911.01547, 2019.

[33] V. Lomonaco and D. Maltoni, "Core50: a new dataset and benchmark for continuous object recognition," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 17–26.

[34] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *CoRR*, vol. abs/1606.04671, 2016.

[35] H. Ahn, D. Lee, S. Cha, and T. Moon, "Uncertainty-based continual learning with adaptive regularization," *CoRR*, vol. abs/1905.11614, 2019.

[36] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *CoRR*, vol. abs/1612.00796, 2016.

[37] S. Lee, J. Kim, J. Ha, and B. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," *CoRR*, vol. abs/1703.08475, 2017.

[38] D. Isele and A. Cosgun, "Selective experience replay for lifelong learning," *CoRR*, vol. abs/1802.10269, 2018.

[39] P. Buzzega, M. Boschini, A. Porrello, and S. Calderara, "Rethinking experience replay: a bag of tricks for continual learning," *CoRR*, vol. abs/2010.05595, 2020.

[40] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne, "Experience replay for continual learning," *arXiv preprint arXiv:1811.11682*, 2018.

[41] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. S. Torr, and M. Ranzato, "Continual learning with tiny episodic memories," *CoRR*, vol. abs/1902.10486, 2019.

[42] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with A-GEM," *CoRR*, vol. abs/1812.00420, 2018.

[43] Y. Guo, M. Liu, T. Yang, and T. Rosing, "Learning with long-term remembering: Following the lead of mixed stochastic gradient," *CoRR*, vol. abs/1909.11763, 2019.

[44] M. Farajtabar, N. Azizan, A. Mott, and A. Li, "Orthogonal gradient descent for continual learning," *CoRR*, vol. abs/1910.07104, 2019.

[45] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[46] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[47] M. van der Ven and A. S. Tolias, "Generative replay with feedback connections as a general strategy for continual learning," *CoRR*, vol. abs/1809.10635, 2018.

[48] N. Kamra, U. Gupta, and Y. Liu, "Deep generative dual memory network for continual learning," *CoRR*, vol. abs/1710.10368, 2017.

[49] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," *CoRR*, vol. abs/1705.08690, 2017.

[50] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[51] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *CoRR*, vol. abs/2005.12872, 2020.

[52]

[53] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017.

[54] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[55] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016.

[56] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, 2015.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[58] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *CoRR*, vol. abs/1612.01105, 2016.

[59] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, vol. abs/1706.05587, 2017.

[60] Y. Yuan, X. Chen, and J. Wang, "Object-contextual representations for semantic segmentation," 2020.

[61] R. P. K. Poudel, S. Liwicki, and R. Cipolla, "Fast-scnn: Fast semantic segmentation network," *CoRR*, vol. abs/1902.04502, 2019.

[62] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[63] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European Conference on Computer Vision (ECCV)*, ser. LNCS, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9906. Springer International Publishing, 2016, pp. 102–118.

[64] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez, "The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes," 2016.

[65] Y. Chen, W. Li, and L. V. Gool, "ROAD: reality oriented adaptation for semantic segmentation of urban scenes," *CoRR*, vol. abs/1711.11556, 2017.

[66] Y. Chen, Y. Lin, M. Yang, and J. Huang, "Crdoco: Pixel-level domain transfer with cross-domain consistency," *CoRR*, vol. abs/2001.03182, 2020.

[67] J. Hoffman, E. Tzeng, T. Park, J. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," *CoRR*, vol. abs/1711.03213, 2017.

[68] D. Stutz, A. Hermans, and B. Leibe, "Superpixels: An evaluation of the state-of-the-art," *CoRR*, vol. abs/1612.01601, 2016.

[69] J. Lafferty, A. Mccallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 01 2001, pp. 282–289.

[70] B. Fulkerson, A. Vedaldi, and S. Soatto, "Class segmentation and object localization with superpixel neighborhoods," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 670–677.

[71] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," *CoRR*, vol. abs/1210.5644, 2012.

[72] D. Sun, S. Roth, and M. J. Black, "Secrets of optical flow estimation and their principles," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2432–2439.

[73] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," *CoRR*, vol. abs/1504.06852, 2015.

[74] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," *CoRR*, vol. abs/1612.01925, 2016.

[75] D. Sun, X. Yang, M. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," *CoRR*, vol. abs/1709.02371, 2017.

[76] Z. Teed and J. Deng, "RAFT: recurrent all-pairs field transforms for optical flow," *CoRR*, vol. abs/2003.12039, 2020.

[77] M. Menze, C. Heipke, and A. Geiger, "Object scene flow," *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[78] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European Conf. on Computer Vision (ECCV)*, ser. Part IV, LNCS 7577, A. Fitzgibbon et al. (Eds.), Ed. Springer-Verlag, Oct. 2012, pp. 611–625.

[79] D. Nilsson and C. Sminchisescu, "Semantic video segmentation by gated recurrent flow propagation," *CoRR*, vol. abs/1612.08871, 2016.

[80] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *CoRR*, vol. abs/1506.02025, 2015.

[81] P. Tokmakov, K. Alahari, and C. Schmid, "Learning motion patterns in videos," *CoRR*, vol. abs/1612.07217, 2016.

[82] S. Jain, B. Xiong, and K. Grauman, "Fusionseg: Learning to combine motion and appearance for fully automatic segmention of generic objects in videos," *arXiv preprint arXiv:1701.05384*, 2017.

[83] W. Wang, J. Shen, F. Porikli, and R. Yang, "Semi-supervised video object segmentation with super-trajectories," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 4, pp. 985–998, 2019.

[84] M. Grinvald, F. Furrer, T. Novkovic, J. J. Chung, C. Cadena, R. Siegwart, and J. Nieto, "Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3037–3044, July 2019.

[85] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. Kelly, and A. Davison, "Slam++: Simultaneous localisation and mapping at the level of objects," *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1352–1359, 2013.

[86] J. McCormac, A. Handa, A. J. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," *CoRR*, vol. abs/1609.05130, 2016.

[87] J. Dong, X. Fei, and S. Soatto, "Visual-inertial-semantic scene representation for 3d object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3567–3577.

[88] T. Whelan, S. Leutenegger, R. Moreno, B. Glocker, and A. Davison, "Elasticfusion: Dense slam without a pose graph," 07 2015.

[89] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[90] A. Douillard, Y. Chen, A. Dapogny, and M. Cord, "PLOP: learning without forgetting for continual semantic segmentation," *CoRR*, vol. abs/2011.11390, 2020.

[91] ——, "Tackling catastrophic forgetting and background shift in continual semantic segmentation," 2021.

[92] D. Galvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[93] J. Postels, H. Blum, C. Cadena, R. Siegwart, L. V. Gool, and F. Tombari, "Quantifying aleatoric and epistemic uncertainty using density estimation in latent space," *CoRR*, vol. abs/2012.03082, 2020.

[94] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 567–576.

[95] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung, "Scenenn: A scene meshes dataset with annotations," in *2016 Fourth International Conference on 3D Vision (3DV)*, 2016, pp. 92–101.

[96] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison, "Scenenet RGB-D: 5m photorealistic images of synthetic indoor trajectories with ground truth," *CoRR*, vol. abs/1612.05079, 2016.

[97] A. Dai, M. Nießner, M. Zollöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," *ACM Transactions on Graphics 2017 (TOG)*, 2017.

[98] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Online continual learning with no task boundaries," *CoRR*, vol. abs/1903.08671, 2019.

[99] Y. Balaji, M. Farajtabar, D. Yin, A. Mott, and A. Li, "The effectiveness of memory replay in large scale continual learning," *CoRR*, vol. abs/2010.02418, 2020.

[100] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *CoRR*, vol. abs/1712.04621, 2017.

[101] D. Berthelot, N. Carlini, I. J. Goodfellow, N. Papernot, A. Oliver, and C. Raffel, "Mixmatch: A holistic approach to semi-supervised learning," *CoRR*, vol. abs/1905.02249, 2019.

[102] A. Tarvainen and H. Valpola, "Weight-averaged consistency targets improve semi-supervised deep learning results," *CoRR*, vol. abs/1703.01780, 2017.

[103] L. N. Smith and N. Topin, "Super-convergence: Very fast training of residual networks using large learning rates," *CoRR*, vol. abs/1708.07120, 2017.

[104] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147.

[105] I. Bilbao and J. Bilbao, "Overfitting problem and the over-training in the era of data: Particularly for artificial neural networks," in *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*, 2017, pp. 173–177.

[106] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[107] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continuum learning," *CoRR*, vol. abs/1706.08840, 2017.

[108] P. Buzzega, M. Boschini, A. Porrello, and S. Calderara, "Rethinking experience replay: a bag of tricks for continual learning," *CoRR*, vol. abs/2010.05595, 2020.

[109] T. L. Hayes, N. D. Cahill, and C. Kanan, "Memory efficient experience replay for streaming learning," *CoRR*, vol. abs/1809.05922, 2018.

[110] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European Conf. on Computer Vision (ECCV)*, ser. Part IV, LNCS 7577, A. Fitzgibbon et al. (Eds.), Ed. Springer-Verlag, Oct. 2012, pp. 611–625.

[111] P. P. Brahma, D. Wu, and Y. She, "Why deep learning works: A manifold disentanglement perspective," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 10, pp. 1997–2008, 2015.

[112] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *CoRR*, vol. abs/1610.06475, 2016.

# Appendix A

## A.1 Continual Learning Results

### A.1.1 Memory Size

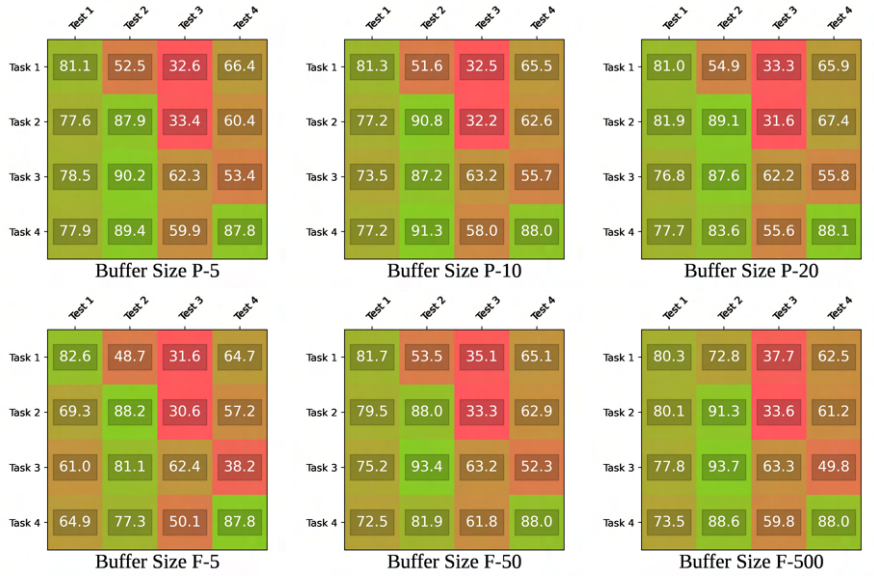



Figure A.1: ScanNet: Accuracy matrix different memory sizes.

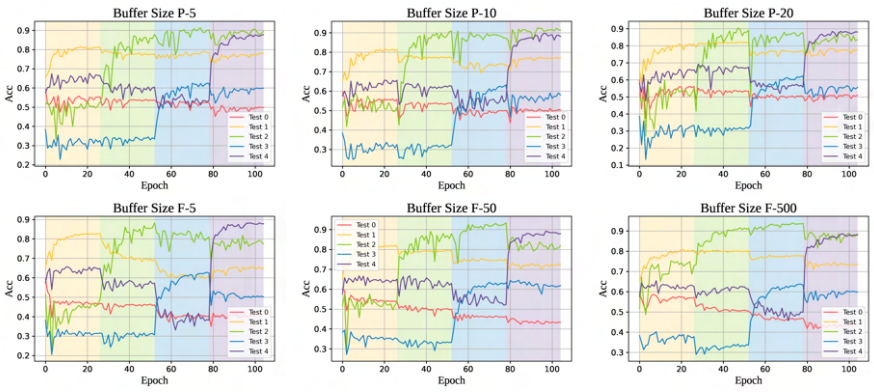



Figure A.2: ScanNet: Learning curve different memory sizes.

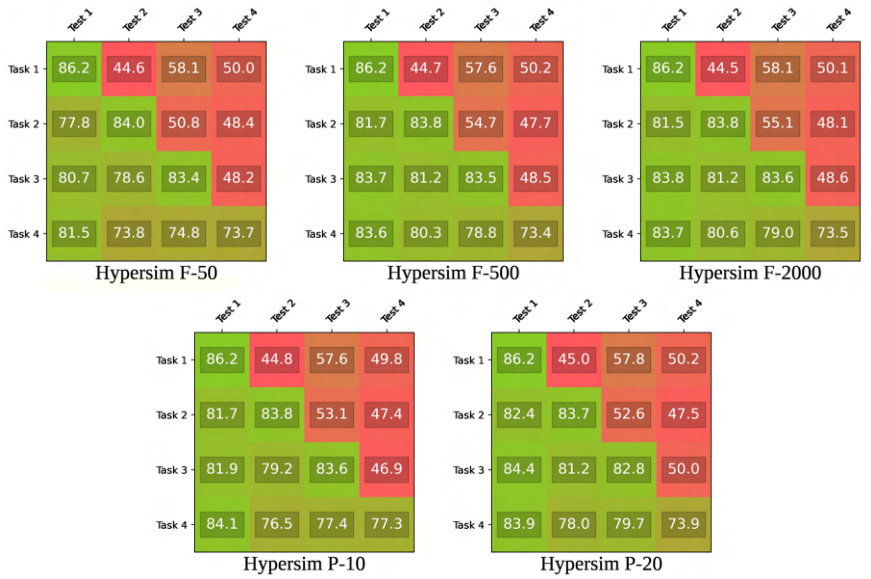



Figure A.4: Hypersim: Learning curve different memory sizes.

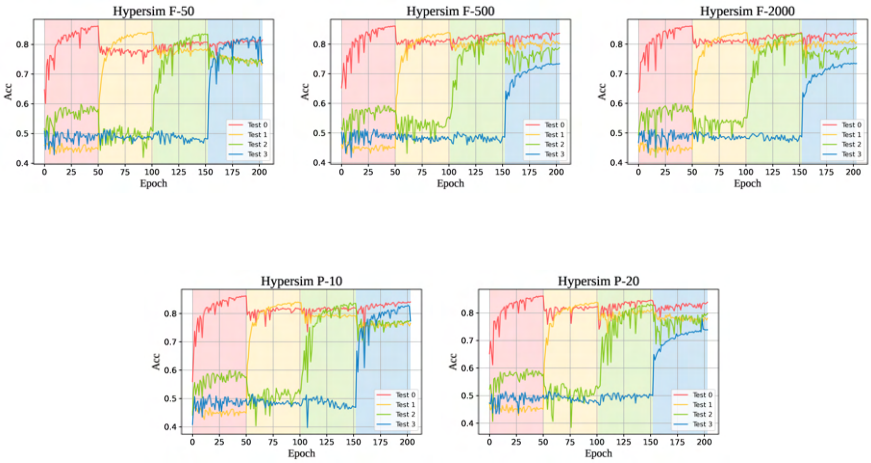



Figure A.3: Hypersim: Learning curve different memory sizes.

### A.1.2 Memory Sampling

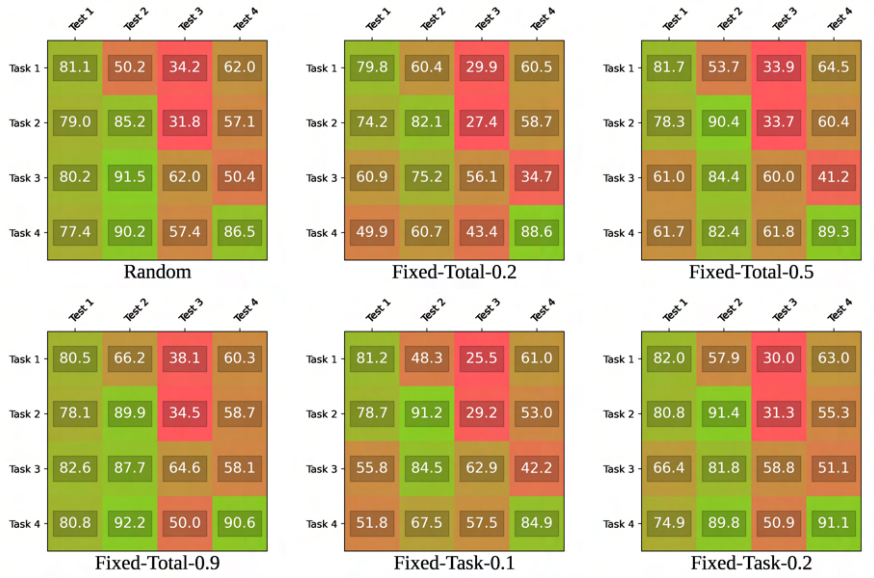



Figure A.5: ScanNet: Accuracy matrix different sampling strategies.

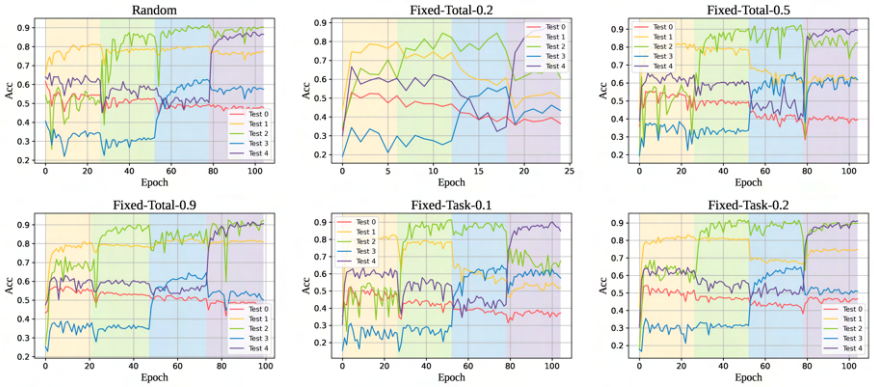



Figure A.6: ScanNet: Learning curve different sampling strategies.

### A.1.3 Augmentation

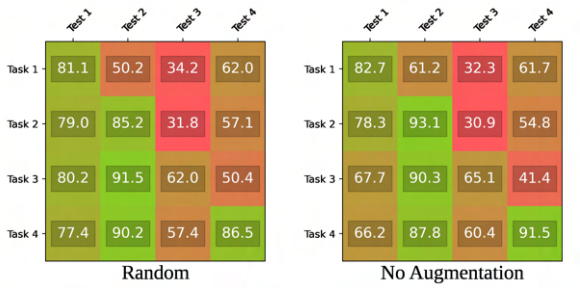



Figure A.7: ScanNet: Accuracy matrix augmentation.

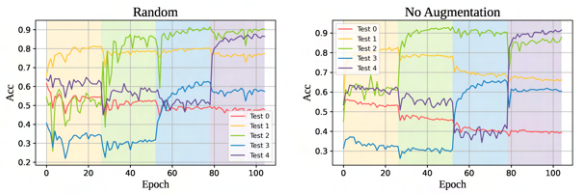



Figure A.8: ScanNet: Accuracy matrix augmentation.

### A.1.4 Labels

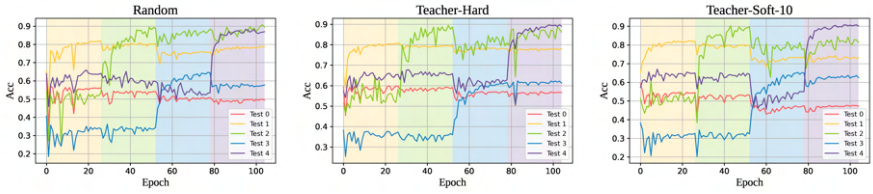



Figure A.9: ScanNet: Learning curve different supervision labels.

### A.1.5 Latent Replay

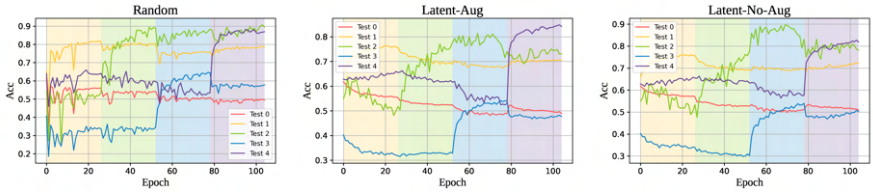



Figure A.10: ScanNet: Learning curve different latent replay.