Object Discovery, Interactive and 3D Segmentation for Large-Scale Computer Vision Tasks

Von der Fakultät für Mathematik, Informatik und
Naturwissenschaften der RWTH Aachen University zur
Erlangung des akademischen Grades einer Doktorin der
Naturwissenschaften genehmigte Dissertation

vorgelegt von

Master of Science
Theodora Kontogianni
aus Ioannina, Griechenland

Berichter: Prof. Dr. Bastian Leibe
Prof. Dr. Konrad Schindler

Tag der mündlichen Prüfung: 29.11.2021

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek verfügbar.

Abstract

Computer vision has made tremendous leaps during the past decade. One of the key factors behind this growth is the vast amount of data that we can generate today: millions of pictures are shared online daily and new specialized sensors allow to easily capture 3D data. Along with the recent advances in deep learning and increased availability of computational power, it is now possible to take advantage of these large amounts of high-quality data. As a result, computer vision achieved impressive performance gains across numerous fields and applications.

However, the increased amount of available data also introduces new challenges. To exploit the large body of available data, we either need efficient unsupervised algorithms to learn patterns from unlabeled data, or we require efficient labeling tools to allow the creation of large-scale labeled datasets. These are essential for the success of most deep learning models.

In this thesis, we deal with issues arising from these different aspects of computer vision: unsupervised algorithms for landmark recognition, fully-supervised methods for semantic segmentation on large-scale 3D point clouds and interactive object segmentation for out-of-domain dataset labeling. More specifically, the main contributions of this thesis are organized into three parts, each one covering an individual computer vision topic:

In the first part, we address the problem of object discovery in time-varying, large-scale image collections. We propose a novel tree structure that closely approximates the Minimum Spanning Tree and present an efficient construction approach to incrementally update the tree structure when new data is added to the image database. This happens either in online-streaming or batch form. Our proposed tree structure is created in a local neighborhood of the matching graph during image retrieval and can be efficiently updated whenever the image database is extended. We show how our tree structure can be incorporated in existing clustering approaches such as Single-Link and Iconoid Shift for efficient large-scale object discovery in image collections.

In the second part of the thesis, we focus on defining novel 3D convolutional and recurrent operators over unstructured 3D point clouds. The goal is to learn point representations for the task of 3D semantic segmentation. The recurrent consolidation unit layer operates on multi-scale and grid neighborhoods along and allows our model to learn long-range dependencies. Additionally, we introduce two types of local neighborhoods for each 3D point that encode local geometry to facilitate the definition and use of convolutions on 3D point clouds.

Finally, in the third part, we address the task interactive object segmentation. Aided by an algorithm, a user segments an object mask in a given image by clicking inside or outside the object. We present a method that significantly reduces the number of required user clicks compared to previous work. In particular, we look at out-of-domain settings where the test datasets are significantly different from the datasets used to train our deep model. We propose to treat user corrections as sparse supervision to adapt our model parameters on-the-fly. Our adaptive method can significantly reduce the number of required clicks to segment an object and handle distribution shifts from small to large, specialize to a new class of objects introduced during test time, and can even handle large domain changes from commercial images to medical and aerial data.

Zusammenfassung

Die Computer Vision hat im letzten Jahrzehnt enorme Sprünge gemacht. Einer der Schlüsselfaktoren für dieses Wachstum ist die riesige Menge an Daten, die wir heute generieren können: Millionen von Bildern werden täglich online geteilt und neue spezialisierte Sensoren ermöglichen die einfache Erfassung von 3D-Daten. Zusammen mit den jüngsten Fortschritten im Deep Learning und der zunehmenden Verfügbarkeit von Rechenleistung ist es nun möglich, diese großen Mengen an hochwertigen Daten zu nutzen. Infolgedessen konnte die Computer Vision in zahlreichen Bereichen und Anwendungen beeindruckende Leistungssteigerungen erzielen.

Allerdings bringt die erhöhte Menge an verfügbaren Daten auch neue Herausforderungen mit sich. Um die große Menge an verfügbaren Daten zu nutzen, benötigen wir entweder effiziente unüberwachte Algorithmen, um Muster aus unmarkierten Daten zu lernen, oder wir benötigen effiziente Markierungswerkzeuge, um die Erstellung großer markierter Datensätze zu ermöglichen. Diese sind für den Erfolg der meisten Deep-Learning-Modelle unerläßlich.

In dieser Arbeit befassen wir uns mit Problemen, die sich aus diesen verschiedenen Aspekten der Computer Vision ergeben: unüberwachte Algorithmen für die Erkennung von Landmarken, vollüberwachte Methoden für die semantische Segmentierung auf großen 3D-Punktwolken und interaktive Objektsegmentierung für die Beschriftung von Datensätzen außerhalb der Domäne. Genauer gesagt sind die Hauptbeiträge dieser Arbeit in drei Teile gegliedert, von denen jeder ein individuelles Thema der Computer Vision abdeckt:

Im ersten Teil befassen wir uns mit dem Problem der Objekterkennung in zeitlich variierenden, großflächigen Bildsammlungen. Wir schlagen eine neuartige Baumstruktur vor, die sich dem Minimum Spanning Tree stark annähert, und präsentieren einen effizienten Konstruktionsansatz, um die Baumstruktur inkrementell zu aktualisieren, wenn neue Daten zur Bilddatenbank hinzugefügt werden. Dies geschieht entweder im Online-Streaming oder in Batch-Form. Die von uns vorgeschlagene Baumstruktur wird in einer lokalen Nachbarschaft des übereinstimmenden Graphen während des Bildabrufs erstellt und kann effizient aktualisiert werden, wenn die Bilddatenbank erweitert wird. Wir zeigen, wie unsere Baumstruktur in bestehende Clustering-Ansätze wie Single-Link und Iconoid Shift zur effizienten, großflächigen Objekterkennung in Bildsammlungen integriert werden kann.

Im zweiten Teil der Arbeit konzentrieren wir uns auf die Definition neuartiger 3D-Faltungsoperatoren und rekurrenter Operatoren über unstrukturierten 3D-Punktwolken. Das Ziel ist es, Punktrepräsentationen für die Aufgabe der semantischen 3D-Segmentierung zu erlernen. Die rekurrente Konsolidierungsschicht operiert auf Multiskalen- und Gitternachbarschaften entlang und erlaubt unserem Modell, weitreichende Abhängigkeiten zu lernen. Zusätzlich führen wir zwei Arten von lokalen Nachbarschaften für jeden 3D-Punkt ein, die lokale Geometrie kodieren, um die Definition und Verwendung von Faltungen auf 3D-Punktwolken zu erleichtern.

Im dritten Teil schließlich widmen wir uns der Aufgabe der interaktiven Objektsegmentierung. Mit Hilfe eines Algorithmus segmentiert ein Benutzer eine Objektmaske in einem gegebenen Bild, indem er innerhalb oder außerhalb des Objekts klickt. Wir stellen eine Methode vor, die die Anzahl der erforderlichen Benutzerklicks im Vergleich zu früheren Arbeiten deutlich reduziert. Insbesondere betrachten wir Out-of-Domain-Einstellungen, bei denen sich die Testdatensätze signifikant von den Datensätzen unterscheiden, die zum Trainieren unseres Deep Models verwendet wurden. Wir schlagen vor, Benutzerkorrekturen als spärliche Überwachung zu behandeln, um unsere Modellparameter on-the-fly anzupassen. Unsere adaptive Methode kann die Anzahl der erforderlichen Klicks zur Segmentierung eines Objekts erheblich reduzieren und Verteilungsverschiebungen von klein bis groß handhaben, auf eine neue Klasse von Objekten spezialisieren, die während der Testzeit eingeführt werden, und kann sogar große Domänenwechsel von kommerziellen Bildern zu medizinischen und Luftbilddaten bewältigen.

Acknowledgments

First and foremost, I would like to thank my advisor Prof. Dr. Bastian Leibe, for the many years of support, interesting discussions and the opportunity to conduct research in his group. I would also like to thank Prof. Dr. Konrad Schindler for his interest in my work and for agreeing to be co-examiner of my thesis defense.

I would also like to thank all my current and former colleagues at the RWTH Computer Vision Group for all the fruitful discussions and valuable feedback who made work exciting and fun. I am lucky to call them friends. In particular, I would like to thank my inspiring co-authors that contributed to the works presented in this thesis (in chronological order): Markus Matthias, Francis Engelmann, Alexander Hermans, Jonas Schult, Vittorio Ferrari, Michael Gygli and Jasper Uijlings.

I also thank my internship hosts for giving me the opportunity to learn on new exciting projects: Tobias Weyand and Fangting Xia (Google Research, 2018).

Last but not least, I would like to thank my parents, my sister and her family for always being supportive.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Contributions . . . 4
1.3 Structure of the Thesis . . . 5

2 State of the Art **9**
2.1 Landmark Object Recognition . . . 9
2.2 3D Semantic Segmentation . . . 12
2.3 Interactive Object Segmentation . . . 14

3 Traditional Unsupervised Methods Preliminaries **17**
3.1 Local Features . . . 18
3.2 G-Equivariance and Invariance . . . 19
3.3 Local Keypoint Detection . . . 19
3.4 Local Descriptors . . . 21
3.5 Tasks . . . 22
  3.5.1 Image Retrieval (Part I) . . . 22
  3.5.2 Matching Graph Construction and Clustering (Part I) . . . 28
3.6 Datasets . . . 28

4 Deep Learning Preliminaries **29**
4.1 Feedforward Neural Networks . . . 30
  4.1.1 Perceptron . . . 30
  4.1.2 Multilayer Perceptron . . . 30
  4.1.3 Training . . . 32
  4.1.4 Backpropagation . . . 33
4.2 Convolutional Neural Networks . . . 33
  4.2.1 The Convolutional Operator . . . 34
  4.2.2 The Discrete Convolutional Operator . . . 34
  4.2.3 Convolutional Layer . . . 35

4.2.4 Depthwise Separable Convolutions 36
4.2.5 Atrous (Dilated) Convolutions 37
4.2.6 Pooling/Unpooling Layers 38
4.3 Convolutions in 3D space 39
4.4 Residual Connections 40
4.5 Recurrent Networks 42
4.6 Tasks 43
4.6.1 Semantic Segmentation (Part II) 43
4.6.2 Interactive Object Segmentation (Part III) 46
4.7 3D Datasets 47
4.8 2D Datasets 48

I Efficient spanning tree construction and update for graph based clustering algorithms. 51

5 Incremental Object Discovery in Time-Varying Image Collections 53
5.1 Introduction 53
5.2 Related Work 55
5.3 Clustering Methods for Object Discovery 56
5.3.1 Single-Link Agglomerative Clustering 56
5.3.2 Iconoid Shift 57
5.4 Limited Horizon Minimum Spanning Tree 59
5.4.1 Offline Construction of LH-MST 59
5.4.1.1 Complexity Analysis. 60
5.4.2 Incremental Update of LH-MST 61
5.4.2.1 Complexity Analysis 62
5.5 Experiments 63
5.5.1 Evaluation of Single - Link Clustering 63
5.5.2 Evaluation of Offline LH-IS 64
5.5.2.1 Results of Offline LH-MST 66
5.5.3 Evaluation of online LH-MST 68
5.6 Conclusion 72

II 3D Scene Understanding 75

6 Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds 77
6.1 Introduction 77

6.2 Related Work 79
6.3 Method 80
  6.3.1 PointNet 80
  6.3.2 Input-Level Context 81
  6.3.3 Output-Level Context 82
  6.3.4 Multi-Scale (MS) Architecture 83
  6.3.5 Grid (G) Architecture 83
6.4 Experiments 84
  6.4.1 Evaluation Measures 85
  6.4.2 Quantitative Results 85
  6.4.3 Qualitative Results 86
6.5 Conclusion 91

7 Know What Your Neighbors Do: 3D Semantic Segmentation of Point Clouds 95
7.1 Introduction 96
  7.1.1 Contributions 97
7.2 Related Work 97
  7.2.1 Voxelized Methods. 98
  7.2.2 Point Cloud Methods. 98
  7.2.3 Feature Networks. 99
7.3 Our Approach 99
  7.3.1 Feature Network 99
  7.3.2 Neighborhoods 102
    7.3.2.1 Feature space neighborhood $\mathcal{N}_F$ 102
    7.3.2.2 World space neighborhood $\mathcal{N}_W$ 102
  7.3.3 Loss Functions 103
    7.3.3.1 Pairwise Similarity Loss $\mathcal{L}_{pair}$ 103
    7.3.3.2 Centroid Loss $\mathcal{L}_{cent}$ 104
7.4 Implementation Details 106
7.5 Evaluation 107
  7.5.1 Indoor Datasets 107
  7.5.2 Outdoor Dataset 107
  7.5.3 Training Details 108
7.6 Conclusions 110

III Continuous Adaptation for Interactive Object Segmentation by Learning from Corrections 115

8 Continuous Adaptation for Interactive Object Segmentation 117
8.1 Introduction . . . 117
8.2 Related Work . . . 119
8.3 Method . . . 121
  8.3.1 Interactive Segmentation Model . . . 121
  8.3.2 Learning from Corrections at Test-Time . . . 122
    8.3.2.1 Adapting to a single image. . . . 123
    8.3.2.2 Adapting to an image sequence. . . . 124
    8.3.2.3 Combined adaptation. . . . 124
  8.3.3 Simulating user corrections . . . 125
8.4 Experiments . . . 125
  8.4.1 Adapting to distribution shift . . . 128
  8.4.2 Adapting to a specific class . . . 129
  8.4.3 Adapting to domain changes . . . 135
  8.4.4 Comparison to Previous Methods . . . 137
  8.4.5 Ablation Study . . . 137
  8.4.6 Subsampling for image sequence adaptation . . . 138
  8.4.7 Robustness to image order . . . 138
  8.4.8 Adaptation speed . . . 138
  8.4.9 Adaptation parameters . . . 139
  8.4.10 Model details . . . 139
8.5 Comparison on the COCO dataset . . . 140
8.6 Additional Qualitative Results . . . 140
8.7 Conclusion . . . 140

9 Conclusion 143
9.1 Discussion of Contributions . . . 143
9.2 Perspectives . . . 145
  9.2.1 Perspectives on Object Discovery. . . . 145
  9.2.2 Perspectives on 3D Data. . . . 146

Bibliography 149

1

Introduction

1.1 Motivation

Computer vision is a scientific field that relies on digital representations of the physical world coming from different sensors. The goal is to teach machines to understand the visual world and eventually act upon it to perform desired tasks. These representations can take many forms such as digital images, videos sequences, depth images or 3D laser point clouds. This ability to perceive and understand their surroundings, move and manipulate objects along with planning and action taking is fundamental to living organisms in order to achieve their goals.

While the above tasks are easy for humans, they are challenging for computer vision algorithms. However, recently computer vision has made great leaps and even achieved superhuman performance in some specific tasks.

Some of the reasons for those recent advances is the abundance of available data, coming from social media platforms (e.g., Facebook, Instagram), new sensors that are readily available and allow everyday users to record previously specialized data formats (e.g., Apple LiDAR sensor, Kinect, Matterport) and new datasets which consist of millions of labeled images such as OpenImages (Krasin *et al.*, 2017), Places (Zhou *et al.*, 2017a), ILSVRC (Russakovsky *et al.*, 2015).

The large quantities of *unlabeled* data from social media platforms gave rise to many *unsupervised* learning algorithms. Datasets consisting of unlabeled data can be created relatively easily since they are usually just a collection of samples such as photos, audio recordings, videos or text. There is no human-created information associated with each data sample. In the context of machine learning, an unsupervised algorithm is tasked with learning patterns from the unlabeled data on its own (like the entire data distribution of the dataset). Clustering methods that group data into clusters of similar examples belong to the unsupervised learning approaches. Gathering unlabeled data from various internet sources might seem relatively easy but it poses its own new challenges. Social media content changes constantly as new data is added at any moment in time, this provides new exciting insights but updating unsupervised learning algorithms with the additional unlabeled examples can be difficult. Most of the unsupervised learning algorithms can be computationally expensive and re-computing their outcome every time a new example is added is in most cases infeasible especially in large-scale internet datasets.

In the first part of this thesis, we address this problem by proposing an efficient construction and update mechanism for *minimum spanning trees*, a key component of many clustering algorithms that are part of large-scale *object discovery* pipelines.

While in the earliest days of computer vision LiDAR and laser-based scanners were bulky and expensive, and thus limited to laboratory and industrial applications, the development of low-cost depth cameras, 3D scanners and LiDARs like the Microsoft Kinect and Apple LiDAR sensor allowed large-scale capturing of 3D data. LiDAR and depth cameras allow accurate geometry and localization of objects in 3D space. These representations led to the rise of several new computer vision tasks including *3D semantic segmentation.* In 3D semantic segmentation each 3D point is associated with a semantic category. It is a crucial component for achieving overall *3D scene understanding* that enables navigating and operating in the real world for autonomous cars and robots. However, the 3D data acquired by these sensors come in many different formats including depths images, meshes and point clouds. 3D point clouds are a commonly used format, in which the 3D data is represented in their raw form as a set of 3D coordinates without any loss of information due to discretization and scale ambiguity. However, deep learning methods on 3D point clouds face several significant challenges like the unstructured and irregular nature of point clouds, the scale of the data and the varied resolution of point clouds that hinders the design of efficient convolutional operators that have proven so successful on the task of 2D semantic segmentation.

In the second part of the thesis, we focus on defining 3D convolutional and recurrent operators over unstructured 3D point clouds to learn informative point representations for the task of 3D semantic segmentation. Our approaches build on the PointNet architecture (Qi *et al.*, 2017b). PointNet-based architectures learn point-wise features by processing each 3D point independently. Then a permutation invariant function (like max-pooling) is used in the set of points and outputs a *global descriptor* that characterizes the point cloud. This way, the representations learned are computed either on individual point level or take into consideration the whole point cloud through the max-pooling operator. In classic convolutional neural networks each feature representation learned is associated with a *receptive* *field* value that identifies the region in the input that produces (influences) the feature. Convolutional operators compute a response at each position as a weighted sum of the features in a *local* neighborhood. The lack of structure of point clouds and therefore point neighborhoods makes the definition and use of convolutions challenging. So PointNets are limited to having a receptive field of one data point with only the use of the max-pooled feature to provide more global context. In chapters 6 and 7 we address the lack of structure in 3D point clouds. We define (i) multi-scale and grid neighborhoods and propose the *recurrent consultation unit* layer that updates the global descriptor based on the global descriptors of the rest of the point cloud and allowing our model to learn *long-range dependencies* and (ii) local neighborhoods for each 3D point that encode *local geometry* to facilitate the definition and use of convolutions on 3D point clouds.

Despite the rise of unsupervised methods that focus on the ability of deep learning models to learn from large quantities of unlabeled examples, most computer vision applications require *supervision* with large *labeled* datasets to achieve good performance. Labeled datasets associate each example with an empty *label* or *target* value that is provided by human instructors to teach our models what is required in a particular task. Common types of labels for computer vision tasks include tags that indicate whether an image contains an object in classification tasks, free text describing an image for image capturing or a table associated to each pixel of an image for semantic segmentation. However, labeling datasets is a cumbersome and slow process that requires a lot of manual effort. Particularly in semantic segmentation an average image of size 640 × 640 requires 409600[1] annotated pixels. In 3D semantic segmentation the effort to assign a semantic label in each 3D point is even larger since some datasets like Semantic3D (Hackel *et al.*, 2017) consist of scenes with up to 22 million points. Furthermore, there are datasets for specialized applications as in the medical domain that not only require significant effort to label but can only be annotated by domain experts (i.e., doctors). In the third part of the thesis, we focus on the task of interactive object segmentation that enables fast and accurate object segmentation which is indispensable for collecting ground-truth segmentation masks at scale. The objective is to infer accurate segmentation masks with as little as possible human interference. In interactive object segmentation a human collaborates with a computer vision model to segment an object of interest. The process iteratively alternates between the user providing corrections on the current segmentation and the model refining the segmentation based on these corrections. These corrections typically refer to point clicks or strokes

---

[1] Supervised algorithms exceed human performance when trained on datasets with at least 10 million labeled examples (Goodfellow *et al.*, 2016).

on mislabeled pixels. Previous interactive object segmentation methods train on large-scale dataset of consumer photos perform well when they are applied to data with similar distribution as the training set. In practice, the distribution almost always changes between training and testing. This happens for example when annotating a new dataset using interactive segmentation. In this scenario a model trained on PASCAL (Everingham *et al.*, 2012) (or any other dataset) might be used to segment classes that are not present in that training dataset or even on different modalities like aerial or medical data. In this thesis, we propose to treat user corrections as sparse supervision to adapt the model parameters on-the-fly. Our adaptive method significantly reduces the number of required clicks to segment an object and handle distribution shifts from small to large, specialize to a new class of objects, and can even handle large domain changes to medical and aerial data.

1.2 Contributions

In detail we make the following contributions:

- We propose a novel spanning tree structure called LH-MST, which approximates a Minimum Spanning Tree (MST) but instead displays efficient construction for online streaming data and can be incrementally updated when several additional data arrive in a batch form. We show how this tree structure can be incorporated in Single-Link (Gammeter *et al.*, 2009; Quack *et al.*, 2008; Zheng *et al.*, 2009) and Iconoid Shift (Weyand and Leibe, 2011) clustering for efficient large-scale object discovery in image collections. We verify experimentally that the resulting clustering approaches using LH-MST achieve very similar results as their offline version, while being both considerably faster and capable of online updates of a significant fraction of the image database.

- We propose two mechanisms that enlarge the receptive field and incorporate larger-scale spatial context in PointNet architectures (Qi *et al.*, 2017b) for the task of semantic segmentation on 3D point clouds. Traditional PointNet architectures operate on single, independent blocks that represent a local neighborhood in space. Our novel recurrent and sequential consolidation units can capture long-range dependencies by incorporating context propagated from other local neighborhoods. Compared to state-of-the-art methods at the time of development, our proposed extensions achieve improved results on challenging indoor and outdoor datasets and were the base of many follow-up works.

- While our previous work focused on mechanisms that pass information between local neighborhoods based on the gained insights, we additionally introduce several techniques that define a local neighborhood. We define point neighborhoods based on two grouping techniques one in the Euclidean and one in feature space based on K-Means and k-NN, respectively. Additionally, we reshape the feature space and thus the neighborhoods with dedicated loss functions. This enables us to learn local features shared among local structures as by 2D convolutions in the 2D domain in contrast to previous works that have no local structure and thus treat each 3D point individually or the simple grid and multi-scale volumes that defined a local neighborhood in our previous work. In a thorough evaluation, we analyze the contribution of each component and are able to present state-of-the art result on 3D semantic segmentation benchmarks.

- We propose a new interactive object segmentation method that updates the model parameters on-the-fly using user corrections as sparse training examples instead of training on large datasets and keeping the model parameters unchanged at test time. Our approach enables the adaptation to a particular object and its background, to distributions shift in a test set, to specific object classes, and even to large domain changes, where the imaging modality changes between training and testing. This semi-automatic annotation procedure can reduce the annotation effort significantly and proves exceedingly important in the case of limited annotated data for training (e.g., medical images). We perform extensive experiments on 8 diverse datasets and compared to a model with frozen parameters our method reduces significantly the required user effort in many challenging scenarios like (i) distribution shifts between training and testing; (ii) specializing to a specific class; and (iii) a complete domain change between training and testing.

1.3 Structure of the Thesis

This thesis consists of three major parts which address different computer vision problems caused in large-scale perception projects: object discovery in large-scale time-varying image databases, semantic segmentation on 3D point clouds with millions of data points and interactive object segmentation to assist the creation of large varying datasets with minimum human effort. Each chapter is dedicated to a specific topic of published work as indicated below. The overall structure of this thesis is as follows.

Chapter 2 - “State of the Art” briefly discusses the relevant concurrent works to methods proposed in this thesis, as well as their perspectives relative to the recent advancements in the field.

Chapter 3 - “Traditional Unsupervised Methods Preliminaries” introduces the foundations of hand-made features and similarity measures along with the high-level tasks of image retrieval and clustering that are crucial components of the landmark object discovery task that we focus on Chapter 5.

Chapter 4 - “Deep Learning Preliminaries” introduces the fundamentals and basic buildings blocks of deep learning required to understand the remaining chapters (Chapters 6, 7 and 8) in the thesis.

Chapter 5 - “Incremental Object Discovery in Time-Varying Image Collections” proposes a novel Limited Horizon Minimum Spanning Tree (LH-MST) structure that closely approximates the Minimum Spanning Tree at a small fraction of the latter’s computational cost. Our proposed tree structure can be created in a local neighborhood of the matching graph during image retrieval and can be efficiently updated whenever the image database is extended. We show how the LH-MST can be used within both single - link hierarchical agglomerative clustering and the Iconoid Shift framework for object discovery in image collections, resulting in significant efficiency gains and making both approaches capable of incremental clustering with online updates. We evaluate our approach on a dataset of 500k images from the city of Paris and compare its results to the batch version of both clustering algorithms. The content of this chapter is based on Kontogianni *et al.* (2016).

Chapter 6 - “Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds” focuses on the task of semantic segmentation performed directly on unstructured 3D point clouds. It extends the recently proposed PointNet architecture (Qi *et al.*, 2017b) that operates on unstructured point clouds learning a local feature for each individual point and then aggregate all point features to a global one. However, PointNet architectures operates independently on individual blocks of limited volume and thus have limited receptive field. Furthermore, they have no hierarchical structure and are limited to process each point separately or all points together. We address these issues by introducing multi-scale and grid neighborhoods each represented by a feature encoding. We update each encoding based on the encoding of its neighbors using novel recurrent and sequential consolidation units. The key idea is to enlarge the receptive field, incorporate larger-scale spatial context and capture long range dependencies. We evaluate the proposed strategies on challenging indoor and outdoor datasets and show improved results in both scenarios over the previous baselines. The content of this chapter is based primarily on (Kontogianni *et al.*, 2017).

Chapter 7 - "Know What Your Neighbors Do: 3D Semantic Segmentation of Point Clouds" expands on the ideas of point neighborhoods introduced in the previous chapter (Kontogianni *et al.*, 2017). PointNet architectures do not encode any local structure where the network can learn local features shared among common local structures. We introduce new grouping techniques which define point neighborhoods both in the Euclidean and the learned feature space. Neighborhoods are important as they allow to compute local or global point features depending on the spatial extent of the neighborhood. Point neighborhoods can be compared to the filter size in traditional convolutions in the 2D image domain. Additionally, we incorporate dedicated loss functions to further structure the learned point feature space: the pairwise distance loss and the centroid loss. We show how to apply these mechanisms to the task of 3D semantic segmentation of point clouds and report state-of-the-art performance on indoor and outdoor datasets over previous baselines of concurrent works. The content of this chapter is based primarily on (Engelmann *et al.*, 2018).

Chapter 8 - "Continuous Adaptation for Interactive Object Segmentation by Learning from Corrections" In interactive object segmentation a user collaborates with a computer vision model to segment an object. Recent works employ convolutional neural networks for this task: Given an image and a set of corrections made by the user as input, they output a segmentation mask. These approaches achieve strong performance by training on large datasets, but they keep the model parameters unchanged at test time. Instead, we recognize that user corrections can serve as sparse training examples and we propose a method that capitalizes on that idea to update the model parameters on-the-fly to the data at hand. Our approach enables the adaptation to a particular object and its background, to distribution shifts in a test set, to specific object classes, and even to large domain changes, where the imaging modality changes between training and testing. We perform extensive experiments on 8 diverse datasets and show: Compared to a model with frozen parameters, our method reduces the required corrections (i) by 9%-30% when distribution shifts are small between training and testing; (ii) by 12%-44% when specializing to a specific class; (iii) and by 60% and 77% when we completely change domain between training and testing.

Note: *The thesis is based on the technical contributions of my respective first author publications. Several images and text passages in the three major parts of this thesis are taken from these articles. However, additional and new content about further extensions has been added in order to provide deeper insights into our approaches. One chapter is based on shared first au-*

thorship publications resulting from collaborations. For these particular parts an additional note will be given in the footnote of the corresponding chapter.

2

State of the Art

In this chapter, we discuss the state of the art in the fields of Landmark Object Recognition, 3D Semantic Segmentation and Interactive Object Segmentation. We start by briefly reviewing previous works in these fields and how they relate to the work in this thesis, and we follow by introducing recent efforts published since the work in this thesis took place.

2.1 Landmark Object Recognition

Feature Embeddings. In the pre-deep learning era the image representation field was dominated by hand-crafted feature descriptors such as the Scale Invariant Feature Transform (SIFT) (Lowe, 2004) like the ones we used in our work presented in Chapter 5. Originally, SIFT was comprised by a detector and a descriptor but most SIFT-based methods usually use a Hessian-Affine detector along with the SIFT descriptor. The 128-dim SIFT descriptors are accumulated to a global vector representing an image using Bag-of-Words (BoW) histograms with a pre-trained codebook (Philbin *et al.*, 2007). More details regarding the methodologies of hand-crafted local feature extraction can be found in Chapter 3.

Even though handcrafted features are still used in some fields of computer vision (e.g., SLAM) since 2012 and the breakthrough works of AlexNet (Krizhevsky *et al.*, 2012a) and ImageNet (Russakovsky *et al.*, 2015) opened a new era of deep learning-based methods focusing on convolutional neural networks and learned features based on convolutional filters. Even though CNNs have shown to outperform hand-crafted features in many computer vision tasks in image retrieval, competitive performance compared to the SIFT and BoW approaches in Oxford5k was only achieved in 2016 (Gordo *et al.*, 2016) due to the local featuresŕobustness to scaling, cropping and image clutter along with the domain gap by the use of pre-trained networks on ImageNet (Russakovsky *et al.*, 2015) that could not generalize to the specialized task of landmark recognition.

(i) *Off-the-shelf deep learning features.* Deep learning methods require training on large-scale datasets that include labelled examples for the task at hand. However, such datasets did not initially exist for image retrieval. So, early works on learned features for image retrieval focused on using deep models trained on datasets for image recognition and classification that were instead readily available. The most straightforward idea is to use CNN pre-trained models on large datasets like ImageNet (Russakovsky *et al.*, 2015). These models can take a whole image as input and extract features that hopefully generalize for the task of image retrieval. Babenko *et al.* (2014); Razavian *et al.* (2014) use the features extracted from the fully connected layer and use the features directly in a standard image retrieval setup as is presented in Fig. 3.3. These early works although they outperform other standard non-deep learning global descriptors, their performance is significantly below the state of the art.

Babenko and Lempitsky (2015); Razavian *et al.* (2016); Yue-Hei Ng *et al.* (2015) focus on local features extracted from the intermediate convolutional layers. A fusion of both can also be used (Wei *et al.*, 2017).

Since fully connected layers lack geometric invariance Gong *et al.* (2014) generate several patches from an input image using sliding window techniques, multiple scales or random sampling and feed them all into the network. Standard encoding schemes can then be applied following the SIFT paradigm for image retrieval. The column features can be aggregated using BoW (Kulkarni *et al.*, 2015) or VLAD (Yue-Hei Ng *et al.*, 2015). Local descriptors can also be pooled into discriminative global features. (Tolias *et al.*, 2016) can create geometry aware descriptors from a single pass using max-activations on convolutional feature maps.

(ii) *Fine-tuned deep learning features.* Image level labelling is time-consuming especially for large scale image retrieval datasets. However, the sole use of deep features extracted from deep networks trained for different tasks other than image retrieval suffer from the domain gap between the tasks. Fine-tuning the pre-trained networks for the specific task of image retrieval with any amount of labelled data is crucial (Ong *et al.*, 2017; Oquab *et al.*, 2014; Radenovic *et al.*, 2016). In fine-tuned deep learning models the network will usually produce the final feature representation of an image without additional encoding or pooling steps. For a comprehensive survey on the latest deep learning methods for content-based image retrieval please see Chen *et al.* (2021).

Image Retrieval. Regardless of how the image features have been obtained or which ones have been proved to be more beneficial the retrieval step of finding the most similar images to a query remains the same as in the standard SIFT-based retrieval pipeline (Fig. 3.3, 2nd step), i.e finding the k instances in the image database that are closer to the query. Approximate nearest neighbor (ANN) search methods such as hashing (Perronnin *et al.*, 2010) or inverted files (Philbin *et al.*, 2007) are used for fast retrieval. For an in-depth review of the topic please refer to Chapter 3.

So despite the significant advances of deep learning features for image retrieval our work in Chapter 5 is orthogonal to the above methods. Our proposed method for efficient landmark recognition in large-scale, time-varying image collections can be used with any type of feature representations.

Object Discovery. Object discovery approaches aim at finding clusters of images showing the same object or landmark building in large image collections. Traditionally image matching techniques are applied based on image feature matching (Philbin *et al.*, 2007) to build up a matching graph of images and to then subdivide the connected components of this matching graph into image clusters. A large range of clustering methods have been proposed for this step, including single-link hierarchical agglomerative clustering (Gammeter *et al.*, 2009; Quack *et al.*, 2008; Zheng *et al.*, 2009), spectral clustering (Heath *et al.*, 2010; Kim *et al.*, 2008; Philbin and Zisserman, 2008), eigenvector centrality (Jing and Baluja, 2008), spherical k-means (Simon *et al.*, 2007), Kernel Vector Quantization (Avrithis *et al.*, 2010) and Medoid Shift mode estimation (Weyand and Leibe, 2011, 2013). Clustering techniques for object discovery require runtime efficiency due to the large scale of the datasets to be mined. Many works focus on parallelization and speed however the focus on incremental object discovery where the clusters have to be efficiently updated instead of re-running the entire clustering process whenever a new set of images are added to the underlying database is very limited.

Incremental Object Discovery. Despite the new advances in image retrieval with the use of deep learning features most deep learning approaches for image retrieval and landmark recognition focus mainly on static datasets where all categories are available during training. However, this is not applicable to real-world applications where new images are added on a daily basis.

Models for image retrieval and recognition have to scale with a fast-increasing number of images and be adaptable to new incoming objects. Recently a few works have been published on the topic of incremental learning (Chen *et al.*, 2020; Wu *et al.*, 2019a) however they focus on fine-grained image retrieval where object categories are added over time and does not extend to landmark recognition.

Our work presented in Chapter 5 instead focuses on content-based image retrieval (also known as instance-based image retrieval) where the aim is to retrieve images that contain the same landmark/object instance and object instance discovery where the task is to identify specific object instances from unstructured databases. Our work is scalable to tens of millions of images (internal communication with Google) and it can retrieve and recognize existing and newly added landmarks accurately and efficiently.

2.2 3D Semantic Segmentation

Conventional Approaches. There are several approaches that process 3D data even before the rise of deep learning methods. Hackel *et al.* (2016) use traditional random forest classifiers with 3D features (without color). Munoz *et al.* (2008) replace the random forest classifier with an associative Markov network following a similar approach. Zhang *et al.* (2015) use random forest classifiers on 2D and 3D data, which they later fuse. Similarly, Xu *et al.* (2013) fuse camera and LiDAR sensor data.

2D Projections. One straightforward way to impose structure on 3D data is to create 2D image projections from multiple views of the 3D scene. Then 2D-CNNs can be applied to each of the views. This method has the advantage of exploiting well research 2D-CNN architectures designed to perform well on the task of 2D semantic segmentation. Boulch *et al.* (2017); Lawin *et al.* (2017); Qi *et al.* (2016) project point-clouds in multi-view synthetic images that are processed by 2D convolutional networks. The predictions of all the multi-view images are fused to a single semantic label and back-projected into each 3D point. The above methods however discard the 3D spatial information and are therefore surpassed by methods that operate on 3D representations of the data.

3D Projections. Similar to 2D projections the 3D space can be divided into a volumetric grid. Each grid cell is called voxel and has specific size and discrete coordinates similar to pixels in a 2D image. With this dense representation with predefined local connectivity and structure we can use 3D convolutional networks. Numerous approaches for 3D semantic segmentation rely on voxelized input data (Dai and Nießner, 2018; Dai *et al.*, 2018; Huang and You, 2016; Maturana and Scherer, 2015; Wu *et al.*, 2015). However, with these dense grid representations, the computational complexity and memory footprints grows significantly. The more fine-grained the voxel grid the higher the computational cost and memory requirements. A coarser representation though leads to discretization artifacts and loss of information.

To alleviate this problem Riegler *et al.* (2017) divide hierarchically the 3D space into irregular 3D volumes using octrees while Klokov and Lempitsky (2017) partition the 3D space with kd-trees structures. However, even though reduced, empty volumes still require memory and computational effort. Graham *et al.* (2018) takes advantage of the natural sparsity of 3D point clouds and proposes a memory and computationally efficient approach that stores only non-empty voxels. 3D Convolutions are then defined only on a non-empty voxel and the filter positions operate also only on voxels that have at least one 3D point. The idea was popularized by Choy *et al.* (2019) that used it as the base of the Minkowski Engine. MinkowskiEngine-based networks dominate the 3D semantic segmentation benchmarks, however not all data can be represented in a voxelized grid (e.g., not all data represent spatial coordinates) like for example graphs or sets. So, in this work (Chapters 6, 7) we focus on methods that operate on raw 3D data directly without any data transformation pre-processing step.

3D Point Clouds. Point clouds by nature are unstructured and with irregular density and require 2D/3D projections to be processed by conventional 2D/3D architectures as proposed by the methods described above. However, recently several methods (Li *et al.*, 2018a; Qi *et al.*, 2017a,b; Wang *et al.*, 2018a,c; Wu *et al.*, 2019b; Xu *et al.*, 2018b) have been proposed that directly operate on raw point clouds for the tasks of semantic segmentation and classification. They take as input usually just the 3D coordinates of the 3D points however sometimes the input features are enhanced by color or normals depending on the availability but require no pre-processing of the input data to 2D or 3D projections. PointNet (Qi *et al.*, 2017b) is a pioneering such work that directly operates on point clouds. PointNet processes each point individually through a series of multi-layer perceptrons (MLPs) and aggregates the global information of a 3D scene through a permutation invariant function (max pooling) over all the points making the network permutation invariant. However, PointNet fails to take advantage of the local structure since the MLPs are processing every point individually and the max-pooling function that operates on a set of points ignores any local structure. Additionally, PointNet-like architectures cannot operate on a full 3D scene and instead split the scene into 3D volumes and process them independently and thus cannot exploit global context. In 2D-CNN local structure is already define by the 2D structure of digital images that allows the use of local filters like 2D discrete convolutions and the series of convolutions expand gradually the receptive field of each neuron incorporating context from increasingly larger parts of the image. In part we address these issues.

In chapter 6 we introduce multi-scale and grid neighborhoods overall the 3D scene with each sub-volume represented by a learned feature encoding. We learn an update for each encoding based on the encoding of neighboring volumes using novel recurrent and sequential consolidation units that incorporate global spatial context and capture long range dependencies.

Several works followed that rely on the definition of spatial neighborhoods (Li *et al.*, 2018a; Qi *et al.*, 2017a; Wang *et al.*, 2018c; Wu *et al.*, 2019b). Most of them though focus on the definition of local neighborhoods, this shown by their preference in the use of limited number of k nearest neighbors (KNN) of every point to represent its neighborhood (Li *et al.*, 2018a; Qi *et al.*, 2017a; Wang *et al.*, 2018c; Wu *et al.*, 2019b) while our work in chapter 6 focuses on the incorporating global context. Only Landrieu and Simonovsky (2018) focus on global context by over-segmenting the whole point cloud scene into super-points using clustering methods. PointNet-based architectures can not only process directly 3D point clouds but also are significantly faster than the Minkowski Engine Choy *et al.* (2019) based architectures making them as popular in 3D applications. However, the latter lead the benchmarks on 3D semantic segmentation. Our work though takes a step towards understanding global context that comes naturally from 3D convolutional networks that process whole scenes like the Choy *et al.* (2019).

In chapter 7 we introduce two types of local neighborhoods to impose structure on the unstructured point clouds. We use k-means to create neighborhoods in the world space and KNN for neighborhoods in the feature space and perform fusion of the two types of neighborhoods. Other works compute only one type of neighborhoods and focus either on the feature (Wang *et al.*, 2018c; Wu *et al.*, 2019b) or the world space (Li *et al.*, 2018a; Qi *et al.*, 2017a). Schult* *et al.* (2020) employ and take advantage also of two types of neighborhoods one on the Euclidean space and one on the geodesic space define over the mesh surface proving that it is beneficial to combine two types of convolutions and it leads to significant performance gains for 3D semantic segmentation.

2.3 Interactive Object Segmentation

Conventional Approaches. Conventional approaches in the pre-deep learning era perform interactive segmentation through energy minimization on a graph defined over the image pixels (Bai and Sapiro, 2009; Boykov and Jolly, 2001; Gulshan *et al.*, 2010; Price *et al.*, 2010; Rother *et al.*, 2004a). The user inputs are used to create an image-specific appearance model based on low-level features (e.g. color), which is then used to predict foreground and background probabilities. These classical methods are based on a weak appearance model which is specialized to one specific image.

Deep Learning Approaches. Recent methods use Convolutional Neural Networks (CNNs) for the task of interactive object segmentation (Agustsson *et al.*, 2019; Benard and Gygli, 2017; Chen *et al.*, 2018a; Hu *et al.*, 2019; Jang and Kim, 2019; Li *et al.*, 2018b; Liew *et al.*, 2017; Mahadevan *et al.*, 2018; Xu *et al.*, 2016). These methods take an image and the user corrections as input and produce foreground and background probabilities. The user corrections are represented as guidance maps and are part of the input as additional channels. However, the deep learning models are optimized over a training dataset and remain frozen at test time. Hence these models have a strong appearance model, but it cannot be adapted to the particularities of the test image or dataset at hand.

Our method described in part III combines the advantages both of energy minimization and the deep learning recent approaches: we learn a strong initial appearance model from a training set using CNNs but during segmentation of a new test image we can adapt the model to it. Furthermore, we can continuously adapt the model not only to a new image but also to new class distribution of the whole test set, which may be significantly different from the one the model is originally trained on.

Optimization at test time. In other applications it is common practice to fine-tune on in-domain data: (i) when transferring a model to a new domain (Caelles *et al.*, 2017; Papadopoulos *et al.*, 2016; Voigtlaender and Leibe, 2017; Zhou *et al.*, 2017c), (ii) iteratively minimize a loss at test time (Gatys *et al.*, 2016; Gygli *et al.*, 2017; Sun *et al.*, 2019) or (iii) on few-shot (Finn *et al.*, 2017; Qi *et al.*, 2018; Ravi and Larochelle, 2016; Snell *et al.*, 2017) and (iv) continual learning scenarios (Farquhar and Gal, 2018; Rebuffi *et al.*, 2017).

In the domain of interactive segmentation, Jang and Kim (2019) update the user corrections guidance maps representation that is part of the input to the model. Sofiiuk *et al.* (2020) expand on that idea and update intermediate feature activations, rather than the guidance maps. In contrast to these methods, we are the first to continuously adapt an interactive segmentation model by updating the model parameters, both, per image and over the whole test sequence. This results in a more general method and allows our model to adapt to individual images as well as sequences.

3

Traditional Unsupervised Methods Preliminaries

In this chapter (Chapter 5), we propose a novel LH-MST spanning tree structure on an edge-weighted directed graph (*matching graph*). LH-MST approximates the MST but it can be incrementally updated very efficiently.

To prove the efficiency and usability of our construction we implore the task of *landmark object recognition*. In such a task, we want to automatically discover and identify objects, and in particular touristic landmarks (e.g., Eiffel Tower) in large and unstructured image collections (*image database*) that change in time.

Most classic landmark object recognition pipelines (including the one described here) consist of the following steps:

Step 1. Image Retrieval discovers image relationships in unstructured image databases.

Step 2. Matching Graph construction, represents the above image relationships through image similarity.

Step 3. Clustering images that represent the same object to refine the Matching Graph results.

In this work we focus on the clustering step and in particular on the part of clustering that involves the Minimum Spanning Tree (MST) construction and update.

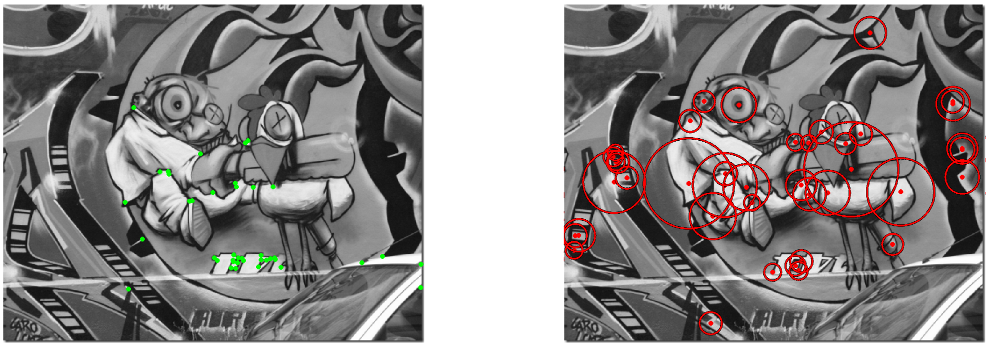
However, Step 1-*Image Retrieval* and Step 2-*Matching Graph Construction* are still key components of the object recognition pipeline. So here we will introduce the basic techniques underlying each step of the object recognition pipeline. The techniques explained (e.g., local features) here have been the state-of-the-art at the time of the publication of our work and are still widely used. Recently several advances have been made especially in the field of image retrieval (local and global

CNN features). We discuss current approaches in image retrieval, matching graph construction and clustering along with state-of-the-art methods for the overall task of object recognition in Chapter 2.1.

3.1 Local Features

A *local feature* is part of an image that differs distinctively from its immediate neighborhood. With the term *detector* we refer to the algorithm used to discover local features in an image. A local feature can be either a point or small image region. Many applications like camera calibration require only a location in space with no spatial extent. In these cases one usually refers to that location as *interest point* or *keypoint*. In this chapter, we use the term keypoint, and it can represent either points or regions.

Regardless of whether the local feature is a point or a region, a local neighborhood surrounding the local feature is still needed. Usually through its statistics described by the local feature *descriptor* the local features can be matched, and corresponding features can be found in other images. The local features along with their corresponding descriptors are used in a wide range of systems and applications. Figure 3.1a shows some example local features (points) based on a Harris detector (Harris and Stephens, 1988) and Figure 3.1b shows some example local features (regions) based on a SURF detector (Bay *et al.*, 2006).



(a) Harris detector **(b)** SURF

Figure 3.1: Example of local features based on points detected by the Harris corner detector (left) and regions based on SURF (right).

3.2 G-Equivariance and Invariance

Repeatability. The goal of the keypoint detector is to detect repeatable keypoints on the input images under various lighting conditions, noise, blur etc. Given two images of the same object or scene a high percentage of the features detected on the scene part visible in both images should be found in both images, regardless of the different viewing conditions. This is an important property for all applications including the image retrieval part that us a crucial component of our pipeline described in this work.

G-equivariance. A function f is called G-equivariant under a group of transformations G if $f(g \cdot x) = g \cdot f(x),\ \forall g \in G$.

G-invariance. A function f is called G-invariant under a group of transformations G if $f(g \cdot x) = f(x),\ \forall g \in G$.

For image retrieval, in order for a *keypoint detector* to be repeatable, it must be G-equivariant with respect to many transformations G. For example if a keypoint detector is rotation-equivariant then rotating an image and then detecting keypoints returns the same result as detecting the keypoints and then rotating them. Transformation equivariant keypoint detectors return invariant *descriptors*.

Keypoints can of course be selected by *random sampling* or *fixed grid sampling*. However, these options suffer from poor repeatability that is crucial in our task as described above. To that end more sophisticated approaches were introduced for *keypoint detectors* like the Harris (Harris and Stephens, 1988) or Hessian detector (Beaudet, 1978) that are gradient based keypoint detectors and rotation equivariant. They can both be extended to be equivariant under various other geometric transformations too.

Most image retrieval methods (including ours) use a Hessian-Affine detector (Mikolajczyk and Schmid, 2004) that is scale- and affine-equivariant. In this chapter we will describe the methods used in our particular implementation and refer the reader to Tuytelaars and Mikolajczyk (2008) for a thorough survey on keypoint detectors.

3.3 Local Keypoint Detection

As mentioned above our local feature extraction pipeline is based on the Hessian-Affine detector proposed by Mikolajczyk and Schmid (2004). Roughly a local feature extraction pipeline is composed of a *(i) keypoint detector* and a *(ii) local descriptor* around the detected keypoint that is used for matching keypoints along images.

The goal of the keypoint detector is to detect repeatable keypoints on the input images under various lighting conditions, noise, blur and various image transformations.

In the following paragraphs we will describe the steps of the Mikolajczyk and Schmid (2004) pipeline: from the initial Hessian Detector to the additional improvements that make it scale and affine invariant.

Hessian Detector. The Hessian-Affine detector (Mikolajczyk and Schmid, 2004) that we used in this work is a multiple scale algorithm. In each scale the algorithm detects keypoints based on the Hessian matrix. It detects keypoints (strong response) in corners and highly lectured areas.
Given an input image I we first compute the second order derivatives of the image:

$$\text{Hessian}(x, y) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{yx} & I_{yy} \end{bmatrix} \tag{3.1}$$

Since derivatives are sensitive to noise we smooth the derivative images by convolving with a Gaussian kernel of bandwidth σ.

$$Hessian(x, y, \sigma) = \begin{bmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{yx}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{bmatrix} = \begin{bmatrix} g(\sigma) * I_{xx} & g(\sigma) * I_{xy} \\ g(\sigma) * I_{yx} & g(\sigma) * I_{yy} \end{bmatrix} \tag{3.2}$$

The intuition is to search for strong derivatives in both directions. The keypoints are the local maxima (computed by *non-maximum suppression*) of the determinant of the Hessian matrix:

$$R_H = \sigma^2 \cdot \det(H) = \sigma^2 \cdot (L_{xx} \cdot L_{yy} - L_{xy}^2) \tag{3.3}$$

Characteristic Scale Selection. While the Hessian detector provides keypoints that are rotation equivariant and is robust to illumination changes and noise however the keypoints returned are not scale invariant. To achieve scale invariance i.e. having the same descriptor independent of the distance of the object to the camera is important since we would like to be able to match keypoints between images on different scales. To that end we need to include in each local neighborhood of a keypoint the same information regardless of the scale of the structure.

Selecting image regions in different scales for both images and then directly comparing all their descriptors until we find the best match is computationally expensive. Instead, we evaluate a *signature function* and plot the result value as a function of the neighborhood scale σ. This function needs to be invariant to the size of the keypoint's neighborhood region. The average intensity of an image region could be used as such a function (the average intensity should be the same regardless of the number of pixels in the region).

The characteristic scale of each image is then detected by searching for extrema of the signature function. If images I and I' have corresponding extrema σ and σ' then the scaling factor between the two images is $\frac{\sigma}{\sigma'}$.

In this work, we follow following Mikolajczyk and Schmid (2004) and use as signature function the Laplacian-of-Gaussians introduced by Lindeberg (1998) that has strong responses in blob like structures. The extended keypoint detector is called *Hessian-Laplace* detector and returns a localized keypoint along with a blob shaped keypoint neighborhood the size of the image's characteristic scale.

Affine Region Estimation. The keypoints detected with the procedures above provide us with features that are translation, rotation and scale invariant. The above Hessian-Laplace detector can be extended to detected keypoints that are covariant with respect to affine transformations. Mikolajczyk and Schmid (2004) obtain affine covariant keypoint neighborhoods using the algorithm by .

The process is iterative. We start with the initial Hessian-Laplace regions and in each iteration, we update the regions using the region's second-moment matrix:

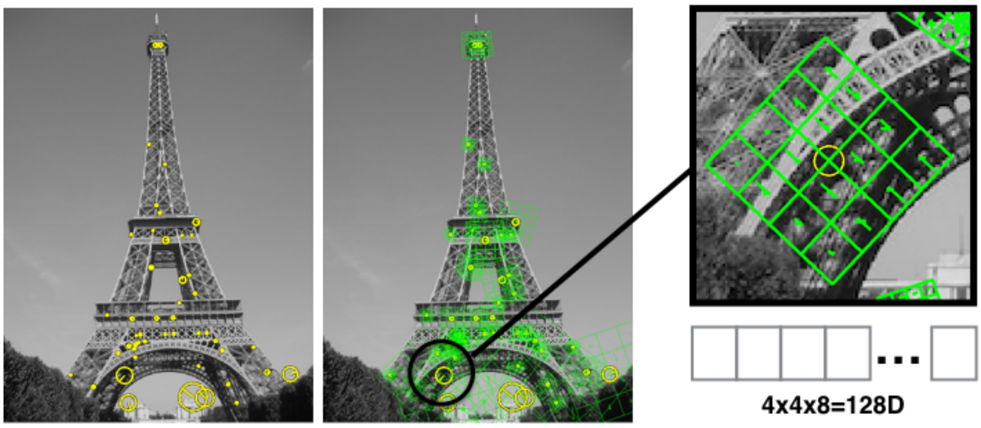
$$M(x, y, \sigma_D, \sigma_I) = \sigma_D^2 \cdot g(\sigma_I) * \begin{bmatrix} L_x^2(x, y, \sigma_D) & L_x(x, y, \sigma_D) \cdot L_y(x, y, \sigma_D) \\ L_x(x, y, \sigma_D) \cdot L_y(x, y, \sigma_D) & L_y^2(x, y, \sigma_D) \end{bmatrix} \tag{3.4}$$

where $L_x(x, y, \sigma) = g(\sigma) * I_x$ and $L_y(x, y, \sigma) = g(\sigma) * I_y$ are the derivative images convolved with a gaussian kernel g of bandwidth σ. We compute the eigenvalues of the matrix M. They represent an elliptical region that defines the affine deformation. We transform the ellipse into a circle and update the location and scale. This procedure stops when both eigenvalues of the second-moment matrix are equal.

The elliptical regions obtained by this process are containing the same information despite of viewpoint changes.

3.4 Local Descriptors

Once a set of keypoints is detected in an image a *keypoint descriptor* is used to describe a keypoint and thus facilitate matching using information from its surrounding neighborhood. In our work we use the SIFT (Lowe, 2004), a popular choice.



Keypoint Detection **LH-IS***

Figure 3.2: Local features example. Every image in an *image database* is described by a set of local keypoints and their corresponding descriptors (here SIFT).

Scale Invariant Feature Transform (SIFT) was introduced by Lowe (2004). It was part of a framework that included not only a feature descriptor but also a keypoint detector based on the Difference-of-Gaussians. Since the two components are independent of each other Mikolajczyk and Schmid (2004) combined the SIFT descriptor with the Hessian-Affine detector.

First a 4×4 grid is placed around a keypoint. Inside each grid location another $4 * 4$ positions are sampled and the gradients' magnitude and orientation is computed and accumulated in a histogram of gradient orientations of 8 bins. Each location is weighted by the gradient magnitude and a Gaussian function with a σ of half the region size. This way gradients further away from the center of the patch contribute less to the histogram. The entries of the histogram of each of the 4×4 coarse grid cells are concatenated to form a single $4 \times 4 \times 8 = 128$ dimensional feature vector.

3.5 Tasks

3.5.1 Image Retrieval (Part I)

Image retrieval is a well-researched topic in computer vision. It addresses the problem of searching an unordered *image database* in order to retrieve images that are similar to a new *query image* and rank them according to their relevance to the query image. For the task of landmark recognition, we need to retrieve those images containing the same object instance. This task belongs to the set of problems called *content-based image retrieval (CBIR)* or *instance retrieval* where the requirement is to find a particular object instance in a large image database (e.g.,

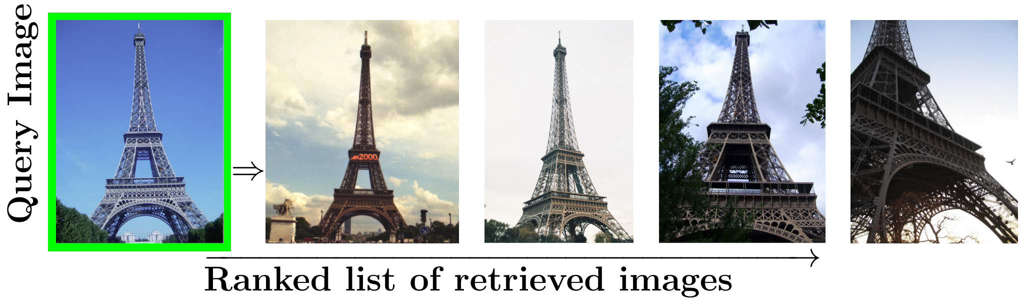


Figure 3.3: Image Retrieval example. Using a *query* image, images similar to the query are retrieved from an unordered *image database* and ranked according to their similarity with the query image.

all images portraying the Eiffel Tower). This task differs from the task of retrieving all images containing objects of a particular class (e.g., chairs). A visual example can be seen in Fig. 3.3.

Classical approaches (Philbin *et al.*, 2007; Sivic and Zisserman, 2003) rely on hand-crafted local features based on image gradients such as *Scale Invariant Feature Transform* (SIFT) (Lowe, 2004) to determine the similarity between the query image and the other images in the database. Direct comparison of every pair of descriptors is very precise for descriptor matching however it is too costly to perform in real-world retrieval systems that operate on millions of images. So the local features are aggregated into visual words using approximate nearest neighbors search methods like KD trees. Each image is then represented by a global feature vector based on a histogram of visual words computed by TD-IDF scoring. The major steps of the image retrieval pipeline can be seen in Fig. 3.4.

Our method implores for the image retrieval step the pipeline proposed by Sivic and Zisserman (2003), the state-of-the-art method for image retrieval at the time of

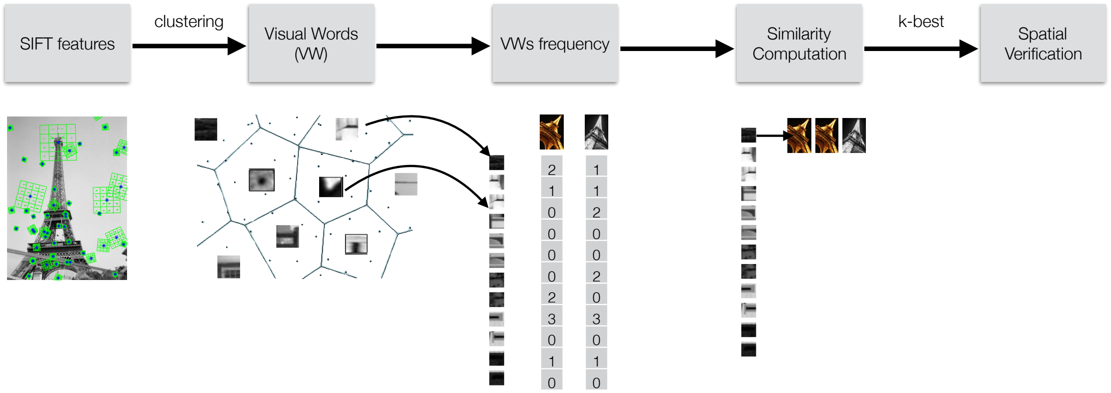


Figure 3.4: The steps of an image retrieval pipeline.

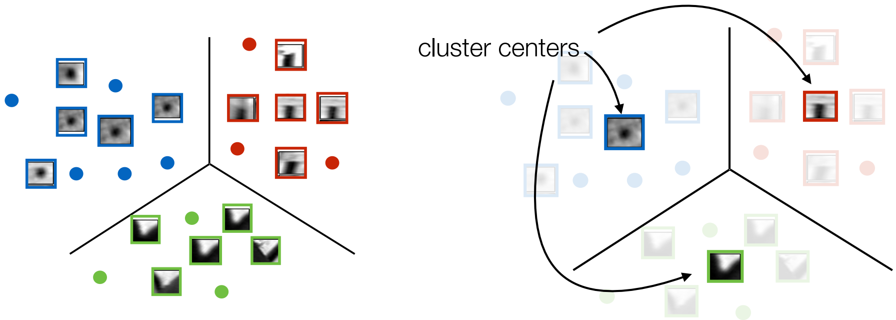


Figure 3.5: SIFT Features clustered into visual words.(*top*) SIFT descriptors that describe similar image patches are grouped into the same cluster. Here, circles (blue), squares (red) and corners (green) are parts of different clusters. (*bottom*) Each cluster is represented by its cluster center and defines a *visual word*. Now nearest neighbors correspondences can be found between visual words instead of SIFT features.

publication and thus uses hand-crafted local features (SIFT). However, our method is independent of the features used in image retrieval step and can be integrated in pipelines that use local or global CNN features.

In the following chapter we will introduce the image retrieval framework originally proposed by Sivic and Zisserman (2003).

Step 1. Local Feature Extraction. As described in Chapter 3.1 each image in the database is described by a set of keypoints with their corresponding descriptors (Fig. 3.2).

Step 2. Visual Words. The most straightforward way to find matching images would be to directly compare all features in all potential matching image pairs. However, this is impractical for most real-world application. Given an N images in the database, M number of SIFT descriptors that have $D = 128$ dimensions exhaustive linear search has processing complexity of $O(N \cdot M^2 \cdot D)$ and memory requirements of $O(N \cdot M \cdot D)$.

Building a visual word vocabulary. Instead Sivic and Zisserman (2003) proposed an efficient framework for image retrieval inspired by text retrieval. The 128-dimensional SIFT descriptors are clustered to representative *visual words*. SIFT descriptors that are supposed to be similar are mapped to the same cluster and thus the same visual word. The clustering is performed using k-means. The cluster centers represent the visual words, the set of visual words defines the *visual vocabulary* and a SIFT descriptor is mapped to the closest visual word (Fig. 3.5). Now each image is represented by a set of visual words instead of SIFT descriptors.

In particular each image I in the *image database* can be represented by a vector of visual word frequencies v_I. Each entry v_I^j represents the number of occurrences of the visual word j in the image I. Now the similarity between two images I_1 and I_2 can be computed between their corresponding visual word frequency vectors.

When a new *query image* appears each of its SIFT descriptors is assigned to the *nearest* cluster (visual word) creating a vector of visual word frequencies and this is used to generate matches with the other images in the database.

The step of building the visual word vocabulary is performed offline before the actual retrieval takes place. The clustered SIFT descriptors and the corresponding cluster centers are extracted from a separate database than the one we are using to retrieve images.

Approximate Nearest Neighbors. Every SIFT descriptor of an image is assigned to a visual word from the vocabulary described above. This requires finding the cluster center that is closest to the SIFT descriptor. Searching for an exact nearest neighbor in the high dimensional space of the SIFT descriptors (128-D) is computationally expensive. Thus, we use an approximate nearest neighbor approach based on k-d trees proposed by Philbin *et al.* (2007) that is more efficient but cannot guarantee finding the actual nearest neighbor. A k-d tree is a binary tree of depth $log_2 N$. Each node of the tree splits the data into two parts called half spaces. One half belongs to the left branch and the other to the right. Typically, the splitting decision is made at the median of the dimension of the highest variance. We stop splitting after reaching a stop criterion. Philbin *et al.* (2007) use 8 random k-d trees where the splitting decision in each level is made at a random point close to the median of a dimension randomly selected from a pool of dimensions with high variance providing robustness for high dimensional cases. For image retrieval a k-d tree is constructed using the cluster centers as data.

Nearest neighbor queries are evaluated by traversing the tree choosing always the half space that contains the data point. When we reach a leaf node, we compute the distance with each point contained in the leaf node and pick the closest as the nearest neighbor to the query. Of course, if the tree construction has not been stopped early there is only one data point in the leaf and this is considered the nearest neighbor. However, this is not necessarily the exact nearest neighbor. The real nearest neighbor could be in neighboring splitting planes (Fig. 3.6). In high dimensional spaces the number of neighboring splitting planes to be explored can be significantly large.

So in Philbin *et al.* (2007)'s randomized case while each tree is traversed the distances to the discriminating boundaries are recorded in a single priority queue for all trees. The most promising alternative cells are explored based on the distances

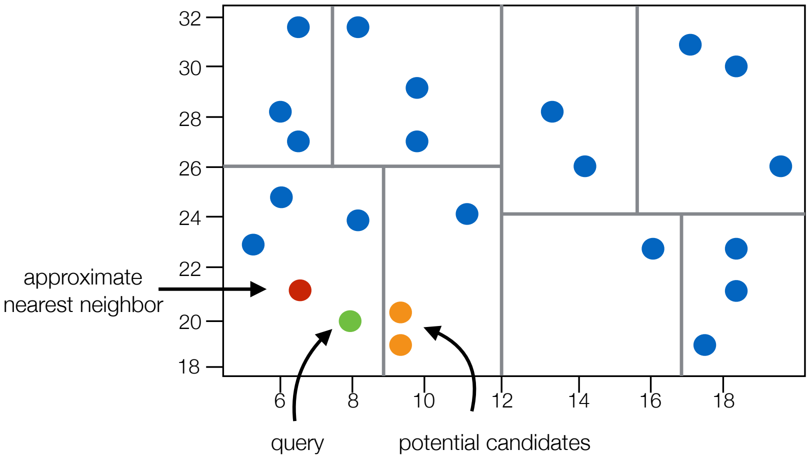


Figure 3.6: Example of nearest neighbor search using a k-d tree. We initially search for the nearest neighbor in the cell containing the query (green point) and pick as nearest neighbor the closest point to the query inside the cell (red point). However, the true nearest neighbor can be in the other side of the splitting plane (orange points).

from the query to the splitting plane. After a certain number of cells has been tested for better nearest neighbor candidates we stop.

Image Similarity. The similarity between two images could be directly computed as the dot product of their respective frequency vectors. However Sivic and Zisserman (2003) suggest an alternative to simple frequency terms, the *term frequency - inverse document frequency (tf-idf)* score that is computed as follows: Each image I is represented by a k-dimensional vector $v_I = (v_I^1, v_I^2 \cdots, v_I^k)$ of

$$v_I^j = \underbrace{\frac{n_{jI}}{n_I}}_{\text{tf}} \cdot \underbrace{\log \frac{N}{n_j}}_{\text{idf}} \tag{3.5}$$

where n_{jI} the number of occurrences of visual word j in image I, n_I the number of visual words in image I, n_j the number of times the visual word j occurs in the whole image database and N the total number of images. If a visual word appears often in an image the tf term increases but if it is too common a word and appears too often in the database the idf term downgrades its score.

Given two vectors v_{I1} and v_{I2} representing two images I_1 and I_2 this similarity is computed as:

$$sim(v_{I1}, v_{I2}) = \frac{v_{I1} \cdot v_{I2}}{||v_{I1}|| \cdot ||v_{I2}||} \tag{3.6}$$

Step 3. Inverted File Index. The *tf* term is sparse and is wasting memory to store it for every image in the database. So, we build a more memory efficient data structure called an *inverted file index* following Sivic and Zisserman (2003). For each visual word we save a list of the images containing it and pre-compute the *idf* score of each visual word and the *tf-idf* score for each image in the database. To retrieve images similar to the query only the images with common words query are considered as potential matches. So the inverted file index is not only efficient in memory but also in computational time since the index is traversed only for the visual words existing in the query image.

Step 4. Spatial Verification. After computing a similarity score between the query image and all images that share at least one visual word with the query the k best matches are verified matching the individual SIFT features, estimate a homography transformation with RANSAC using the feature matches. If a certain number of inliers is found between the query and an image, then the pair is considered a match.

A *homography* is a transformation model between two images. A transformation maps each pixel in one image to a corresponding location in the second (*correspondences*). The type of transformation one is trying to estimate depends on the camera motion and the object geometry. Homography estimation assumes a mapping between two perspective projections of a planar object. In homography straight lines are projected as straight lines although parallel lines are not necessarily preserved.

A homography matrix estimating the transformation between two planes is a 3×3 matrix H:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{3.7}$$

H is 3×3 matrix with $h_{33} = 1$. It has 8 DoF and can be estimated by at least 4 correspondences of 2D points. A homography transformation between two points is computed according to:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.8}$$

It is a system of linear equations that can be solved by singular value decomposition (SVD). More detailed information regarding the estimation of a homography along with other transformations is available in Hartley and Zisserman (2003).

Good homography estimation requires accurate point matches in both images but which correspondences are good is determined by the estimated homography. A solution to the problem is the RANSAC algorithm by Fischler and Bolles (1981). RANSAC is an iterative algorithm. In each iteration a random number of correspondences is drawn (4 in our case for homography), estimates the homography transformation and then checks how many of the total correspondences are consistent with the transformation (*inliers*). The algorithm stops when a sufficient number of inliers exists. The final transformation is estimated from all inliers. There are several variations of the RANSAC algorithm, here we employ the Spatially Consistent Random Sample Consensus (SCRAMSAC, Sattler *et al.* (2009))

3.5.2 Matching Graph Construction and Clustering (Part I)

A crucial step in landmark discovery is the building of a *matching graph* (Avrithis *et al.*, 2010; Gammeter *et al.*, 2009; Quack *et al.*, 2008). In the matching graph the nodes are representing images and the edges represent a match between images. The matching graph is a directed graph, image I_1 could be a match for image I_2 but not vice versa. The graph can also have weighted edges that represent the similarity between the two images. The matching graph is constructed by using each image in the database as a query and retrieving all possible matches using the image retrieval algorithm described above. Each match is used now as a follow up query. The construction of the matching graph is usually followed by a clustering step like hierarchical agglomerative clustering (Gammeter *et al.*, 2009) that refines the retrieval results and offers insights using the graph structure.

3.6 Datasets

Paris 500k. by Weyand *et al.* (2010) is a dataset that consists of 501356 geotagged images, "naturally" distributed over the center of Paris. The images were collected from Flickr and Panoramio. The collection used a geographic bounding box instead of keyword queries. The dataset is considered challenging due to the presence of duplicates and near-duplicates, as well as a large fraction of distractor images (images unrelated to the task of landmark retrieval such as photos of everyday objects, pets etc.) For more details regarding the construction and characteristics of the dataset please see Weyand *et al.* (2010).

4

Deep Learning Preliminaries

Deep learning is a machine learning subfield that relies on a deep hierarchy of learned representations. These representations are learned from training data with the use of an optimization algorithm between the algorithm's predictions and the gathered ahead objective ground truth data. Other machine learning approaches like the one described on the previous chapters (Chapter 5) rely on handcrafted input features and the optimization process does not involve any ground truth information but instead focuses on optimizing internal metrics to measure the relative quality of different algorithms.

Deep learning has become the standard approach to many computer vision tasks. In the following chapter we will present our novel algorithms for the tasks of *semantic segmentation* on 3D point clouds and *interactive object segmentation* on 2D images.

Our suggested approaches make use of many different deep learning algorithms and components. In this chapter (Chapter 4) we will introduce the basic deep learning concepts that are required for the tasks of semantic segmentation on 3D point clouds and 2D images and their key differences. After, we will introduce briefly the goals of each task and the datasets used to prove the performance gain with our suggested approaches. For a detailed introduction on deep learning theory and methods please see also Goodfellow *et al.* (2016).

4.1 Feedforward Neural Networks

4.1.1 Perceptron

The *perceptron* (Rosenblatt, 1958) is a classification algorithm that decides if an input data point belongs to a specific class or not, based on the following equation:

$$y = \sigma(\sum_{i=1}^{D} x_i \cdot w_i + b) \tag{4.1}$$

As it can be seen also in Fig. 4.1a it consists of the following components: (1) a D dimensional *input* vector: $x \in R^D$, (2) the corresponding learnable *weights* (for each input entry) and *bias*: $w \in R^D, b \in R$, (3) a *summation* function and (4) an *activation* function: σ.

The inputs are multiplied with the corresponding weight values and then are added together. A non-linear *activation* function is then applied to the weighted sum producing the final output $y \in [0, 1]$. The learned weights represent the importance of each input entry and the activation function maps the weighted sum output to the required values for binary classification. The weights and bias are learned during the *learning phase* (Sec. 4.1.3).

A perceptron can be extended from binary to multi-class classification using a *neural network* (4.1b) of perceptrons/neurons with one output node per class:

$$y_k = \sigma(\sum_{i=1}^{D} x_i \cdot w_{ki} + b_k) \tag{4.2}$$

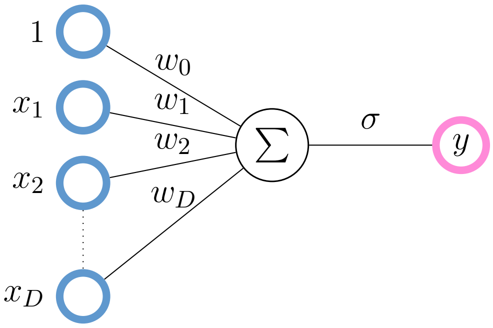
Perceptrons can be seen as a *single layer* neural network also known as *fully connected layer* (FC), since all of the inputs are connected to all the outputs. In matrix form defined as:

$$\mathbf{y} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b}) \tag{4.3}$$

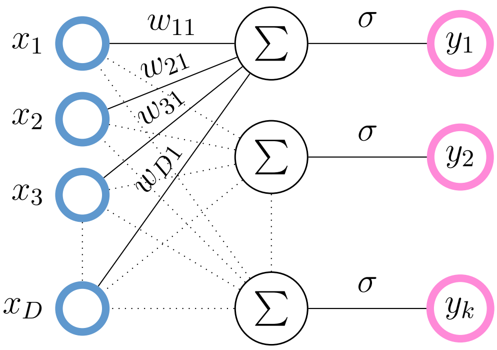
where $\mathbf{x} \in R^D$ is the input in the form of a vector, $\mathbf{W} \in R^{D\times M}$ the learnable weights matrix, $\mathbf{b} \in R^M$ the learnable bias term and $\mathbf{y} \in R^M$ the output to which the input $\mathbf{x}$ has been transformed. Each y_i is a perceptron unit referred to as neuron and a set of connected neurons create a *neural network*.

4.1.2 Multilayer Perceptron

A single layer neural network corresponds only to generalized linear discriminants and is thus very limited. The solution is the multilayer perceptron that is composed



(a) A perceptron



(b) A multi-class perceptron

of multiple single layers stacked on top of each other where the output of the previous layer $l-1$ is the input for the next layer l.

Multilayer perceptrons are called universal approximators because during learning they can approximate any function $f_\theta : y = f_\theta(x)$ which maps a D-dimensional input $x \in R^D$ to a k-dimensional output $y \in R^k$. With θ we represent the parameters of the network that we learn during the optimization process and correspond to a good representation of the task at hand.

All layers except the input and output are called hidden layers since their values are not observable outside of the network.

A network with L layers is represented as:

$$f_\theta(x) = f^L_{\theta_L}(f^{L-1}_{\theta_{L-1}}(\cdots f^1_{\theta_1}(x))) \tag{4.4}$$

Fully connected layers. Each layer l is a multiple output perceptron (Eq. 4.3). Using matrix notation:

$$y^l = \sigma^l(z^l) = \sigma^l(W^l \cdot y^{l-1} + b^l) \tag{4.5}$$

where $y^{l-1} \in R^{D^{l-1}}$ is the input to the current layer l that also corresponds to the output of the output of previous layer $l-1$, $W^l \in R^{D^{l-1} \times D^l}$ is the weight matrix and $b^l \in R^{D^l}$ is the bias vector, σ^l is the activation function and $y^l \in R^{D^l}$ the output of layer l.

A network that consists of only fully connected layers is called a *multilayer perceptron* (MLP).

Activation functions provide the non-linearity to the whole network. Without them the whole multilayer perceptron output would have been a linear function of the input. A popular choice is the rectified linear unit (ReLU) introduced by Glorot *et al.* (2011):

$$\sigma^{ReLU}(z_i) = max\{0, z_i\} \tag{4.6}$$

The sigmoid or tanh function are other popular choices. The activation function of the final layer L enforces the output to be represented as a probability distribution over usually k number of classes in the case of classification.

The activation function σ^L is usually a softmax function given by:

$$\sigma^{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \forall j \in k \tag{4.7}$$

All output values are between 0 and 1 and their sum equal to 1 like a proper probability distribution $p(k|x)$ of the class k given input x.

4.1.3 Training

In a feedforward network like MLPs we input an input vector x and produce an output y. This is usually called the *forward propagation.* During training the network f_θ we need to define a loss function between the network output y and the ground truth $\tilde{y}$.

For classification tasks, it is common the use of a cross entropy loss E^{CE}.

$$E^{CE}(\theta) = -\sum_k \tilde{y}_k \log(y_k) \tag{4.8}$$

where $\tilde{y}_k$ is the ground truth class, and $\log(y_k)$ is the softmax probability for class k.

Most deep learning algorithms have some optimization procedure. Optimization is maximizing or minimizing a function. In our case we are minimizing the loss function.

These types of optimization processes that minimize a loss function requires the use of prediction-ground truth pairs (training examples) and is called *unsupervised learning.* The success of the learning algorithm is measured though the *generalization error* between the prediction and the ground truth of unseen examples. All algorithms presented in Part II are fully supervised.

In *semi-supervised learning* a small portion of the data can be used for training. The method presented in Part III belongs to the semi-supervised methods since only a few pixels per image have ground-truth labels.

In contrast to all the above, the algorithm presented in Part I belongs to *unsupervised methods* where the algorithm discovers patterns in data without the use of any ground truth information to guide the optimization process.

Gradient Descent. Given a function $y = f(x)$ with $x, y \in R$ and $f'(x)$ the derivative of the function. A small change ϵ in the input results in a corresponding change in the output $f(x+\epsilon) \approx f(x) + \epsilon \cdot f'(x)$. We can then reduce the value of function $f(x)$ by moving in small steps towards the opposite direction of the gradient $f(x = \epsilon \cdot sign(f'(x)))$.

In our deep learning case after the loss E is computed, we update the parameters θ in the direction of the steepest descent. One update step of the parameters is given by:

$$\theta^{k+1} = \theta^k - \lambda \frac{\partial E}{\partial \theta}\bigg|_{\theta=\theta^k} \tag{4.9}$$

where λ is a hyper-parameter that has to be chosen carefully. More advanced optimization techniques like Adam automatically choose the learning rate.

Traditional gradient descent assumes the computation of the loss function is happening overall the data points in the training set. A common problem in machine learning is the big size of the training set. *Stochastic gradient descent* is an extension of the original gradient descent algorithm where in each iteration only one data point is used to compute the loss function. Alternative we can sample a relatively small *minibatch* of examples from the training set to compute the sum of the loss as a compromise between standard gradient descent and stochastic gradient descent.

4.1.4 Backpropagation

Computing the gradients $\frac{\partial E}{\partial \theta_i}$ of the loss function E with respect to all parameters θ from Eq. 4.9 requires the efficient backprogation algorithm (Rumelhart *et al.*, 1988). We will explain the basic back propagation steps with respect to the weighted input z_i^l instead of the activation output for simplicity reasons.

It is based on the chain rule of differentiation and requires two passes: the forward and the backward. During the forward pass all neutron outputs are computed with respect to its input: $z^l = \sigma^l(w^l \cdot z^{l-1} + b^l)$. During the backward pass we propagate the error from the last layer to the first one: $\frac{\partial E}{\partial z^{l-1}} = ((w^{l+1})^T \cdot \frac{\partial E}{\partial z^{l+1}}) \odot \sigma^{l\prime}(z^l)$.

4.2 Convolutional Neural Networks

Neural networks based on layers of fully connected layers (MLPs) have two disadvantages: they require a large number of parameters and are sensitive to small shifts of the input data. These drawbacks are more prominent when we consider a common input to most computer vision tasks: an RGB image of size 640 × 460 pixels. Flattening such an image results in a $640 \times 460 \times 3 \approx 890K$ input entries.

Using a hidden layer of 1024 neutrons results in approximately 900M parameters on a single layer. Additionally, different weights are learned and associated with different parts of the image producing different activations for the same object depending its position in an image.

Convolutional Neural Networks (CNNs) (Lecun, 1989) alleviate these issues by replacing one or more layers of a network with a *convolutional layer*. In a convolutional layer, instead of connecting all its units to all the units in the preceding layer, the weighted sums are computed within a small local window. Additionally, weights are identical regardless of the location of the pixel. Numerous architectures appeared since Krizhevsky *et al.* (2012b) used CNNs to win the ImageNet Large Scale Visual Recognition Challenge outperforming all other methods by a large margin. His seminal work started a new era in computer vision and now CNNs are used to solve most perception tasks. Most CNN architectures use the basic building blocks described in this chapter.

4.2.1 The Convolutional Operator

Convolution is a mathematical operator between two functions f and g denoted as $f * g$. The convolution of two real multivariate functions $f, g : R^D \to R^M$ is defined as:

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(y)g(x-y)dy \tag{4.10}$$

It is computed as the integral of the product of two functions where one of them is shifted and reversed. In the field of machine learning $f(x)$ represents the *input* and $g(x-y)$ the *kernel* of learnable weights. The convolution represents the overlap between function f and kernel g.

4.2.2 The Discrete Convolutional Operator

Usually in machine learning and especially in computer vision the input is not continuous and it can only take discrete values. We can then define the *discrete convolution* as:

$$(f * g)(x) = \sum_{-\infty}^{+\infty} f(y)g(x-y) \tag{4.11}$$

If the function f ranges over a finite set of values N (e.g., number of pixels in an image) then the discrete convolutional operator can be represented as:

$$(f * g)(x) = \sum_{N} f(y)g(x-y) \tag{4.12}$$

In particular when the input data is two-dimensional (e.g., an image) then we require a two-dimensional kernel of size $n \times m$ that represents a local neighborhood of size $m \times n$ around each pixel (x, y). Then the convolution in the two-dimensional discretized, finite space is defined as:

$$(\mathbf{F} * \mathbf{G})(x, y) = \sum_{m} \sum_{n} \mathbf{F}(m, n)\mathbf{G}(x-m, y-n) \tag{4.13}$$

Where $\mathbf{F}$ represents usually the 2D image and $\mathbf{G}$ the two-dimensional kernel function.

4.2.3 Convolutional Layer

The *convolutional layer* is a common building block of modern deep networks (CNNs) that take as input two-dimensional images. The input of every convolutional layer then is not a flattened vector like in a fully connected layer but a tensor of $H \times W \times D$ where $H \times W$ are the spatial dimensions of the 2D input and D the number of feature channels: in the first layer usually D=3 for each of the image color channels. The layer outputs a tensor of the same spatial dimensions $H \times W$ as the input and M number of output feature channels.

$$\begin{aligned}\mathbf{Y}^l &= \sigma^l(\mathbf{Y}^{l-1} * \mathbf{W}^l + \mathbf{B}^l) \\ &= \sigma^l(\mathbf{Z}^l + \mathbf{B}^l)\end{aligned} \tag{4.14}$$

where $\mathbf{Y}^{l-1} \in R^{W \times H \times D}$ the input (which is usually the output of the previous layer), $\mathbf{Y}^l \in R^{W \times H \times M}$ the output of this layer, $\mathbf{W}^l \in R^{K \times K \times D \times M}$ the learnable filters of kernel size K and $\mathbf{B}^l \in R^{D \times M}$ the bias. The convolutional operation is represented with * and σ the non-linear activation function for layer l.

The x, y position in the $i_{th} \in \{1, 2 \cdots M\}$ output feature map $\mathbf{Z}_i^l(x, y)$ is computed as:

$$\begin{aligned}\mathbf{Z}_i^l(x, y) &= (\mathbf{Y}^{l-1} * \mathbf{W}_i^l)(x, y) \\ &= \sum_{d=1}^{D} \sum_{m=-K/2}^{K/2} \sum_{n=-K/2}^{K/2} Y^{l-1}(x+m, y+n, d) \cdot W_i(m, n, d)\end{aligned} \tag{4.15}$$

Even though the layer is called a convolutional layer, in practice it is usually implemented as a *cross-correlation* operation instead of a convolution as shown in Eq. 4.15. The limitation of the filter kernel at the borders of an image is usually dealt by zero-padding the image borders before applying the filter.

As it can be seen from Eq. 4.15 convolutional operations are equivariant with respect to translation. In practice that means the convolutional operator output is translated if the input is translated. Convolutional operators however are not translation invariant. For a formal explanation of equivariance and invariance please see Chapter 3.2.

A convolutional layer decreases the number of needed parameters to learn by sharing them across the image spatial positions. However, in contrast to a fully connected layer neuron that incorporates information from all input positions now each convolutional layer neuron "sees" only a limited part of the input equal to the size of its kernel $K \times K$. This is usually referred to as the *receptive field*. CNNs solve this problem by stacking multiple layers of convolutional layers each one increasing the receptive field of each neuron further. We would like to increase the receptive field to the size of an image. However, this requires many layers of convolution that not only increase the number of parameters needed to learn but also reduce the gradient values making the network more difficult to train.

4.2.4 Depthwise Separable Convolutions

Given an input feature map with input channel dimension D and output channel dimension M and a kernel of $K \times K \times D$ the convolution process results in $K \times K \times D$ multiplications every time one of the M kernel moves and $K \times K \times D \times M$ parameters.

Both the time number of parameters and computational time can be significantly reduced with the use of *Depthwise Separable Convolutions*(Chollet, 2017). The depthwise separable convolution consists of two separate operations: the depthwise convolution and a pointwise convolution.

In the first part, the depthwise convolution D different $K \times K$ kernels are used. Each kernel produces one feature map and then the D feature maps are concatenated together. The depthwise convolution step requires $K \times K \times D$ parameters and $K \times K \times D \times H \times W$ multiplications.

In the second part, the pointwise convolution a 1×1 convolutions is used that iterates over each point, it allows interactions among the channels and transforms the D dimensional channel to an M dimensional one. It requires $D \times M$ parameters. Thus, the depthwise separable convolution needs $K \times K \times D + D \times M$ parameters in total, which are much less than the $K \times K \times D \times M$ parameters needed for the standard convolution. It also requires $D \times M \times H \times W$ multiplications making it in total $(K \times K \underline{+M}) \times M \times H \times W$ compared to the $(K \times K \underline{\times M}) \times D \times H \times W$ of the standard convolution.

4.2.5 Atrous (Dilated) Convolutions

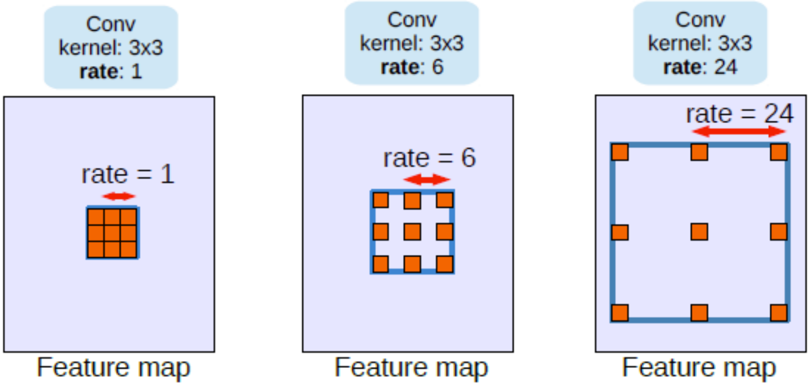


Figure 4.2: Atrous Convolutions with different atrous rates. The standard convolution (*left*) represented as an atrous convolution with atrous rate equal to 1 uses a dense filter with learnable parameters in each position of the grid. Atrous convolutions (*middle, right*) replace filter locations in both dimensions with zeros based on the atrous rate. Source (Chen *et al.*, 2018b)

In standard convolutions the size of the kernel K represents the size of the pixel's neighborhood information that influences the value of each pixel and is directly connected to the *receptive field* of the layer. We can increase the size of the receptive field by either increasing the size of K or by stacking multiple convolutional layers of kernel size K. Two layers with kernel size 3×3 have the same receptive field as a 7×7. However, in both case the increase of the receptive field comes at the cost of more parameters to learn.

Chen *et al.* (2016) and Yu and Koltun (2016) proposed *atrous convolutions* in order to increase the receptive field of the convolutional layer without increasing the number of learnable parameters. In atrous convolutions the filter is not applied to all neighboring points densely but instead its learnable parameters are arranged in a sparse grid where we insert zeros among the filter elements (Fig. 4.2).

The number of zeros inserted is defined by the atrous rate. If the atrous rate equal to 1 the atrous convolution is equivalent to the standard convolution. If the atrous rate is equal to 2 then the two learnable parameters are two points apart in each direction. Since the skipped locations in the filter equal to zero an atrous filter with kernel 7×7 and atrous rate 3 has the same number of learnable parameters as a 3×3 traditional filter but it incorporates spatial context from locations as far as a standard 7×7 filter.

4.2.6 Pooling/Unpooling Layers

An important component of CNNs is the pooling layers. Max-pooling layers are a common variant and have no learnable parameters. Pooling operators divide the spatial space in block of $p_h \times p_h$ and in the max-pooling case take the maximum of all the values in the region. There is a pooling operator for each separate feature channel. The grid of blocks operates in a stride the size of pooling region's height and width thus reducing the spatial dimension of the input. For example, a typical max-pooling operator of kernel size 2 and stride 2 divides the spatial dimensions of the input tensor $\mathbf{X} \in R^{H \times W \times D}$ in blocks with height and width equal to 2. It then takes the maximum value in each 2 block and outputs a tensor with both the spatial dimension halved $\mathbf{Y} \in R^{H/2 \times W/2 \times D}$

Pooling operations increase the receptive field of the network without the use of additional parameters. Following convolutional layers require less parameters due to the decreased spatial size. The drawback is that the output has reduced spatial dimension. This can be useful when we would like to produce a global representation of the whole image that through a fully connected layer can make a decision regarding the presence or not of a class (e.g., *cat*) in the image. However, when we would like to make a decision for each one of the pixels in an image (e.g., semantic segmentation) we would like to return the feature maps to the original spatial size.

Unpooling layers increase the spatial dimensions of a feature map using upsampling techniques like interpolation or nearest-neighbors.

Typical CNNs are composed by multiple layers (this is why they are called deep models also) of convolutions stacked one after the other alternating with max-pooling layers. This is usually called the *encoder* part of the network. If the task requires the original spatial dimensions several unpooling layers follow, this is called the *decoder* part of the network.

4.3 Convolutions in 3D space

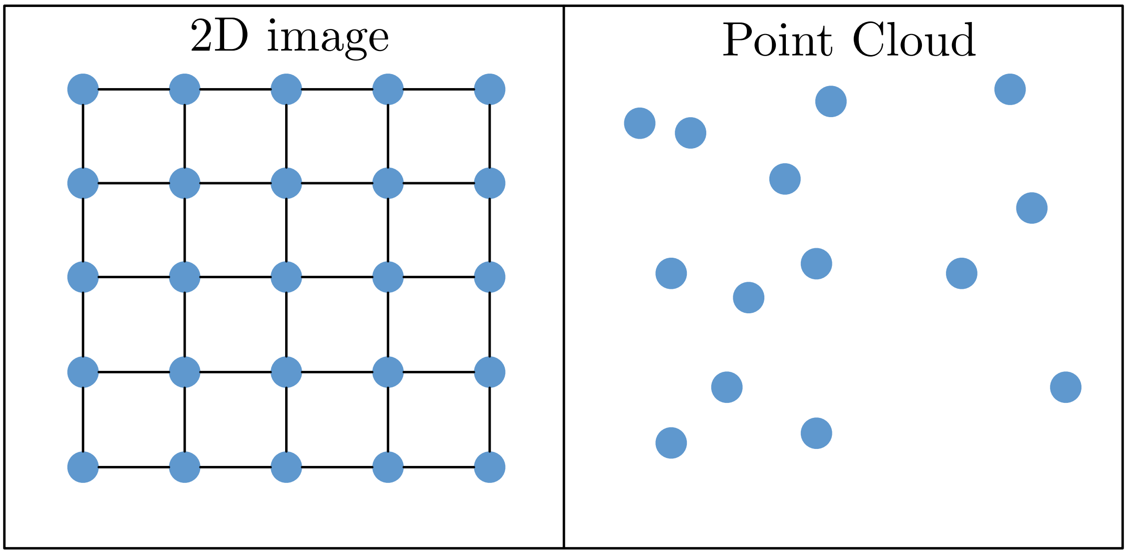


Figure 4.3: Image Grid vs Point Cloud. On the left side we see a typical two-dimensional grid representing the pixels in an image. The relative positions of points in such a grid are fixed. This results on fixed distances between points and fixed neighborhoods. On the other hand, on the right side, we see a point cloud representation as a set of points. The coordinates of each point are not located on a fixed grid and can take any continuous value.

Digital images are one way of representing the 3D world around us by projecting the 3D space into the image plane. Images are a discrete and finite representation of data where each point in the image has a spatial neighborhood based on a two-dimensional grid space that is predefined by the digital camera sensor. The feedforward and convolutional layers described above are well suited to model data with an underlined grid structure (e.g., 1D vectors, 2D images etc.).

However, most sensors that provide a direct representation of the 3D space return 3D data that are not ruled by any underlying grid structure. Depth sensors return an unstructured set of 3D points (see Fig. 4.3).

Initial works (Qi *et al.*, 2017b) that operate on raw 3D data they have no convolutional layers. They do not have fully connected layers either since they do not provide permutation invariance, a critical property for 3D point clouds. Context from neighboring points is distributed only in the form of a global max-pooling layer over all points:

$$\max_{x_i \in N_x} \{f(x_i)\} \tag{4.16}$$

where in the case of Qi *et al.* (2017b) N_x is the whole point cloud.

The follow-up work of Qi *et al.* (2017a) replaces the global neighborhood of Eq. 4.16 with spherical neighborhoods N_i of various radius but has not defined any convolutional layers with learned finite kernels.

Most works that directly work with 3D data representations are focusing on defining the neighborhood N_x for every point x. Without a support domain the definition of a convolution in 3D space is not possible.

A general 3D point convolution at a point $x \in R^3$ based on Monte-Carlo integration with sampling the continuous space can be defined as:

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(y)g(x-y)dy \approx \frac{1}{N} \sum_{x_i \in N_x} f(x_i)g(x-x_i) \tag{4.17}$$

where x_i are points in the set of 3D points P representing a 3D scene $P \in R^{N \times 3}$ and f_i its corresponding feature from $F \in R^{N \times D}$. $N_x = \{x_1, x_2 \cdots x_N\}$ defined the neighborhood of point x.

An implementation of Eq. 4.17 can be obtained by modelling the continuous kernel function g using multi-layer perceptrons (MLP) that are continuous function approximators:

$$g(x - x_i, \theta) = MLP(x - x_i, \theta) \tag{4.18}$$

The above approximation is used by many approaches (Wang *et al.*, 2018b, 2019; Wu *et al.*, 2019c)

4.4 Residual Connections

Neural networks are universal function approximators. This means that they can approximate any function given enough layers and units. The common understanding in computer vision community was that deeper and deeper networks will result in increased accuracy. He *et al.* (2016a) realized that this is true up to a certain depth and after a certain number of layers accuracy starts to saturate and the error rate increases both in training and test set.

He *et al.* (2016a) proposed residual connections as a solution. In classic neural networks the output of a layer l is fed into the following layer $l+1$. As it can be

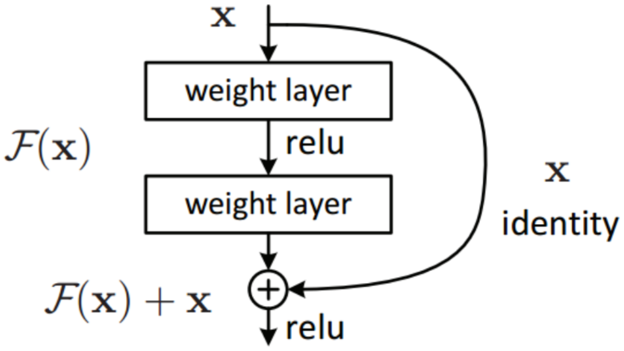

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4.4: Residual Block. Source (Chen *et al.*, 2018b)

seen in Fig. 4.4 in residual networks the output of a layer l is directly added to the input of layers further away $l+2, l+3\cdots$. Another way to think it is that the input to layer l is added to the output of layer $l+1$.

Formally that means given a block of network layers $\{l-n, l-n+1, l-n+2, \cdots, l\}$ representing a function $F(x)$ where x is the input to layer $l-n$ then the output of layer l is modelled by:

$$H(x) = F(x) + x \tag{4.19}$$

This forces the block of layers to learn the *residual*(difference) information between the input and the output than directly learning a desired output. According to He *et al.* (2016a) it is easier to learn the residual $F(x)$ instead of directly learning $H(x)$. And as an added advantage adding more layers does not hurt the performance of the network since the layer can always default to the identity function if this is considered the optimal route by setting the residual to zero. This way the network is allowed to choose the number of layers to train by skipping the layers that do not contribute to the network's performance. Networks constructed with these types of connections are called *residual networks* or *ResNets*.

ResNet-50 and ResNet-101 with 50 and 101 hidden layers respectively are popular backbone architectures that have been widely adopted my many follow-up works. The idea was successfully applied to other architectures also.

The Xception architecture used in Xception-65 (Chollet, 2017) that is part of the DeepLabV3+ framework (Chen *et al.*, 2018b) (Chapter 4.6.1) used in Part III of the thesis, is similar to the ResNet architecture where the standard convolution are replaced with depthwise separable convolutions (Chapter 4.2.4).

4.5 Recurrent Networks

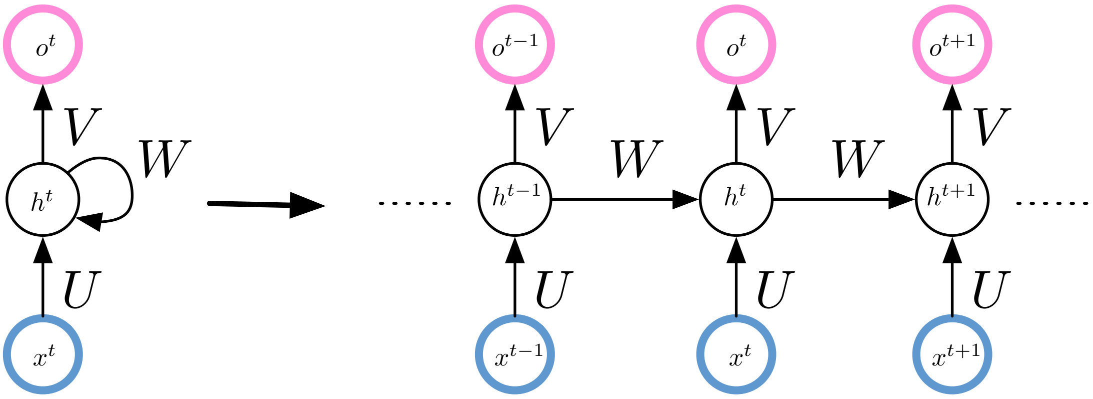


Figure 4.5: Unrolled recurrent neural network. A recurrent neural network that maps a sequence of input x values to a corresponding output sequence o. The model representation shown in the figure above is called a *computational graph*, where each node corresponds to a mathematical operation (*left*). The forward pass of the model presented in this computational graph is described in the equations 4.20 and 4.21 (*right*). An RNN can be seen as a regular neural network in its unrolled state where each time step is represented by its own node in the computational graph.

So far we have described fully connected layers that model successfully vector data, 2D convolutional layers for image data and 3D convolutional layers that operate on unstructured 3D point data. However, none of the above are suited to model sequential data (e.g., time series, videos, audio).

Recurrent Neural Networks (RNNs)(Rumelhart *et al.*, 1986) share layers across time and are commonly used for this purpose. The input to an RNN is a sequence of vectors of length T: $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^T$ where $x^t \in R^D, t \in \{1, 2, \cdots, T\}$.

A hidden layer at time step t:

$$h^t = \sigma(\mathbf{W}\mathbf{h}^{t-1} + \mathbf{U}\mathbf{x}^t + \mathbf{b}) \tag{4.20}$$

where $\mathbf{x}^t$ is the input at time step t, $\mathbf{h}^{t-1}$ the hidden state of previous time step $t-1$, $\mathbf{W}$ the weight matrix of the hidden-to-hidden connections, $\mathbf{W}$ the weight matrix of the input-to-hidden connections and σ the activation function.

The output layer at time step t:

$$o^t = \sigma(\mathbf{V}\mathbf{h}^t + \mathbf{c}) \tag{4.21}$$

where $\mathbf{V}$ the weight matrix of the hidden-to-output connections and $\mathbf{c}$ the bias term.

Some RNN versions have only a new hidden state without a new output at every time step. In general the above equations are computed for each time step $t \in \{1, 2, \cdots, T\}$ for an RNN that outputs a sequence of the same length as the input.

Backpropagation through time (BPTT). RNN gradients are computed using the Backpropagation through time algorithm. BPTT computes the forward and backward pass in the unrolled computational graph, computing the gradients not only backwards to the previous layers but also to the previous time steps.

Long-term dependencies. RNNs are designed to remember and use information presented in the network in the past (the past does not refer only to time it might actually represent a previously seen part of a sequence). However, in practice they suffer from the *vanishing gradients* problem that makes it difficult to handle ”long-term dependencies” and remember information for a long period of time. The gradient multiplication in each RNN time step results diminishes the overall gradient value. Longer sequences yield increasingly small gradients making the training procedure slower. *Long Short-Term Memory* (LSTMs) (Hochreiter and Schmidhuber, 1997) units have been designed to correct these issues. The LSTMs extend the RNN architecture with the use an additional hidden state. Gates add or remove selectively information from the hidden state. The gates are composed by point wise multiplication with a sigmoid layer where its zero to one value range represents how much of that piece of information will be retained.

4.6 Tasks

In the following chapters we propose methods to improve on several computer vision tasks. In the following chapters we focus on deep learning algorithm that solve the task of semantic segmentation through fully and semi-supervised methods.

4.6.1 Semantic Segmentation (Part II)

A standard task in computer vision is *semantic segmentation*. Semantic segmentation can be applied in 2D images or 3D point clouds. The task consists of assigning a semantic label to every 3D point (or pixel) in a point cloud (or image) as it can

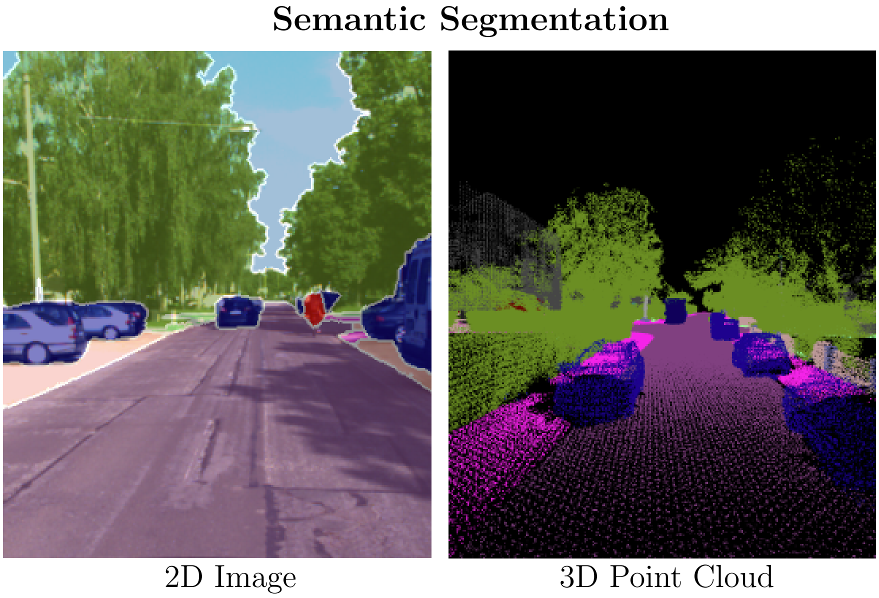


Figure 4.6: Semantic Segmentation on 2D image vs 3D point cloud. On the left side the result of semantic segmentation on a 2D image input. Each pixel is assigned to a semantic class. On the right the result of semantic segmentation where the input is an unstructured 3D point cloud. Each 3D point is assigned to a semantic class. The point cloud is sparse and unstructured compared to the grid like image. The examples are from the KITTI dataset (Xie *et al.*, 2016)

be seen in Fig. 4.6. we describe our work on the task of 3D semantic segmentation in Part II. Possible semantic classes for indoor scenes are *chair*, *table*, *sofa* and many others, or *car*, *tree*, *road* etc. for outdoor scenarios. In (Part III) we focus on a variation of 2D semantic segmentation called interactive object segmentation.

Semantic Segmentation in 3D point clouds using PointNet. PointNet is a seminal work by Qi *et al.* (2017b) that is used for the task of semantic segmentation in 3D point clouds. What differentiates PointNet from previous works is that it operates directly on unstructured raw 3D points without the need to enforce 3D structure through a regular volumetric grid or through 2D projections, like common approaches until then.

As it can be seen in Fig. 4.10 PointNet takes as input n points. After a series of Multi-Layer-Perceptrons (MLP) per point, the points are mapped into a higher dimensional space, these are called local point-features. Max-pooling is applied to aggregate information from all the points resulting in a common global-feature invariant to input permutations. The global-feature is then concatenated with all the point-features. After another series of MLPs these combined features are used to predict the m output class scores. Batch normalization is used for all layers that use ReLU as an activation function. Dropout layers are used for the last layer.

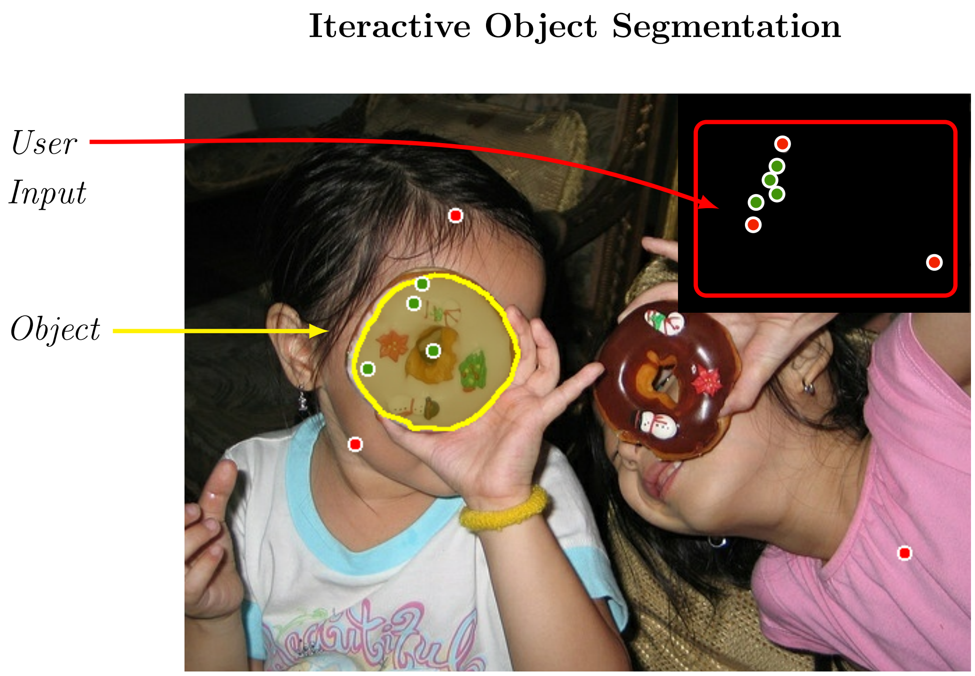


Figure 4.7: Interactive Object Segmentation in 2D Images. In the task of interactive object segmentation a human user collaborates with a computer vision model to segment an object of interest, here the ”donut” defined by the yellow border. The user provides feedback in the form of positive clicks and negative clicks, mouse clicks that are on the object of interest and mouse clicks outside the object respectively.

Semantic Segmentation in 2D images using DeepLabV3+. A successful deep network architecture for semantic segmentation of 2D images that is used in Part III of the thesis, is DeepLabV3+ (Chen *et al.*, 2018b). The overall architecture can be seen in image 4.8. It is based on a strong encoder-decoder network. The encoder is a modified Xception-65(Chollet, 2017) model originally introduced for image classification. In addition to the encoder-decoder network, the additionally applied *depth-wise separable convolutions* increase the computational efficiency. The encoder makes use of *atrous spatial pyramid pooling* (ASSP) that have been introduced in DeeplabV2(Chen *et al.*, 2016). Atrous convolutions are applied to the feature maps with different atrous rates corresponding to different sampling rates of the input feature map. The resulted feature maps are concatenated together, and their dimensionality reduced by 1 × 1 convolutions. The ASSP helps to extract information at different scales and combine it together.

The decoder is based on an output stride of 16. The encoded features are up sampled by a factor of 4 and concatenated to the corresponding low-level features

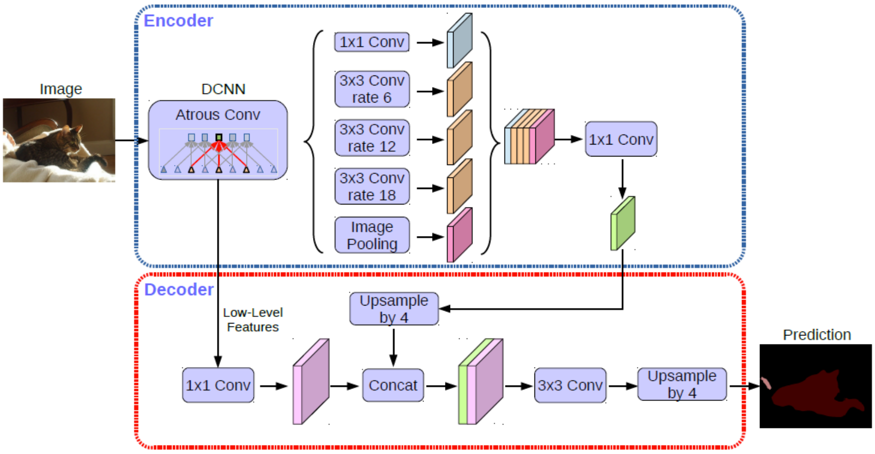


Figure 4.8: Overview of the DeepLabV3+ architecture. The model consist of an encoder and a decoder. The backbone of the encoder part is a modified Exception-65 architecture. For more details regarding the encoder part please see Fig. 4.9. Source (Chen *et al.*, 2018b)

from the encoder to the corresponding features with the same spatial dimensions after their dimensionality has been reduced with 1×1 convolutions. Another set of 3×3 convolutions are applied and the features are up-sampled by a factor of 4 to produce an output with the original image resolution.

We use a modified version of DeepLabV3+ as part of our approach in Part III. Our task is interactive object segmentation is to perform foreground segmentation where each image pixel can belong to only one out of two classes. It is either object or background. The object class is assigned to only one object instance per image. So DeepLabV3+ in that case outputs a distribution over two possible classes for each image pixel. Semantic segmentation has the same output as interactive object segmentation in images depicting only one object instance.

4.6.2 Interactive Object Segmentation (Part III)

The task is related to semantic segmentation but there are a few key differences. The input is a 2D image where we would like to assign each pixel to a semantic class. But in interactive object segmentation there are only two possible semantic classes: the *foreground* class that is assigned to all the image pixels belonging to an object of interest and the *background* class for everything else. Additionally, in interactive object segmentation a human user collaborates with a computer vision model to segment that object of interest. The computer vision models predicts a segmentation mask the user corrects possible errors. The user feedback is incorporated as part of

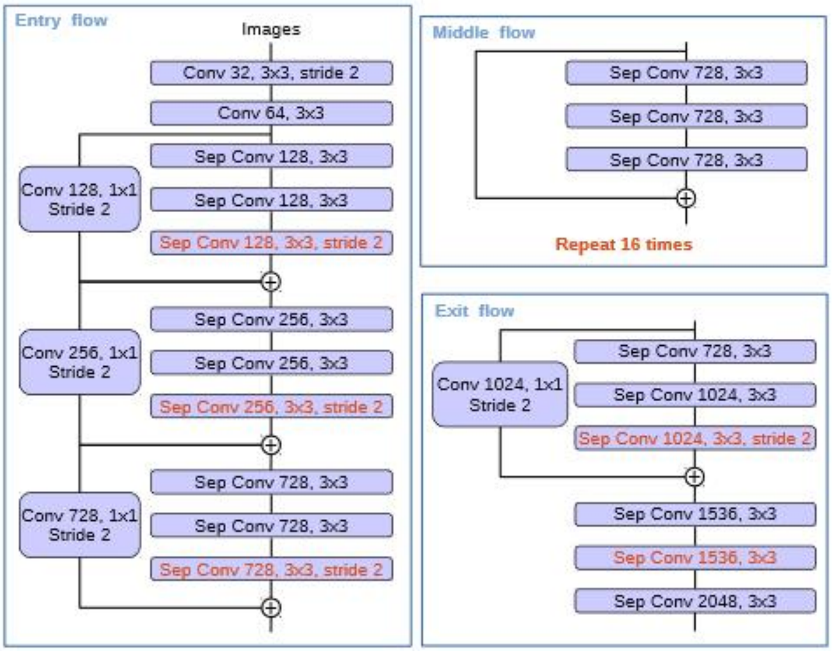


Figure 4.9: Overview of the modified Exception model used in DeeplabV3+. Modifications are presented in red. Max pooling operations have been replaced by separable convolutions (*Sep Conv*) with stride 2 and more layers have been added compared to the original. Source (Chen *et al.*, 2018b)

the input to the model and the model predicts an updated segmentation mask. The process iterates until the user is satisfied with the result.

4.7 3D Datasets

We train and evaluate our proposed models and algorithms for the task of 3D semantic segmentation on a number of different datasets. The datasets span a variety of scenarios and conditions from indoor to outdoor scenes and from synthetic to real. In contrast to Chapter 5 that presented unsupervised methodologies, the algorithms presented for the task of 3D semantic segmentation are fully supervised where the use of ground truth data helps optimizing our deep networks. We use the following datasets to evaluate the semantic segmentation methods presented in Part II.

Stanford Large-Scale 3D Indoor Spaces (S3DIS) by Armeni *et al.* (2016). The S3DIS dataset consists of dense 3D point clouds captured by Matterport scanners.

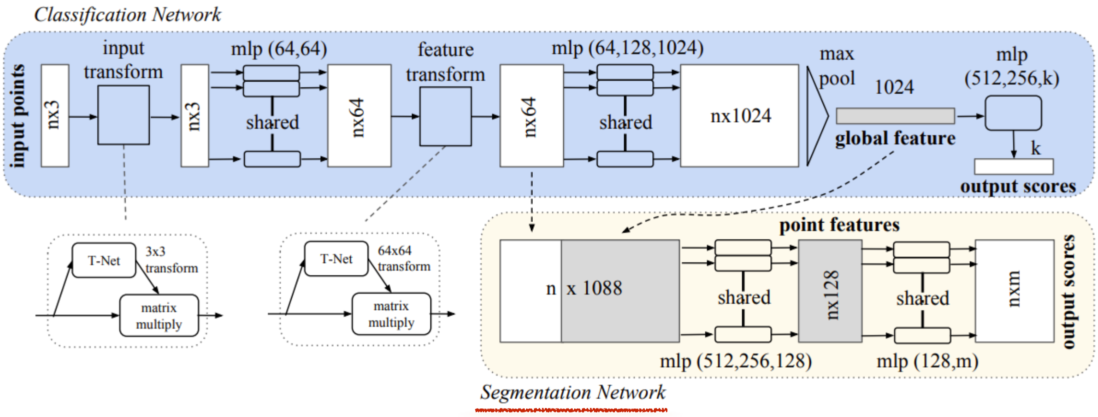


Figure 4.10: Overview PointNet semantic segmentation model. Modifications are presented in red. . The segmentation network takes n points as input and transforms the features to different dimensions through a series of multi-layer-perceptrons (**local feature**), and then aggregates point features by max pooling (**global feature**). It then concatenates global and local features and outputs per point scores. The output dimension of each layer is in brackets. Source (Qi *et al.*, 2017b)

It spans 271 rooms from 3 different buildings grouped into 6 large-scale indoor areas. Each point is assigned to one of 13 semantic classes (e.g., chair, door, table, wall) along with instance level segmentation. The dataset additionally provides normal information but, in our experiments, we only use the point coordinates.

ScanNet by Dai *et al.* (2017). The ScanNet benchmark dataset contains a wide variety of 3D reconstructed indoor scenes representing different rooms styles and layouts. It consists of more than 1500 scans of rooms. For semantic segmentation, we predict and evaluate on the 20 semantic classes used in the benchmark. In all experiments we follow the public training, validation, and test split containing 1201, 312 and 100 scans, respectively.

Virtual KITTI by Gaidon *et al.* (2016). The Virtual KITTI dataset is a photo-realistic synthetic dataset that imitates the real world KITTI dataset (Geiger *et al.*, 2013). The rendered virtual worlds were generated by computer graphics along with detailed annotations for many different tasks. The dataset contains labels for object detecting, tracking, dense depth and semantic instance annotations. We use the dataset for semantic segmentation of 3D outdoor point clouds by projecting the annotated depth and semantics into the 3D space to obtain semantically labeled point cloud representations that imitates the LiDAR data of KITTI dataset.

4.8 2D Datasets

We train our proposed method for the task of interactive object segmentation solely on PASCAL VOC12 training set (Everingham *et al.*, 2012) described below. Since our method can adapt to different datasets on-the-fly during test time we evaluate our method in numerous datasets that differ from the training dataset of PASCAL VOC12 in various degrees. From the small distribution shifts of the PASCAL VOC12 validation set to the significant domain change of the medical images of DRIONS-DB (Carmona *et al.*, 2008). We briefly describe the datasets in the following section. With **train** we refer to a dataset used only for training and with **test** to a dataset used only for testing. Please note that the hyper parameter tuning was performed on a dataset separate from both train and testing datasets, ADE20K. We use the following datasets for the experiments presented in Part III.

PASCAL VOC12 TRAIN (Train) by (Everingham *et al.*, 2012). We pre-train the model parameters only on PASCAL VOC12 augmented with the Semantic Boundaries Dataset (SBD) (Hariharan *et al.*, 2011). The training set consists of 10582 images with 24125 segmented instances of 20 object classes.

PASCAL VOC12 VAL (Test) by (Everingham *et al.*, 2012). We test our method on PASCAL VOC12 validations set.

GrabCut (Test) by (Rother *et al.*, 2004b). The Grabcut datasets contains 49 images with segmentation masks. Each image depicts a single foreground object.

Berkeley (Test) by (McGuinness and O'Connor, 2010). The Berkeley datasets consists of a 100 images with a single foreground object.

YouTube - VOS (Test) by (Xu *et al.*, 2018a) is a large video object segmentation dataset. In our experiments we use the test set of the 2019 challenge. We select the first frame of each video clip with its ground truth resulting with 1169 object instances. Each image is downscaled to 855 480 maximal resolution.

Microsoft Common Objects in Context (COCO) (Test) by (Lin *et al.*, 2014) is a large segmentation dataset with 80 object classes. 20 of those overlap with the ones in the PASCAL VOC12 dataset and are thus seen during training. The other 60 are unseen. We sample 10 objects per class from the validation set and separately report results for seen (200 objects) and unseen classes (600 objects) as in (Majumder and Yao, 2019; Xu *et al.*, 2016). We also study how image sequence adaptation behaves on longer sequences of 100 objects for each unseen class (named COCO unseen 6k).

DAVIS16 (Test) by (Perazzi *et al.*, 2016) contains 50 high-resolution videos (30 sequence reserved usually for training and 20 for testing). In our set up we make use of all sequences out of which we sample 10% of the frames uniformly at random as in (Jang and Kim, 2019; Li *et al.*, 2018b). We note that the standard evaluation protocol of DAVIS16 favors adaptive methods, as the same objects appear repeatedly in the test sequence since the dataset was usually used to evaluate Video Object Segmentation (VOS) methods. For each sequence in DAVIS16 dataset only a single object is annotated with ground truth even though other objects are present.

Rooftop Aerial by (Sun *et al.*, 2014) is a dataset of 65 aerial images with segmented rooftops of buildings.

DRIONS-DB by (Carmona *et al.*, 2008) is a dataset of 110 retinal images with a segmentation of the optic disc of the eye fundus. We use the masks of the first expert.

ADE20k by (Zhou *et al.*, 2016) We optimize the hyperparameters for all adaptation methods on a subset of the ADE20k dataset. Hence, the hyperparameters are optimized for adapting from PASCAL VOC12 to ADE20k, which is distinct from the distribution shifts and domain changes we evaluate on.

I

Efficient spanning tree construction and update for graph based clustering algorithms.

5

Incremental Object Discovery in Time-Varying Image Collections

In this chapter, we address the problem of object discovery in time-varying, large-scale image collections. A core part of our approach is a novel Limited Horizon Minimum Spanning Tree (LH-MST) structure that closely approximates the Minimum Spanning Tree at a small fraction of the latter's computational cost. Our proposed tree structure can be created in a local neighborhood of the matching graph during image retrieval and can be efficiently updated whenever the image database is extended. We show how the LH-MST can be used within both single-link hierarchical agglomerative clustering and the Iconoid Shift framework for object discovery in image collections, resulting in significant efficiency gains and making both approaches capable of incremental clustering with online updates. We evaluate our approach on a dataset of 500k images from the city of Paris and compare its results to the batch version of both clustering algorithms.

5.1 Introduction

Social media platforms have become favorite storage and sharing sites for all kinds of images. A large part of those images are touristic photos, resulting in a dense image coverage of famous monuments and landmarks up to the scale of entire cities (Avrithis *et al.*, 2010; Gammeter *et al.*, 2009). This has created a call for computer vision algorithms that can perform efficient object discovery (Philbin and Zisserman, 2008; Quack *et al.*, 2008; Weyand and Leibe, 2011), clustering, and matching in image collections for applications such as large-scale 3D reconstruction (Agarwal *et al.*, 2009; Frahm *et al.*, 2010; Li *et al.*, 2008), scene summarization

(Simon *et al.*, 2007), automatic image annotation (Gammeter *et al.*, 2009), or visual search (Avrithis *et al.*, 2010; Zheng *et al.*, 2009).

However, the content of large-scale image repositories is never static. Millions of images are added to such repositories each day, while others are withdrawn or deleted. Current object discovery algorithms (Avrithis *et al.*, 2010; Philbin and Zisserman, 2008; Quack *et al.*, 2008; Weyand and Leibe, 2011; Zheng *et al.*, 2009) do not yet address this issue. They typically operate in a static setting, making it necessary to re-run the entire clustering process whenever the underlying image database changes, even though only a small part of the clusters may be affected by the changes. Image retrieval and recognition web services using those algorithms waste thousands of core hours of computation because of this.

Even leading providers of visual search engines such as Google are bound to rebuild the clustering from scratch every week [personal communication] due to changes in the image database. Currently this problem of dynamic database changes is not well researched in the community despite its large effect on practical applications.

In this chapter, we address the image clustering and object discovery problem in an incremental setting. We propose a novel clustering method that allows for efficient reuse of the stored data. A key idea of this paper is that many clustering methods can be efficiently implemented with the help of a spanning tree. In order to enable incremental clustering, we therefore propose efficient techniques for incrementally constructing and updating the spanning tree. Because of the incremental updates, information about unaffected clusters can be preserved and only those clusters need to be recomputed that contain updated parts of the spanning tree.

At the core of our approach is a novel Limited Horizon Minimum Spanning Tree (LH-MST) data structure that closely approximates a Minimum Spanning Tree (MST), while being significantly faster to construct and to update whenever new images are added to the matching graph. The LH-MST can be used in any object discovery approach that operates on spanning trees, such as single-link clustering (Gammeter *et al.*, 2009; Quack *et al.*, 2008; Zheng *et al.*, 2009) or Iconoid-Shift mode estimation (Weyand and Leibe, 2011, 2013). In contrast to a regular MST that always requires the full matching graph to be known a priori, an LH-MST can be computed starting from a local seed, making it suitable for local exploration of a matching graph through Query Expansion (Chum *et al.*, 2007). We experimentally verify that in realistic use cases, the differences to a regular MST are minimal, making our algorithm an efficient alternative for parallel and distributed clustering applications.

We demonstrate our approach's practical feasibility for large-scale incremental object discovery by applying it to the *Iconoid Shift* (IS) object discovery approach (Weyand and Leibe, 2011, 2013). As our results will show, replacing the MST by an LH-MST translates to a 5-fold improvement in runtime already during batch operation. In addition, we experimentally show that IS with LH-MST achieves stable incremental clustering results even when the image database is extended by a significant fraction.

Contributions. In detail, the work presented in this chapter makes the following contributions:

1. We propose a novel LH-MST spanning tree structure, which approximates the MST and which can be incrementally updated very efficiently.

2. We show how this tree structure can be used for efficient approximative single-link clustering and how it can be incorporated into the Iconoid Shift approach (Weyand and Leibe, 2011) for large-scale object discovery in image collections.

3. We verify experimentally that the resulting IS using LH-MST (or LH-IS) approach achieves very similar results as the offline version of IS, while being both considerably faster and capable of online updates of a significant fraction of the image database.

5.2 Related Work

Object discovery approaches aim at finding clusters of images showing the same object or landmark building in large image collections. The standard procedure for this is to apply local feature based matching techniques (Philbin *et al.*, 2007) to build up a matching graph of images and to then subdivide the connected components of this matching graph into image clusters. A large range of clustering methods have been proposed for this step, including single-link hierarchical agglomerative clustering (Gammeter *et al.*, 2009; Quack *et al.*, 2008; Zheng *et al.*, 2009), spectral clustering (Heath *et al.*, 2010; Kim *et al.*, 2008; Philbin and Zisserman, 2008), eigenvector centrality (Jing and Baluja, 2008), spherical k-means (Simon *et al.*, 2007), Kernel Vector Quantization (Avrithis *et al.*, 2010) and Medoid Shift mode estimation (Weyand and Leibe, 2011, 2013). Because of the large size of the datasets to be mined, runtime efficiency and ease of parallelization are a prime concern.

In this work, we focus on two classes of object discovery algorithms that have proven their worth in large-scale applications: single-link clustering (Gammeter *et al.*, 2009; Quack *et al.*, 2008; Zheng *et al.*, 2009) and Iconoid Shift (Weyand and Leibe, 2011, 2013). Single-link clustering is directly related to the Minimum Spanning Tree (MST). Given the MST of a graph, the corresponding single-link clusters can be obtained by cutting all MST edges above a chosen distance threshold (Rohlf, 1973). Iconoid Shift (Weyand and Leibe, 2011) uses Medoid Shift mode estimation with a special kernel in order to find images that have a locally maximal mutual homography overlap with their neighbors. Those images often correspond to central, iconic views of a landmark building, and the overlapping images under the kernel form the resulting cluster. In order to be able to apply the Medoid Shift formalism, the used distance measure needs to fulfill the triangle inequality. Since this is usually not the case for local-feature based matching scores, IS defines a transitive pairwise image distance measure that is computed over the edges of an MST. In practice, the MST construction step is responsible for a significant fraction of both clustering algorithms' runtime.

As shown in (Chin and Houck, 1978; Spira and Pan, 1975), the MST of a graph with N nodes can be updated in $O(N)$ when a new node is added to the graph, which is simultaneously the lower bound for an exact update. This may not seem like much, but the key issue here is that such an exact update requires the full matching graph to be available on a single machine. For practical object discovery in large-scale image databases, even individual connected components of the matching graph may get so large that processing them on a single computing node becomes inefficient (in our experiments with 500k images, the largest connected component comprises $>$ 50k images). We therefore aim at an approximation of the MST that can be locally grown from a seed image, such that computation can be distributed over a computing cluster.

5.3 Clustering Methods for Object Discovery

In this work we are primarily interested in performing efficient object discovery in large image collections. To that end, we propose a novel tree structure which can speed-up MST based clustering algorithms and has the additional benefit of allowing incremental database updates without the need of rebuilding the entire database. In the following we will review two such clustering algorithms which rely on the creation of an MST. In the remainder of the paper we will then show that these algorithms benefit from our proposed MST approximation.

5.3.1 Single-Link Agglomerative Clustering

Single-Link Agglomerative Clustering is a widely used clustering method especially in the landmark discovery community (Gammeter *et al.*, 2009; Quack *et al.*, 2008; Zheng *et al.*, 2009). It offers the advantage of creating clusters without any assumptions on their shape and the number of clusters. One of its drawbacks is that it is prone to long chain-like clusters that require rigorous post-processing in order to achieve competitive results. Additionally, it does not allow for incremental database updates, requiring a full re-computation after adding nodes.

Single-Link clustering can be performed in a bottom-up fashion with each image starting as an individual cluster. Then the two closest clusters are iteratively joined until either the whole database is represented by a single cluster or a cut-off threshold θ on the between-cluster distance $d(A, B)$ is reached. This cluster distance is defined as $d(A, B) = \min_{a \in A, b \in B} d(a, b)$.

An efficient $\mathrm{O}(N^2)$ implementation of Single-Link clustering can be obtained by first building the full matching graph and then computing its MST. Removing all edges above a certain threshold θ generates clusters of connected components. This dependency on the MST creation makes it an ideal candidate for MST approximation experiments.

5.3.2 Iconoid Shift

Iconoid Shift (IS) discovers landmark objects in image collections by searching for so-called *iconic* images. Starting from a set of seed images (typically obtained by Geometric MinHash (Chum and Matas, 2010)), IS first builds up local matching graphs G by recursively applying image retrieval (Chum *et al.*, 2007; Philbin *et al.*, 2007) with the seed image r as a query. The edges between images i and j are weighted by a distance measure $\mathrm{d}_{\mathrm{ovl}}(i, j)$ that quantifies the overlap region between the two directly connected images. In addition to those direct distances, IS proposes a transitive overlap distance $\mathrm{t}_{\mathrm{ovl}}(i, j)$ that propagates content overlaps between images that are not directly connected via homography chains. The algorithm adds nodes to G as long as the transitive distance to the root node r is below a cut-off point β (the so-called kernel radius). As a result of this local exploration, each seed image is connected to a set of similar images via the graph structure G.

On this graph, IS first builds up an MST and then computes the transitive distances $\mathrm{t}_{\mathrm{ovl}}(i, j)$ between all pairs of nodes (i, j) in G by propagating homography overlaps via the MST. It then uses the *Medoid Shift* (MS) algorithm (Sheikh *et al.*, 2007) to find the modes of the density implicitly defined by the transitive overlap distance, which correspond to the *iconic* images. Medoid Shift is a generalization of the Mean Shift mode estimation algorithm (Comaniciu and Meer, 2002) that operates on graphs. In each iteration, it computes the medoid of the graph nodes under its kernel (defined by the pairwise image distances) and then shifts the kernel window to this medoid. Whenever such a shift happens, the local matching graph is extended using the medoid as a new seed and the MST is recomputed. It iterates until convergence (see Algorithm 1).

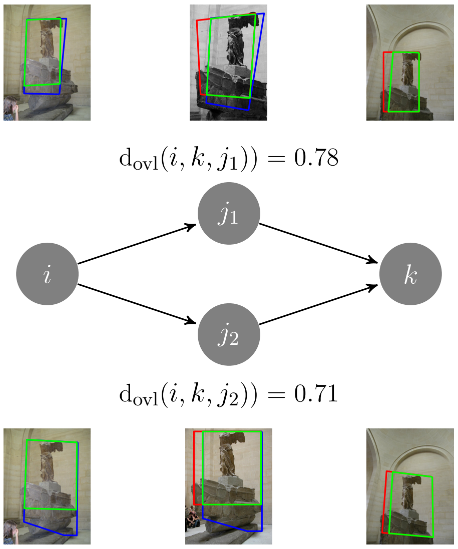


Figure 5.1: Transitive overlap distance

Transitive overlap distance ($\mathrm{t}_{\mathrm{ovl}}$): The transitive overlap distance forms an important part of calculating the overlap over paths in the graph. It defines a distance for a pair of images (i,k) which are not directly connected in G, only through a path j of nodes. All directly connected images in G have corresponding homographies and bounding boxes enclosing the matching feature points. As shown in Figure 5.1, the transitive overlap distance between images i and k through images j_1 or j_2 is computed by projecting the matching bounding box from images i (blue box) and k (red box) to their common neighbor j_1 or j_2. The overlap region between these two projections is found by intersecting the boxes (green box in j_1 and j_2) and back-projecting them to images i and k. Then the transitive overlap distance $\mathrm{t}_{\mathrm{ovl}}(i, k, j)$ is defined as 1 minus the minimum overlap of the projected boxes divided by the image area (see (Weyand and Leibe, 2011)).

Figure 5.1 also shows that the path j influences the final distance between i and k, i.e. the overlap via j_1 is much larger than via j_2. There are as many distances as paths from i to k. The choice of the spanning tree can therefore crucially influence the behavior of IS.

Caveats. The use of MS in a distance graph G requires the edge weights to form a metric. However due to the limited repeatability of the features used for image retrieval, the graph G may contain cycles which violate the triangular inequality for d_{ovl} in G, i.e. $\mathrm{d}_{\mathrm{ovl}}(i, k) \not\leq \mathrm{d}_{\mathrm{ovl}}(i, j) + \mathrm{d}_{\mathrm{ovl}}(j, k)$. Although d_{ovl} is symmetric, the retrieval system is not. As such, it is not possible to use MS directly on G (at this point G is not fully connected). The spanning tree construction is necessary in order to circumvent this problem. By computing the MST on G, all potentially inconsistent cycles are removed, before the fully-connected, consistent graph G' is

Algorithm 1: Iconoid Shift Steps

```
Require: photo collection $C$, seed image $s$
  while $s$ shifts to a new mode do
    Step 1. Build graph $G$ around $s$ that connects similar images via distance $\mathrm{d}_{\mathrm{ovl}}$.
    → Edge weights are not a metric.
    Step 2. Create minimum spanning tree $T$ of $G$ to remove cycles.
    Step 3. Construct fully connected $G'$ from $T$ by replacing missing edges with $\mathrm{t}_{\mathrm{ovl}}$.
    → Edge weights are now a metric.
    Step 4. Perform Medoid Shift in $G'$. Mode $m$ is now the new $s$.
```

constructed by connecting nodes via transitive connections $\mathrm{t}_{\mathrm{ovl}}$ (see Algorithm 1). Through the definition of $\mathrm{t}_{\mathrm{ovl}}$ (see (Weyand and Leibe, 2011)), the resulting graph fulfills the triangular inequality, making it possible to perform an iteration of MS.

When the mode shifts to a new node, the retrieval has to be performed again, as new nodes might now be within reach of the kernel distance, while others might drop out. This means that the MST has to be recomputed on the new local neighborhood. Although this new neighborhood is usually smaller than the full connected component, this step can still incur heavy computation. An efficient alternative to the MST creation can therefore result in significant speed-ups.

In summary, we see that, as for single-link clustering, the expensive calculation of the MST ($\mathrm{O}(N^2)$ in the number of tree nodes N) is a major bottleneck of IS and it restricts its usability as an online algorithm.

5.4 Limited Horizon Minimum Spanning Tree

In this section we introduce a novel tree structure called ***Limited Horizon Minimum Spanning Tree*** (LH-MST). This tree structure (Section 5.4.1) is built during image retrieval (e.g. in Step 1 in Algorithm 1). It is considerably faster to compute than the regular MST (Section 5.4.1.1) and allows to incrementally add new nodes to an initial dataset instead of rebuilding the whole database when new imagery is available.

5.4.1 Offline Construction of LH-MST

Algorithm 2 shows the creation of our ***Limited Horizon Minimum Spanning Tree*** (LH-MST). The algorithm is based on the priority queue q_{update}, which contains tuples of edges and their corresponding edge cost $\mathrm{d}_{\mathrm{ovl}}$. The edges are directed from a parent node p to a child node n and the queue is sorted by the edge cost in ascending order. As before, we start with a seed, the root image r. The priority queue is initialized by a directed edge from $p = r$ to $n = r$ with weight

Algorithm 2: Compute LH-MST

```
Require: root $r$, priority queue $q_{update}$
  $q_{update} = \emptyset$
  $push(q_{update}, \{\text{edge}(r,r), d_{ovl}(r,r)\})$
  while $q_{update} \neq \emptyset$ do
    $\text{edge}(p,n) = pop(q_{update})$
    if $n \notin$ spanning tree $T$ then
      $add(\text{edge}(p,n))$ to $T$;
      $children = retrieve(n)$
      for $c \in children$ do
        if $\phi(t_{ovl(}r,c)) < \theta$ then
          $push(q_{update}, \{\text{edge}(n,c), d_{ovl}(n,c)\})$
  return $T$
```

Algorithm 3: Incremental Update LH-MST- Addition

```
Require: root $r$, priority queue $q_{update}$
Require: spanning tree $T$
  $q_{update} = \emptyset$
  for $\text{edge}(p,n) \in T$ do
    $push(q_{update}, \{\text{edge}(p,n), d_{ovl}(p,n)\})$
  while $q_{update} \neq \emptyset$ do
    $\text{edge}(p,n) = pop(q_{update})$
    if $n \notin T$ then
      $add(\text{edge}(p,n))$ to $T$
      $children = retrieve(n)$
      for $c \in children$ do
        if $\phi(t_{ovl(}\text{r,c})) > \theta$ then
          $push(q_{update}, \{\text{edge}(n,c), d_{ovl}(n,c)\})$
  return T
```

equal to one. As long as the priority queue contains elements, we remove the first tuple $\{\text{edge}(p,n), d_{ovl}(p,n)\}$ from q_{update} and perform image retrieval on image n. Newly retrieved images c are added to the q_{update} (as $\{\text{edge}(n,c), d_{ovl}(n,c)\}$) if they are within kernel distance to the root node and do not create cycles. A few iterations of the algorithm are visualized in Figure 5.2.

LH-MST is an MST approximation constructed in a greedy way. After iteration k the resulting spanning tree contains at most k nodes and has a maximal depth of k. The resulting spanning tree would be equivalent to the MST if the retrieval system were symmetric.

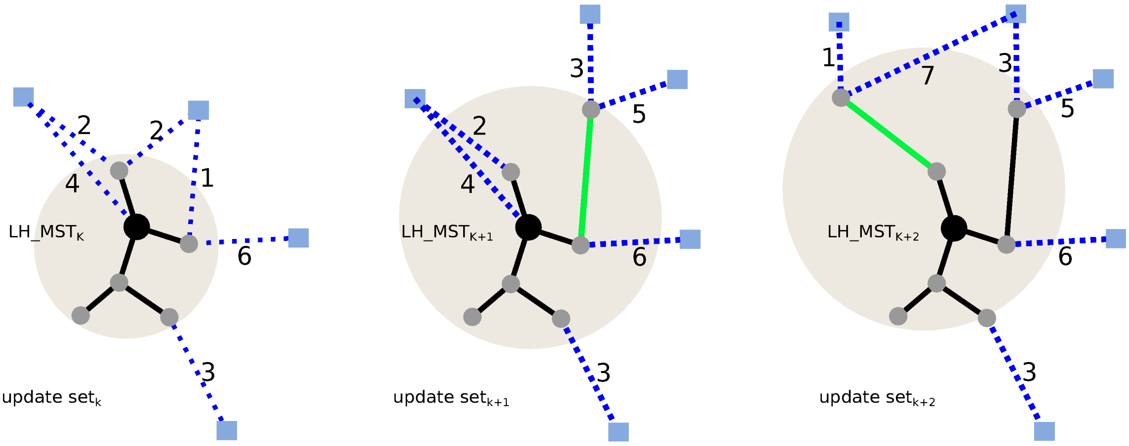


Figure 5.2: Three Steps of the Limited Horizon Minimum Spanning Tree. **LH-MST** $_k$: The current tree (solid lines) contains k nodes, the update set new node candidates with their corresponding weights (dashed lines). **LH-MST** $_{k+1}$: the node with the minimal distance to any existing tree node (green line) is added to the tree; redundant edges are removed; the retrieval system finds new image candidates (within kernel distance) and adds them to **update set**$_{k+1}$.

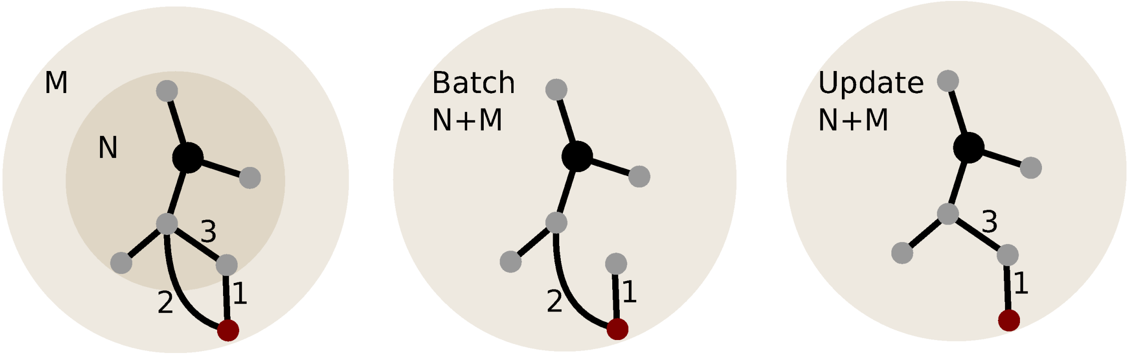


Figure 5.3: The incremental update of the LH-MST can result in slightly different trees. The red node belongs to set M, all other nodes to set N. In batch mode (middle), the red node would be retrieved during tree-construction with cost “2”. The incremental update (right) starts with a pre-existing tree resulting in a slightly different path.

5.4.1.1 Complexity Analysis.

A major advantage of using LH-MST over MST to enforce the triangular inequality in G is speed. The LH-MST is constructed during retrieval and causes negligible maintenance overhead during the update. **Step 2** of the IS pipeline (see Algorithm 1) is not needed anymore. Saving the overhead for the MST construction (O(N^2)) makes the proposed method significantly faster.

In contrast, one might suggest to use the basic IS algorithm as is and only replace the MST creation by a faster approximate MST algorithm (Karger *et al.*, 1995). The resulting speed-up would be limited, as any faster MST algorithm still needs to build a full image graph first (not a spanning tree) and then compute the MST as a separate step.

5.4.2 Incremental Update of LH-MST

Performing any kind of clustering on databases with hundreds of thousands of images is very time consuming. Although we are able to speed up the tree creation, it might well happen that the image database has already been outdated before the clustering finished. Instead of collecting new data and re-running the clustering, the LH-MST is able to perform incremental updates. In this section we will describe the update procedure; Section 5.5.3 will then experimentally show its validity.

Algorithm 3 shows the incremental update of an LH-MST. The algorithm extends an already existing spanning tree LH-MST $^{(N)}$ consisting of N nodes with a set of new nodes M to create the spanning tree LH-MST $^{(N+M)}$. The update procedure is similar to the initial construction of an LH-MST. Instead of starting from the root node r, we start with a complete tree built based on set N, and fill the priority queue with edges from this initial tree. The retrieval is performed in the order of the priority queue, but only retrieves images from the new set M. This adds new edges to the existing tree from N to M. As such, the resulting trees might slightly differ, as shown in Figure 5.3.

5.4.2.1 Complexity Analysis

We consider the worst case scenario (in terms of complexity) of a fully connected retrieval graph, where each node upon retrieval returns all other nodes in the connected component. As complexity we consider the number of distance measure computations between nodes. Our full database has N elements.

Offline Version considers the construction of an LH-MST consisting of N nodes (including the seed s). In iteration k=1, there is only one node already in the LH-MST, the seed s and N-1 nodes to be considered for addition. Thus, N-1 distance computations between node pairs are needed. Each iteration of adding a node to the tree reduces the number of distance computations by one, resulting in a complexity of

$$\begin{aligned} \text{compl}_{\text{offline}} &= 1 + 2 + 3 + 4 + \cdots + N - 1 && (5.1) \\ &= \frac{(N) \cdot (N-1)}{2} && (5.2) \end{aligned}$$

Online Update considers a pre-existing LH-MST with an initial set of nodes N_1 and updates the initial tree with a new set of nodes N_2, where $N_1 + N_2 = N$. In iteration k=1 there are $N_1 \cdot N_2$ distances computed between all the N_1 nodes of the initial spanning tree and all the N_2 nodes of the update set in order to pick a new edge to add in the spanning tree. Similar to the offline update, in each iteration elements from N_2 will be added to the tree resulting in a decrement of distance computations:

$$\text{compl}_{\text{update}} = N_1 \cdot N_2 + \frac{(N_2) \cdot (N_2 - 1)}{2} \tag{5.3}$$

As expected, the complexity of the offline version is larger than the complexity of incrementally updating the existing initial tree with the additional nodes by

$$\text{compl}_{\text{offline}} - \text{compl}_{\text{update}} = \frac{(N_1) \cdot (N_1 - 1)}{2}, \tag{5.4}$$

which is the number of distance computations to create the spanning tree with the initial set of nodes N_1. Note that this analysis includes the effort for retrieval and distance computations. Without considering retrieval, the remaining effort of node addition is in O(1).

5.5 Experiments

In this paper we propose to replace MST by LH-MST in order to speed up computation and allow for efficient online updates. We will now use LH-MST to perform Single-Link Agglomerative clustering and Iconoid Shift as follows.

Limited Horizon Single-Link Clustering (LH-SL). In the tree construction we replace MST by LH-MST.

Limited Horizon Iconoid Shift (LH-IS). We replace steps 1 and 2 in Algorithm 1 by creating the tree T' directly.

LH-MST is one instance of many possible spanning tree alternatives that would also allow for fast creation and online updates. The alternatives could also be constructed during the retrieval step (similar to LH-MST) and thus would be faster than the original MST. We will show the advantages of LH-MST over several of such spanning tree baselines: **BFT** - *Breadth First Spanning Tree* (siblings with higher priority in queue), **DFT** - *Depth First Spanning Tree* (children with higher priority in queue), **RAND** - *Random Spanning Tree* (random order priority queue), **SPT** - *Shortest Path Spanning Tree* (queue sorted by shortest path to initial medoid).

All experiments are performed on the full 500k images of the Paris dataset (Weyand *et al.*, 2010) collected from Flickr and Panoramio.

5.5.1 Evaluation of Single - Link Clustering

In Single-Link Clustering changing the cut-off threshold θ affects the number of resulting clusters starting from one cluster for the whole database to individual clusters for each element. Replacing the MST creation with LH-MST will likely result in a slightly different tree structure and therefore in different clusters. To assess the magnitude of this difference we compare the resulting trees by their edit-distance E_d as a function of θ:

$$\begin{aligned} E_d(S,T) = \; & \tfrac{1}{2}(|\{(i,j)|(i,j) \in S \wedge (i,j) \notin T\}| \\ & +|\{(i,j)|(i,j) \notin S \wedge (i,j) \in T\}|) \end{aligned}$$

where T is the MST and S is the candidate tree. First the two full trees are compared based on the number of edges that need to be removed/added to convert the approximation (LH-MST) back into MST. We then keep track of the tree differences when changing θ.

Figure 5.4 shows the percentage of changed edges in the alternative trees (BFS, DFS, RAN, LH-MST) compared to the MST when varying θ. The LH-MST is the closest to the MST by a large margin.

While this result shows that the approximation does not deviate by more than 0.027 percent for the original MST, it should be noted that the performed comparison is still slightly too strict. From a practical point of view, the underlying tree structure is irrelevant as long as the same images are clustered together by both methods.

5.5.2 Evaluation of Offline LH-IS

After using LH-MST for the case of Agglomerative Clustering, we now show its validity of performing Iconoid Shift with LH-MST called LH-IS. We compare our results to the ones of Weyand and Leibe (2011). In the following, we introduce the general setup and evaluation measures used for the evaluation.

Seeds. We use the seeds provided by Weyand and Leibe (2011): A seed group S with geometric min-hash (Chum *et al.*, 2007) using 5 min-Hash sketches of size 2 returns a total 10487 images. Duplicates are removed resulting in a set of 477 images. Empirically this limited number of seeds offers a good coverage of the database.

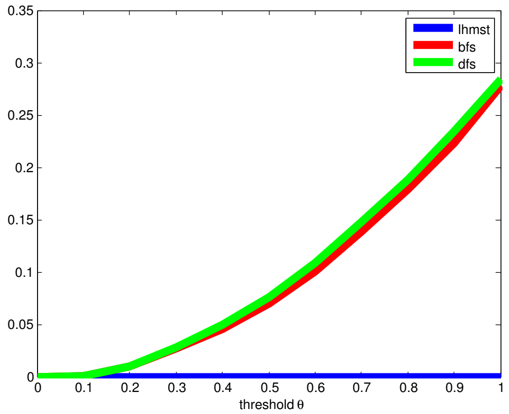


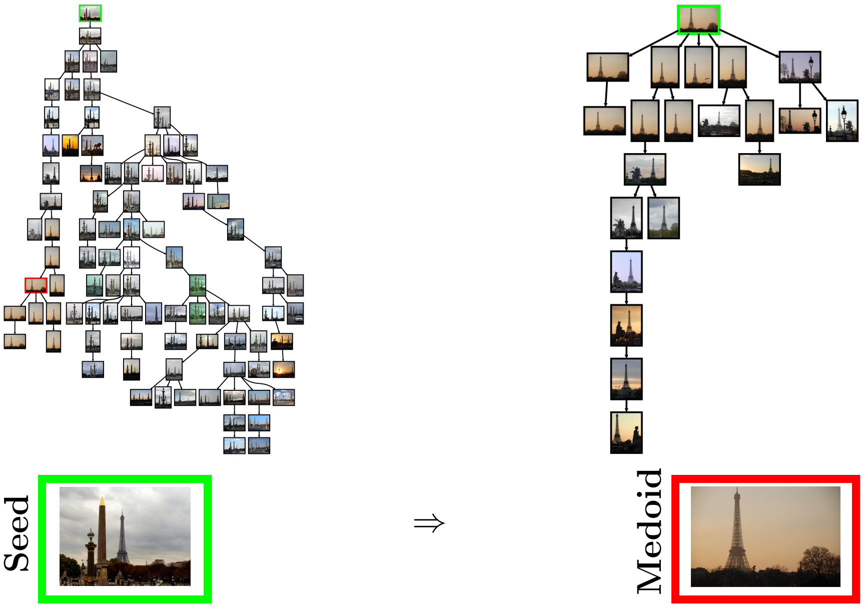
Figure 5.4: Evaluation of trees alternative to MST for Single-Link Clustering. The cumulative histogram of image distances in the cases of disagreeing tree edges.

τ	$\|\mathbb{M}\|\uparrow$	$\|\mathbb{M}_A\|\uparrow$	$\|\mathbb{M}_O\|\downarrow$	$\|\mathbb{M}_U\|\downarrow$	$\|\mathbb{K}\|\downarrow$	*Rank* $\downarrow$ avg.	$\|\mathbb{S}_{sup}\|$ avg.	*DB cover*
IS	**324**	-	-	-	90	-	380.42	73249
LH-IS	321	**282**	28	4	87	18.71	380.43	70762
BFT-IS	396	150	213	14	153	172.6	1146.27	73886
DFT-IS	365	137	136	87	154	75.0	331.65	68023
SPT-IS	256	88	148	12	220	132.39	1334.86	61828
RAN-IS	346	194	135	15	99	61.29	981.7	70414

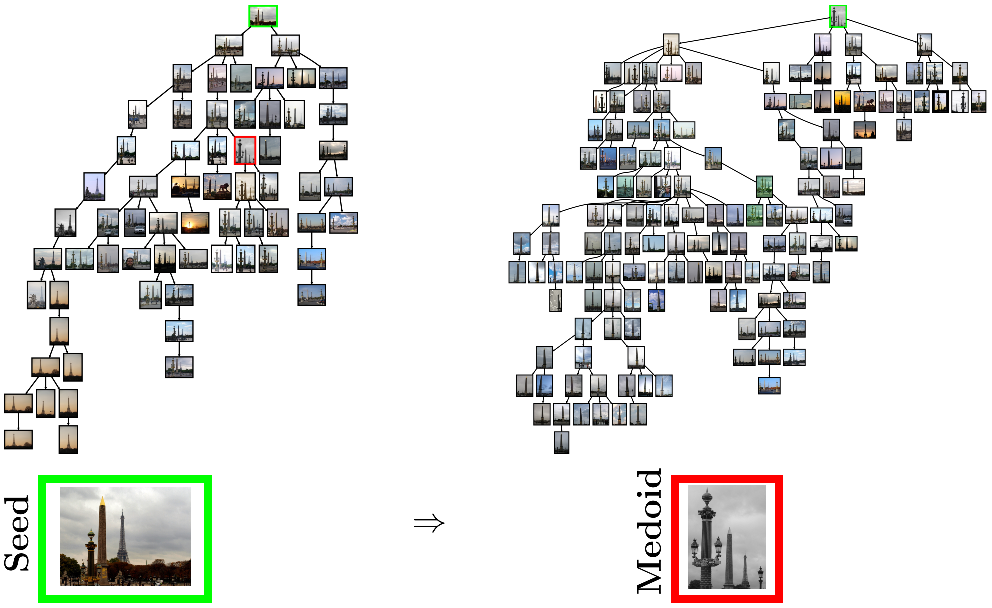
Table 5.1: Performance of the LH-MST compared to several baseline tree when they are replacing MST in the IS framework.

Medoid Shift kernel. We used a hinge kernel for the MS with a cut-off threshold of 0.9, which means that we allow images with at least 10% overlap to the support set of the *Iconoid*.

Evaluation measures glossary. In the following we compare various evaluation measures to the “ground truth”, the set of medoids $\mathbb{M}^{MST}$ originating from the original (offline) version of IS. In Section 5.5.3 we compare to the online version and use the sets $\mathbb{M}^{LH-IS}$ as “ground truth”.



(a) 2 iterations of MST (Weyand and Leibe, 2011)



(b) 2 iterations of LH-MST

Figure 5.5: Example iterations of IS using (a) MST and (b) LH-MST. Both algorithms are initialized with the same seed image (green border) that depicts the *Eiffel Tower* and *Cleopatra's Needle* but they converge to different medoids (red border). The MST's final cluster (5.5a) shows only one of the landmarks (*Eiffel Tower*) depicted in the seed image. As it can be seen, the final medoid of LH-MST (5.5b) does not belong to the support set of the final medoid of MST and visa versa. This is one of the rare cases where medoids cannot be matched (see Table 5.1, column $\mathbb{M}_U$).

τ	potential spanning tree are: $\tau \in \{ \text{MST}, \text{LH-MST}, \text{BFT}, \text{DFT}, \text{RAN}, \text{SPT} \}$
$\mathbb{M}^{\tau}$	the set of medoids resulting from method τ
$\mathbb{M}_A$	the set of medoids common between ground truth and method τ.
$\mathbb{M}_O$	the set of medoids of method τ that do not match the medoid selected in the ground truth. Still, the medoids are members of the ground truth cluster. How close they are to the ground truth is expressed by the *Rank*.
$\mathbb{M}_U$	the set of medoids of method τ that are not even part of any ground truth cluster.
$\mathbb{K}$	the set of seeds for which the MS is stuck in initialization.
$\mathbb{S}_{sup}$	the cardinality of the final clusters
Rank	the smallest relative position (rank) that the medoids of method τ achieve in all ground truth clusters. The smaller the value of $\mathbb{R}_d$ is, the closer the result is to the optimum.
DB Cover	the size of the database explored

Table 5.2: Glossary: Evaluation Measures

LH-MST			$\lvert\mathbb{M}\rvert$	$\lvert\mathbb{M}_A\rvert$	$\lvert\mathbb{M}_O\rvert$	$\lvert\mathbb{M}_U\rvert$	$\lvert\mathbb{S}\rvert$	*Rank*	$\lvert\mathbb{S}_{sup}\rvert$	*DB cover*
	Split									
	100	-	321	-	-	-	87	-	416	70762
S1	90	10	327.33 ± 1.89	263.0 ± 8.83	53.67 ± 8.96	3.67 ± 1.89	90.0 ± 2.16	79.93 ± 74.43	424.04 ± 32.72	61716
S1	70	30	327.0 ± 7.79	264.0 ± 22.73	56.67 ± 25.62	2.67 ± 1.7	86.33 ± 0.94	36.13 ± 19.29	435.14 ± 28.13	69689
S1	50	50	325.33 ± 2.49	234.0 ± 6.53	85.67 ± 9.84	2.0 ± 0.0	91.0 ± 1.41	81.1 ± 57.32	445.94 ± 65.38	68570
S2	70	30	340.33 ± 2.49	212.0 ± 21.65	116.0 ± 19.6	5.0 ± 2.16	87.33 ± 10.87	32.4 ± 8.36	423.41 ± 31.73	69659
S2	50	50	338.0 ± 4.32	180.33 ± 12.5	144.0 ± 14.31	4.0 ± 1.41	83.33 ± 3.86	153.12 ± 107.98	390.75 ± 23.2	67246
S1	90	-	329.67 ± 0.47	252.33 ± 1.25	65.0 ± 1.41	4.0 ± 2.16	90.0 ± 2.16	178.53 ± 4.38	365.61 ± 24.18	55583
S1	70	-	352.33 ± 12.5	170.33 ± 15.8	128.0 ± 16.97	5.67 ± 2.49	126.0 ± 35.39	100.29 ± 53.06	277.42 ± 13.65	45980
S1	50	-	347.33 ± 4.64	84.33 ± 5.79	184.33 ± 12.68	4.67 ± 2.05	135.0 ± 5.72	192.81 ± 43.39	176.64 ± 22.28	31321

Table 5.3: Online LH-MST vs offline LH-MST. Performance of the online version of the LH-MST compared to the offline version of the LH-MST.

5.5.2.1 Results of Offline LH-MST

Table 5.1 summarizes the results of our evaluation of using different spanning trees in place of the MST. As it can be seen, using LH-IS provides the closest approximation of IS. The majority of clusters are represented with the same medoid (87% or 282 medoids). In 28 cases the algorithm converges to a different medoid which is part of the same cluster. Figures 5.7 (e-f) and 5.8 (e-f) show exemplary samples of the result of LH-IS and IS. Most picked medoids are the same, in case of divergence the images are still similar. This is also expressed by the low rank difference of 18.71. In 4 cases the medoid does not have a direct correspondence, i.e. the medoid is not part of the support set of MST. Such an example is shown in Figure 5.5.

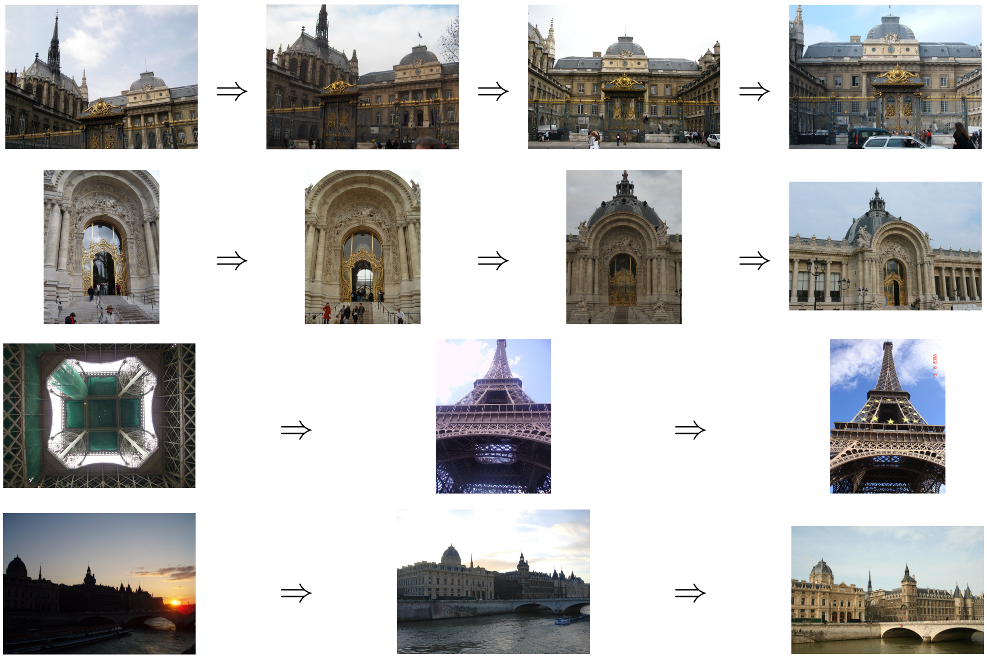
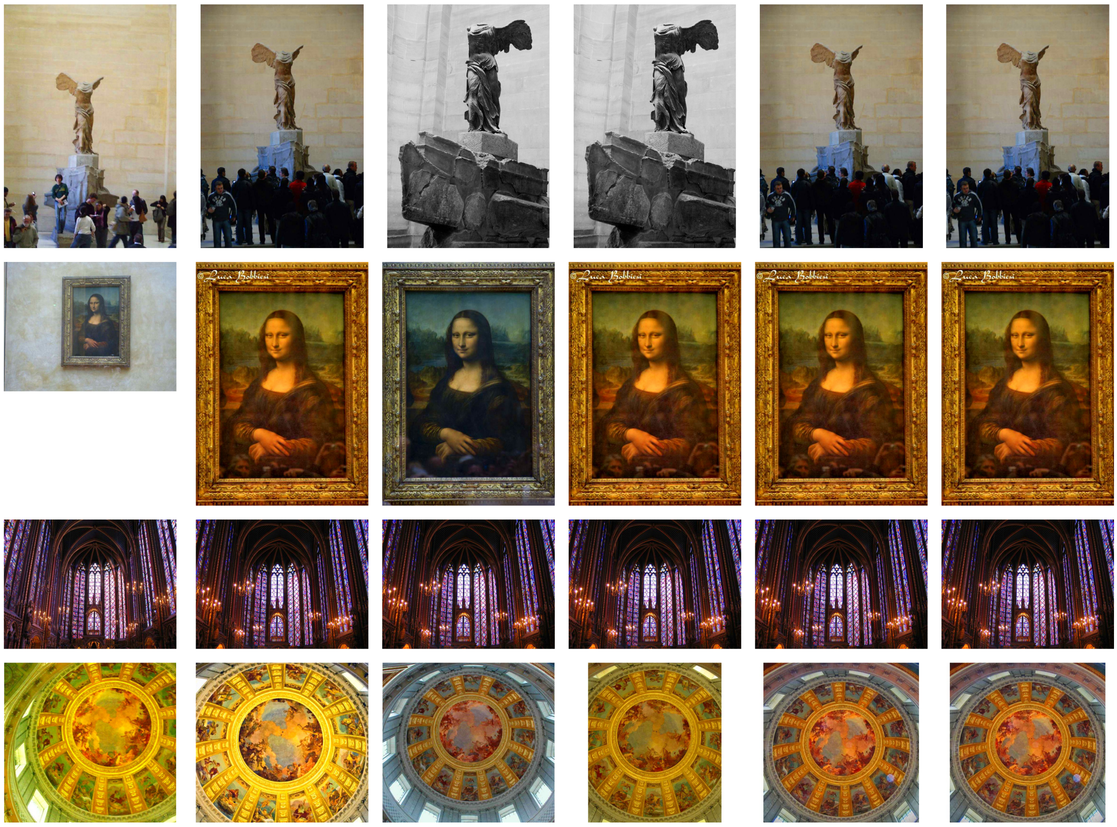


Figure 5.6: Examples of IS shift sequences (series of medoids until convergence) using LH-MST.

Interestingly, the second most similar result to IS is achieved by RAN-IS, confirming that in principle any tree could perform well as long as it is not biased towards certain nodes. An important property of 'good' spanning trees is to allow IS to shift away from its initial seed. SPT-IS is performing worst with almost half of its initial seeds unable to shift to a better medoid. It is unsuitable by design as it is biased towards the root by keeping the edges that minimize the distance to the root.



Initial Seed (a)

LH-IS* 50/50 (b)

LH-IS* 70/30 (c)

LH-IS* 90/10 (d)

LH-IS 100 (e)

IS 100 (f)

Figure 5.7: Here the figure shows various example medoids resulting from different splits (Scenario 1). In some cases the medoids are different, but visually very similar images are selected (e.g. Mona Lisa in LH-IS, 70/30 split).

Summary. Compared to the original IS, LH-IS converges to exactly the same medoids in most cases (87% of the clusters); 97% of the medoids depict the same object with only minor viewpoint/color changes. For the result in Table 5.1, the runtime of IS using LH-MST (∼1 day) is 5 times faster than using MST (∼5 days). LH-MST is therefore not only a suitable alternative in terms of quality but is also considerably faster than MST.

5.5.3 Evaluation of online LH-MST

As has been shown in the previous section, LH-MST is suitable for performing IS. Besides the speed advantage in an offline setting, LH-MST also offers the ability to be used with online updates of the image database (Section 5.4.2).

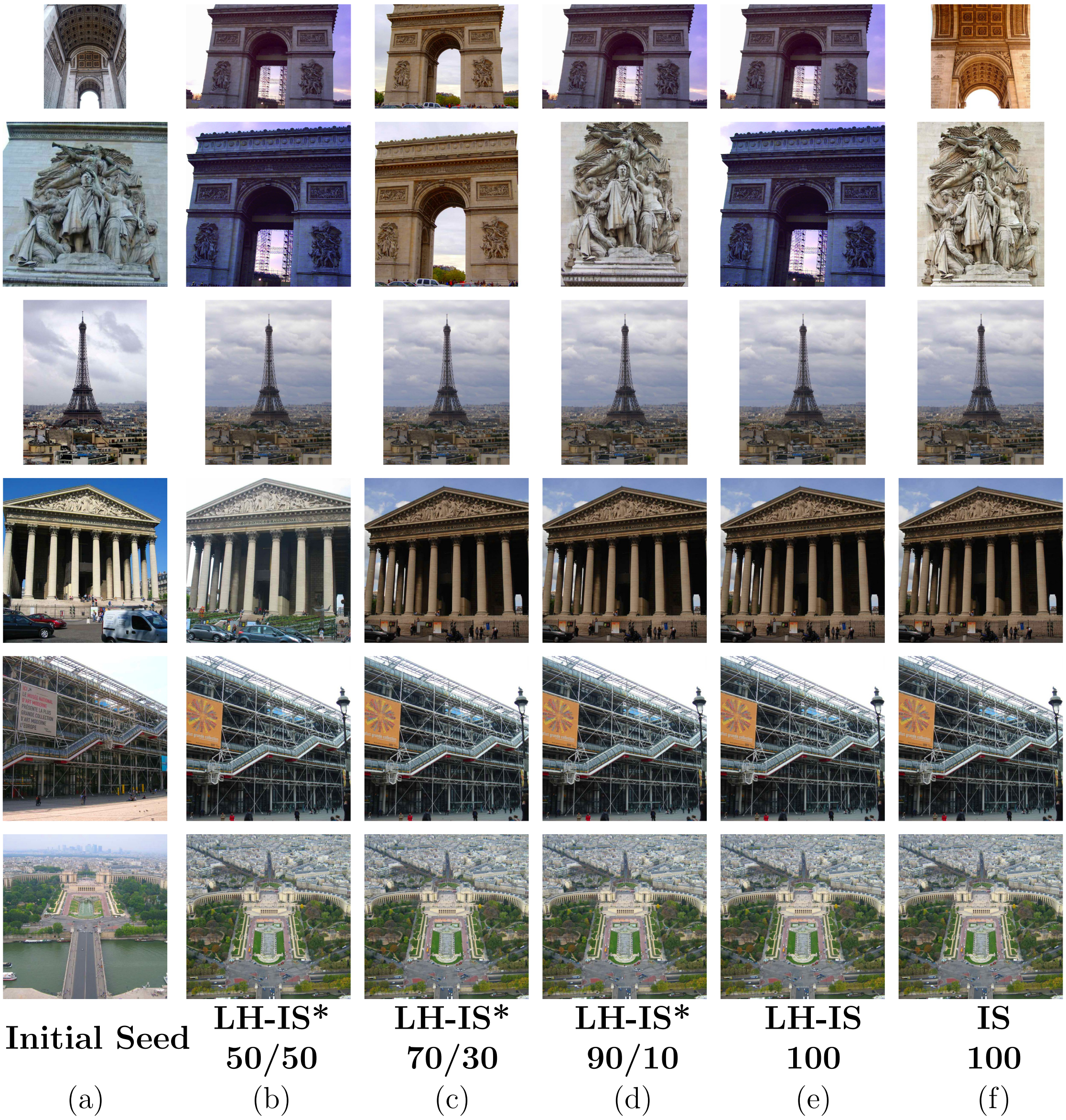


Figure 5.8: Here we shows additional example medoids resulting from different splits (Scenario 1).

In the following, we compare the online version of LH-IS with the offline/batch version and with the original IS. For each experiment, we split the dataset into initial set and update set. After performing IS using LH-MST on this initial set, we add the remaining data via online update. All experiments are repeated 3 times to account for different random splits. To keep the results comparable, we enforce the seed images to be contained in the initial set. Two basic scenarios are considered:
Scenario 1. The split is performed randomly. This scenario mirrors real world setting.

Scenario 2. First, the dataset is split randomly. Second, images that represent medoids in offline LH-IS are shifted into the update set. As such no seed has converged to the "ground truth" cluster prior to the update. This setting gives further insights into the ability of our online version to perform useful shifts during the update step.

Comparison of online LH-IS to the batch LH-IS. (Table 5.3) Table 5.3 compares offline LH-IS to various settings of online LH-IS. For both scenarios, we report results using different ratios of images in the initial and update set (from 50/50 to 90/10).

As it can be seen from Table 5.3 the online version of LH-IS offers comparable results to the offline version. As expected, the selected medoids are closer with a smaller update set. Even in the case of a 50/50 split, many of the resulting medoids agree with offline LH-IS. Note that such a set-up is quite unrealistic considering the drastic change of the database (doubles in size). Yet, in Scenario 1 on average still 234 of the medoids agree with the offline LH-IS compared to 84 if no update is performed. For Scenario 2, although none of the correct medoids were in the initial database ($\mathbb{M}_A = 0$), more than 2/3 of them are recovered through the update (in the 70/30 split) confirming that our update step is able to add meaningful connections in the graph and to also shift medoids to their rightful place.

Comparison of online LH-IS to the original IS. (Table 5.1) We evaluated above the performance of online LH-IS compared to the offline/batch version LH-IS (Table 5.3). Additionally, we showed that the performance of batch LH-IS is very similar to the original IS (Weyand and Leibe, 2011) (Table 5.1). Here we will directly compare the online LH-IS to the original IS rather than offline LH-IS. The experiments show that the agreement of medoids between online and offline LH-IS (Table 5.3) is slightly better than between online LH-IS and the original IS (Table 5.4). At the same time, the difference in *Rank* shows an opposing trend: Even though more medoids disagree, the rank difference is smaller. This indicates that disagreeing medoids are closer to each other and more likely to depict a similar view of the same iconoid.

Qualitative Results. In Figures 5.7 and 5.8 we show and compare various example medoid images resulting from performing online LH-IS on different splits (Scenario 1), offline LH-IS and original IS. All three methods agree on the medoid in most cases. In the cases of disagreement (e.g., in Fig. 5.7: 4th-row where all online versions result to a different medoid, 2nd-row Mona Lisa in LH-IS, 70/30 split) the chosen medoids are similar in appearance and viewpoint. The overlap distance of all proposed methods is based only on the region overlap and disregards differences e.g., in color (due to the used SIFT features) like in the Mona Lisa example. The example medoid in the second row of Figure 5.8, showing the complete Arc de Triomphe is a more favorable choice to many applications, than ones depicting a small detail.

Figure 5.9 shows instances where the algorithms disagree in the selection of the exact medoid while showing the same object. Figure 5.10 finally shows some of the rare cases where the medoids show different object instances. As can be observed, in these cases the medoids of both algorithms are rather unspecific. The clusters are still valid since they contain similar to the medoid images as seen from the cluster members in Figure 5.11 of the top left medoid of Figure 5.10.

Summary. In the realistic scenario of a random initialization of 90% and an update of 10% using online LH-IS, 82% of the medoids are exactly the same as in offline LH-IS; 77% when compared to IS; 98% of the clusters depict the same object. The runtime of the LH-IS update is ∼3 hours, 8 times faster than re-computing LH-MST from scratch; 40 times faster than rerunning the original IS algorithm.

5.6 Conclusion

In this chapter we introduced the Limited Horizon Minimum Spanning Tree (LH-MST), an approximation of the well-known Minimum Spanning Tree (MST). In contrast to the MST, the LH-MST can be constructed much faster and has the additional property of allowing online updates of the tree structure, even for large changes in the image database. Yet various experiments have shown that this approximation is very close to the original algorithm. Our novel LH-MST can potentially be used in many algorithms which involve a costly Minimum Spanning Tree (MST) calculation. Finally, we demonstrated its applicability for two distinct methods of landmark discovery in large datasets: Single-Link Agglomerative and Iconoid Shift clustering.

MST			$\lvert\mathbb{M}\rvert$	$\lvert\mathbb{M}_A\rvert$	$\lvert\mathbb{M}_O\rvert$	$\lvert\mathbb{M}_U\rvert$	$\lvert\mathbb{S}\rvert$	*Rank*	$\lvert\mathbb{S}_{sup}\rvert$	*DB cover*
	Split									
	100	-	324	-	-	-	90	-	380.42	73249
S1	90	10	327.33 ± 1.89	248.33 ± 3.86	59.67 ± 5.44	3.67 ± 1.89	90.0 ± 2.16	42.69 ± 11.11	424.04 ± 32.72	61716
S1	70	30	327.0 ± 7.79	246.0 ± 18.83	64.0 ± 19.3	2.33 ± 1.25	86.33 ± 0.94	34.33 ± 10.6	435.14 ± 28.13	69689
S1	50	50	325.33 ± 2.49	220.0 ± 5.72	92.33 ± 8.73	1.67 ± 0.47	91.0 ± 1.41	42.52 ± 28.95	445.94 ± 65.38	68570
S2	70	30	340.33 ± 2.49	212.67 ± 15.84	112.67 ± 13.3	4.67 ± 2.49	87.33 ± 10.87	44.2 ± 15.47	423.41 ± 31.73	69659
S2	50	50	338.0 ± 4.32	169.67 ± 9.29	149.0 ± 13.59	3.33 ± 1.25	83.33 ± 3.86	123.41 ± 82.06	390.75 ± 23.2	67246
S1	90	-	329.67 ± 0.47	240.33 ± 3.77	68.67 ± 4.64	4.0 ± 2.16	90.0 ± 2.16	37.76 ± 10.39	365.61 ± 24.18	55583
S1	70	-	352.33 ± 12.5	169.67 ± 15.33	120.67 ± 13.91	5.67 ± 2.49	126.0 ± 35.39	38.15 ± 13.88	277.42 ± 13.65	45980
S1	50	-	347.33 ± 4.64	80.33 ± 6.24	184.0 ± 13.06	4.0 ± 2.16	135.0 ± 5.72	116.68 ± 69.14	176.64 ± 22.28	31321

Table 5.4: Performance of the online version of the LH-MST compared to the offline version of the original IS algorithm that uses an MST.

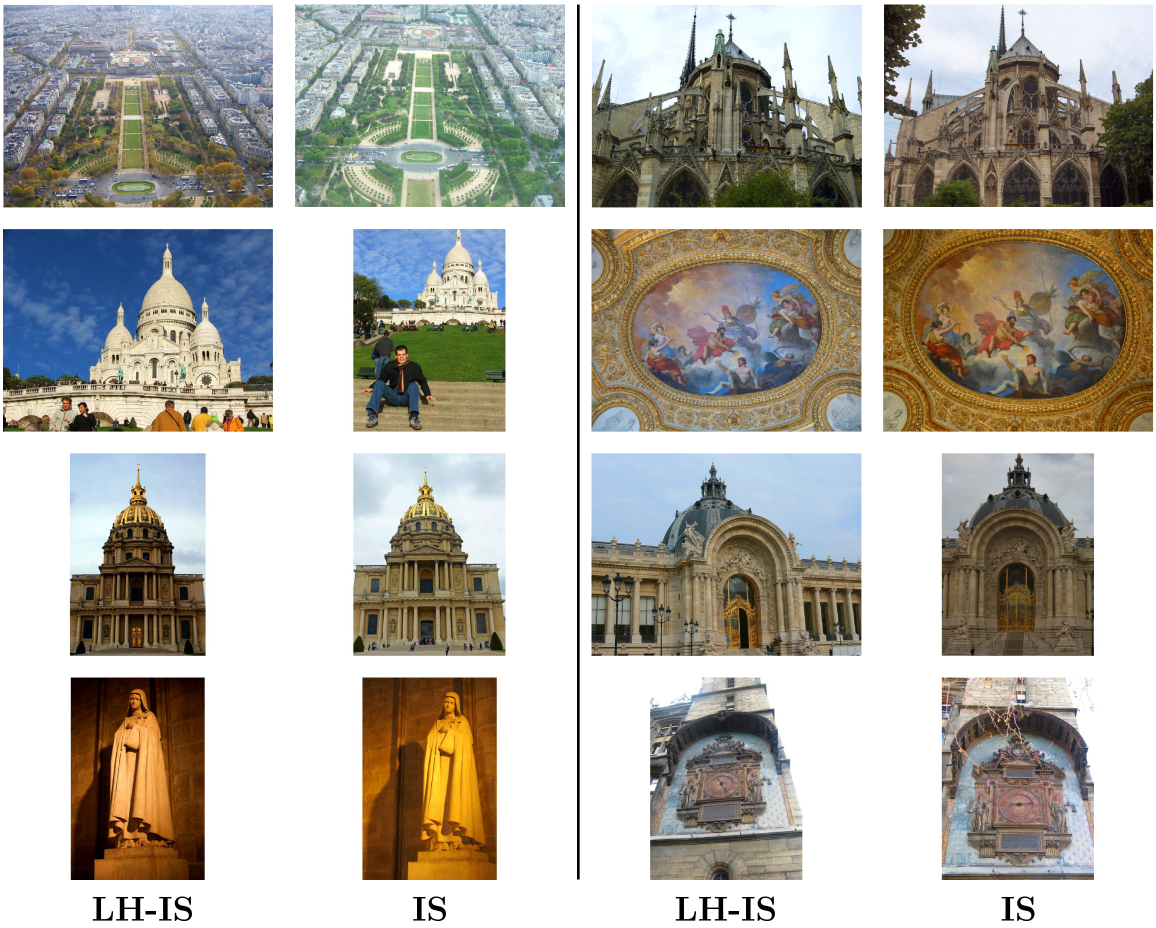


Figure 5.9: Cluster centers of offline LH-IS and original IS (Weyand and Leibe, 2011). Examples of cluster centers where LH-MST converges to a *different medoid* than IS. As it can be seen the quality of the result is the *rather the same*. In most situations there is only minor color or camera pose differences.

Acknowledgements. We thank Tobias Weyand for his invaluable input regarding the IS framework. This work is supported by the Google Faculty Research Award.

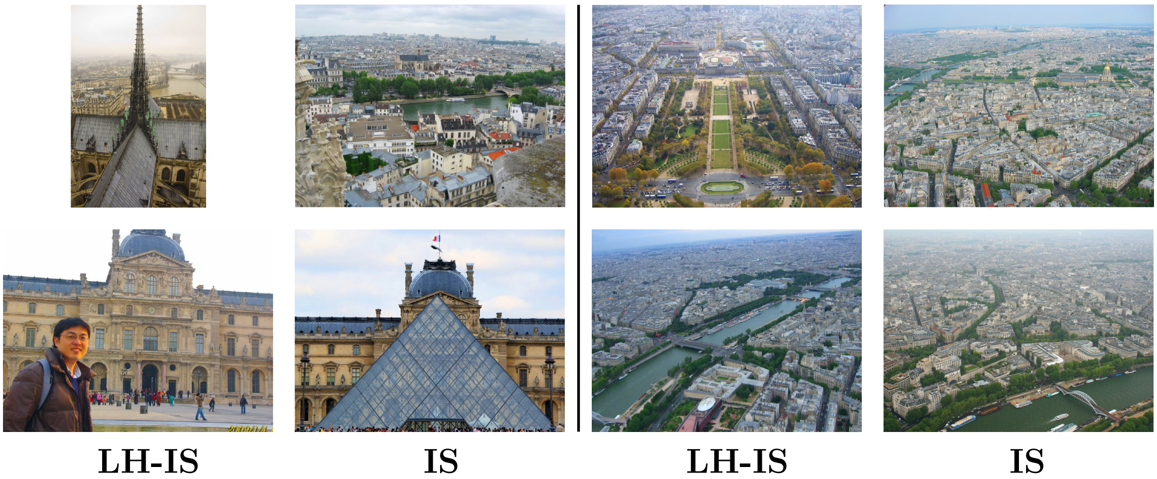


Figure 5.10: Cluster centers of offline LH-IS and original IS (Weyand and Leibe, 2011). Examples of cluster centers where LH-MST converges to a *different medoid* than IS but the object depicted in the image is *very different*.

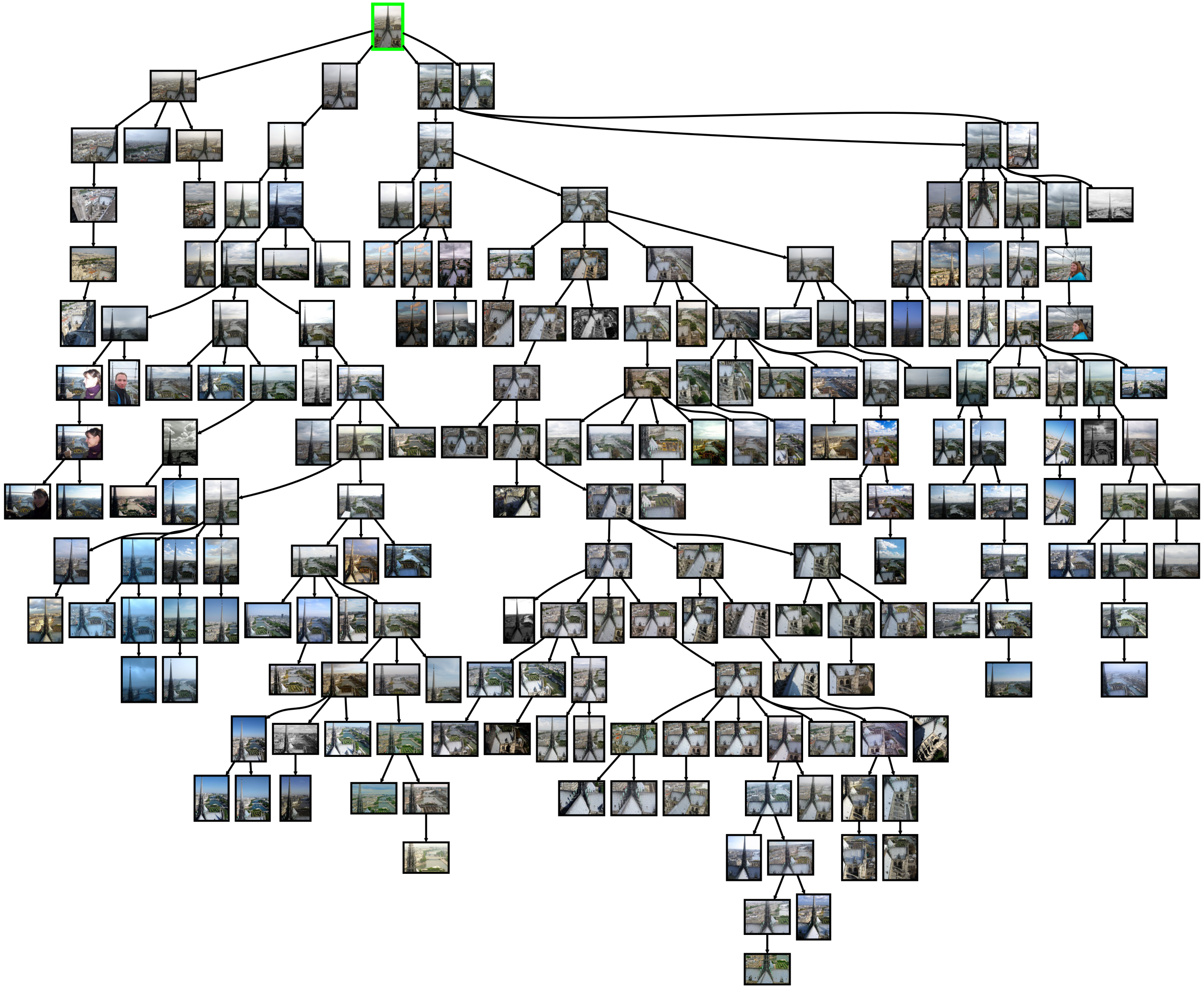


Figure 5.11: Example of cluster members of top left LH-MST medoid in Figure 5.10.

II

3D Scene Understanding

6

Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds

Deep learning approaches have made tremendous progress in the field of semantic segmentation over the past few years. However, most current approaches operate in the 2D image space. Direct semantic segmentation of unstructured 3D point clouds is still an open research problem. The recently proposed PointNet architecture presents an interesting step ahead in that it can operate on unstructured point clouds, achieving encouraging segmentation results. However, it subdivides the input points into a grid of blocks and processes each such block individually. In this chapter, we investigate the question how such an architecture can be extended to incorporate larger-scale spatial context. We build upon PointNet and propose two extensions that enlarge the receptive field over the 3D scene. We evaluate the proposed strategies on challenging indoor and outdoor datasets and show improved results in both scenarios.

6.1 Introduction

Semantic segmentation is an important capability for intelligent vehicles, such as autonomous cars or mobile robots. Identifying the semantic meaning of the observed 3D structure around the vehicle is a prerequisite for solving subsequent tasks such as navigation or reconstruction (Engelmann *et al.*, 2016, 2017). Consequently, the problem has attracted a lot of attention, and notable successes have been achieved

---

[0] This chapter is based on Kontogianni* et al. (2017) originally published at ICCVW'17 and is joint work together with Francis Engelmann. It includes my ideas on recurrent consolidation units applied to grid neighborhoods and Francis Engelmann's ideas on multi-scale input blocks and sequential consolidation units.

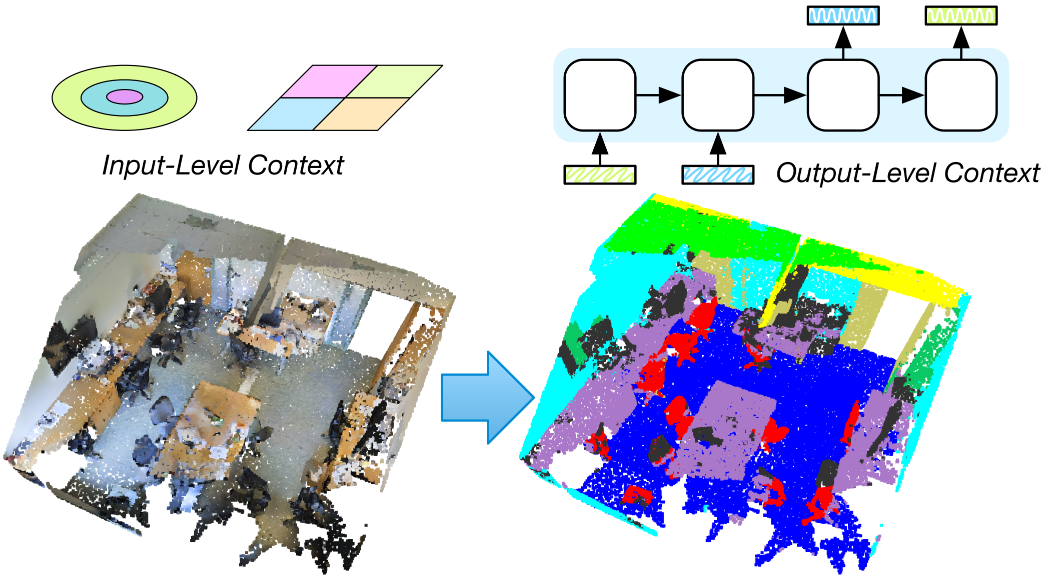


Figure 6.1: We explore mechanisms to extend the spatial context for semantic segmentation of 3D point clouds.

with the help of deep learning techniques. However, most state-of-the-art semantic segmentation approaches operate on 2D images, which naturally lend themselves to processing with Convolutional Neural Networks (CNNs) (Chen *et al.*, 2017; Long *et al.*, 2015; Noh *et al.*, 2015; Wu *et al.*, 2016).

Processing unstructured 3D point clouds, such as those obtained from LiDAR or stereo sensors, is a much harder problem, and it is only recently that first successful deep learning approaches have been proposed for this task (Huang and You, 2016; Maturana and Scherer, 2015; Qi *et al.*, 2017b; Yi *et al.*, 2016). Such point clouds can be obtained from LiDAR sensors mounted on top of a recording vehicle or they can be obtained from visual SLAM approaches operating on the vehicle's cameras (Kasyanov *et al.*, 2017). Finding approaches that can directly operate on point cloud data is highly desirable, since it avoids costly preprocessing and format conversion steps. However, the question what the best network architecture is to process unstructured 3D point clouds is still largely open.

In this work, we take inspiration from the recent PointNet work by Qi *et al.* (2017b), which currently defines the state of the art in 3D semantic segmentation. PointNet learns a higher dimensional spatial feature representation for each 3D point and then aggregates all the points within a small 3D volume (typically an occupancy grid cell) in order to bring in some form of 3D neighborhood context. However, this neighborhood context is very restricted, as each grid cell is processed independently.

In this paper, we investigate possible mechanisms to incorporate context into a point cloud processing architecture. We focus on spatial context, which has been identified as being very important for semantic segmentation (Mottaghi *et al.*, 2014; Pohlen *et al.*, 2017). We introduce two mechanisms to add spatial context to an existing PointNet. The first mechanism incorporates neighborhood information by processing input data from multiple scales or multiple adjacent regions together (input-level context). The second mechanism operates on the estimated point descriptors and aims at consolidating them by exchanging information over a larger spatial neighborhood (output-level context). For both mechanisms, we explore several possible realizations and compare them experimentally. As our results show, both mechanisms improve semantic segmentation quality.

Contributions. The key contributions of our work can be summarized as follows:

1. We present two mechanisms that can be used to incorporate spatial context into semantic 3D point cloud segmentation.
2. We show how these mechanisms can be incorporated into the PointNet pipeline.
3. We verify experimentally that our proposed extensions achieve improved results on challenging indoor and outdoor datasets.

6.2 Related Work

Unstructured Point Clouds. A varied number of sensors and setups exist which help to obtain unstructured point clouds: areal data from airborne laser scanners, laser scanners mounted on dynamic setups in a push-broom configuration (Maddern *et al.*, 2017), rotating lasers e.g., Velodyne (Geiger *et al.*, 2013), or static lasers (Hackel *et al.*, 2017). Additionally, indoor spaces can be scanned using devices such as the Microsoft Kinect (Silberman *et al.*, 2012) or Matterport cameras (Armeni *et al.*, 2016). All these devices produce point clouds of different quality and density. We apply our method to indoor data from (Armeni *et al.*, 2016) and to synthetic urban outdoor data from (Gaidon *et al.*, 2016).

Traditional Methods. Hackel *et al.* (2016) use traditional random forest classifiers with 3D features (without color). Their method is based on eigenvalues and eigenvectors of covariance tensors created by the nearest neighbors of the points. Their main contribution is an efficient approximate nearest neighbors computation at different scales. Munoz *et al.* (2008) follow a similar approach but replace the random forest classifier with an associative Markov network. Random forest classifiers are also used in Zhang *et al.* (2015) to classify data from 2D images and 3D point clouds, which they later fuse. Similarly, Xu *et al.* (2013) fuse camera and LiDAR sensor data. Xiong *et al.* (2011a) propose a sequential parsing procedure that learns the spatial relationships of objects. Lai *et al.* (2014) introduce a hierarchical sparse coding technique for learning features from synthetic data. Vosselman (2013) combines multiple segmentation and post-processing methods to achieve useful point cloud segmentations.

Deep-learning Methods. In a deep learning context, point clouds can be represented in a regular volumetric grid in order to apply 3D convolutions (Huang and You, 2016; Maturana and Scherer, 2015). Alternatively, 3D points can be mapped to a 2D representation followed by 2D convolutions (Qi *et al.*, 2016). In Boulch *et al.* (2017), the authors are performing 2D convolutions in 2D snapshots of a 3D point cloud and then project the labels back to 3D space. In Ondruska *et al.* (2016) a deep learning framework learns semantic segmentation by tracking point clouds. Yi *et al.* (2016) use spectral CNNs on 3D models represented as shape graphs for shape part segmentation. Recent methods operate directly on raw point clouds with KD-trees (Klokov and Lempitsky, 2017) or fully convolutional layers (Qi *et al.*, 2017b).

6.3 Method

In this section we start by reviewing the PointNet model (Qi *et al.*, 2017b), then we introduce our mechanisms of extending context and finish by describing our two exemplary architectures.

6.3.1 PointNet

PointNet (Qi *et al.*, 2017b) is a deep neural network that, when used for semantic segmentation, takes as input a point cloud and outputs the per point semantic class labels. First, it splits a point cloud into 3D blocks, then it takes N points inside a block and after a series of Multi-Layer-Perceptrons (MLP) per point, the points are mapped into a higher dimensional space D', these are called local *point-features*. Max-pooling is applied to aggregate information from all the points resulting in a common *global feature* invariant to input permutations. The global feature is then concatenated with all the point-features. After another series of MLPs these combined features are used to predict the M output class scores. Figure 6.2 shows a simplified model.

Caveats. The global features in PointNet summarize the context of a single block (block-feature), as a result the aggregated information is passed only among points

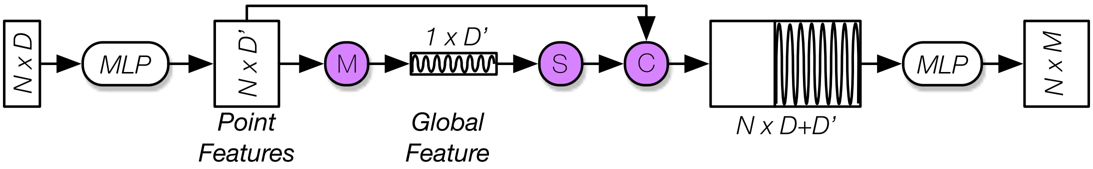


Figure 6.2: Simplified PointNet Architecture. In this work, we build upon the PointNet architecture for semantic segmentation. In short, it computes a global feature which summarizes a set of input points. Specifically, the network takes N points as input, applies a series of multi-layer-perceptrons transformations and aggregates the point features by max pooling them into a global feature. Global and local features are concatenated and the per point class scores are returned. (MLP): Multi-Layer-Perception, (M): Max-Pool, (S): Vertical Stack, (C): Concatenate. See text and Qi *et al.* (2017b) for more details.

inside the same block.

Context outside a block is equally important and could help make more informed class label predictions. Therefore, we introduce two mechanisms to add context: **input-level context** – which operates directly on the input point clouds – and **output-level context** – which consolidates the output from the input-level context.

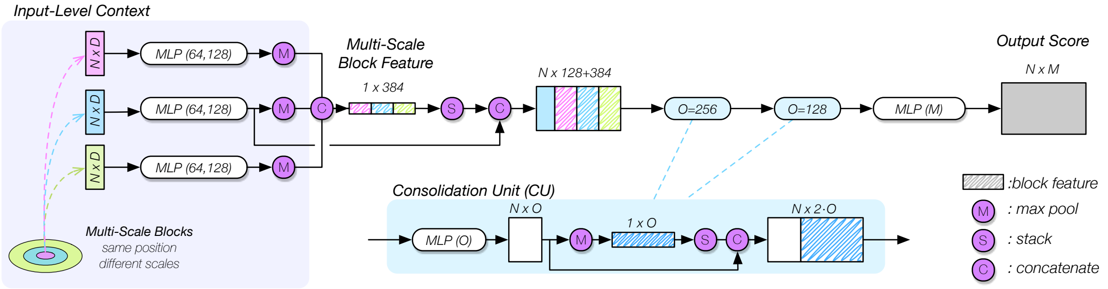


Figure 6.3: Architecture with multi-scale input blocks and consolidation units (MS-CU). The network takes as input three blocks from multiple scales, each one containing N D-dimensional points. Separately, for each scale, it learns a block-feature similarly to the PointNet mechanism. The concatenated block-features are appended to the input-features and then transformed by a sequence of consolidation units (see Section 6.3.3). The network outputs per point scores. Shaded fields represent block-features.

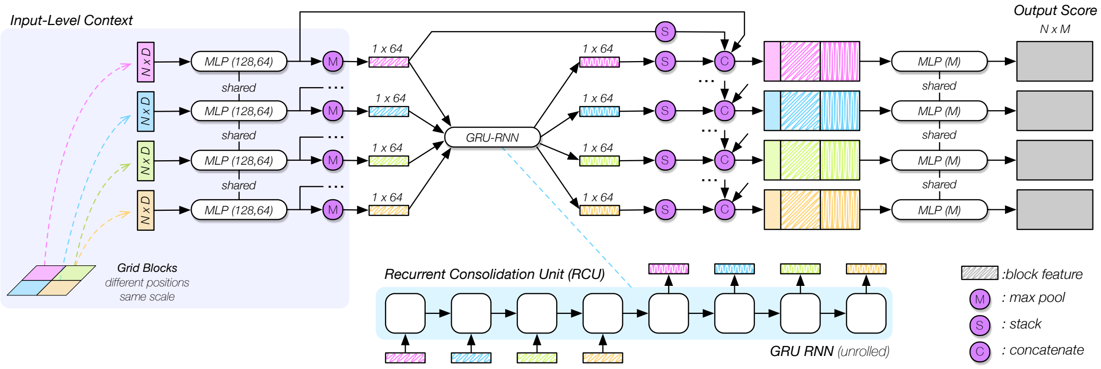


Figure 6.4: Architecture with grid input blocks and a recurrent consolidation unit (GB-RCU). The network takes as input *four blocks* from a grid structure, each one containing N D-dimensional points. It then learns the block-features using the same MLP weights for each block. All block-features are passed through a recurrent consolidation unit (see Section 6.3.3) which shares the spatial context among all blocks and returns updated block-features. The updated block-features are appended to the input-features together with the original block-features and used to compute the output per point scores. Shaded fields represent block-features. Some skip-connections are omitted for clarity.

6.3.2 Input-Level Context

In this straightforward addition, we increase the context of the network by considering a group of blocks simultaneously instead of one individual block at a time as done in PointNet. Context is shared among all blocks in a group. These groups of blocks are selected either from the same position but at multiple different scales (**Multi-Scale Blocks**, see Figure 6.3, left) or from neighboring cells in a regular grid (**Grid Blocks**, see Figure 6.4, left). For each input block, we compute a block-feature using the mechanism from PointNet. For the multi-scale version, we train a block-descriptor for each scale individually to obtain scale-dependent block-features. In the case of grid blocks, all block features are computed by a shared single-scale block-descriptor. In the end, both approaches output a set of block-features corresponding to the input blocks.

6.3.3 Output-Level Context

At this stage, we further consolidate the block-features obtained from the previous stage. Here, we differ between two consolidation approaches:

Consolidation Units (CU) consume a set of point features, transform them into a higher dimensional space using MLPs and apply max-pooling to generate a common block-feature which is again concatenated with each of the high dimensional input features (see Figure 6.3, blue box). This procedure is similar to the block-feature mechanism of PointNet. The key point is that CUs can be chained together into a sequence CUs forming a deeper network. The intuition behind this setup is as follows: In the beginning each point sees only its own features. After appending the block-features, each point is additionally informed about the features of its neighboring points. By applying CUs multiple times, this shared knowledge is reinforced.

Recurrent Consolidation Units (RCU) are the second type of context consolidation we employ. RCUs take as input a sequence of block-features originating from spatially nearby blocks and return a sequence of corresponding updated block-features. The core idea is to create block-features that take into consideration neighboring blocks as well. In more detail, RCUs are implemented as RNNs, specifically GRUs (Cho *et al.*, 2014), which are a simpler variation of standard LSTMs (Hochreiter and Schmidhuber, 1997). GRUs have the capability to learn long range dependencies. That range can either be over time (as in speech recognition) or over space as in our case. The cells of the unrolled GRU are connected in an unsynchronized many-to-many fashion (see Figure 6.4, blue box). This means that the updated block-features are returned only after the GRU has seen the whole input sequence of block-features. Intuitively, GRU retain relevant information about the scene in their internal memory and update it according to new observations. We use this memory mechanism to consolidate and share the information across all input blocks. For example, the decision about whether a point belongs to a chair is changed if the network remembers that it has seen a table further down in the room.

In the following, we describe two exemplary architectures which combine the previously introduced components. For those, we provide a detailed evaluation and report improved results in Section 8.4.

6.3.4 Multi-Scale (MS) Architecture

The full MS architecture is displayed in Figure 6.3. The learned block-features from the multi-scale blocks, (see Section 6.3.2) are concatenated into one **multi-scale block-feature**. This multi-scale block-feature is further concatenated with the transformed input point-features and passed through a series of CUs (see Section 6.3.3). Applying a final MLP results in output scores for each input point.

Specific for this architecture is the sampling procedure to select the positions of the multi-scale blocks: We randomly pick a D-dimensional point from the input point cloud as the center of the blocks and we group together N randomly selected points that fall within a specified radius. This procedure is repeated at the same point for multiple radii.

6.3.5 Grid (G) Architecture

Figure 6.4 shows the pipeline of the architecture with grid input blocks. It consists of the following components: The input level context is a group of four blocks from a 2x2 grid-neighborhood (see Section 6.3.2) is fed into a series of MLPs that transform the point features, with weights shared among all blocks. These block-features are passed to an RCU that updates the individual block-features with common context from all neighboring blocks. The updated block-features are then concatenated with the original block-features. They are then used, along with the local features, for class predictions. After a series of fully connected layers the output of class scores is computed for each point.

6.4 Experiments

For experimental evaluation, we compare our two architectures with PointNet (Qi *et al.*, 2017b), the current state-of-the-art semantic segmentation method directly operating on point clouds. We produce quantitative results for our models and the baseline on two challenging datasets: *Stanford Large-Scale 3D Indoor Spaces* (S3DIS) (Armeni *et al.*, 2016) and on *Virtual KITTI* (vKITTI) (Gaidon *et al.*, 2016). Additionally, we provide qualitative results on point clouds obtained from a Velodyne HDL-64E LiDAR scanner from the *KITTI* dataset (Geiger *et al.*, 2013). We will now describe these datasets in more detail.

Stanford Large Scale 3D Indoor Scenes. This dataset is composed of 6 different large scale indoor areas, mainly conference rooms, personal offices and open spaces. It contains dense 3D point clouds scanned using a *Matterport* camera. Each point is labeled with one of the 13 semantic classes listed in Table 7.4. Using the reference implementation of PointNet, we were able to reproduce the results reported by Qi *et al.* (2017b), see Table 6.4. Throughout the paper, we follow the same evaluation protocol used in Qi *et al.* (2017b), which is a 6-fold cross validation over all the areas.

Virtual KITTI. Due to the lack of semantically annotated large-scale outdoor datasets, we rely on the photo-realistic synthetic vKITTI dataset which closely mimics the real-world KITTI dataset. It consists of 5 different monocular video sequences in urban settings, fully annotated with depth and pixel-level semantic labels. In total there are 13 semantic class, listed in Table 6.2. For our purposes, we project the given 2D depth into 3D space to obtain semantically annotated 3D point clouds. Conveniently, this procedure results in point clouds that resemble the varying density of real-world point clouds obtained by Velodyne LiDAR scanners (see Figure 6.5). For test and training, we split the original sequences into 6 non-overlapping subsequences. The final train-test sets are created by choosing point clouds from each subsequence at regular time-intervals. For evaluation, we also follow the 6-fold cross validation protocol.

6.4.1 Evaluation Measures

As in Qi *et al.* (2017b), we evaluate on the intersection over union (IoU), the average per class accuracy and overall accuracy. Intersection over union is computed as:

$$IoU = \frac{TP}{TP + FP + FN} \tag{6.1}$$

where TP is the number of true positives, FP the number of false positives and FN the number of false negatives.

6.4.2 Quantitative Results

In this section, we analyze the effectiveness of the input-block schemes and the consolidation units exemplary on the two previously introduced models. As input features, we differentiate between geometry (XYZ) and geometry with color (XYZ+RGB).

Geometry with Color. (Table 6.4, Table 6.1) First, we compare the grid-blocks in combination with a recurrent consolidation block (G+RCU) to the original

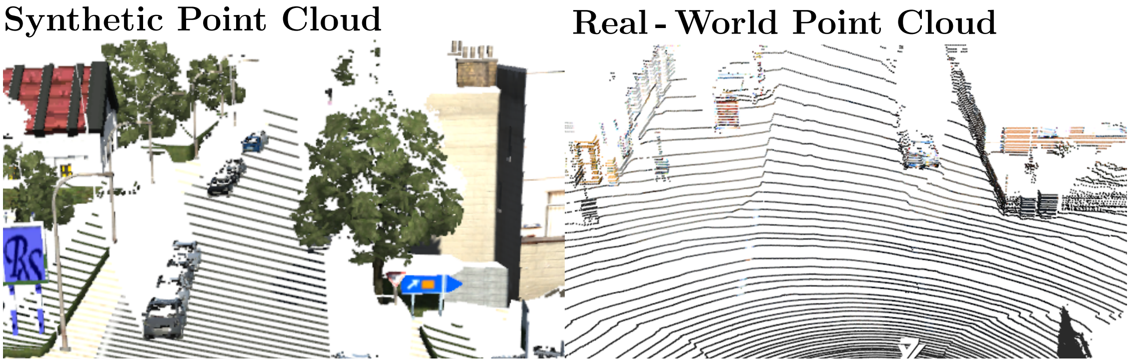


Figure 6.5: We train our network on synthetic point clouds generated from vKITTI (Gaidon *et al.*, 2016) (left) and apply it onto real-world Velodyne LiDAR point clouds (right). The structure and the varying density are comparable.

PointNet. Using the same evaluation setup as described in Qi *et al.* (2017b) we are able to show improved results over PointNet, see Table 6.4 and Table 6.1. This proves our hypothesis that RCU are able to convey context among blocks and thus improving results. During training, each room is split into blocks of 1x1 m on the ground plane. Each block extends over the whole room height. Neighboring blocks are overlapping by 0.5 meters in both directions. We select four blocks simultaneously from a 2x2 grid-neighborhood (see Figure 6.4, left). Each block contains 4096 points. The unrolled GRU is 8 cells long (4 input, 4 output). Its memory size is 64. During testing, the room is split into non-overlapping blocks and evaluated on all 2x2 groups of blocks. Each block is evaluated only once.

Next, we take a look at the multi-scale input block with consolidation units (MS-CU) model. To build the multi-scale blocks, we follow the process described in Section 6.3.4. As radii, we choose [0.25, 0.5, 1.0] m. As distance metric we choose the *Chebyshev*-distance which generates axis-aligned rectangular blocks. The middle scale block is equal to the PointNet block regarding shape and size.

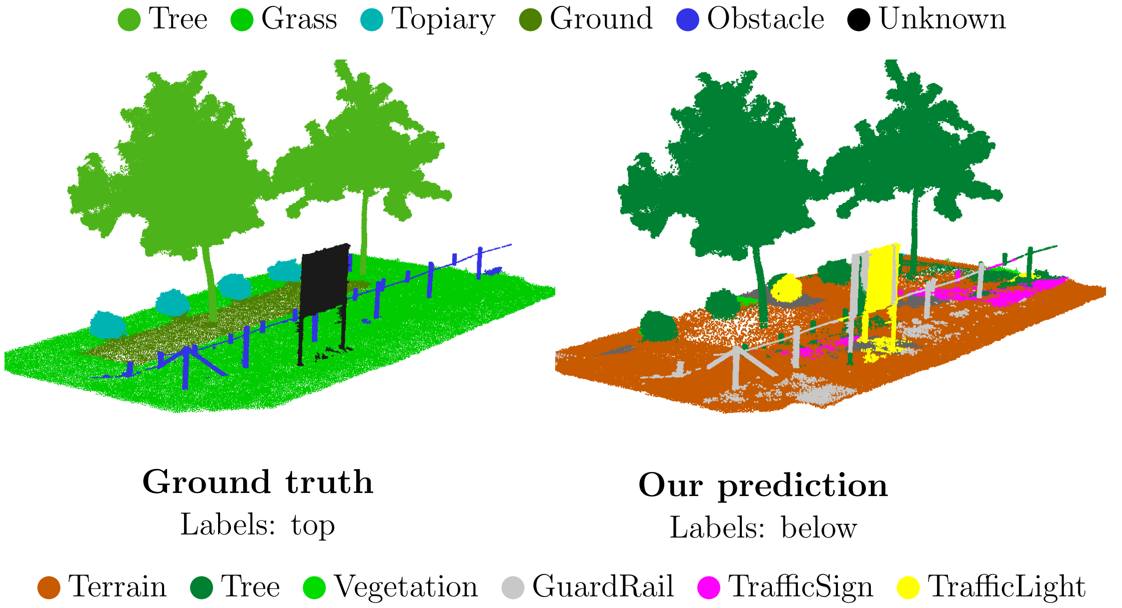
By using sampling (necessary for the multi-scale block construction), we diverge from the previous training procedure, so we re-run all experiments under these new conditions.

We validate the influence of each of the architecture's components by adding them one-by-one to our pipeline and evaluating after each step, see Table 6.4 and Table 6.1. First, we only consider the input-level context i.e., the multi-scale block feature (MS) as input to our pipeline while skipping the consolidation units. This shows some performance benefits over PointNet but not as much as one would expect considering the enlarged input context. Next, we take only single-scale input blocks and add one consolidation unit (SS+CU(1)). The results show that the CU outperforms the MS input blocks. It also shows that CUs provide a simple technique to boost the network's performance. Finally, we combine both the MS blocks and the CU while appending another CU to the network (MS+CC(2)). This full model is depicted in Figure 6.3.

Geometry only. (Table 6.2, 6.3) Until now, each input point was described by a 9-dimensional feature vector $[X, Y, Z, R, G, B, X', Y', Z']$ where $[X, Y, Z]$ are the spatial coordinates of a point, $[R, G, B]$ its color and $[X', Y', Z']$ the normalized coordinated based on the size of the environment, see Qi *et al.* (2017b) for further details. Without doubt, color is a very strong input feature in the context of semantic segmentation. In this section, we pose the question what will happen if no color information is available like it is the case with point clouds obtained from laser scanners. To simulate the missing colors, we simply discard the color information from the input feature and re-run the experiments. Table 6.2 and 6.3 show the obtained results. See caption for discussion of the results.

6.4.3 Qualitative Results

We present qualitative results of our models applied to indoor scenarios in Figure 6.8 and outdoor results in Figure 6.9 along with a short discussion. Additionally, we applied our pre-trained geometry-only model (vKITTI) to real-world laser data. The results are shown in Figure 6.7 and Figure 6.6.


● Tree ● Grass ● Topiary ● Ground ● Obstacle ● Unknown
● Terrain ● Tree ● Vegetation ● GuardRail ● TrafficSign ● TrafficLight

Ground truth
Labels: top

Our prediction
Labels: below

Figure 6.6: Qualitative results on 3DRMS'17 Challenge. We trained our model on vKITTI point clouds without color and applied it to the 3DRMS laser data. Training and test datasets do not have the same semantic labels. Despite that, common classes like trees are successfully segmented and plausible ones are given otherwise (e.g., terrain instead of grass, guardrail instead of obstacle).

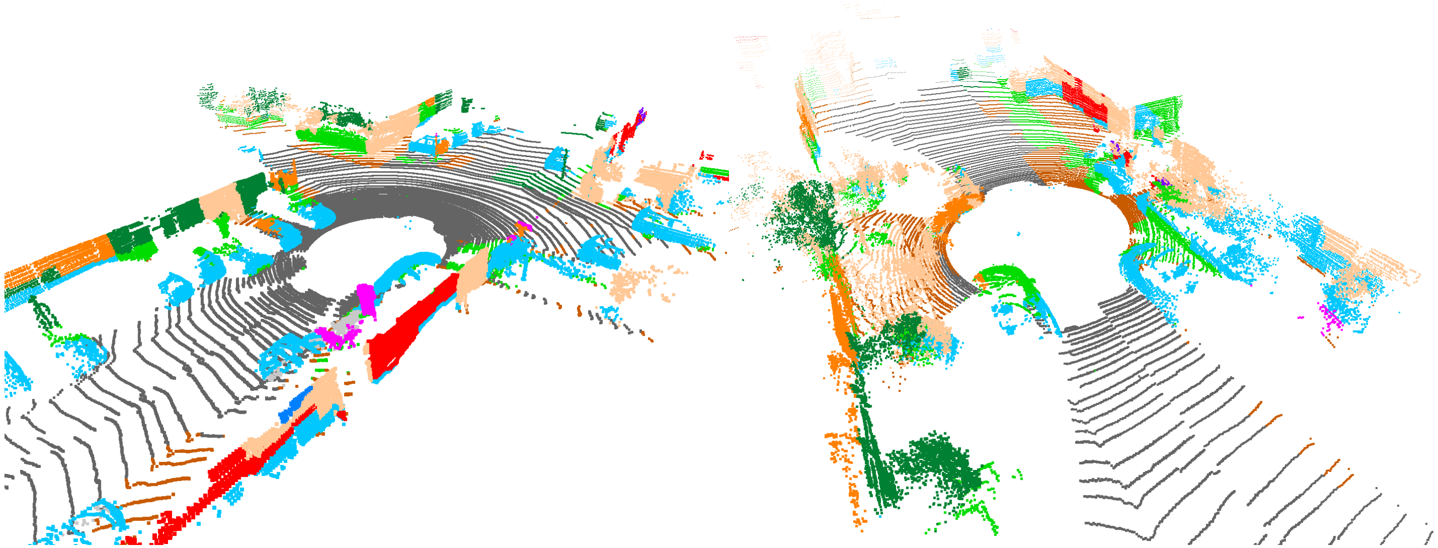


Figure 6.7: Qualitative results on laser point clouds. Dataset: Velodyne HDL-64E laser scans from KITTI Raw (Geiger *et al.*, 2013). We trained our model on vKITTI point clouds without color and applied it on real-world laser point clouds. So far, only classes like road, building and car give reasonable results.

S3DIS Dataset (Armeni *et al.*, 2016)	mIoU	*Ceiling*	*Floor*	*Wall*	*Beam*	*Column*	*Window*	*Door*	*Table*	*Chair*	*Sofa*	*Bookcase*	*Board*	*Clutter*
*PointNet (Qi *et al.*, 2017b)	43.5	81.5	86.7	64.8	29.4	16.3	39.1	48.1	52.5	42.5	5.4	37.6	30.4	31.4
*MS	44.4	82.2	86.9	64.2	33.8	22.8	43.3	52.0	51.0	38.6	9.2	36.1	23.6	33.7
*MS + RCU	45.5	83.6	86.9	**67.5**	**40.5**	17.1	37.0	48.8	**53.9**	42.3	6.8	**39.7**	**32.8**	34.2
*SS + CU(1)	45.9	**88.6**	92.6	66.3	36.2	23.6	47.1	51.2	50.2	36.9	**12.6**	33.7	22.7	35.3
*MS + CU(2)	**47.8**	**88.6**	**95.8**	67.3	36.9	**24.9**	**48.6**	**52.3**	51.9	**45.1**	10.6	36.8	24.7	**37.5**
PointNet (Qi *et al.*, 2017b)	47.6	88.0	88.7	**69.3**	42.4	23.1	47.5	**51.6**	54.1	42.0	**9.6**	38.2	29.4	35.2
G + RCU	**49.7**	**90.3**	**92.1**	67.9	**44.7**	**24.2**	**52.3**	51.2	**58.1**	**47.4**	6.9	**39.0**	**30.0**	**41.9**

Table 6.1: IoU per semantic class. S3DIS dataset with XYZ-RGB input features. We compare our models with different components against the original PointNet baseline. By adding different components, we can see an improvement of mean IoU. We obtain state-of-the-art results in mean IoU and all individual class IoU. *Entries marked with * use random sampling for input block selection instead of discrete positions on a regular grid.*

S3DIS Dataset (Armeni *et al.*, 2016)	mIoU	*Ceiling*	*Floor*	*Wall*	*Beam*	*Column*	*Window*	*Door*	*Table*	*Chair*	*Sofa*	*Bookcase*	*Board*	*Clutter*
*PointNet (Qi *et al.*, 2017b)	40.0	84.0	87.2	57.9	37.0	19.6	**29.3**	35.3	**51.6**	42.4	11.6	26.4	12.5	25.5
*MS + CU(2)	**43.0**	**86.5**	**94.9**	**58.8**	**37.7**	**25.6**	28.8	**36.7**	47.2	**46.1**	**18.7**	**30.0**	**16.8**	**31.2**

vKITTI Dataset (Gaidon *et al.*, 2016)	mean IoU	*Terrain*	*Tree*	*Vegetation*	*Building*	*Road*	*GuardRail*	*TrafficSign*	*TrafficLight*	*Pole*	*Misc*	*Truck*	*Car*	*Van*
*PointNet (Qi *et al.*, 2017b)	17.9	32.9	76.4	11.9	17.7	49.9	3.6	2.8	3.7	3.5	0.7	1.5	25.1	3.4
*MS + CU(2)	**26.4**	**38.9**	**87.1**	**14.6**	**44.0**	**58.4**	**12.4**	**9.4**	**10.6**	**5.3**	**2.2**	**3.6**	**43.0**	**13.3**

Table 6.2: IoU per semantic class. S3DIS and vKITTI datasets both with XYZ input features (no color). Our methods not only outperform PointNet consistently on two datasets, the improvements in mean IoU are also more considerable when no color is available. This suggests that our network architectures are able to learn improved geometric features and are more robust to varying point densities as they occur in the outdoor vKITTI dataset.

6.5 Conclusion

In this work, we investigated the question how to incorporate spatial context into a neural network architecture for 3D semantic segmentation. Building upon PointNet, we proposed two extension (Input-level context and Output-level context) which we successfully applied onto indoor and outdoor datasets. Still, numerous other combinations remain possible. The full exploration of the design space is left for future work.

	mean IoU	overall accuracy	avg. class accuracy
S3DIS Dataset (Armeni *et al.*, 2016) – no RGB			
*PointNet (Qi *et al.*, 2017b)	40.0	72.1	52.9
*MS + CU(2)	**43.0**	**75.4**	**55.2**
vKITTI Dataset (Gaidon *et al.*, 2016) – no RGB			
*PointNet (Qi *et al.*, 2017b)	17.9	63.3	29.9
*MS + CU(2)	**26.4**	**73.2**	**40.9**

Table 6.3: S3DIS and vKITTI datasets with only XYZ input features, without RGB. We show improved results on indoor (S3DIS) and outdoor (vKITTI) datasets. Our presented mechanisms are even more important when no color is available.

S3DIS Dataset (Armeni *et al.*, 2016) XYZ-RGB	mean IoU	overall accuracy	avg. class accuracy
*PointNet (Qi *et al.*, 2017b)	43.5	75.0	55.5
*MS	44.4	75.5	57.6
*MS + RCU	45.5	77.2	57.2
*SS + CU(1)	45.9	77.8	57.7
*MS + CU(2)	**47.8**	**79.2**	**59.7**
PointNet (Qi *et al.*, 2017b)	47.6	78.5	66.2
G + RCU	**49.7**	**81.1**	**66.4**

Table 6.4: S3DIS Dataset with XYZ-RGB input features. Comparison of different context expansion techniques on input- and output-level (see Sections 6.3.2–6.3.3). MS: Multi-Scale, SS: Single-Scale, G: Grid, CU: Consolidation Unit, RCU: Recurrent Consolidation Unit. *Entries marked with * use random sampling for input block selection instead of discrete positions on a regular grid.*

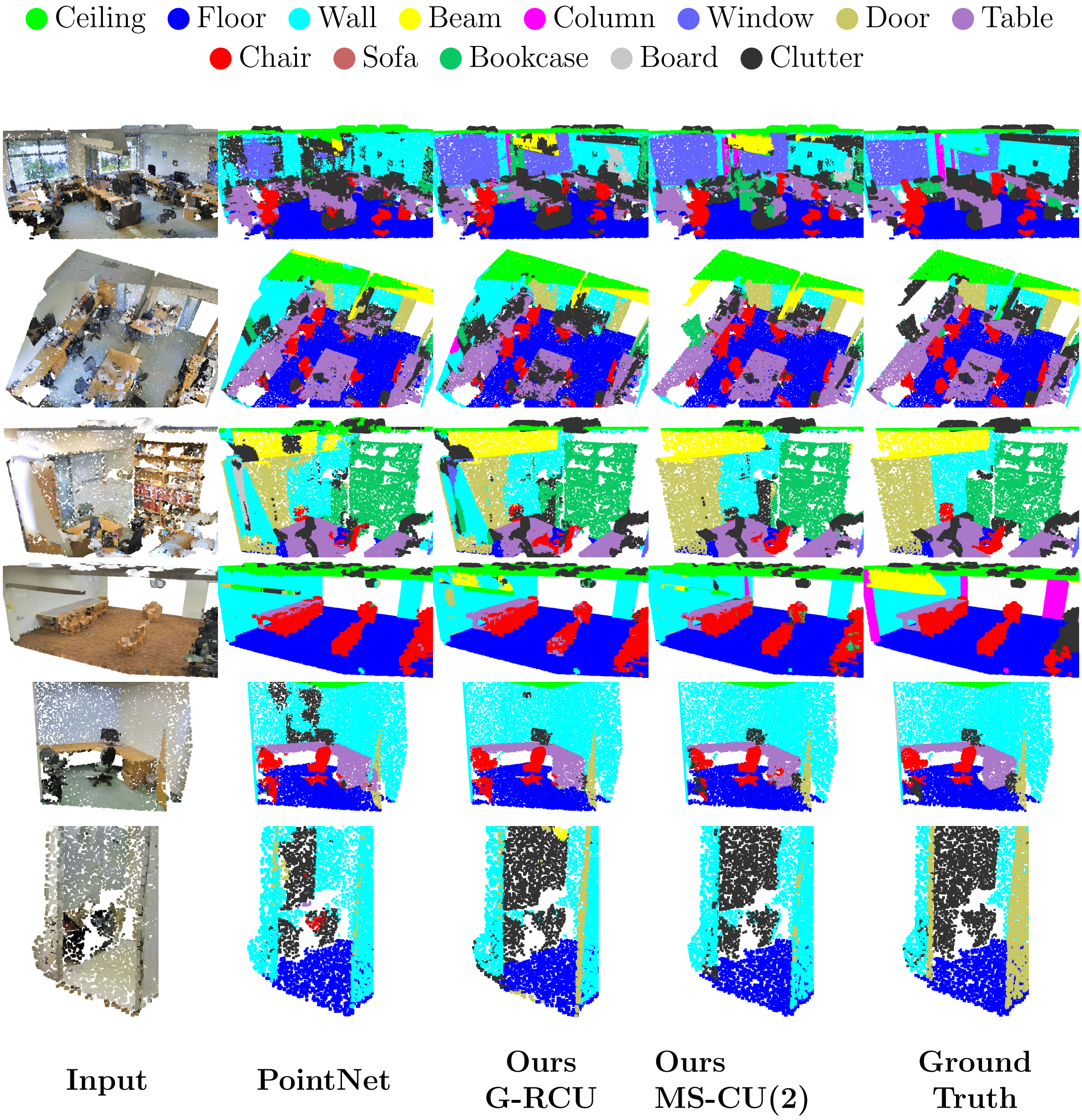
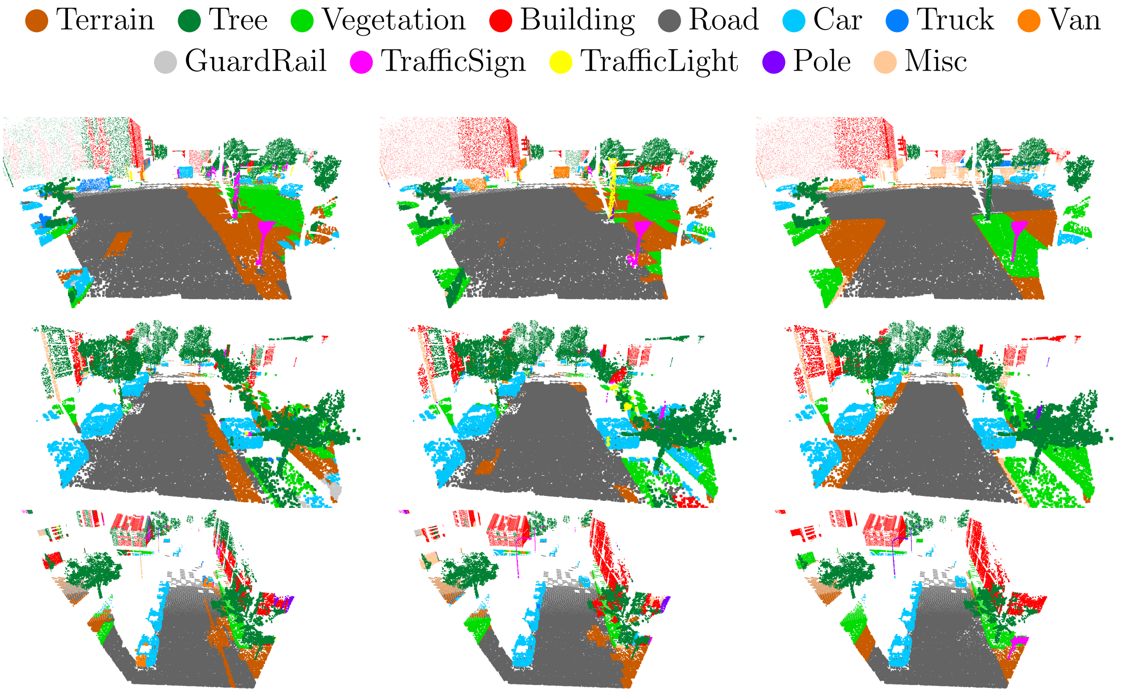


Figure 6.8: Indoor qualitative results. Dataset: S3DIS (Armeni *et al.*, 2016) with XYZ-RGB input features. From left to right: input point cloud, baseline method PointNet (Qi *et al.*, 2017b), our results using the G-RCU model (see Figure 6.4), our results using the MS-CU(2) model (see Figure 6.3), ground truth semantic labels. Our models produce more consistent and less noisy labels.



PointNet (Qi *et al.*, 2017b)

Ours MS-CU(2)

Ground Truth

Figure 6.9: Outdoor qualitative results. Dataset: Virtual KITTI (Gaidon *et al.*, 2016). Results were obtained using only XYZ coordinates as input, no color information was used. Left: baseline method PointNet. Center: our results using the MS-CU model as illustrated in Figure 6.3. Right: ground truth semantic labels. The outputs of our method are less fragmented (cars, houses) and finer structures like street lights and poles are recognized better.

7

Know What Your Neighbors Do: 3D Semantic Segmentation of Point Clouds

In this chapter, we present a deep learning architecture which addresses the problem of 3D semantic segmentation of unstructured point clouds. Compared to previous work, we introduce new grouping techniques which define *point neighborhoods*

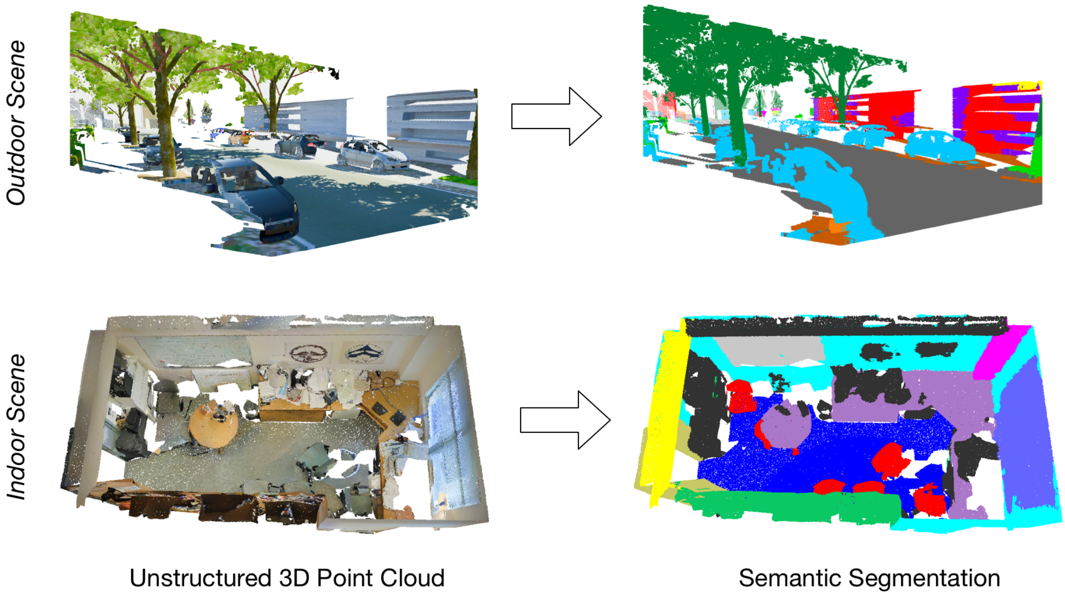


Figure 7.1: Our deep learning framework takes as input an unordered 3D point cloud *(left)* and predicts a semantic label for each point in the point cloud *(right)*. The main components of our approach are the definition of *point neighborhoods* in different feature spaces and the introduction of loss functions which help to refine the learned feature spaces. We apply our method to large scale indoor- and outdoor-scenes.

in the initial world space and the learned feature space. Neighborhoods are important as they allow to compute local or global point features depending on the spatial extend of the neighborhood. Point neighborhoods can be compared the filter size in traditional convolutions in the 2D image domain. Additionally, we incorporate dedicated loss functions to further structure the learned point feature space: the *pairwise distance loss* and the *centroid loss*. We show how to apply these mechanisms to the task of 3D semantic segmentation of point clouds and report state-of-the-art performance on indoor and outdoor datasets.

7.1 Introduction

In the field of 3D scene understanding, semantic segmentation of 3D point clouds becomes increasingly relevant. Point cloud analysis has found its application in indoor scene understanding and more recently has become an essential component for autonomous driving. This is due to the increasing availability and affordability of 3D sensors such as LiDAR or the Matterport scanner. The data obtained from these sensors comes mainly in the form of unstructured point clouds.

The point cloud representation is challenging for multiple reasons: there is no inherent order (such as pixel neighborhoods in 2D images), they are generally sparse in 3D space and the point density varies with the distance to the sensor. Moreover, the number of points in a cloud can easily exceed the number of pixels in a high-resolution image by multiple orders of magnitude. All these properties make it difficult to process point clouds directly using traditional convolutional networks (CNN) which have shown to be extremely effective in the 2D image domain (Chen *et al.*, 2017; Long *et al.*, 2015; Noh *et al.*, 2015; Wu *et al.*, 2016). Recently, a lot of effort was put into extending the successes from 2D scene understanding into the 3D world (Klokov and Lempitsky, 2017; Kontogianni *et al.*, 2017; Landrieu and Simonovsky, 2018; Qi *et al.*, 2016, 2017a,b,c; Simonovsky and Komodakis, 2017; Tchapmi *et al.*, 2017). In this work, we aim to further narrowing down the gap between 2D and 3D semantic scene understanding. The trivial approach of applying CNNs in the 3D space is by pre-processing the point cloud into a voxel representation in order to apply 3D convolutions (Qi *et al.*, 2016). However, by doing so, we introduce discretization artifacts (especially for thin structures) and loose geometric information such as point density. Methods directly operating on the point cloud representation (e.g.,. (Qi *et al.*, 2017a,b)) produce promising results. However, in these methods, the point neighborhoods over which point features are aggregated are either global (Qi *et al.*, 2017b) or defined in a static coarse-to-fine approach (Qi *et al.*, 2017a). In this work, we propose to define neighborhoods in an adaptive manner that is firstly sensitive to the local geometry by using KMeans clustering on the input point cloud features in the world space and secondly defining dynamic neighborhoods in the learned feature space using k neatest neighbors (KNN). Next is the observation that a well-structured feature space is essential for learning on point clouds. Thus, we add dedicated loss functions which help shaping the feature space at multiple locations in the network.

We showcase the effectiveness of our approach on the task of semantic segmentation of 3D point clouds. We present a comprehensive ablation study evaluating all introduced mechanisms. We apply our method in different scenarios: indoor data from the Stanford 3D Indoor Scene dataset (Armeni *et al.*, 2016) and outdoor data from the Virtual KITTI 3D dataset (Gaidon *et al.*, 2016; Kontogianni *et al.*, 2017). Our method outperforms the current state-of-the art on these datasets. The paper is structured as follows: we start by giving on overview of current related work in Section 8.2. Our method is presented in Section 8.3 followed by implementation details in Section 7.4. Finally, we evaluate our approach in Section 7.5.

7.1.1 Contributions

In summary, our contributions are as follows:

- We introduce a deep learning architecture for end-to-end 3D semantic segmentation of point clouds.
- We present a novel and intuitive feature network to learn strong features.
- We define two grouping techniques based on neighborhoods from KNNs and KMeans.
- Furthermore, dedicated loss functions help shaping the feature space.
- In a thorough evaluation, we analyze the contribution of each component and are able to present state-of-the art result on 3D semantic segmentation benchmarks.

7.2 Related Work

Before the introduction of deep learning methods, there have been numerous traditional approaches (Hackel *et al.*, 2016; Lai *et al.*, 2014; Munoz *et al.*, 2008; Xiong *et al.*, 2011a) applied to the task of semantically labelling 3D point clouds. Since then, methods relying on deep learning can be roughly split into two groups: methods that impose structure on the unstructured 3D point cloud (by voxelization or projection) followed by standard convolutions, and methods that operate directly on the 3D point clouds:

7.2.1 Voxelized Methods.

Up until recently, the standard method to perform semantic segmentation of 3D data involved *voxelization*. Voxelization approaches transform the unstructured 3D point clouds into regular volumetric 3D grids (*voxels*). By doing so, 3D convolutions can be directly applied on the voxels (Maturana and Scherer, 2015; Wu *et al.*, 2015). Alternatively, *projection* approaches map the 3D points into 2D images as seen by virtual cameras. Then, 2D convolutions are applied on the projections (Boulch *et al.*, 2017; Qi *et al.*, 2016). These methods suffer from major disadvantages: the mapping from a sparse representation to a dense one leads to an increased memory footprint. Moreover, the fixed grid resolution results in discretization artifacts and loss of information.

7.2.2 Point Cloud Methods.

A new set of methods started with the work of PointNet (Qi *et al.*, 2017b). PointNet operates directly on 3D points. The key idea is the extraction of point features through a sequence of MLPs processing the points individually (point features) followed by a max-pooling operation that describes the points globally (global features). Point- and global-representations are fused (concatenation + MLP) before making the final label predictions. Many methods followed the PointNet paradigm to operate directly on point clouds. Where PointNet partitions the space into cuboidal blocks of arbitrary fixed size, others use octrees (Tatarchenko *et al.*, 2017) or kd-trees (Klokov and Lempitsky, 2017) to partition the space in a more meaningful way. Furthermore, PointNet does not take into consideration the local geometry and surface information. Clustering has been used in many classical approaches as a way of imposing structure, mostly as a pre-processing step (Xiong *et al.*, 2011a,b). So, Qi *et al.* (2017a) and Klokov and Lempitsky (2017) were introduced trying to apply hierarchical grouping of the points and incorporate local structure information. The former used farthest point sampling and the latter kd-trees. Simonovsky and Komodakis (2017) generalize the convolution operator on a spatial neighborhood. Taking it further from local neighborhoods, Landrieu and Simonovsky (2018) organizes the points into super-points of homogeneous elements and defines relationships between them with the use of graph neural networks on the so-called super-point graph. Kontogianni *et al.* (2017) also use cuboidal blocks, which act as super-points and updates their respective global features based on the surrounding blocks in space or scale using GRUs.

We now compare our method to the recent PointNet++ (Qi *et al.*, 2017a) which is an hierarchical extension of the original PointNet (Qi *et al.*, 2017b). PointNet (PN) globally aggregates point features using max-pooling. PN++ forms local groups (or neighborhoods) based on the metric world space and collapses each group onto a single representative point. This technique is repeated to increase the receptive field in each iteration. In our work, we follow a similar approach by iterative applications of the feature-space neighborhoods (N_F-modules, introduced later): In every $\mathcal{N}_F$ iteration, each point is updated with the aggregated feature information from its KNN. Repeating this procedure allows the information to flow over many points, one hop per iteration. Unlike (Qi *et al.*, 2017a), we build the neighborhood based on the feature space, this allows the network to learn the grouping. In PN++, neighborhoods are statically defined by a metric distance.

7.2.3 Feature Networks.

In a first step of our network, we learn strong features using a dedicated feature network. The idea of extracting initial strong features is prominent in the field: In (Qi *et al.*, 2017c), features are learned in 2D image space using CNN for the task of 2.5 semantic segmentation. For the problem of object detection, VoxelNet (Zhou and Tuzel, 2018) uses a cascade of PointNet like architectures named *Voxel Feature Encoding* to obtain a more meaningful representation.

7.3 Our Approach

In the following subsections, we describe the main components of our network. Starting from the initial point features (e.g.,. position and color), we learn more powerful feature representations. This step is handled by the *Feature Network*, a novel architecture described in Section 7.3.1. Next, we define two kinds of neighborhoods (Section 7.3.2.1) within the point cloud, one defined on the learned feature space ($\mathcal{N}_F$) and one on the input world space ($\mathcal{N}_W$). Based on these groupings, we learn regional descriptors which we use to inform the feature points about their neighborhood. Finally, we further enforce structure on the learned feature space by defining two dedicated loss functions (Section 7.3.3.1).

7.3.1 Feature Network

In this section, we describe our simple yet powerful feature network, a novel base architecture for feature point learning. The goal of this component is to transform input features - such as position and color - into stronger learned features. It can be seen as the distillation of important elements from previous works, in particular *PointNet* (Qi *et al.*, 2017b) and *Consolidation Units* (Kontogianni *et al.*, 2017). A schematic visualization is shown in Figure 7.2.

The network is built from a sequence of *feature blocks*. Feature blocks can be seen as layers which can be stacked to arbitrary depth. Each feature block performs the following tasks: Starting from a set of N point, each with feature dimension F, it produces refined *point features* by passing the incoming features F through a multi-layer-perceptron (MLP). Then, a global representation is computed by aggregating all point features using max-pooling. This *global feature* is again passed through an MLP. Finally, after vertically stacking the global feature N times, it is concatenated with the point features. This is in line with the popular PointNet architecture.

In addition to the feature blocks, we introduce pathway connections which allow the individual feature blocks to consult features from previous layers. We distinguish between the point features (local point pathway) and global features (global pathway). Inspired by DenseNet (Huang *et al.*, 2017) and ResNet (He *et al.*, 2016b), these features can be combined either by concatenation or summation. Our findings are that concatenation gives slightly inferior results over addition with the cost of a higher memory footprint. At the same time, increasing the number of feature blocks in the network is even more important. Thus, in the interest of scalability, in our final feature network we use additive features with 17 features blocks. Finally, the feature network provides us with strong features required for the following components. Our experiments on different number of feature blocks and aggregation functions are summarized in Table 7.6.

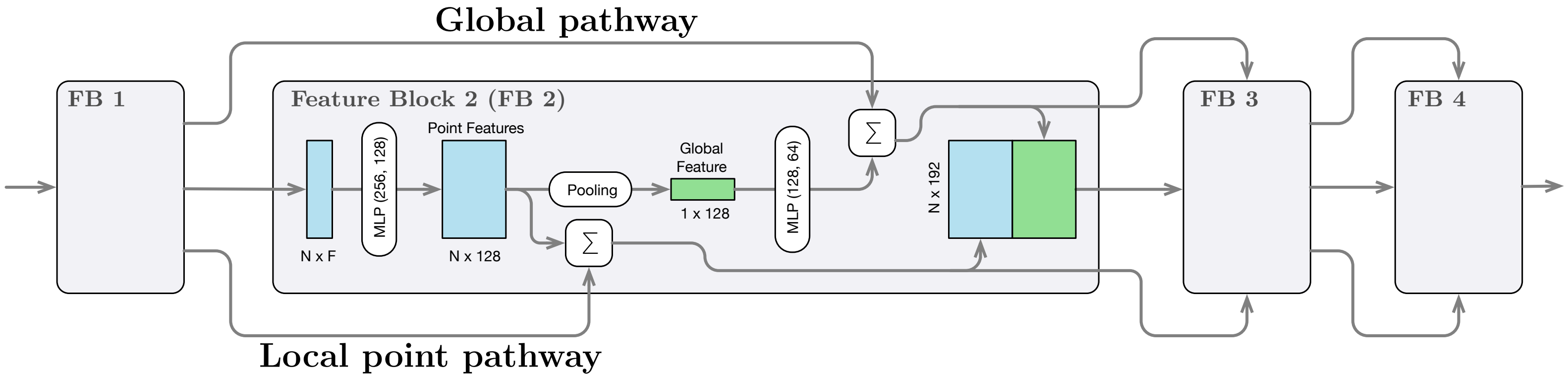


Figure 7.2: The Feature Network. In a first step, our model learns strong features using a *feature network* consisting of multiple *feature blocks* (FBs). Above, we illustrate such an example feature network with four feature blocks. Feature Block 2 (FB 2) shows the mechanism of feature blocks in detail. We refer the reader to Section 7.3.1 for further details and our motivation for the architecture.

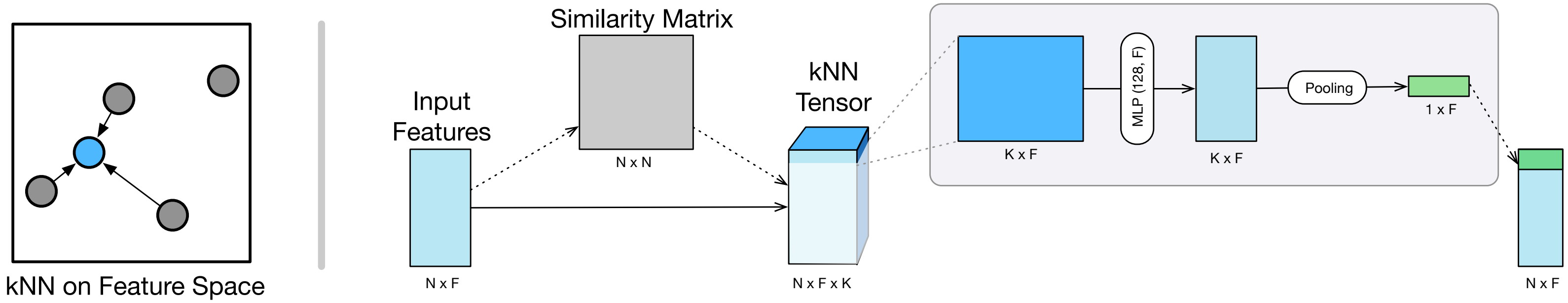


Figure 7.3: Feature space neighborhood $\mathcal{N}_F$. We use kNN in the feature space to compute the *feature space neighborhood* $\mathcal{N}_F(\mathbf{x})$ of a point $\mathbf{x} \in \mathbb{R}^F$. Example for $k = 3$. Based on this neighborhood, the features of a point (shown in blue) are updated (left). Architecture for learning point features over a set of N points based on $\mathcal{N}_F$. The feature learning is shown exemplarily for one point in the set and is identical for the others. Further details are in the text (right).

7.3.2 Neighborhoods

We employ two different grouping mechanism to define the neighborhoods over the point cloud: (1) the *feature space neighborhood* $\mathcal{N}_F$ is obtained by computing the k nearest neighbors (kNN) for each point in the learned feature space, and (2) the *world space neighborhood* $\mathcal{N}_W$ is obtained by clustering points using K-means in the world space. In this context, the world space corresponds to the features of the input point cloud, such as position and color. In the following, we explain the two neighborhoods in more detail and show how they are used to update each point feature.

7.3.2.1 Feature space neighborhood $\mathcal{N}_F$

From the set of N input features of dimensionality F, we compute an $N \times N$ similarity matrix based on the pairwise distance between all the feature points $\mathbf{x}_i$ (See Figure 7.3). For each point, we select the k nearest neighbors to construct the kNN tensor. Each slice in the tensor corresponds to an $\mathcal{N}_F(\mathbf{x}_i)$ neighborhood of feature point $\mathbf{x}_i$. Next, we learn a new feature representation of this neighborhood using an MLP and we generate the updated feature of point i by applying max-pooling. This procedure is the same for each input feature and can be efficiently implemented using convolutions. In the following chapters, we will refer to this architecture as an $\mathcal{N}_F$-Module.

As such, an $\mathcal{N}_F$-module updates the local feature of a point based on its neighborhood in the feature space. By concatenating multiple $\mathcal{N}_F$-modules, we can increase the receptive field of the operation, one hop at a time, which is comparable to applying multiple convolutions in an image space.

7.3.2.2 World space neighborhood $\mathcal{N}_W$

Unlike kNN, k-Means assigns a variable number of points to a neighborhood. K-Means clustering is an iterative method, it alternatively assigns points to the nearest mean which represents the cluster center. Then it recomputes the means based on the assigned points. When applied to the world space, k-Means can be seen as a pooling operation which reduces the input space and increases the receptive field by capturing long-range dependencies. Additionally, we are offered a feature point representative per cluster by averaging over all cluster members in the feature space.

We now describe the corresponding $\mathcal{N}_W$-Module: We perform k-means clustering in the world space and represent each cluster by the average over all feature points in the cluster. Next, we concatenate this average to all the feature points within

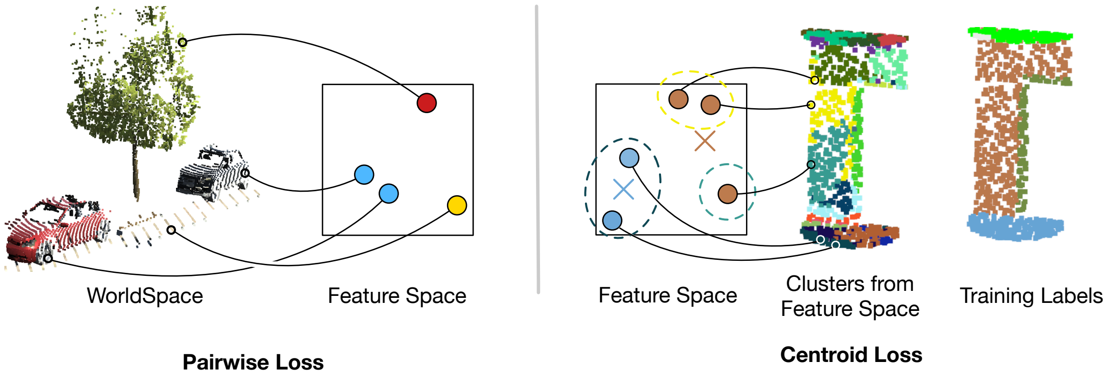


Figure 7.4: Left: The pairwise distance loss $\mathcal{L}_{pair}$ minimizes the distance in the feature space between points of the same semantic class. It increases the distance between points from different classes. **Right:** The centroid loss $\mathcal{L}_{cent}$ minimizes the distance between features and their corresponding centroids, shown as crosses. The feature space is sketched as a 2D embedding. The point colors in the feature space represent training labels. To demonstrate the quality of our embedding, we further show clustering results (dashed lines) and their projection into world space (middle). See Section 7.3.3.2 for details.

the same cluster. We then again apply max-pooling which produces a regional descriptor for this cluster. A visualization is shown in Figure 7.5.

7.3.3 Loss Functions

In this section, we define the loss function $\mathcal{L}$ minimized during the training of our network. The classification loss $\mathcal{L}_{class}$ at the end of our network is realized as the cross entropy between predicted per-class probabilities and one-hot encoded ground truth semantic labels. Beside the classification loss, we introduce two additional losses $\mathcal{L}_{pair}$ and $\mathcal{L}_{cent}$ which further help to shape the feature space. The final loss is computed as $\mathcal{L} = \mathcal{L}_{class} + \mathcal{L}_{pair} + \mathcal{L}_{cent}$.

7.3.3.1 Pairwise Similarity Loss $\mathcal{L}_{pair}$

So far, we assumed that points from the same semantic class are likely to be nearby in the feature space. The *pairwise similarity loss*, described in this section, explicitly enforces this assumption. Similar to Chopra *et al.* (2005), we notice that semantic similarity can be measured directly as a distance in the feature space. By minimizing pairwise distances, we can learn an embedding where two points sampled from the same object produce nearby points in the feature space. And, equivalently, two points originating from different objects have a large pairwise distance in the feature space. This goal is illustrated with a 2D embedding in Figure 7.4.

All we need is a pairwise distance matrix, which we already compute in the $\mathcal{N}_F$-Module (Sec. 3.2). Hence, the distance loss is a natural extension of our network and comes at almost no additional memory cost. For a pair of points (i, j) with features $\mathbf{x}_i$ and $\mathbf{x}_j$, the loss is defined as follows:

$$\ell_{i,j} = \begin{cases} \max(||\mathbf{x}_i - \mathbf{x}_j|| - \tau_{\text{near}}, 0) & \text{if } \mathcal{C}_i = \mathcal{C}_j \\ \max(\tau_{\text{far}} - ||\mathbf{x}_i - \mathbf{x}_j||, 0) & \text{if } \mathcal{C}_i \neq \mathcal{C}_j \end{cases} \tag{7.1}$$

where τ_{near} and τ_{far} are threshold values and $\mathcal{C}_i$ is the semantic class of point i. Finally, the loss $\mathcal{L}_{pair}$ is computed as the sum over all pairwise losses $\ell_{i,j}$.

7.3.3.2 Centroid Loss $\mathcal{L}_{cent}$

This loss reduces the within-class distance by minimizing the distance between point features $\mathbf{x}_i$ and a corresponding representative feature $\overline{\mathbf{x}}_i$ (centroid). It makes the features in the feature space more compact. During training, the representative feature can be computed as the mean feature over all points from the same semantic class. An illustration is shown in Figure 7.4. We define the centroid loss as the sum over all $(\mathbf{x}_i, \overline{\mathbf{x}}_i)$ pairs:

$$\mathcal{L}_{cent} = \sum_{i \in \mathcal{N}} ||\mathbf{x}_i - \overline{\mathbf{x}}_i|| \tag{7.2}$$

where $\mathcal{N}$ is the total number of points. As distance measure $|| \cdot ||$, we found the Cosine distance to be more effective than the L_1 or L_2 distance measures.

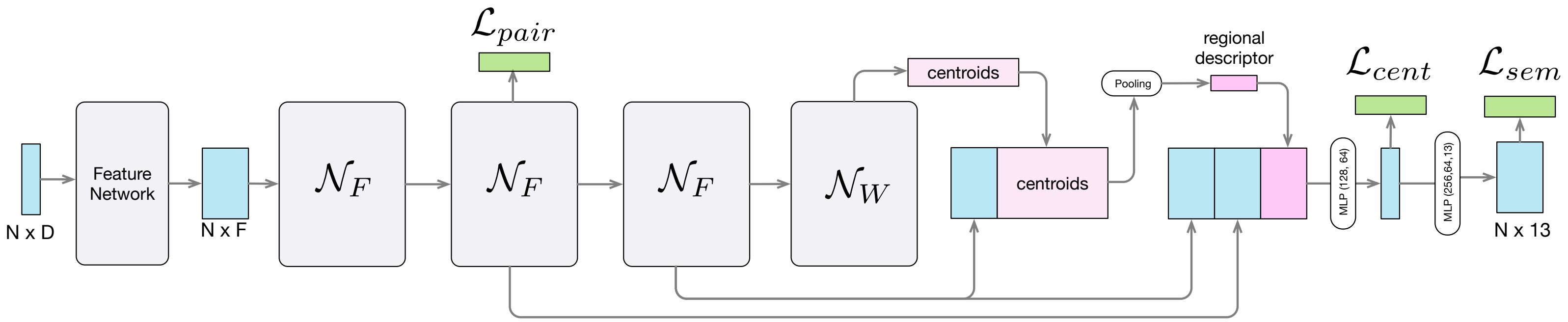


Figure 7.5: Our network architecture. It consists of a Feature Network (Figure 7.2), followed by three $\mathcal{N}_F$-modules (Figure 7.3) and one $\mathcal{F}_W$-module. Features are represented by light blue blocks, losses are shown in green and pink blocks represent features computed over clusters in the world space. We refer the reader to the text for additional details.

Components	oAcc	mAcc	mIoU
Feature Network (FN) (Sec. 7.3.1)	82.43	56.96	47.25
$\mathcal{N}_F$*3 (Section 7.3.2.1)	81.70	55.14	47.37
$\mathcal{N}_F$*3 + $\mathcal{L}_{pair}$ (Section 7.3.3.1)	82.51	58.20	49.41
FN + $\mathcal{N}_F$*3 + $\mathcal{L}_{pair}$	83.84	58.29	50.56
FN + $\mathcal{N}_F$*3 + $\mathcal{N}_W$ + $\mathcal{L}_{pair}$ (Section 7.3.2.2)	84.31	59.18	50.95
FN + $\mathcal{N}_F$*3 + $\mathcal{N}_W$ + $\mathcal{L}_{pair}$ + $\mathcal{L}_{cent}$ (Section 7.3.3.2)	84.19	60.59	51.56

Table 7.1: Ablation study highlighting the contribution of the individual components of our pipeline. The reported numbers are obtained by training on our validation set.

7.4 Implementation Details

In this section, we describe the integration of the aforementioned components into a deep learning architecture. The architecture of our model is depicted in Figure 7.5. We start by learning strong features using our *Feature Network* (Section 7.3.1) producing F-dimensional features. See Table 7.6 for an evaluation and discussion on the architecture. We then feed these features into three stacked $\mathcal{N}_F$ modules. Then, the $\mathcal{N}_W$ module computes regional descriptors for each cluster (based on world space with descriptors form the feature space). We concatenate the regional descriptors to its corresponding feature points of the second and third $\mathcal{N}_F$-Module. The concatenated features are passed through another MLP after which we compute the centroid loss. Finally, we reduce the feature points to the dimensions corresponding to the number of semantic classes in our datasets. The pairwise distance loss is computed in the beginning in the network. This informs the networks as early as possible which points should have similar features. This provides early layers a stronger signal of what should be learned and simplifies propagating as the gradient is passed through fewer layers (Harley *et al.*, 2017).

Position of the embedding loss. Although the distance loss could be appended at each point where a similarity matrix is computed, we found it most effective to add it to the second $\mathcal{N}_F$-module. An ablation study is provided in Table 7.1 and shows the contribution of each component in the performance.

S3DIS (Armeni *et al.*, 2016): 6-fold CV	oAcc	mAcc	mIoU
PointNet (Qi *et al.*, 2017b)	78.62	-	47.71
G + RCU (Kontogianni *et al.*, 2017)	81.1	66.4	49.7
SPG (Landrieu and Simonovsky, 2018)	82.90	64.45	54.06
DGCNN † (Wang *et al.*, 2019)	84.1	-	56.1
PointNet++ (Qi *et al.*, 2017a)	81.03	67.05	54.49
RSN (Huang *et al.*, 2018)	-	66.45	56.47
Ours	**83.95**	**67.77**	**58.27**

Table 7.2: Stanford Large-Scale 3D Indoor Spaces. 6-fold cross validation results. We can present state-of-the-art results in the more difficult CV and slightly inferior results on Area 5. Entries marked with † are unpublished concurrent work.

7.5 Evaluation

In this section, we quantitatively evaluate our model, and we show qualitative results of our approach over previous methods. We evaluate our method on multiple datasets: two indoor- and one outdoor dataset showing the versatility of our approach. For each dataset, we report the overall accuracy (oAcc), the mean class accuracy (mAcc) and the mean class intersection-over-union (mIoU).

7.5.1 Indoor Datasets

We evaluate our model on the *Stanford Large-Scale 3D Indoor Spaces* (S3DIS) dataset (Armeni *et al.*, 2016) and the *ScanNet* dataset (Dai *et al.*, 2017). Both datasets have recently become popular to evaluate 3D semantic segmentation methods. The S3DIS dataset consists of 6 different indoor areas, totaling to 272 rooms. Each point is labeled as one of 13 semantic classes as shown in Figure 7.6. The ScanNet dataset contains 1523 RGB-D scans labeled with 20 different semantic classes.

7.5.2 Outdoor Dataset

We apply our approach to the vKITTI3D dataset, a large-scale outdoor data set in an autonomous driving setting. Introduced in Kontogianni *et al.* (2017), this dataset is an adaptation of the synthetic *Virtual KITTI* dataset (Gaidon *et al.*, 2016) for the task of semantic segmentation of 3D point clouds. It is split in 5 different sequences containing 13 semantic classes listed in Figure 7.7.

S3DIS (Armeni *et al.*, 2016): Area 5	oAcc	mAcc	mIoU
PointNet (Qi *et al.*, 2017b)	-	48.98	41.09
MS + CU(2) ‡ (Kontogianni *et al.*, 2017)	-	52.11	43.02
G + RCU ‡ (Kontogianni *et al.*, 2017)	-	54.06	45.14
SEGCloud (Tchapmi *et al.*, 2017)	-	57.35	48.92
SPG (Landrieu and Simonovsky, 2018)	**85.14**	**61.75**	**54.67**
Ours	84.15	59.10	52.17

Table 7.3: Stanford Large-Scale 3D Indoor Spaces. Area 5 results. We can present competitive results on Area 5. Entries marked with ‡ are scores obtained from authors.

7.5.3 Training Details

For the experiments on the S3DIS dataset, we follow a similar training procedure as (Qi *et al.*, 2017b) i.e. we split the rooms in cuboidal blocks of $1m^2$ on the ground plane. From each block we randomly sample 4096 points. During testing, we predict class labels for all points. Additionally, we add translation augmentation to the block positions. Each point is represented by a 9D feature vector $[x, y, z, r, g, b, x', y', z']$ consisting of the position $[x, y, z]$, color $[r, g, b]$ and normalized coordinates $[x', y', z']$ as in (Qi *et al.*, 2017b). The hyperparameters of our method are set as follows: For the kNN clustering, we set $k = 30$ and use the L_1 distance measure. For K-means, we dynamically set $\mathrm{K} = \lfloor \mathrm{N}/52 \rfloor$ where N is the number of points per block. We report scores on a 6-fold cross validation across all areas in Table 7.2 along with the detailed scores of per class IoU in Table 7.4. Additionally, we provide scores for Area 5 in Table 7.3 to compare ourself to (Tchapmi *et al.*, 2017) and (Landrieu and Simonovsky, 2018).

On the ScanNet dataset (Dai *et al.*, 2017), we use the reference implementation of PointNet++ to train and evaluate our model. This approach allows us to focus on the comparison of the models while abstracting from the training procedures. All hyperparameters remain the same. The results are shown in Table 7.5

Method	mIoU	Ceiling	Floor	Wall	Beam	Column	Window	Door	Table	Chair	Sofa	Bookcase	Board	Clutter
PointNet (Qi *et al.*, 2017b)	47.6	88.0	88.7	69.3	42.4	23.1	47.5	51.6	54.1	42.0	9.6	38.2	29.4	35.2
MS+CU(2) (Kontogianni *et al.*, 2017)	47.8	88.6	95.8	67.3	36.9	24.9	48.6	52.3	51.9	45.1	10.6	36.8	24.7	37.5
SegCloud(Tchapmi *et al.*, 2017)	48.9	90.1	**96.1**	69.9	0.0	18.4	38.4	23.1	**75.9**	**70.4**	**58.4**	40.9	13.0	42.0
G+RCU (Kontogianni *et al.*, 2017)	49.7	90.3	92.1	67.9	**44.7**	24.2	**52.3**	51.2	58.1	47.4	6.9	39.0	30.0	41.9
SPG (Landrieu and Simonovsky, 2018)	54.1	**92.2**	95.0	72.0	33.5	15.0	46.5	60.9	65.1	69.5	56.8	38.2	6.9	51.3
Ours	**58.3**	92.1	90.4	**78.5**	37.8	**35.7**	51.2	**65.4**	64.0	61.6	25.6	**51.6**	**49.9**	**53.7**

Table 7.4: IoU per semantic class on the S3DIS dataset. We compare our model against the original PointNet and other recent methods. On average our method outperforms the current state-of-the-art by a large margin, specifically on *bookcase* and *board* while being slightly worse on *beam* and *sofa*.

ScanNet (Dai *et al.*, 2017)	oAcc	mAcc
PointNet++ (Qi *et al.*, 2017a)	71.40	24.51
Our method	**75.53**	**25.39**

Table 7.5: ScanNet. Overall point accuracy (oAcc), mean semantic class accuracy (mAcc), mean Intersection-over-Union (mIoU). ScanNet dataset using the official training and test split from (Dai *et al.*, 2017), scores are shown on a per-point basis as computed by the PN++ reference implementation. To train on our hardware, we set the batch size to 32 and number of points to 1024.

On the VKITTI3D dataset, we follow again the same training procedure as on the S3DIS dataset. The scenes are split into blocks of $9m^2$ on the ground plane. Since the dataset is much sparser, we set the number of points sampled per block to N = 256. Training on a 6-fold cross validation is performed as in (Kontogianni *et al.*, 2017). We use the same input features as in the indoor dataset and additionally, we analyze how well our method performs if we take into consideration only geometric features (xyz-position) while leaving out color information. This is an interesting experiment, as color is not always imminently available e.g.,. point clouds from laser scanners. We show quantitative results in Table 7.7. Qualitative results are shown in Figure 7.7.

7.6 Conclusions

In this work, we presented a deep learning architecture for 3D semantic segmentation of point clouds. We provide a novel feature learning network that can easily be added into new and existing networks. More importantly, we present two approaches of incorporating neighborhood information from the feature space and from the world space. We also introduce the pairwise distance loss function and the centroid loss function in the context of 3D semantic segmentation. In a thoroughly evaluation, we analyze the contributions of each presented component and are able to outperform the current state-of-the art on current popular datasets. As our results show, at this stage it makes sense to further explore the design space in the field of 3D semantic segmentation.

# Layers	Fusion	mIoU
3	additive	42.15
12	additive	44.46
17	additive	**45.15**
17	concat.	44.23
12	concat.	43.35
3	concat.	41.73

Table 7.6: Study on the feature network. We evaluate the number of layers and compare feature fusion using concatenation or addition. Deeper networks perform better, in general feature addition is slightly stronger while also less memory demanding than concatenation.

VKITTI3D (Kontogianni *et al.*, 2017): 6-fold CV	oAcc	mAcc	mIoU
PointNet (Qi *et al.*, 2017b) from (Kontogianni *et al.*, 2017)	63.3	29.9	17.9
MS+CU(2) (Kontogianni *et al.*, 2017)	73.2	40.9	26.4
Ours	**78.19**	**56.43**	**33.36**
Ours (+ color)	**79.69**	**57.59**	**35.59**

Table 7.7: Virtual KITTI 3D. Comparison of our method to previous methods. We outperform the state-of-the-art by a large margin. The upper part of the tables shows results trained on position only. In the lower part, we additionally trained with color. As it shows, geometric features alone are already quite powerful. By adding color, we can further improve the scores.

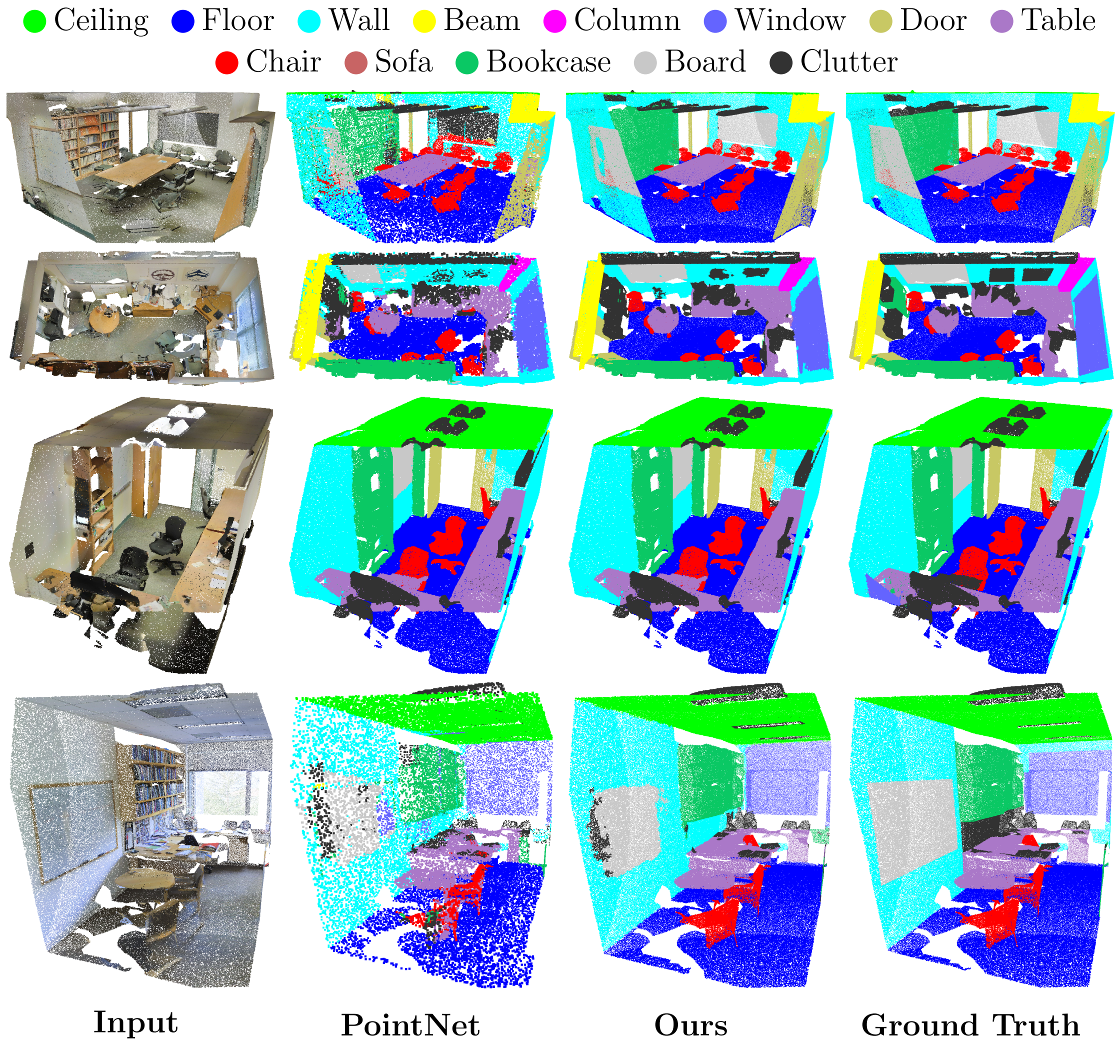


Figure 7.6: Qualitative results on S3DIS dataset. We show three example rooms. Our method compared to PointNet (Qi *et al.*, 2017b) provides segmentations of objects with reduced noise and clear boundaries. As pointed out in the quantitative results section (see Table 7.4), our method performs quite well in challenging objects like *board* and *bookcase*.

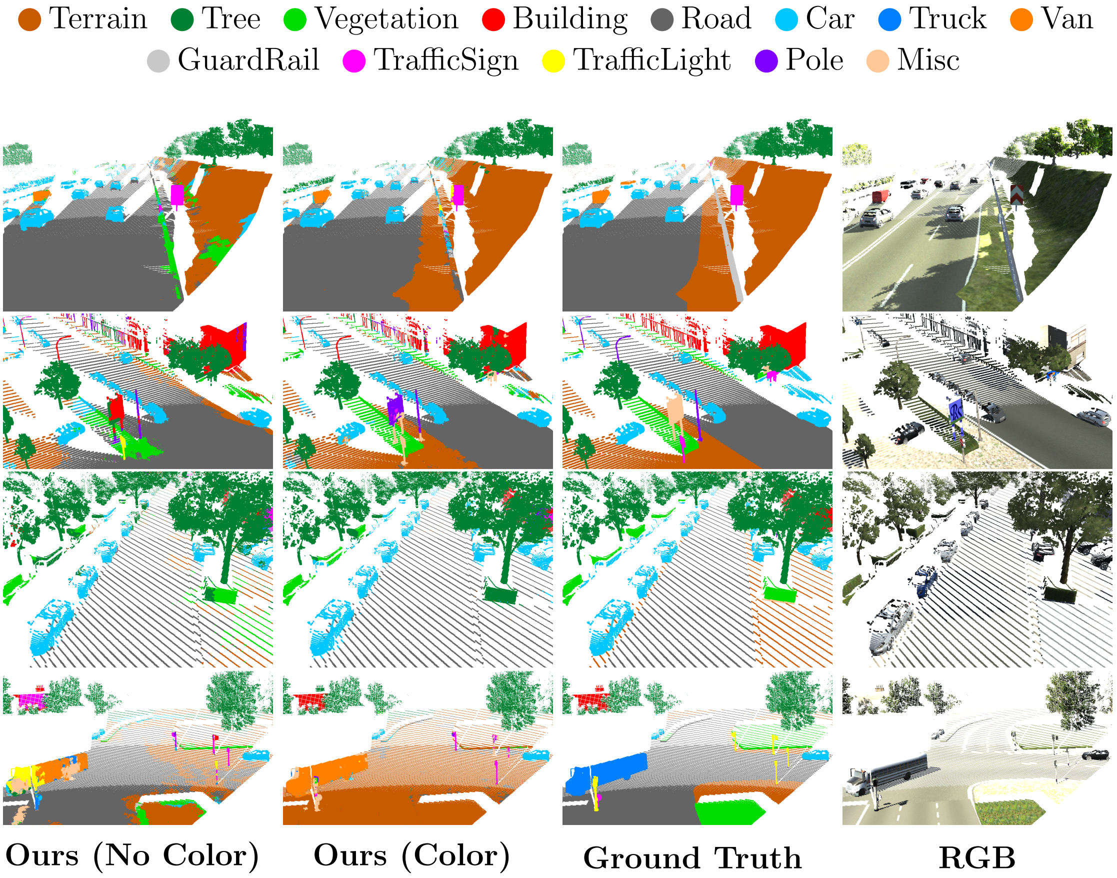


Figure 7.7: Qualitative results on VKITTI3D dataset. In general, color is an important attribute to distinguish between shapes that have similar structure e.g.,. *terrain* and *road*. The last row shows a failure case, during training our model was not able to differentiate between *van* and *truck*, and between *terrain* and *road*.

III

Continuous Adaptation for Interactive Object Segmentation by Learning from Corrections

8

Continuous Adaptation for Interactive Object Segmentation by Learning from Corrections

In interactive object segmentation a user collaborates with a computer vision model to segment an object. Recent works employ convolutional neural networks for this task: Given an image and a set of corrections made by the user as input, they output a segmentation mask. These approaches achieve strong performance by training on large datasets, but they keep the model parameters unchanged at test time. Instead, we recognize that user corrections can serve as sparse training examples and we propose a method that capitalizes on that idea to update the model parameters on-the-fly to the data at hand. Our approach enables the adaptation to a particular object and its background, to distributions shifts in a test set, to specific object classes, and even to large domain changes, where the imaging modality changes between training and testing. We perform extensive experiments on 8 diverse datasets and show: Compared to a model with frozen parameters, our method reduces the required corrections (i) by 9%-30% when distribution shifts are small between training and testing; (ii) by 12%-44% when specializing to a specific class; (iii) and by 60% and 77% when we completely change domain between training and testing.

8.1 Introduction

In interactive object segmentation a human collaborates with a computer vision model to segment an object of interest (Benenson *et al.*, 2019; Boykov and Jolly, 2001; Rother *et al.*, 2004a; Xu *et al.*, 2016). The process iteratively alternates between the user providing corrections on the current segmentation and the model refining the segmentation based on these corrections. The objective of the model

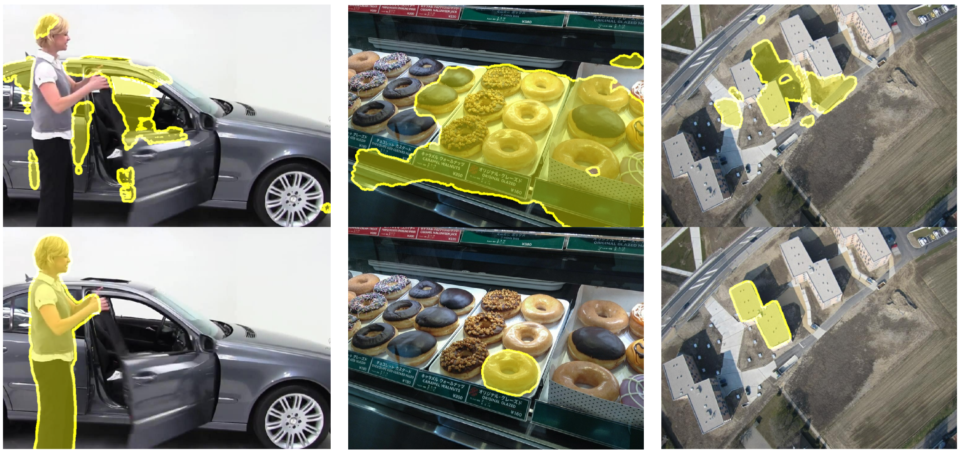


Figure 8.1: Example results for a frozen model (top) and our adaptive methods (bottom). A frozen model performs poorly when foreground and background share similar appearance (left), when it is used to segment new object classes absent in the training set (center, donut class), or when the model is tested on a different image domain (aerial) than it is trained on (consumer) (right). By using corrections to adapt the model parameters to a specific test image, or to the test image sequence, our method substantially improves segmentation quality. The input is four corrections in all cases shown.

is to infer an accurate segmentation mask from as few user corrections as possible (typically point clicks (Bearman *et al.*, 2016; Chen *et al.*, 2018a) or strokes (Gulshan *et al.*, 2010; Rother *et al.*, 2004a) on mislabeled pixels). This enables fast and accurate object segmentation, which is indispensable for image editing (Adobe, 2018) and collecting ground-truth segmentation masks at scale (Benenson *et al.*, 2019).

Current state-of-the-art methods train a convolutional neural network (CNN) which takes an image and user corrections as input and predicts a foreground / background segmentation (Benard and Gygli, 2017; Benenson *et al.*, 2019; Jang and Kim, 2019; Li *et al.*, 2018b; Liew *et al.*, 2017; Mahadevan *et al.*, 2018; Xu *et al.*, 2016). At test time, the model parameters are frozen and corrections are only used as additional input to guide the model predictions. But in fact, user corrections directly specify the ground-truth labelling of the corrected pixels. In this paper we capitalize on this observation: we treat user corrections as training examples to adapt our model on-the-fly. We use these user corrections in two ways: (1) in *single image adaptation* we iteratively adapt model parameters to one specific object in an image, given the corrections produced while segmenting that object; (2) in *image sequence adaptation* we adapt model parameters to a sequence of images with an online method, given the set of corrections produced on these images. Each of these leads to distinct advantages over using a frozen model:

During *single image adaptation* our model learns the specific appearance of the current object instance and the surrounding background. This allows the model to adapt even to subtle differences between foreground and background for that specific example. This is necessary when the object to be segmented has similar color to the background (Fig. 8.1, 1st column), has blurry object boundaries, or low contrast. In addition, a frozen model can sometimes ignore the user corrections and overrule them in its next prediction. We avoid this undesired behavior by updating the model parameters until its predictions respect the user corrections.

During *image sequence adaptation* we continuously adapt the model to a sequence of segmentation tasks. Through this, the model parameters are optimized to the image and class distribution in these tasks, which may consist of different types of images or a set of new classes which are unseen during training. An important case of this is specializing the model for segmenting objects of a single class. This is useful for collecting many examples in high-precision domains, such as *pedestrians* for self-driving car applications. Fig. 8.1, middle column shows an example of specializing to the single, unseen class *donut*. Furthermore, an important property of image sequence adaptation is that it enables us to handle large domain changes, where the imaging modality changes dramatically between training and testing. We demonstrate this by training on consumer photos while testing on medical and aerial images (Fig. 8.1, right column).

Naturally, single image adaptation and image sequence adaptation can be used jointly, leading to a method that combines their advantages.

In summary: Our innovative idea of treating user corrections as training examples allows to update the parameters of an interactive segmentation model at *test time*. To update the parameters, we propose a practical online adaptation method. Our method operates on sparse corrections, balances adaptation vs. retaining old knowledge and can be applied to any CNN-based interactive segmentation model. We perform extensive experiments on 8 diverse datasets and show: Compared to a model with frozen parameters, our method reduces the required corrections (i) by 9%-30% when distribution shifts are small between training and testing; (ii) by 12%-44% when specializing to a specific class; (iii) and by 60% and 77% when we completely change domain between training and testing. (iv) Finally, we evaluate on four standard datasets where distribution shifts between training and testing are minimal. Nevertheless, our method did set a new state-of-the-art on all of them, when it was initially released (Kontogianni *et al.*, 2019).

8.2 Related Work

Interactive Object Segmentation. Traditional methods approach interactive segmentation via energy minimization on a graph defined over pixels (Bai and Sapiro, 2009; Boykov and Jolly, 2001; Gulshan *et al.*, 2010; Price *et al.*, 2010; Rother *et al.*, 2004a). User inputs are used to create an image-specific appearance model based on low-level features (e.g. color), which is then used to predict foreground and background probabilities. A pairwise smoothness term between neighboring pixels encourages regular segmentation outputs. Hence these classical methods are based on a weak appearance model which is specialized to one specific image.

Recent methods rely on Convolutional Neural Networks (CNNs) to interactively produce a segmentation mask (Agustsson *et al.*, 2019; Benard and Gygli, 2017; Chen *et al.*, 2018a; Hu *et al.*, 2019; Jang and Kim, 2019; Li *et al.*, 2018b; Liew *et al.*, 2017; Mahadevan *et al.*, 2018; Xu *et al.*, 2016). These methods take the image and user corrections (transformed into a guidance map) as input and map them to foreground and background probabilities. This mapping is optimized over a training dataset and remains frozen at test time. Hence these models have a strong appearance model but it is not optimized for the test image or dataset at hand.

Our method combines the advantages of traditional and recent approaches: We use a CNN to learn a strong initial appearance model from a training set. During segmentation of a new test image, we adapt the model to it. It thus learns an appearance model specifically for that image. Furthermore, we also continuously adapt the model to the new image and class distribution of the test set, which may be significantly different from the one the model is originally trained on.

Gradient Descent at test time. Several methods iteratively minimize a loss at test time. The concurrent work of Sun *et al.* (2019) uses self-supervision to adapt the feature extractor of a multi-tasking model to the test distribution. Instead, we directly adapt the full model by minimizing the task loss. Others iteratively update the inputs of a model (Gatys *et al.*, 2016; Gygli *et al.*, 2017; Jang and Kim, 2019), e.g. for style transfer (Gatys *et al.*, 2016). In the domain of interactive segmentation, Jang and Kim (2019) update the guidance map which encodes the user corrections and is input to the model. Sofiiuk *et al.* (2020) made this idea more computationally efficient by updating intermediate feature activations, rather than the guidance maps. Instead, our method updates the model parameters, making it more general and allowing it to adapt to individual images as well as sequences.

In-domain Fine-Tuning. In other applications it is common practice to fine-tune on in-domain data when transferring a model to a new domain (Caelles *et al.*, 2017; Papadopoulos *et al.*, 2016; Voigtlaender and Leibe, 2017; Zhou *et al.*, 2017c).

For example, when supervision for the first frame of a test video is available (Caelles *et al.*, 2017; Perazzi *et al.*, 2016; Voigtlaender and Leibe, 2017), or after annotating a subset of an image dataset (Papadopoulos *et al.*, 2016; Zhou *et al.*, 2017c). In interactive segmentation the only existing attempt is Acuna *et al.* (2018), which performs polygon annotation (Acuna *et al.*, 2018; Castrejón *et al.*, 2017; Ling *et al.*, 2019). However, it does not consider adapting to a particular image; their process to fine-tune on a dataset involves 3 different models, so they do it only a few times per dataset; they cannot directly train on user corrections, only on complete masks from previous images; finally, they require a bounding box on the object as input.

Few-shot and Continual Learning. Our method automatically adapts to distribution shifts and domain changes. It performs domain adaptation from limited supervision, similar to few-shot learning (Finn *et al.*, 2017; Qi *et al.*, 2018; Ravi and Larochelle, 2016; Snell *et al.*, 2017). It also relates to continual learning (Farquhar and Gal, 2018; Rebuffi *et al.*, 2017), except that the output label space of the classifier is fixed. As in other works, our method needs to balance between preserving existing knowledge and adapting to new data. This is often done by fine-tuning on new tasks while discouraging large changes in the network parameters, either by penalizing changes to important parameters (Aljundi *et al.*, 2018, 2019; Kirkpatrick *et al.*, 2017; Zenke *et al.*, 2017) or changing predictions of the model on old tasks (Li and Hoiem, 2017; Michieli and Zanuttigh, 2019; Shmelkov *et al.*, 2017). Alternatively, some training data of the old task is kept, and the model is trained on a mixture of the old and new task data (Belouadah and Popescu, 2019; Rebuffi *et al.*, 2017).

8.3 Method

We adopt a typical interactive object segmentation process (Benenson *et al.*, 2019; Boykov and Jolly, 2001; Jang and Kim, 2019; Li *et al.*, 2018b; Mahadevan *et al.*, 2018; Xu *et al.*, 2016): the model is given an image and makes an initial foreground / background prediction for every pixel. The prediction is then overlaid on the image and presented to the user, who is asked to make a correction. The user clicks on a single pixel to mark that it was incorrectly predicted to be foreground instead of background or vice versa. The model then updates the predicted segmentation based on all corrections received so far. This process iterates until the segmentation reaches a desired quality level.

We start by describing the model we build on (Sec. 8.3.1). Then, we describe our core contribution: treating user corrections as training examples to adapt the model on-the-fly at test-time (Sec. 8.3.2). Lastly, we describe how we simulate user corrections to train and test our method (Sec. 8.3.3).

8.3.1 Interactive Segmentation Model

As the basis of our approach, we use a strong re-implementation of Mahadevan *et al.* (2018), an interactive segmentation model based on a convolutional neural network. The model takes an RGB image and the user corrections as input and produces a segmentation mask. As in Benenson *et al.* (2019) we encode the position of user corrections by placing binary disks into a *guidance map*. This map has the same resolution as the image and consists of two channels (one channel for foreground and one for background corrections). The guidance map is concatenated with the RGB image to form a 5-channel map $\mathbf{x}$ which is provided as input to the network.

We use DeepLabV3+ (Chen *et al.*, 2018c) as our network architecture, which has demonstrated good performance on semantic segmentation. However, we note that our method does not depend on a specific architecture and can be used with others as well.

For training the model we need a training dataset $\mathcal{D}$ with ground-truth object segmentations, as well as user corrections which we simulate as in Mahadevan *et al.* (2018) (Sec. 8.3.3). We train the model using the cross-entropy loss over all pixels in an image:

$$\mathcal{L}_{\mathrm{CE}}(\mathbf{x}, \mathbf{y}; \theta) = \frac{1}{|\mathbf{y}|}\{-\mathbf{y}\log \mathbf{f}(\mathbf{x};\theta) - (1-\mathbf{y})\log(1-\mathbf{f}(\mathbf{x};\theta))\} \tag{8.1}$$

where $\mathbf{x}$ is the 5-channel input defined above (image plus guidance maps), $\mathbf{y} \in \{0,1\}^{H\times W}$ are the pixel labels of the ground-truth object segmentations, and $\mathbf{f}(\mathbf{x};\theta)$ represents the mapping of the convolutional network parameterized by θ. $|\cdot|$ denotes the l_1 norm.

We produce the initial parameters θ^* of the segmentation model by minimizing $\sum_{(\mathbf{x}_i,\mathbf{y}_i)\in\mathcal{D}} \mathcal{L}_{\mathrm{CE}}(\mathbf{x}_i, \mathbf{y}_i; \theta)$ over the training set using stochastic gradient descent.

8.3.2 Learning from Corrections at Test-Time

Previous interactive object segmentation methods do not treat corrections as training examples. Thus, the model parameters remain unchanged/frozen at test time (Benard and Gygli, 2017; Benenson *et al.*, 2019; Jang and Kim, 2019; Li *et al.*, 2018b; Mahadevan *et al.*, 2018; Xu *et al.*, 2016) and corrections are only used as inputs to guide the predictions. Instead, we treat corrections as ground-truth labels to adapt the model at test time. We achieve this by minimizing the generalized cross-entropy loss over the corrected pixels:

$$\mathcal{L}_{\mathrm{GCE}}(\mathbf{x}, \mathbf{c}; \theta) = \frac{\mathbf{1}[\mathbf{c}\neq -1]^T}{|\mathbf{1}[\mathbf{c}\neq -1]|}\Bigg\{ -\mathbf{c}\log \mathbf{f}(\mathbf{x};\theta) - (1-\mathbf{c})\log(1-\mathbf{f}(\mathbf{x};\theta)\Bigg\} \tag{8.2}$$

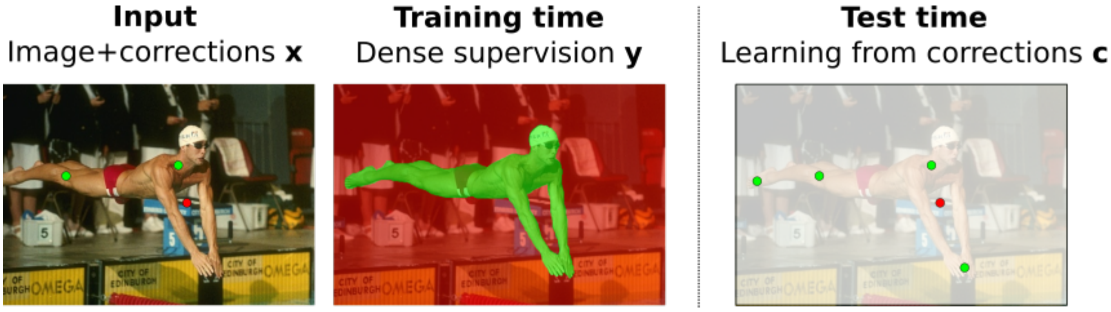


Figure 8.2: Corrections as training examples. For learning the initial model parameters, full supervision is available, allowing to compute a loss over all the pixels in the image. At test time, the user provides sparse supervision in the form of corrections. We use these to adapt the model parameters.

where $\mathbf{1}$ is an indicator function and $\mathbf{c}$ is a vector of values $\{1, 0, -1\}$, indicating what pixels were corrected to what label. Pixels that were corrected to be positive are set to 1 and negative pixels to 0. The remaining ones are set to -1, so that they are ignored in the loss. As there are very few corrections available at test time, this loss is computed over a sparse set of pixels. This is in contrast to the initial training which had supervision at every pixel (Sec. 8.3.1). We illustrate the contrast between the two forms of supervision in Fig. 8.2.

Dealing with label sparsity. In practice, corrections $\mathbf{c}$ are extremely sparse and consist of just a handful of scattered points (Fig. 8.3). Hence, they offer limited information on the spatial extent of objects and special care needs to be taken to make this form of supervision useful in practice. As our model is initially trained with full supervision, it has learned strong shape priors. Thus, we propose two auxiliary losses to prevent forgetting these priors as the model is adapted. First, we regularize the model by treating the initial mask prediction $\mathbf{p}$ as ground-truth and making it a target in the cross-entropy loss, i.e., $\mathcal{L}_{\mathrm{CE}}(\mathbf{x}, \mathbf{p}; \theta)$. This prevents the model from focusing only on the user corrections while forgetting the initially good predictions on pixels for which no corrections were given.

Second, inspired by methods for class-incremental learning (Aljundi *et al.*, 2018; Kirkpatrick *et al.*, 2017; Zenke *et al.*, 2017), we minimize unnecessary changes to the network parameters to prevent it from forgetting crucial patterns learned on the initial training set. Specifically, we add a cost for changing important network parameters:

$$\mathcal{L}_{\mathrm{F}}(\theta) = \boldsymbol{\Omega}^T \left(\theta - \theta^*\right)^{\odot 2} \tag{8.3}$$

where θ^* are the initial model parameters, θ are the updated parameters and $\boldsymbol{\Omega}$ is the importance of each parameter. $(\cdot)^{\odot 2}$ is the element-wise square (Hadamard square). Intuitively, this loss penalizes changing the network parameters away from their initial values, where the penalty is higher for important parameters. We compute $\mathbf{\Omega}$ using Memory-Aware Synapses (MAS) (Aljundi *et al.*, 2018), which estimates importance based on how much changes to the parameters affect the prediction of the model.

Combined loss. Our full method uses a linear combination of the above losses:

$$\mathcal{L}_{\text{ADAPT}}(\mathbf{x}, \mathbf{p}, \mathbf{c}; \theta) = \lambda \mathcal{L}_{\text{GCE}}(\mathbf{x}, \mathbf{c}; \theta) + (1 - \lambda)\mathcal{L}_{\text{GCE}}(\mathbf{x}, \mathbf{p}; \theta) + \gamma \mathcal{L}_{\text{F}}(\theta) \quad (8.4)$$

where λ balances the importance of the user corrections vs. the predicted mask and γ defines the strength of parameter regularization. Next, we introduce *single image adaptation* and *image sequence adaptation*, which both minimize Eq. (8.4). Their difference lies in how the model parameters θ are updated: individually for each object or over a sequence.

8.3.2.1 Adapting to a single image.

We adapt the segmentation model to a particular object in an image by training on the click corrections. We start from the segmentation model with parameters θ^* fit to the initial training set (Sec. 8.3.1). Then we update them by running several gradient descent steps to minimize our combined loss Eq. (8.4) every time the user makes a correction (Algo. in supp. material). We choose the learning rate and the number of update steps such that the updated model adheres to the user corrections. This effectively turns corrections into constraints. This process results in a segmentation mask $\mathbf{p}$, predicted using the updated parameters θ.

Adapting the model to the current test image brings two core advantages. First, it learns about the specific appearance of the object and background in the current image. Hence corrections have a larger impact and can also improve the segmentation of distant image regions which have similar appearance. The model can also adapt to low-level photometric properties of this image, such as overall illumination, blur, and noise, which results in better segmentation in general. Second, our adaptation step makes the corrections effectively hard constraints, so the model will preserve the corrected labeling in later iterations too.

This adaptation is done for each object separately, and the updated θ is discarded once an object is segmented.

8.3.2.2 Adapting to an image sequence.

Here we describe how to continuously adapt the segmentation model to a sequence of test images using an online algorithm. Again, we start from the model parameters θ^* fit to the initial training set (Sec. 8.3.1). When the first test image arrives, we perform interactive segmentation using these initial parameters. Then, after segmenting each image $I_t = (\mathbf{x}_t, \mathbf{c}_t)$, the model parameters are updated to θ_{t+1} by doing a single gradient descent step to minimize Eq. (8.4) for that image. Thereby we subsample the corrections in the guidance maps to avoid trivial solutions (predict the corrections given the corrections themselves, see supp. material). The updated model parameters are used to segment the next image I_{t+1}.

Through the method described above our model adapts to the whole test image sequence, but does so gradually, as objects are segmented in sequence. As a consequence, this process is fast, does not require storing a growing number of images, and can be used in a online setting. In this fashion it can adapt to changing appearance properties, adapt to unseen classes, and specialize to one particular class. It can even adapt to radically different image domains as we demonstrate in Sec. 8.4.3.

8.3.2.3 Combined adaptation.

For a test image I_t, we segment the object using single image adaptation (Algo. in supp. material). After segmenting a test image, we gather all corrections provided for that image and apply a image sequence adaptation step to update the model parameters from θ_t to θ_{t+1}. At the next image, the image adaptation process will thus start from parameters θ_{t+1} better suited for the test sequence. This combination allows to leverage the distinct advantages of the two types of adaptation.

8.3.3 Simulating user corrections

To train and test our method we rely on simulated user corrections, as is common practice (Benard and Gygli, 2017; Jang and Kim, 2019; Li *et al.*, 2018b; Liew *et al.*, 2017; Mahadevan *et al.*, 2018; Xu *et al.*, 2016).

Test-time corrections. When interactively segmenting an object, the user clicks on a mistake in the predicted segmentation. To simulate this, we follow Benard and Gygli (2017); Mahadevan *et al.* (2018); Xu *et al.* (2016), which assume that the user clicks on the largest error region. We obtain this error region by comparing the model predictions with the ground-truth and select its center pixel.

Train-time corrections. Ideally one wants to train with the same user model that is used at test-time. To make this computationally feasible, we train the model in two stages as in Mahadevan *et al.* (2018). First, we sample corrections using ground-truth segmentations (Benard and Gygli, 2017; Jang and Kim, 2019; Li *et al.*, 2018b; Liew *et al.*, 2017; Xu *et al.*, 2016). Positive user corrections are sampled uniformly at random on the object. Negative user corrections are sampled according to three strategies: (1) uniformly at random from pixels around the object, (2) uniformly at random on other objects, and (3) uniformly around the object. We use these corrections to train the model until convergence. Then, we continue training by iteratively sampling corrections following Mahadevan *et al.* (2018). For each image we keep a set of user corrections **c**. Given **c** we predict a segmentation mask, simulate the next user correction (as done at test time), and add it to **c**. Based on this additional correction, we predict a new segmentation mask and minimize the loss (Eq. (8.1)). Initially, **c** corresponds to the corrections simulated in the first stage, and over time more user corrections are added. As we want the model to work well even with few user corrections, we thus periodically reset **c** to the initial clicks (Mahadevan *et al.*, 2018).

8.4 Experiments

We extensively evaluate our single image adaptation and image sequence adaptation methods on several standard datasets as well as on aerial and medical images. These correspond to increasingly challenging adaptation scenarios.

Adaptation scenarios. We first consider *distribution shift*, where the training and test image sets come from the same general domain, consumer photos, but differ in their image and object statistics (Sec. 8.4.1). This includes differences in image complexity, object size distribution, and when the test set contains object classes absent during training. Then, we consider a *class specialization* scenario, where a sequence of objects of a single class has to be iteratively segmented (Sec. 8.4.2). Finally we test how our method handles large *domain changes* where the imaging modality changes between training and testing. We demonstrate this by going from consumer photos to aerial and medical images (Sec. 8.4.3).

Model Details. We use a strong re-implementation of Mahadevan *et al.* (2018) as our interactive segmentation model (Sec. 8.3.1). We pre-train its parameters on PASCAL VOC12 (Everingham *et al.*, 2012) augmented with SBD (Hariharan *et al.*, 2011) (10582 images with 24125 segmented instances of 20 object classes). As a baseline, we use this model as in Mahadevan *et al.* (2018), i.e. without updating its parameters at test time. We call this the *frozen model*. This baseline already achieves state-of-the-art results on the PASCAL VOC12 validation set, simply by increasing the encoder resolution compared to Mahadevan *et al.* (2018) (3.44 clicks). This shows that using a fixed set of model parameters works well when the train and test distributions match. We evaluate our proposed method by adapting the parameters of that same model at test time using *single image adaptation* (IA), *image sequence adaptation* (SA), and their combination (IA + SA).

Evaluation metrics. We use two standard metrics (Benard and Gygli, 2017; Benenson *et al.*, 2019; Jang and Kim, 2019; Li *et al.*, 2018b; Liew *et al.*, 2017; Mahadevan *et al.*, 2018; Xu *et al.*, 2016): (1) **IoU@k**, the average intersection-over-union between the ground-truth and predicted segmentation masks, given k corrections per image, and (2) **clicks@q%**, the average number of corrections needed to reach an IoU of $q\%$ on every image (thresholded at 20 clicks). We always report mean performance over 10 runs (standard deviation is negligible at ≈ 0.01 for clicks@q%).

Hyperparameter selection. We optimize the hyperparameters for both adaptation methods on a subset of the ADE20k dataset (Zhou *et al.*, 2017b, 2018). Hence, the hyperparameters are optimized for adapting from PASCAL VOC12 to ADE20k, which is distinct from the distribution shifts and domain changes we evaluate on.

Table 8.1: Adapting to distribution shifts. Mean number of clicks required to attain a particular mIoU score on Berkeley, YouTube-VOS and COCO datasets (Lower is better). Both of our adaptive methods, single image adaptation (IA) and image sequence adaptation (SA) improve over the model that keeps the weights frozen at test time.

	Berkeley (McGuinness and O'Connor, 2010)	**YouTube-VOS** (Xu *et al.*, 2018a)	**COCO** (Lin *et al.*, 2014)		
			seen	unseen	unseen 6k
Method	clicks@90%	clicks@85%	clicks@85%		
Frozen model (Mahadevan *et al.*, 2018)	5.4	7.9	10.0	11.9	13.2
IA	4.9	7.0	9.1	10.7	10.6
SA	5.3	6.9	9.7	10.6	10.0
IA+SA	**4.9**	**6.7**	**9.1**	**9.9**	**9.3**
Δ over frozen model	8.5%	15.2%	9.0%	16.8%	29.5%

8.4.1 Adapting to distribution shift

We test how well we can adapt the model which is trained on PASCAL VOC12 to other consumer photos datasets.

Datasets. We test on: (1) *Berkeley* (McGuinness and O'Connor, 2010), 100 images with a single foreground object. (2) *YouTube-VOS* (Xu *et al.*, 2018a), a large video object segmentation dataset. We use the test set of the 2019 challenge, where we take the first frame with ground truth (1169 objects, downscaled to 855×480 maximal resolution). (3) *COCO* (Lin *et al.*, 2014), a large segmentation dataset with 80 object classes. 20 of those overlap with the ones in the PASCAL VOC12 dataset and are thus *seen* during training. The other 60 are *unseen*. We sample 10 objects per class from the validation set and separately report results for seen (200 objects) and unseen classes (600 objects) as in (Majumder and Yao, 2019; Xu *et al.*, 2016). We also study how image sequence adaptation behaves on longer sequences of 100 objects for each unseen class (named *COCO unseen 6k*).

Results. We report our results in Tab. 8.1 and Fig. 8.4. Both types of adaptation improve performance on all tested datasets. On the first few user corrections *single image adaptation* (IA) performs similarly to the frozen model as it is initialized with the same parameters. But as more corrections are provided, it uses these more effectively to adapt its appearance model to a specific image. Thus, it performs particularly well in the high-click regime, which is most useful for objects that are challenging to segment (e.g., due to low illumination, Fig. 8.3), or when very accurate masks are desired.

During *image sequence adaptation* (SA), the model adapts to the test image distribution and thus learns to produce good segmentation masks given just a few clicks (Fig. 8.4a). As a result, SA outperforms using a frozen model on all datasets with distribution shifts (Tab. 8.1). By adapting from images to the video frames of YouTube-VOS, SA reduces the clicks needed to reach 85% IoU by 15%. Importantly, we find that our method adapts fast, making a real difference after just a few images, and then keeps on improving even as the test sequence becomes thousands of images long (Fig. 8.4b). This translates to a large improvement given a fixed budget of 4 clicks per object: on the COCO unseen 6k split it achieves 69% IoU compared to the 57% of the frozen model (Fig. 8.4a).

Generally, the curves for image sequence adaptation grow faster in the low click regime than the single image adaptation ones, but then exhibit stronger diminishing returns in the higher click regime (Fig. 8.4a). Hence, combining the two compounds their advantages leading to a method that considerably improves over the frozen model on the full range of number of corrections and sequence lengths (Fig. 8.4a). Compared to the frozen model, our combined method significantly reduces the

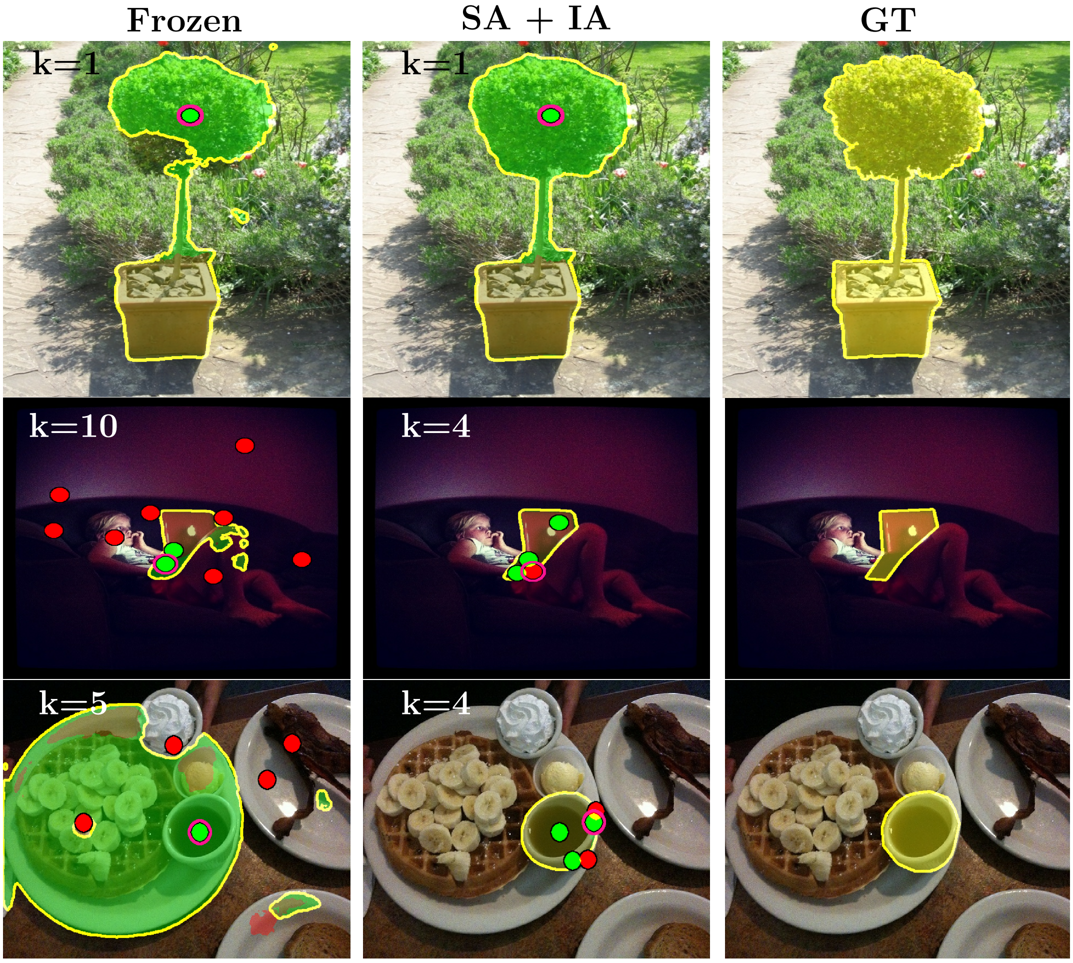


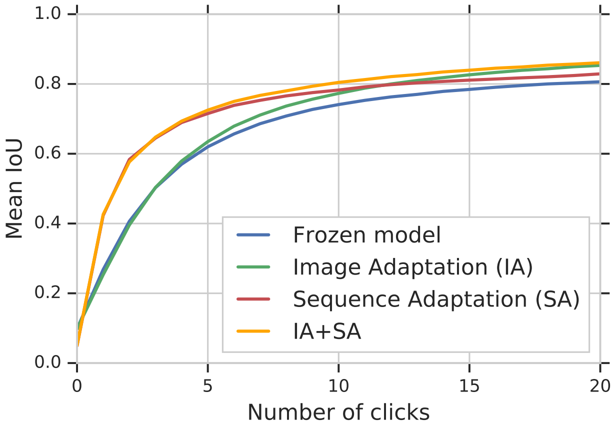
Figure 8.3: Qualitative results of the frozen model and our model with combined adaptation. Red circles are negative clicks and green ones are positive. The red area shows the pixels that turned to background and the green the ones turned to foreground with the latest clicks. Our method produces accurate masks with fewer clicks **k**.

number of clicks needed to reach the target accuracy on all datasets: from a 9% reduction on Berkeley and COCO seen, to a 30% reduction on COCO unseen 6k.

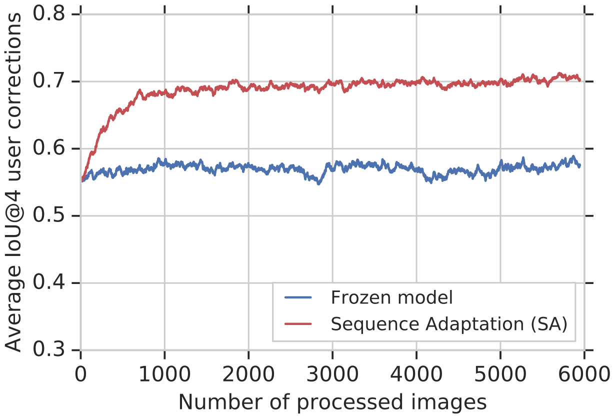
8.4.2 Adapting to a specific class

When a user segments objects of a single class at test-time, image sequence adaptation naturally specializes its appearance model to that class. We evaluate this phenomenon on 4 COCO classes. We form 4 test image sequences, each focusing on a single class, containing objects of varied appearance. The classes are selected based on how they perform using image sequence adaptation compared to the frozen model from Sec. 8.4.1. We selected the following classes, with increasing order of difficulty for image sequence adaptation: (1) donut (2540 objects) (2) bench (3500) (3) umbrella (3979) and (4) bed (1450).

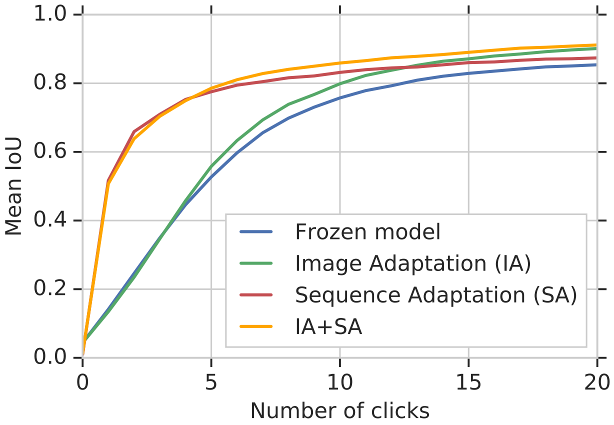
Results. Tab. 8.2, Fig. 8.4c present results. The class specialization brought by our image sequence adaptation (SA) leads to good masks from very few clicks. For example, on the donut class it reduces clicks@85% by 39% compared to the frozen model and by 44% when combined with single image adaptation (Tab. 8.2). Given just 2 clicks, SA reaches 66% IoU for that class, compared to 25% IoU for the frozen model (Fig. 8.4c). The results for the other classes follow a similar pattern, showing that image sequence adaptation learns an effective appearance model for a single class.



(a) Average IoU@k for varying k on the COCO unseen 6k split. Both forms of adaptation lead to a significant improvement over the frozen model.

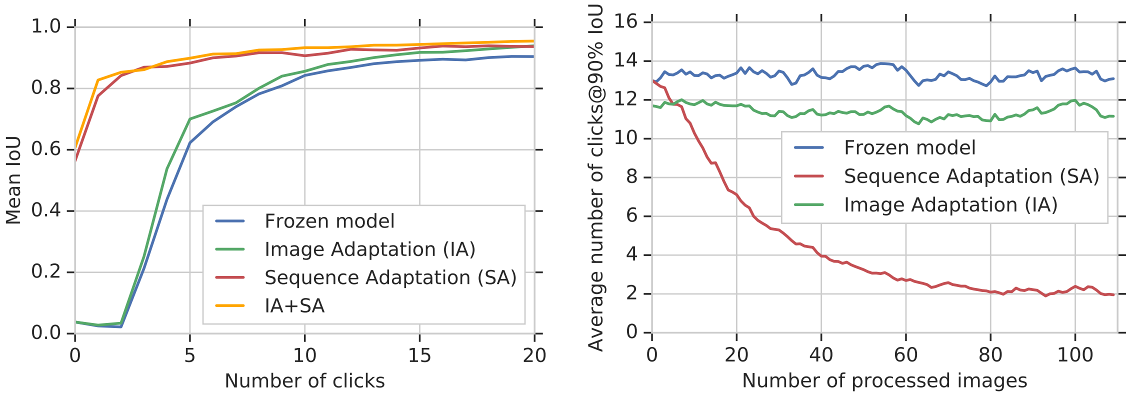


(b) IoU@4 clicks as a function of the number of images processed. Image sequence adaptation quickly improves over the model with frozen weights.

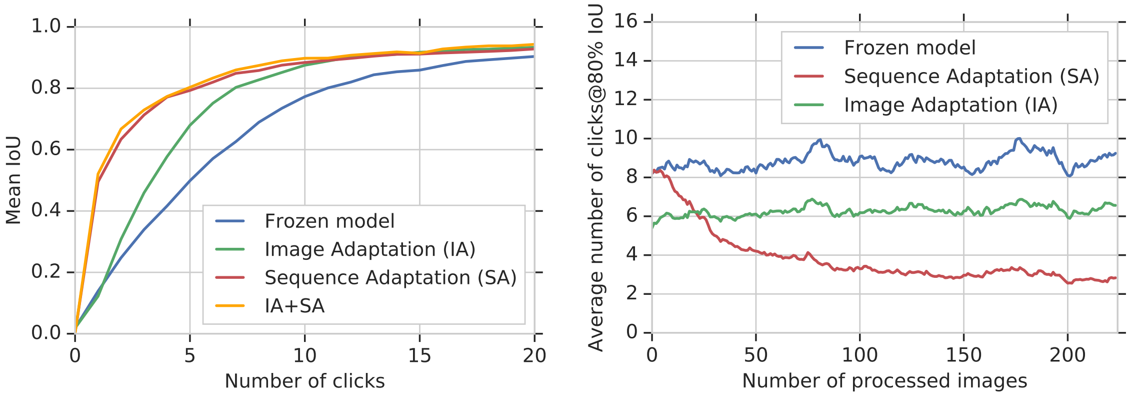


(c) IoU@k for varying k when specializing to *donuts*. SA offers large improvements by learning an appearance model specifically for that class.

Figure 8.4: Results adapting to distribution shifts (a,b) and to a specific class (c).



(a) DRIONS-DB dataset.



(b) Rooftop Aerial dataset.

Figure 8.5: Results for adapting to domain change. For each dataset, we show the average IoU@k for varying number of user clicks k (left in 8.5a, 8.5b) and the number of clicks to reach the target IoU as a function of the number of images processed (right in 8.5a and 8.5b). Single image adaptation provides a consistent improvement over the test sequences. Instead, image sequence adaptation adapts its appearance model to the new domain gradually, improving with every image processed (right in 8.5a and 8.5b).

Table 8.2: Class specialization. We test segmenting objects of only one specific class. Our adaptive methods outperforms the frozen model on all tested classes. Naturally, gains are larger for image sequence adaptation, as it can adapt to the class over time.

	clicks @ 85% IoU			
	Donut	**Bench**	**Umbrella**	**Bed**
Frozen model (Mahadevan *et al.*, 2018)	11.6	15.1	13.1	6.8
IA (Ours)	9.2	14.1	11.9	5.5
SA (Ours)	7.1	14.0	11.1	5.5
IA+SA (Ours)	**6.5**	**13.3**	**10.2**	**5.0**
Δ over frozen model	44.0%	11.9%	22.1%	26.5%

Table 8.3: Domain change results. We evaluate our model on 2 datasets that belong to different domains: aerial (Rooftop) and medical (DRIONS-DB). Both types of adaptation (IA and SA) outperform the frozen model on both datasets.

Method	**DRIONS-DB** (Carmona *et al.*, 2008) clicks@90% IoU	**Rooftop** (Sun *et al.*, 2014) clicks@80% IoU
Frozen model (Mahadevan *et al.*, 2018)	13.3	8.9
IA (Ours)	11.4	6.3
SA (Ours)	3.6	3.6
IA+SA (Ours)	**3.1**	**3.6**
Δ over frozen model	76.7%	59.6%

8.4.3 Adapting to domain changes

We test our method's ability of adapting to domain changes by training on consumer photos (PASCAL VOC12) and evaluating on aerial and medical imagery.

Datasets. We explore two test datasets: (1) *Rooftop Aerial* (Sun *et al.*, 2014), a dataset of 65 aerial images with segmented rooftops and (2) *DRIONS-DB* (Carmona *et al.*, 2008), a dataset of 110 retinal images with a segmentation of the optic disc of the eye fundus. (we use the masks of the first expert). Importantly, the initial model parameters θ^* were optimized for the PASCAL VOC12 dataset, which consists of consumer photos. Hence, we explore truly large domain changes here.

Results. Both our forms of adaptation significantly improve over the frozen model (Tab. 8.3, Fig. 8.5). Single image adaptation can only adapt to a limited extent, as it independently adapts to each object instance, always starting from the same initial model parameters θ^*. Nonetheless, it offers a significant improvement, reducing the number of clicks needed to reach the desired IoU by 14%-29%. Image sequence adaptation (SA) shows extremely strong performance, as its adaptation effects accumulate over the duration of the test sequence. It reduces the needed user input by 60% for the Rooftop Aerial dataset and by over 70% for DRIONS-DB. When combining the two types of adaptation, the reduction increases to 77% for the DRIONS-DB dataset (Tab. 8.3). Importantly, our method adapts fast: on DRIONS-DB clicks@90% drops quickly and converges to just 2 corrections, as the length of the test sequence increases (Fig. 8.5a). In contrast, the frozen model performs poorly on both datasets. On the Rooftop Aerial dataset, it needs even more clicks than there are points in the ground truth polygons (8.9 vs. 5.1). This shows that even a state-of-the-art model like (Mahadevan *et al.*, 2018) fails to generalize to truly different domains and highlights the importance of adaptation.

To summarize: We show that our method can bridge large domain changes spanning varied datasets and sequence lengths. With just a single gradient descent step per image, our image sequence adaptation successfully addresses a major shortcoming of neural networks, for the case of interactive segmentation: Their poor generalization to changing distributions (Alcorn *et al.*, 2019; Recht *et al.*, 2018).

Table 8.4: The focus of our work is handling distribution shifts and domain changes between training and testing (Tab. 8.1, 8.2 & 8.3). For completeness, we also compare our method against existing interactive segmentation models on standard datasets, where the distribution mismatch between training and testing is small. At the time of initially releasing our work (Kontogianni *et al.*, 2019), our method outperformed all previous state-of-the-art models on all datasets, including methods that developed specialized network architectures in order to predict at full resolution (Jang and Kim, 2019; Li *et al.*, 2018b) (see text). Later, F-BRS (Sofiiuk *et al.*, 2020) (CVPR 2020) achieved even better results.

	VOC12	**GrabCut**	**Berkeley**	**DAVIS**
	(Everingham *et al.*, 2012)	(Rother *et al.*, 2004a)	(McGuinness and O'Connor, 2010)	(Perazzi *et al.*, 2016)
	validation			10% of frames
Method	clicks@85%	clicks@90%	clicks@90%	clicks@85%
iFCN w/ GraphCut (Xu *et al.*, 2016)	6.88	6.04	8.65	-
RIS (Liew *et al.*, 2017)	5.12	5.00	6.03	-
TSLFN (Hu *et al.*, 2019)	4.58	3.76	6.49	-
VOS-Wild (Benard and Gygli, 2017)	5.6	3.8	-	-
ITIS (Mahadevan *et al.*, 2018)	3.80	5.60	-	-
CAG (Majumder and Yao, 2019)	3.62	3.58	5.60	-
Latent Diversity (Li *et al.*, 2018b)	-	4.79	-	5.95
BRS (Jang and Kim, 2019)	-	3.60	5.08	5.58
F-BRS (Sofiiuk *et al.*, 2020) (Concurrent Work)	-	2.72	4.57	5.04
IA+SA combined (Ours)	**3.18**	**3.07**	**4.94**	**5.16**

8.4.4 Comparison to Previous Methods

While the main focus of our work is tackling challenging adaptation scenarios, we also compare our method against state-of-the-art interactive segmentation methods on standard datasets. These datasets are typically similar to PASCAL VOC12, hence have a small distribution mismatch between training and testing.

Datasets. (1) Berkeley, introduced in Sec. 8.4.1 (2) *GrabCut* (Rother *et al.*, 2004a), 49 images with segmentation masks. (3) *DAVIS16* (Perazzi *et al.*, 2016), 50 high-resolution videos out of which we sample 10% of the frames uniformly at random as in (Jang and Kim, 2019; Li *et al.*, 2018b) (We note that the standard evaluation protocol of DAVIS16 favors adaptive methods, as the same objects appear repeatedly in the test sequence.) and (4) *PASCAL VOC12 validation*, with 1449 images.

Results. Tab. 8.4 shows results. Our adaptation method achieves strong results: At the time of initially releasing our work (Kontogianni *et al.*, 2019), it outperformed all previous state-of-the-art methods on all datasets (it was later overtaken by (Sofiiuk *et al.*, 2020)). It brings improvements even when the previous methods (which have frozen model parameters) already offers strong performance and need less than 4 clicks on average (PASCAL VOC12, GrabCut). The improvement on PASCAL VOC12 further shows that our method helps even when the training and testing distributions match exactly (the frozen model needs 3.44 clicks).

Importantly, we find that our method outperforms (Jang and Kim, 2019; Li *et al.*, 2018b), even though we use a standard segmentation backbone (Chen *et al.*, 2018c) which predicts at $\frac{1}{4}$ of the input resolution. Instead (Jang and Kim, 2019; Li *et al.*, 2018b) propose specialized network architectures in order to predict at full image resolution, which is crucial for their good performance (Jang and Kim, 2019). We note that our adaptation method is orthogonal to these architectural optimizations and can be combined with them easily.

8.4.5 Ablation Study

We ablate the benefit of treating corrections as training examples (on COCO unseen 6k). For this, we selectively remove them from the loss (Eq. (8.4)). For single image adaptation, this leads to a parameter update that makes the model more confident in its current prediction, but this does not improve the segmentation masks. Instead, training on corrections improves clicks@85% from 13.2 to 10.6. For image sequence adaptation, switching off the corrections corresponds to treating the predicted mask as ground-truth and updating the model with it. This approach implicitly contains corrections in the mask and thus improves clicks@85% from 13.2 for the frozen model to 11.9. Explicitly using correction offers an additional gain of almost 2 clicks, down to 10. This shows that treating user corrections as training examples is key to our method: They are necessary for single image adaptation and highly beneficial for image sequence adaptation.

8.4.6 Subsampling for image sequence adaptation

We subsample the corrections in the guidance maps to avoid trivial solutions. If all corrections were used equally in the loss and guidance maps, the model could eventually degrade to predict the corrections given the corrections themselves. Specifically, it could degrade to only use the information in the guidance maps as its prediction, without relying on image appearance. We avoid this by subsampling the clicks given to the network as guidance but using all clicks to compute the adaptation loss (Eq. (4)). This forces the network to rely on appearance for propagating the corrections to the rest of the image, where the loss is sparsely evaluated at the pixel locations which were corrected.

8.4.7 Robustness to image order

Since our *image sequence adaptation* (SA) and *combined adaptation* (IA+SA) methods are processing images sequentially, we tested our method's sensitivity to the image order. We repeated all our experiments 10 times by randomizing the image order and computed the variance of the results. We found the variance to be minimal (≤ 0.01 standard deviation) verifying that our adaptation methods are not sensitive to the order in which the images are processed. Hence, we only report averages to improve readability.

8.4.8 Adaptation speed

While our method updates the parameters at test time, it remains fast enough for interactive usage. For the model used throughout our paper a parameter update step takes 0.16 s (Nvidia V100 GPU, mixed-precision training, Berkeley dataset). Image sequence adaptation only needs a single update step, done *after* an object is segmented (Sec. 8.3.2.2). Thus, the adaptation overhead is negligible here. For single image adaptation we used 10 update steps, for a total time of 1.6 s. We chose this number of steps based on hyperparameter search (see supp. material). In practice, fewer update steps can be used to increase speed, as they quickly show diminishing returns (Fig. 8.6). In practice, we recommend using 3 update steps, reducing adaptation time to 0.5 s, with a negligible effect on the number of

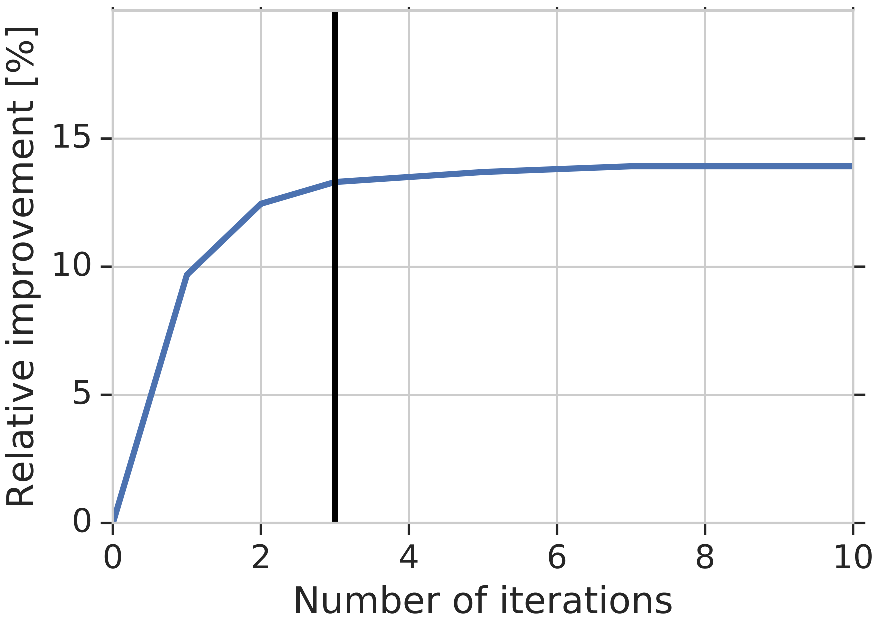


Figure 8.6: Iterations vs. relative improvement over a frozen model, averaged over all 8 datasets.

corrections required (average difference of less than 1%, over all datasets).
To increase speed further, the following optimizations are possible: (1) Using a faster backbone, e.g., with a ResNet-50 (He *et al.*, 2015), the time for an update step reduces to 0.06 s; (2) Using faster accelerators such as Google Cloud TPUs; (3) Employing a fixed feature extractor and only updating a light-weight segmentation head (Li *et al.*, 2018b).

8.4.9 Adaptation parameters

Our adaptation methods use Adam optimizer Kingma and Ba (2015) with learning rate of 10^{-6} and batch size 1. For single image adaptation we do 10 SGD steps and regularize with $\lambda = 1$ and $\gamma = 1$. For image sequence adaptation we do a 1 SGD step and use $\lambda = 0.5$ and $\gamma = 2$. For the DRIONS-DB dataset we use a larger learning rate of 10^{-5}.

8.4.10 Model details

We use DeeplabV3+ Chen *et al.* (2018c) with Xception-65 Chollet (2017) as our backbone architecture (pre-trained on ImageNet Deng *et al.* (2009) and PASCAL VOC12 Everingham *et al.* (2012)). We extend this model with 2 extra channels for the guidance maps and train it for interactive segmentation model using SGD with momentum and an initial learning rate 0.0002 with polynomial decay. We use a batch size of 2, and atrous rates $\{12, 24, 36\}$. We use in input image resolution of 513 × 513 and an output stride of 8 for the encoder and 4 for the decoder, respectively. For generating corrections, we sample at most 5 foreground and 5 background corrections for stage one of training (see Sec. 3.3 in the main paper). Corrections are encoded with disks of radius 3.

8.5 Comparison on the COCO dataset

We have showed that all our adaptation methods are exhibiting substantial improvement compared to the frozen model in many datasets including the COCO dataset. The improvement is especially large on the unseen classes of COCO (16.8% improvement, Table 1 in the main paper) and on adaptation to a particular unseen class (44% improvement for the *donut* class, Table 2 in the main paper), two cases where adaptation is particularly useful.

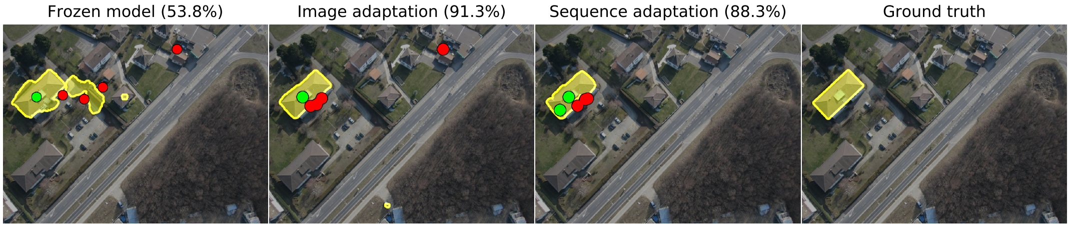
While we outperform all existing methods on PASCAL VOC12, GrabCut, Berkeley and DAVIS16, some existing works report better clicks@q% than us on COCO. e.g. Li *et al.* (2018b) reports 7.86 compared to 9.69 for our method. We however note that these results are not directly comparable. 10 instances are sampled per class to form a test set and the selected instances have not been made available by previous works. But how the selection is done is crucial, as segmenting smaller objects is more challenging. If we ignore objects smaller than 80 × 80 pixels as in Benenson *et al.* (2019), for example, our IA+SA improves from 9.9 to 5.4 (4.5 clicks less). Optimizing the architecture to better handle small objects is however not the focus of our work, as our adaptation methods work with any network architecture and can hence be combined with architectural improvements easily.

8.6 Additional Qualitative Results

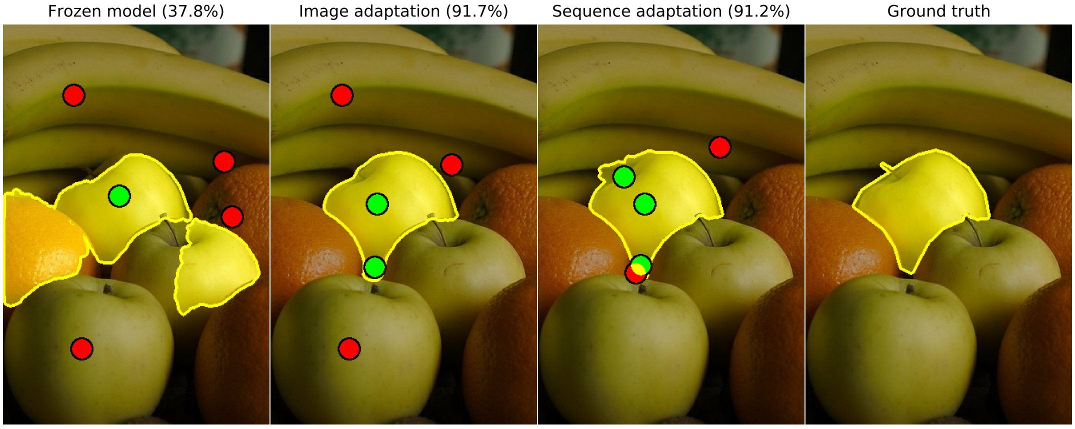
We show additional results for our two adaptation methods compared to the frozen model in Fig. 8.7.

8.7 Conclusion

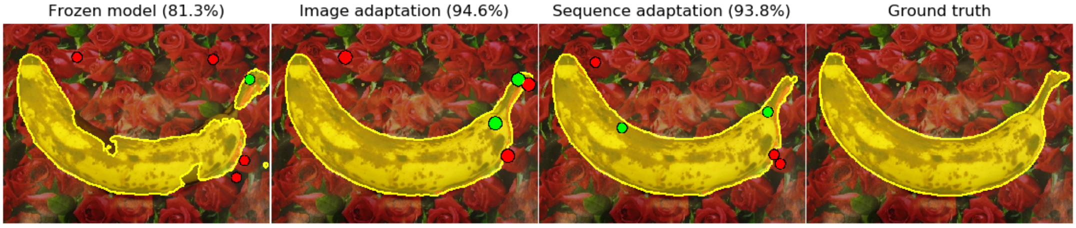
We propose to treat user corrections as sparse training examples and introduce a novel method that capitalizes on that idea to update the model parameters on-the-fly at test time. Our extensive evaluation on 8 datasets shows the benefits of our method. When distribution shifts between training and testing are small, our methods offers gains of 9%-30%. When specializing to a specific class, our gains are



(a) Rooftop Aerial dataset Sun *et al.* (2014)



(b) COCO dataset Lin *et al.* (2014)



(c) GrabCut Rother *et al.* (2004a)

Figure 8.7: Additional results. For each image we show results for our two adaptation methods and the frozen model for the same number of user corrections (IoU@5 is given in parenthesis). When the frozen model is applied to classes that are unseen during training, it sometimes produces segmentation masks that span multiple objects (8.7a & 8.7b) or do not respect object boundaries (8.7c). Single image adaptation handles such cases much better, by adapting the model parameters to that specific object and its background. This allows it to correctly segment objects even when the foreground and background have similar appearance (8.7b). Image sequence adaptation optimizes the model parameters for the test sequence. This allows it to produce good masks from very few clicks, and additional clicks are only necessary required close to the object to refine the exact boundary (8.7a & 8.7c).

12%-44%. For large domain changes, where the imaging modality changes between training and testing, it reduces the required number of user corrections by 60% and 77%.

Acknowledgement. We thank Rodrigo Benenson, Jordi Pont-Tuset, Thomas Mensink and Bastian Leibe for their inputs on this work.

9

Conclusion

In this thesis, we have presented solutions for multiple computer vision problems: object discovery in large-scale and time-varying image databases, semantic segmentation on 3D point clouds, and interactive object segmentation. While these tasks seem varied and diverse in scope, they all serve towards taking advantage of the vast amounts of available data that come in many different formats to solve computer vision problems while tackling the arising challenges. In this final chapter of the thesis, we summarize the individual contributions and discuss potential future research directions.

9.1 Discussion of Contributions

In Chapter 5, we focused on the problem of image clustering and object discovery in an incremental setting where landmarks are discovered in large-scale image collections that change over time. Most algorithms focus on static datasets and are not easily applicable to real-world scenarios where image collections are extended daily with new images.

Algorithms for object discovery need to scale to an ever-growing size of image collections (e.g., social media platforms have access to billions of images) and they need to quickly adapt to new incoming images without re-running the object discovery algorithm from scratch. Even an efficient object discovery algorithm that scales linearly with the number of images in the database can still be infeasible in practice, if a re-run is required on a daily basis. In particular such a re-run can last days or even weeks for just a mere 5% change* in the database.

Many object discovery algorithms rely on image clustering methods that can be efficiently implemented using a minimum spanning tree. However, typical landmark

---

*the percentage could still refer to millions of images discovery algorithms that operate in a static setting have to re-run the entire clustering process whenever the image database changes – even though only a small part of the clusters may be affected by the newly added images.

We addressed these issues by proposing a novel spanning tree structure called LH-MST, which approximates Minimum Spanning Trees. Our proposed tree structure can be created in a local neighborhood of the matching graph during image retrieval and can be efficiently updated whenever the image database is extended. So LH-MST can be constructed much faster and has the additional property of allowing online updates of the tree structure. Our novel LH-MST can potentially be used in many algorithms which involve a costly Minimum Spanning Tree (MST) computation. We demonstrated its applicability for two distinct methods of landmark discovery in large datasets: Single-Link Agglomerative Clustering and Iconoid Shift mode estimation. We experimentally verified that in various use cases, ranging from realistic to extreme scenarios (where a large portion of the existing database has been updated) the difference between our approximated MST and a regular MST is minimal (0.03% difference in edit distance). More importantly, it reduces the computation time from 5 days to 3 hours (on the 500k images of the Paris dataset (Weyand *et al.*, 2010)). Thus, our algorithm offers an efficient alternative for parallel and distributed clustering applications.

In the second part of this thesis (Chapters 6 and 7), we have tackled the task of semantic segmentation on 3D point clouds. We proposed various mechanisms that define and utilize long range as well as local dependencies on unstructured 3D point clouds. Specifically, in Chapter 6 we extended the PointNet architecture that splits a 3D scene into individual blocks of restricted volume and operates on them independently to take into consideration non-local dependencies. We introduced novel recurrent and sequential consolidation units that operate on multi-scale and grid neighborhoods. Our proposed mechanisms lead to enlarged receptive fields and can capture long range dependencies. We showed increased performance on the task of semantic segmentation on 3D point clouds over previous approaches in multiple challenging indoor and outdoor datasets. In Chapter 7, we expanded on the initial idea of point neighborhoods but now from the perspective of defining and encoding local- instead of global-structure. We introduced two grouping techniques which define point neighborhoods both in the Euclidean and the learned feature space based on K-Means and k-NN respectively. Additionally, we reshaped the feature space and thus the neighborhoods with dedicated loss functions. While previous works cannot exploit the local structure and treat 3D points individually or collectively together, we learn local features that are commonly shared among similar local structures as in the well-established 2D convolutions. As our experiments showed, our suggested improvements boosted the performance of deep learning models that operate directly on raw point clouds as we were able to outperform the state-of-the-art (at the time) on several popular datasets.

Finally, in Part III we proposed a new interactive object segmentation method for images. Standard approaches train on large datasets but keep the model parameters unchanged during test time. This way they can achieve good performances as long as the training and testing data distributions are similar or identical. However, this is not the case for real-life applications where the distribution almost always changes between training and testing while neural networks do not generalize well to new distributions. In our approach, we used the user corrections as sparse training examples and updated the model parameters on-the-fly during test time. This way our proposed method could not only adapt to the particular distribution of a specific image but also to new object classes introduced during test time. Our approach can even handle large domain changes as proven in our extensive experiments on 8 different datasets, where we outperform the previous state-of-the-art baseline by up to 77%.

9.2 Perspectives

9.2.1 Perspectives on Object Discovery.

Incremental object discovery. The physical world around us changes continuously and so does its digital representations. Most computer vision approaches though continuously treat it as static. Supervised techniques mostly assume a fixed number of object classes that are all available during training and testing. Unsupervised methods like clustering are bound to rebuild the clusters from scratch every time new images become available. However, this is not applicable to real-world applications. Millions of images and videos are uploaded daily on social media. Models for image retrieval and recognition have to scale with an ever-increasing number of images and adapt to new incoming images showing previously unseen objects. Only recently, a few works have been published on the topic of incremental learning for image retrieval (Chen *et al.*, 2020; Wu *et al.*, 2019a), however they focus on fine-grained image retrieval where object categories are added over time and do not extend to landmark recognition like our work presented in Chapter 5. We believe future approaches should focus on the image clustering and object discovery problem in an incremental setting in order to provide solutions for real-world use cases.

Furthermore, continuous learning is a general problem where the field of *lifelong learning* tries to tackle the issue of learning across tasks and data. Lifelong learning mostly focuses on image classification where each classification task is learned one after another. These tasks might include different datasets or labels learned in a sequential manner. The general assumption is that the training data for all the tasks are not available all the time. Still, learning under an open world setting without constraints is a new topic that receives increasing attention in the computer vision community and is a step towards more general practical pipelines.

The setups discussed in this thesis, as well as the setting of life-long learning, the applications and developed algorithms focus on ever growing image collections or extended tasks. However, specifically in the light of recent data protection laws and privacy concerns, it might be useful to consider the opposite scenario as well: users of such internet based services and social media platforms might want to delete or restrict some of their contributions. In particular they might want to entirely remove images from the database. This is a topic not yet explored, which consists in developing efficient and robust algorithms that incorporate deletions of data, while so far only the addition of new data was considered.

The long tail. Social media community photos contain millions of images depicting hundreds of thousands of classes. However, in general and for landmarks in particular, these image collections are highly *imbalanced*. There are thousands, if not millions of images depicting the *Eifel Tower* making it easily discoverable by landmark recognition algorithms but there are only a few instances available on lesser-known landmarks. An additional challenge is that these image collections contain predominately photos of other object categories besides landmarks (for example images of cats) that increase the scale of the problem and act as distractors. New large-scale datasets (Weyand *et al.*, 2020), catered to real-life applications hopefully help to kickstart a new interest in these under-represented but very practical computer vision problems that have widespread implications for many computer vision applications.

9.2.2 Perspectives on 3D Data.

Semi-supervised learning. Deep learning algorithms that focus on popular tasks like image classification and semantic segmentation, require large amounts of labelled data to be trained effectively in a supervised manner. Acquiring such data is labor-intensive and time-consuming. Compared to 2D images, these difficulties are even more pronounced when we need to obtain per point labels on 3D point clouds or video sequences. In this case, the amount of data to be labelled grows significantly. Efficient semi-supervised algorithms such as interactive object segmentation can help to reduce the required human annotation effort. These algorithms can be combined with adaptation techniques from the fields of life-long learning as in our work (presented in Part III of this thesis). Then it becomes possible to label datasets from challenging domains like the medical imaging were training examples are limited and require contributions from experts. Similar techniques can be useful on 3D and video data where labelled examples for training are equally limited and challenging to obtain.

Dynamic point clouds and actions. 3D data is becoming increasingly popular as input in many computer vision problems. 3D input is used for 3D scene understanding and in particular with the popular tasks of 3D semantic (instance) segmentation and 3D object detection. However, these tasks are performed on static 3D scenes. Yet, in most real-life scenarios 3D scenes are dynamic, ranging from rigid object movements like furniture or driving cars to non-rigid movements by humans. On one hand, this leads to new and varied number of low level tasks on 3D motion analysis; for example registration of 3D point clouds (Wang and Solomon, 2019a,b) and 3D scene flow (Liu *et al.*, 2019; Niemeyer *et al.*, 2019; Puy *et al.*, 2020; Rempe *et al.*, 2020). On the other hand, these scenes enable interactions with objects in a 3D environment. While the first set of tasks started recently to get some attention, the second set is still a largely under explored field. One of the reason for the lack of such approaches is the limited availability of fitting datasets.

Bibliography

Acuna, D., Ling, H., Kar, A., and Fidler, S. (2018). Efficient Interactive Annotation of Segmentation Datasets with Polygon-RNN++. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Adobe (2018). Select a subject with just one click. `https://helpx.adobe.com/photoshop/how-to/select-subject-one-click.html`.

Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S., and Szeliski, R. (2009). Building Rome in a Day. In *IEEE International Conference on Computer Vision (ICCV)*.

Agustsson, E., Uijlings, J. R., and Ferrari, V. (2019). Interactive full image segmentation by considering all regions jointly. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Alcorn, M. A., Li, Q., Gong, Z., Wang, C., Mai, L., Ku, W.-S., and Nguyen, A. (2019). Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. (2018). Memory aware synapses: Learning what (not) to forget. In *European Conference on Computer Vision (ECCV)*.

Aljundi, R., Kelchtermans, K., and Tuytelaars, T. (2019). Task-free continual learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Armeni, I., Ozan, S., Amir, R. Z., Helen, J., Ioannis, B., Martin, F., and Silvio, S. (2016). 3D Semantic Parsing of Large-Scale Indoor Spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Avrithis, Y., Kalantidis, Y., Tolias, G., and Spyrou, E. (2010). Retrieving Landmark and Non-landmark Images from Community Photo Collections. In *ACM International Conference on Multimedia (MM).*

Babenko, A. and Lempitsky, V. (2015). Aggregating local deep features for image retrieval. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV).*

Babenko, A., Slesarev, A., Chigorin, A., and Lempitsky, V. (2014). Neural codes for image retrieval. In *Computer Vision – ECCV 2014*, pages 584–599. Springer International Publishing.

Bai, X. and Sapiro, G. (2009). Geodesic matting: A framework for fast interactive image and video segmentation and matting. *International Journal of Computer Vision.*

Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. (2006). Speeded-Up Robust Features (SURF).

Bearman, A., Russakovsky, O., Ferrari, V., and Fei-Fei, L. (2016). What's the point: Semantic segmentation with point supervision. In *European Conference on Computer Vision (ECCV).*

Beaudet, P. (1978). Rotationally invariant image operators. In *International Joint Conference on Pattern Recognition.*

Belouadah, E. and Popescu, A. (2019). Il2m: Class incremental learning with dual memory. In *International Conference on Computer Vision (ICCV).*

Benard, A. and Gygli, M. (2017). Interactive video object segmentation in the wild. *arXiv.*

Benenson, R., Popov, S., and Ferrari, V. (2019). Large-scale interactive object segmentation with human annotators. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Boulch, A., Saux, B. L., and Audebert, N. (2017). Unstructured Point Cloud Semantic Labeling Using Deep Segmentation Networks. In *Eurographics Workshop on 3D Object Retrieval.*

Boykov, Y. and Jolly, M. P. (2001). Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *International Conference on Computer Vision (ICCV'01).*

Caelles, S., Maninis, K.-K., Pont-Tuset, J., Leal-Taixé, L., Cremers, D., and Van Gool, L. (2017). One-shot video object segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Carmona, E. J., Rincón, M., García-Feijoó, J., and Martínez-de-la Casa, J. M. (2008). Identification of the optic nerve head with genetic algorithms. *Artificial Intelligence in Medicine*.

Castrejón, L., Kundu, K., Urtasun, R., and Fidler, S. (2017). Annotating object instances with a Polygon-RNN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chen, D.-J., Chien, J.-T., Chen, H.-T., and Chang, L.-W. (2018a). Tap and shoot segmentation. In *Association for the Advancement of Artificial Intelligence*.

Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2017). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. (2016). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PP**.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. (2018b). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018c). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision (ECCV)*.

Chen, W., Liu, Y., Wang, W., Tuytelaars, T., Bakker, E. M., and Lew, M. S. (2020). On the exploration of incremental learning for fine-grained image retrieval. In *31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020*. BMVA Press.

Chen, W., Liu, Y., Wang, W., Bakker, E. M., Georgiou, T., Fieguth, P., Liu, L., and Lew, M. S. (2021). Deep image retrieval: A survey. *CoRR*, **abs/2101.11282**.

Chin, F. and Houck, D. (1978). Algorithms for Updating Minimal Spanning Trees. *Journal of Computer and System Sciences*, **16**, 333–344.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Choy, C., Gwak, J., and Savarese, S. (2019). 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Chum, O. and Matas, J. (2010). Web Scale Image Clustering–Large Scale Discovery of Spatially Related Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **32**(2), 371–377.

Chum, O., Philbin, J., Sivic, J., Isard, M., and Zisserman, A. (2007). Total Recall: Automatic Query Expansion with a Generative Feature Model for Object Retrieval. In *IEEE International Conference on Computer Vision (ICCV).*

Comaniciu, D. and Meer, P. (2002). Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **24**(5), 603–619.

Dai, A. and Nießner, M. (2018). 3DMV: Joint 3D-Multi-View Prediction for 3D Semantic Scene Segmentation. In *IEEE European Conference on Computer Vision (ECCV).*

Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017). ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Dai, A., Ritchie, D., Bokeloh, M., Reed, S., Sturm, J., and Nießner, M. (2018). ScanComplete: Large-Scale Scene Completion and Semantic Segmentation for 3D

Scans. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Engelmann, F., Stückler, J., and Leibe, B. (2016). Joint Object Pose Estimation and Shape Reconstruction in Urban Street Scenes Using 3D Shape Priors. In *Proc. of the German Conference on Pattern Recognition (GCPR)*.

Engelmann, F., Stückler, J., and Leibe, B. (2017). SAMP: Shape and Motion Priors for 4D Vehicle Reconstruction. In *Winter Conference on Applications of Computer Vision (WACV)*.

Engelmann, F., Kontogianni, T., Schult, J., and Leibe, B. (2018). Know What Your Neighbors Do: 3D Semantic Segmentation of Point Clouds. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Farquhar, S. and Gal, Y. (2018). Towards robust evaluations of continual learning. *arXiv*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*.

Fischler, M. A. and Bolles, R. C. (1981). *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*, volume 24. Association for Computing Machinery, New York, NY, USA.

Frahm, J.-M., Fite-Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.-H., Dunn, E., Clipp, B., Lazebnik, S., and Pollefeys, M. (2010). Building Rome on a Cloudless Day. In *European Conference on Computer Vision (ECCV)*.

Gaidon, A., Wang, Q., Cabon, Y., and Vig, E. (2016). Virtual Worlds as Proxy for Multi-Object Tracking Analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gammeter, S., Bossard, L., Quack, T., and Van Gool, L. (2009). I know what you did last summer: Object-level auto-annotation of holiday snaps. In *IEEE International Conference on Computer Vision (ICCV)*.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets Robotics: The Dataset. *International Journal of Robotics Research (IJRR)*, **32**(11).

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. JMLR Workshop and Conference Proceedings.

Gong, Y., Wang, L., Guo, R., and Lazebnik, S. (2014). Multi-scale orderless pooling of deep convolutional activation features. In *European Conference on Computer Vision (ECCV)*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Gordo, A., Almazán, J., Revaud, J., and Larlus, D. (2016). Deep image retrieval: Learning global representations for image search. In *European Conference on Computer Vision (ECCV)*.

Graham, B., Engelcke, M., and van der Maaten, L. (2018). 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gulshan, V., Rother, C., Criminisi, A., Blake, A., and Zisserman, A. (2010). Geodesic star convexity for interactive image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gygli, M., Norouzi, M., and Angelova, A. (2017). Deep value networks learn to evaluate and iteratively refine structured outputs. In *International Conference on Machine Learning (ICML)*.

Hackel, T., Wegner, J. D., and Schindler, K. (2016). Fast Semantic Segmentation of 3D Points Clouds with Strongly Varying Density. *Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS)*, **3**(3).

Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., and Pollefeys, M. (2017). Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark. *arXiv preprint arXiv:1704.03847*.

Hariharan, B., Arbeláez, P., Bourdev, L., Maji, S., and Malik, J. (2011). Semantic contours from inverse detectors. In *International Conference on Computer Vision (ICCV)*.

Harley, A. W., Derpanis, K. G., and Kokkinos, I. (2017). Segmentation-Aware Convolutional Networks Using Local Attention Masks. In *IEEE International Conference on Computer Vision (ICCV)*.

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Alvey Vision Conference*.

Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press, USA, 2 edition.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Heath, K., Gelfand, N., Ovsjanikov, M., Aanjaneya, M., and Guibas, L. (2010). Image webs: Computing and Exploiting Connectivity in Image Collections. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural computation (NC)*.

Hu, Y., Soltoggio, A., Lock, R., and Carter, S. (2019). A fully convolutional two-stream fusion network for interactive image segmentation. *Neural Networks*.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Huang, J. and You, S. (2016). Point cloud labeling using 3D Convolutional Neural Network. In *International Conference on Pattern Recognition (ICPR).*

Huang, Q., Wang, W., and Neumann, U. (2018). Recurrent Slice Networks for 3D Segmentation on Point Clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Jang, W.-D. and Kim, C.-S. (2019). Interactive image segmentation via backpropagating refinement scheme. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Jing, Y. and Baluja, S. (2008). PageRank for Product Image Search. In *International Conference on World Wide Web (WWW).*

Karger, D. R., Klein, P. N., and Tarjan, R. E. (1995). A Randomized Linear-time Algorithm to Find Minimum Spanning Trees. *J. ACM*, **42**(2), 321–328.

Kasyanov, A., Engelmann, F., Stückler, J., and Leibe, B. (2017). Keyframe-Based Visual-Inertial Online SLAM with Relocalization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Kim, G., Faloutsos, C., and Hebert, M. (2008). Unsupervised Modeling of Object Categories Using Link Analysis Techniques. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR).*

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., *et al.* (2017). Overcoming catastrophic forgetting in neural networks.

Klokov, R. and Lempitsky, V. (2017). Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Kontogianni, T., Mathias, M., and Leibe, B. (2016). Incremental Object Discovery in Time-Varying Image Collections. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Kontogianni, T., Engelmann, F., Hermans, A., and Leibe, B. (2017). Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds. In *IEEE International Conference on Computer Vision, 3DRMS (ICCV) Workshop.*

Kontogianni, T., Gygli, M., Uijlings, J., and Ferrari, V. (2019). Continuous adaptation for interactive object segmentation by learning from corrections. *arXiv preprint arXiv:1911.12709v1*.

Krasin, I., Duerig, T., Alldrin, N., Ferrari, V., Abu-El-Haija, S., Kuznetsova, A., Rom, H., Uijlings, J., Popov, S., Kamali, S., Malloci, M., Pont-Tuset, J., Veit, A., Belongie, S., Gomes, V., Gupta, A., Sun, C., Chechik, G., Cai, D., Feng, Z., Narayanan, D., and Murphy, K. (2017). OpenImages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://g.co/dataset/openimages*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012a). Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012b). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, **25**, 1097–1105.

Kulkarni, P., Zepeda, J., Jurie, F., Perez, P., and Chevallier, L. (2015). Hybrid multi-layer deep cnn/aggregator feature for image classification. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1379–1383.

Lai, K., Bo, L., and Fox, D. (2014). Unsupervised Feature Learning for 3D Scene Labeling. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Landrieu, L. and Simonovsky, M. (2018). Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lawin, F. J., Danelljan, M., Tosteberg, P., Bhat, G., Khan, F. S., and Felsberg, M. (2017). Deep projective 3d semantic segmentation. In *International Conference on Computer Analysis of Images and Patterns*, pages 95–107. Springer.

Lecun, Y. (1989). *Generalization and network design strategies*. Elsevier.

Li, X., Wu, C., Zach, C., Lazebnik, S., and Frahm, J.-M. (2008). Modeling and Recognition of Landmark Image Collections Using Iconic Scene Graphs. In *European Conference on Computer Vision (ECCV)*.

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., and Chen, B. (2018a). PointCNN: Convolution On X-Transformed Points. In *Neural Information Processing Systems (NIPS)*.

Li, Z. and Hoiem, D. (2017). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Li, Z., Chen, Q., and Koltun, V. (2018b). Interactive image segmentation with latent diversity. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liew, J., Wei, Y., Xiong, W., Ong, S.-H., and Feng, J. (2017). Regional interactive image segmentation networks. In *International Conference on Computer Vision (ICCV)*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. (2014). Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*.

Lindeberg, T. (1998). Feature Detection with Automatic Scale Selection. In *International Joint Conference on Pattern Recognition*.

Ling, H., Gao, J., Kar, A., Chen, W., and Fidler, S. (2019). Fast interactive object annotation with Curve-GCN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, X., Qi, C. R., and Guibas, L. J. (2019). FlowNet3D: Learning Scene Flow in 3D Point Clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, **60**(2), 91–110.

Maddern, W., Pascoe, G., Linegar, C., and Newman, P. (2017). 1 Year, 1000km: The Oxford RobotCar Dataset. *International Journal of Robotics Research (IJRR)*, **36**(1).

Mahadevan, S., Voigtlaender, P., and Leibe, B. (2018). Iteratively trained interactive segmentation. In *British Machine Vision Conference*.

Majumder, S. and Yao, A. (2019). Content-aware multi-level guidance for interactive instance segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Maturana, D. and Scherer, S. (2015). VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

McGuinness, K. and O'Connor, N. E. (2010). A comparative evaluation of interactive segmentation algorithms. *Pattern Recognition.*

Michieli, U. and Zanuttigh, P. (2019). Incremental learning techniques for semantic segmentation. In *International Conference on Computer Vision (ICCV) Workshops.*

Mikolajczyk, K. and Schmid, C. (2004). Scale & affine invariant interest point detectors. *International Journal of Computer Vision (IJCV).*

Mottaghi, R., Chen, X., Liu, X., Cho, N.-G., Lee, S.-W., Fidler, S., Urtasun, R., and Yuille, A. (2014). The Role of Context for Object Detection and Semantic Segmentation in the Wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Munoz, D., Vandapel, N., and Hebert, M. (2008). Directional Associative Markov Network for 3-D Point Cloud Classification. In *International Symposium on 3D Data Processing, Visualization and Transmission.*

Niemeyer, M., Mescheder, L., Oechsle, M., and Geiger, A. (2019). Occupancy Flow: 4D Reconstruction by Learning Particle Dynamics. In *International Conference on Computer Vision (ICCV).*

Noh, H., Hong, S., and Han, B. (2015). Learning Deconvolution Network for Semantic Segmentation. In *IEEE International Conference on Computer Vision (ICCV).*

Ondruska, P., Dequaire, J., Zeng Wang, D., and Posner, I. (2016). End-to-End Tracking and Semantic Segmentation Using Recurrent Neural Networks. In *Robotics: Science and Systems Workshops (RSSW).*

Ong, E.-J., Husain, S., and Bober, M. (2017). Siamese network of deep fisher-vector descriptors for image retrieval. *ArXiv*, **abs/1702.00338**.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Papadopoulos, D. P., Uijlings, J. R. R., Keller, F., and Ferrari, V. (2016). We don't need no bounding-boxes: Training object class detectors using only human verification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Perazzi, F., Pont-Tuset, J., McWilliams, B., Van Gool, L., Gross, M., and Sorkine-Hornung, A. (2016). A benchmark dataset and evaluation methodology for video object segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Perronnin, F., Liu, Y., Snchez, J., and Poirier, H. (2010). Large-scale image retrieval with compressed fisher vectors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Philbin, J. and Zisserman, A. (2008). Object Mining using a Matching Graph on Very Large Image Collections. In *Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP).*

Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2007). Object Retrieval with Large Vocabularies and Fast Spatial Matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Pohlen, T., Hermans, A., Mathias, M., and Leibe, B. (2017). Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Price, B. L., Morse, B., and Cohen, S. (2010). Geodesic graph cut for interactive image segmentation. In *CVPR.*

Puy, G., Boulch, A., and Marlet, R. (2020). FLOT: Scene Flow on Point Clouds Guided by Optimal Transport. In *European Conference on Computer Vision (ECCV).*

Qi, C. R., Su, H., Niessner, M., Dai, A., Yan, M., and Guibas, L. J. (2016). Volumetric and Multi-View CNNs for Object Classification on 3D Data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017a). PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *Conference on Neural Information Processing Systems (NIPS).*

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017b). PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Qi, H., Brown, M., and Lowe, D. G. (2018). Low-shot learning with imprinted weights. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Qi, X., Liao, R., Ya, J., Fidler, S., and Urtasun, R. (2017c). 3D Graph Neural Networks for RGBD Semantic Segmentation. In *IEEE International Conference on Computer Vision (ICCV).*

Quack, T., Leibe, B., and Van Gool, L. (2008). World-scale Mining of Objects and Events from Community Photo Collections. In *International Conference on Content-based Image and Video Retrieval (CIVR).*

Radenovic, F., Tolias, G., and Chum, O. (2016). Cnn image retrieval learns from bow: Unsupervised fine-tuning with hard examples. In *European Conference on Computer Vision (ECCV)*, pages 3–20.

Ravi, S. and Larochelle, H. (2016). Optimization as a model for few-shot learning. In *International Conference on Learning Representations (ICLR).*

Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops.*

Razavian, A. S., Sullivan, J., Carlsson, S., and Maki, A. (2016). Visual instance retrieval with deep convolutional networks.

Rebuffi, S., Kolesnikov, A., Sperl, G., and Lampert, C. (2017). icarl: Incremental classifier and representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. (2018). Do CIFAR-10 classifiers generalize to CIFAR-10? *arXiv.*

Rempe, D., Birdal, T., Zhao, Y., Gojcic, Z., Sridhar, S., and Guibas, L. J. (2020). CaSPR: Learning Canonical Spatiotemporal Point Cloud Representations. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Riegler, G., Ulusoy, A. O., and Geiger, A. (2017). OctNet: Learning Deep 3D Representations at High Resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Rohlf, F. (1973). Hierarchical Clustering using the Minimum Spanning Tree. *The Computer Journal (CJ)*, **16**, 93–95.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65**(6), 386.

Rother, C., Kolmogorov, V., and Blake, A. (2004a). GrabCut - Interactive Foreground Extraction using Iterated Graph Cut. *Special Interest Group on Graphics and Interactive Techniques*, **23**.

Rother, C., Kolmogorov, V., and Blake, A. (2004b). Grabcut: Interactive foreground extraction using iterated graph cuts. In *Special Interest Group on Graphics and Interactive Techniques*.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., and Hinton, G. E. (1986). *Schemata and Sequential Thought Processes in PDP Models*, page 7?57. MIT Press, Cambridge, MA, USA.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). *Learning Representations by Back-Propagating Errors*, pages 696–699. MIT Press, Cambridge, MA, USA.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, **115**(3), 211–252.

Sattler, T., Leibe, B., and Kobbelt, L. (2009). Scramsac: Improving ransac‘s efficiency with a spatial consistency filter. In *IEEE International Conference on Computer Vision (ICCV)*.

Schult*, J., Engelmann*, F., Kontogianni, T., and Leibe, B. (2020). DualConvMesh-Net: Joint Geodesic and Euclidean Convolutions on 3D Meshes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sheikh, Y. A., Khan, E., and Kanade, T. (2007). Mode-seeking by Medoidshifts. In *IEEE International Conference on Computer Vision (ICCV)*.

Shmelkov, K., Schmid, C., and Alahari, K. (2017). Incremental learning of object detectors without catastrophic forgetting. In *International Conference on Computer Vision (ICCV)*.

Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor Segmentation and Support Inference from RGBD Images. In *European Conference on Computer Vision (ECCV)*.

Simon, I., Snavely, N., and Seitz, S. (2007). Scene Summarization for Online Image Collections. In *IEEE International Conference on Computer Vision (ICCV)*.

Simonovsky, M. and Komodakis, N. (2017). Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sivic, J. and Zisserman, A. (2003). Video Google: A Text Retrieval Approach to Object Matching in Videos. In *IEEE International Conference on Computer Vision (ICCV)*.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Neural Information Processing Systems*.

Sofiiuk, K., Petrov, I., Barinova, O., and Konushin, A. (2020). F-BRS: Rethinking Backpropagating Refinement for Interactive Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Spira, P. and Pan, A. (1975). On Finding and Updating Spanning Trees and Shortest Paths. *SIAM Journal on Computing*, **4**(3), 375–380.

Sun, X., Christoudias, C. M., and Fua, P. (2014). Free-shape polygonal object localization. In *European Conference on Computer Vision (ECCV)*.

Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A. A., and Hardt, M. (2019). Test-time training for out-of-distribution generalization. *arXiv*.

Tatarchenko, M., Dosovitskiy, A., and Brox, T. (2017). Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs. In *IEEE International Conference on Computer Vision (ICCV)*.

Tchapmi, L. P., Choy, C. B., Armeni, I., Gwak, J., and Savarese, S. (2017). SEGCloud: Semantic Segmentation of 3D Point Clouds. In *International Conference on 3D Vision (3DV)*.

Tolias, G., Sicre, R., and Jégou, H. (2016). Particular object retrieval with integral max-pooling of CNN activations. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Tuytelaars, T. and Mikolajczyk, K. (2008). *Local Invariant Feature Detectors: A Survey*. Now Publishers Inc., Hanover, MA, USA.

Voigtlaender, P. and Leibe, B. (2017). Online adaptation of convolutional neural networks for video object segmentation. In *British Machine Vision Conference*.

Vosselman, G. (2013). Point Cloud Segmentation for Urban Scene Classification. *Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS)*.

Wang, S., Suo, S., Ma, W., Pokrovsky, A., and Urtasun, R. (2018a). Deep Parametric Continuous Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wang, S., Suo, S., Ma, W.-C., Pokrovsky, A., and Urtasun, R. (2018b). Deep parametric continuous convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wang, Y. and Solomon, J. M. (2019a). Deep Closest Point: Learning Representations for Point Cloud Registration. In *International Conference on Computer Vision (ICCV)*.

Wang, Y. and Solomon, J. M. (2019b). PRNet: Self-Supervised Learning for Partial-to-Partial Registration. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2018c). Dynamic Graph CNN for Learning on Point Clouds. *Computing Research Repository CoRR*, **abs/1801.07829**.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019). Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics (TOG)*.

Wei, X.-S., Luo, J.-H., Wu, J., and Zhou, Z.-H. (2017). Selective convolutional descriptor aggregation for fine-grained image retrieval. *IEEE transactions on image processing*, **26**(6), 2868–2881.

Weyand, T. and Leibe, B. (2011). Discovering Favorite Views of Popular Places with Iconoid Shift. In *IEEE International Conference on Computer Vision (ICCV)*.

Weyand, T. and Leibe, B. (2013). Hierarchical Iconoid Shift Discovering Details and Scene Structure with Hierarchical Iconoid Shift. In *IEEE International Conference on Computer Vision (ICCV)*.

Weyand, T., Hosang, J., and Leibe, B. (2010). An Evaluation of Two Automatic Landmark Building Discovery Algorithms for City Reconstruction. In *European Conference on Computer Vision (ECCVW)*.

Weyand, T., Araujo, A., Cao, B., and Sim, J. (2020). Google Landmarks Dataset v2 - A Large-Scale Benchmark for Instance-Level Recognition and Retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, D., Dai, Q., Liu, J., Li, B., and Wang, W. (2019a). Deep incremental hashing network for efficient image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, W., Qi, Z., and Li, F. (2019b). PointConv: Deep Convolutional Networks on 3D Point Clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, W., Qi, Z., and Fuxin, L. (2019c). Pointconv: Deep convolutional networks on 3d point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3D ShapeNets: A Deep Representation for Volumetric Shape Modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, Z., Shen, C., and van den Hengel, A. (2016). High-performance Semantic Segmentation Using Very Deep Fully Convolutional Networks. *arXiv preprint arXiv:1604.04339*.

Xie, J., Kiefel, M., Sun, M.-T., and Geiger, A. (2016). Semantic instance annotation of street scenes by 3d to 2d label transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xiong, X., Munoz, D., Bagnell, J. A., and Hebert, M. (2011a). 3-D Scene Analysis via Sequenced Predictions over Points and Regions. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Xiong, X., Munoz, D., Andrew, J., and Hebert, B. M. (2011b). 3-D Scene Analysis via Sequenced Predictions over Points and Regions. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Xu, N., Price, B., Cohen, S., Yang, J., and Huang, T. (2016). Deep interactive object selection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xu, N., Yang, L., Fan, Y., Yue, D., Liang, Y., Yang, J., and Huang, T. (2018a). YouTube-VOS: A Large-Scale Video Object Segmentation Benchmark. *arXiv*.

Xu, P., Davoine, F., Bordes, J., Zhao, H., and Denoeux, T. (2013). Information Fusion on Oversegmented Images: An Application for Urban Scene Understanding. In *MVA*.

Xu, Y., Fan, T., Xu, M., Zeng, L., and Qiao, Y. (2018b). SpiderCNN: Deep Learning on Point Sets with Parameterized ConvolutionalFilters. In *IEEE European Conference on Computer Vision (ECCV)*.

Yi, L., Su, H., Guo, X., and Guibas, L. J. (2016). SyncSpecCNN: Synchronized Spectral CNN for 3D Shape Segmentation. *arXiv preprint arXiv:1612.00606*.

Yu, F. and Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*.

Yue-Hei Ng, J., Yang, F., and Davis, L. S. (2015). Exploiting local features from deep networks for image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *International Conference on Machine Learning (ICML)*.

Zhang, R., Candra, S. A., Vetter, K., and Zakhor, A. (2015). Sensor Fusion for Semantic Segmentation of Urban Scenes. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Zheng, Y.-T., Zhao, M., Song, Y., Adam, H., Buddemeier, U., Bissacco, R., Brucher, O., Chua, T.-S., and Neven, H. (2009). Tour the World: Building a Web-Scale

Landmark Recognition Engine. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2016). Semantic understanding of scenes through the ADE20K dataset. *arXiv preprint arXiv:1608.05442*.

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. (2017a). Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2017b). Scene parsing through ADE20K dataset. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2018). Semantic understanding of scenes through the ADE20K dataset. *International Journal of Computer Vision*.

Zhou, Y. and Tuzel, O. (2018). VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhou, Z., Shin, J., Zhang, L., Gurudu, S., Gotway, M., and Liang, J. (2017c). Fine-tuning convolutional neural networks for biomedical image analysis: actively and incrementally. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Curriculum Vitae

Theodora Kontogianni

E-Mail	kontogianni@vision.rwth-aachen.de
Place of Birth	Ioannina, Greece
Citizenship	Greek

Education

May. 2014 – Aug. 2021	Ph.D. in Computer Vision RWTH Aachen University Supervisor: Prof. Dr. Bastian Leibe
Oct. 2009 – Nov. 2011	M.Sc. in Media Informatics RWTH Aachen University

Professions

June. 2019 – Nov. 2019	Research Intern at Google Switzerland in Zurich, Switzerland
May. 2018 – Aug. 2018	Software Engineer Intern at Google LLC in Los Angeles, USA.
Mar. 2012 – Apr. 2014	Data Scientist at Zalando AG in Berlin, Germany
Jan. 2011 – Mar. 2011	Student Research Assistant at the Computer Vision Group, RWTH Aachen University

Fakultät für
Mathematik, Informatik
und Naturwissenschaften

RWTH AACHEN
UNIVERSITY

Eidesstattliche Erklärung

Ich, Theodora Kontogianni

erklärt hiermit, dass diese Dissertation und die darin dargelegten Inhalte die eigenen sind und selbstständig, als Ergebnis der eigenen originären Forschung, generiert wurden.

Hiermit erkläre ich an Eides statt

1. Diese Arbeit wurde vollständig oder größtenteils in der Phase als Doktorand dieser Fakultät und Universität angefertigt;

2. Sofern irgendein Bestandteil dieser Dissertation zuvor für einen akademischen Abschluss oder eine andere Qualifikation an dieser oder einer anderen Institution verwendet wurde, wurde dies klar angezeigt;

3. Wenn immer andere eigene- oder Veröffentlichungen Dritter herangezogen wurden, wurden diese klar benannt;

4. Wenn aus anderen eigenen- oder Veröffentlichungen Dritter zitiert wurde, wurde stets die Quelle hierfür angegeben. Diese Dissertation ist vollständig meine eigene Arbeit, mit der Ausnahme solcher Zitate;

5. Alle wesentlichen Quellen von Unterstützung wurden benannt;

6. Wenn immer ein Teil dieser Dissertation auf der Zusammenarbeit mit anderen basiert, wurde von mir klar gekennzeichnet, was von anderen und was von mir selbst erarbeitet wurde;

7. Ein Teil oder Teile dieser Arbeit wurden zuvor veröffentlicht (siehe Auflistung nächste Seite).

29.11.2021

Publications

T. Kontogianni†, M. Gygli†, J. Uijlings, V. Ferrari. Continuous Adaptation for Interactive Object Segmentation by Learning from Corrections. In European Conference on Computer Vision (ECCV), 2020.

J. Schult, F. Engelmann, T. Kontogianni, B. Leibe. DualConvMesh-Net: Joint Geodesic and Euclidean Convolutions on 3D Meshes. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

C. Elich, F. Engelmann, T. Kontogianni, B. Leibe. 3D Bird's-Eye-View Instance Segmentation. In German Conference on Pattern Recognition (GCPR), 2019.

F. Engelmann, T. Kontogianni, J. Schult, B. Leibe. Know What Your Neighbors Do: 3D Semantic Segmentation of Point Clouds. In European Conference on Computer Vision (ECCV), GMDL Workshop, 2018.

T. Kontogianni†, F. Engelmann†, A. Hermans, B. Leibe. Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds. In International Conference on Applications of Computer Vision (ICCV), 3DRMS Workshop, 2017.

T. Kontogianni, M. Mathias, B. Leibe. Incremental Object Discovery in Time-Varying Image Collections. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.