Learning new representations for 3D point cloud semantic segmentation

Hugues Thomas

▶ To cite this version:

Hugues Thomas. Learning new representations for 3D point cloud semantic segmentation. Machine Learning [cs.LG]. Université Paris sciences et lettres, 2019. English. NNT : 2019PSLEM048 . tel-02458455

HAL Id: tel-02458455
https://pastel.archives-ouvertes.fr/tel-02458455
Submitted on 28 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES ParisTech

Apprentissage de nouvelles représentations pour la sémantisation de nuages de points 3D

Learning new representations for 3D point cloud semantic segmentation

Soutenue par
Hugues THOMAS
Le 19 novembre 2019

Ecole doctorale n° 432
Sciences et Métiers de l'Ingénieur

Spécialité
Mathématiques, Informatique Temps-Réel, Robotique

Composition du jury :

Paul CHECCHIN Professeur, Université Clermont Auvergne (UCA)	*Rapporteur*
Bruno VALLET Chargé de recherche, Institut national de l'information géographique et forestière (IGN)	*Rapporteur*
Pascal, MONASSE Professeur, Ecole Nationale des Ponts et Chaussées	*Président*
Martin WEINMANN Chargé de recherche, Karlsruhe Institute of Technology (KIT)	*Examinateur*
Beatriz MARCOTEGUI Professeur, Mines ParisTech	*Examinateur*
Jean-Emmanuel DESCHAUD Chargé de recherche, Mines ParisTech	*Examinateur*
François GOULETTE Professeur, Mines ParisTech	*Directeur de thèse*

MINES ParisTech
60 Boulevard Saint-Michel
75006 Paris
France

Université PSL
60 Rue Mazarine
75006 Paris
France

Acknowledgments

I would like to start by thanking my thesis supervisors, François Goulette, Beatriz Marcotegui and Jean-Emmanuel Deschaud, for accepting me as their students. Each one of them provided useful insights, advice and support in their own way. I particularly appreciated how they managed to give me the trust and freedom I needed to explore promising research directions, while knowing when I was to be redirected to avoid scattering. I am also grateful to the rest of my committee for devoting their precious time to reviewing this thesis and for coming up with many great, thought-provoking questions.

I would also like to express my gratitude to Terra3D for securing funding for this three years and providing me with a great work environment. Yann Le Gall in particular, my industrial supervisor, has been a precious collaborator, with sharp thinking and challenging discussions, but also a great friend, with surprisingly deadly ping-pong lobs.

Whether they work at the lab or at Terra3D, I was blessed to have amazing colleagues and friends. Between the chats, the help they could provide, or the ideas we have exchanged, they all contributed to the results of this thesis. They are also the reason why the lab had such a warm and relaxed atmosphere, with ping-pong or board games. Thank you, Xavier Roynard, Paul Chauchat, Hassan Bouschiba, Yann Le Gall, Kourosh Beroukhim, Andres Serna, Antoine Plat, Michelle Valente, Guillaume Devineau, Daniele Sportillo, Martin Dubreuil, Emmanuel Moisan, Sami Jouaber, Philip Polack, Martin Brossard, Sofiane Horache, Marin Toromanoff, Simon Tamayo, Arthur Gaudron, Marion Pilte, Grégoire Dupont de Dinechin et Aubrey Clausse.

I had the chance to be able to visit the geometric computing group in the Computer Science Department of Stanford University during the second year of my thesis. I am obliged to Leonidas Guibas, who welcomed me in his team during three months, and help me a lot. There, I was fortunate enough to collaborate with international researchers and in particular, Charles Ruizhongtai Qi, whose ideas contributed to the success of our methods.

Very special thanks go to my dear parents, Christine and François, for their love and unconditional support, through this thesis but also through the rest of my studies.

Eventually, to my love, my partner in life, in the good and difficult moments, thank you Aurélie. You are and always will be the pillar that supports me.

Table of Contents

I. Introduction 17
- French Abstract, résumé français 18
  I.1. Context 20
  I.2. Clarification of the subject 21
  I.3. The three tiers of representations 24
  I.4. Thesis outline and contributions 26
II. About 3D point clouds 29
  II.1. Point Cloud in theory 32
    II.1.1. Point cloud definition 32
    II.1.2. Point cloud properties 33
  II.2. Point cloud in practice 35
    II.2.1. Scanning technologies 35
    II.2.2. Point cloud datasets 38
  II.3. Point neighborhoods 45
    II.3.1. Neighborhood types 45
    II.3.2. Accelerating structures 46
  II.4. Point Cloud Sampling 48
    II.4.1. Principle 48
    II.4.2. Deterministic versus stochastic subsampling methods 49
III. Hand-crafted approaches for point cloud processing 51
  III.1. Overview of non-learning approaches 54
    III.1.1. Primitive extraction 54
    III.1.2. Contour extraction 57
    III.1.3. Point cloud segmentation 59
  III.2. Learning-based methods using hand-crafted features 60
    III.2.1. Pointwise classification 60
    III.2.2. Segmented object classification 62
    III.2.3. Graph-based classification 64
  III.3. Proposed pointwise classification method 67
    III.3.1. Unresolved issues from the state of the art 67

III.3.2. New multiscale spherical neighborhoods 68

III.3.3. Comparison with multiscale KNN 70

III.3.4. Learning strategy with a limited amount of points 73

III.3.5. Performances on large scale datasets 76

III.3.6. Study of the influence of density in our multiscale neighborhood 87

III.3.7. Conclusion for the design of neural network representations 88

IV. Deep learning architectures for point cloud processing 89

IV.1. Overview of 3D deep learning approaches 92

IV.1.1. Projective networks 92

IV.1.2. Graph convolution networks 96

IV.1.3. Pointwise multi-layer perceptrons 99

IV.1.4. Point convolution networks 103

IV.2. KPConv: a new point convolution 110

IV.2.1. A convolution kernel defined by points 111

IV.2.2. Layout of the kernel points 112

IV.2.3. A flexible operator 113

IV.2.4. Rigid or deformable kernels 116

IV.2.5. KPConv network layers 119

IV.2.6. Kernel point network architectures 121

IV.2.7. Implementation details 123

IV.2.8. Learning strategy for different tasks 126

IV.3. Experiments 130

IV.3.1. 3D shape classification on ModelNet40 130

IV.3.2. 3D shape segmentation on ShapeNetPart 131

IV.3.3. 3D scene segmentation on several datasets 134

IV.3.4. Model sizes and speeds 141

IV.3.5. Impact of the number of kernel points 141

IV.3.6. Visualization of learned features 142

IV.3.7. Study of the Effective Receptive Field 142

Conclusion 145

Overview 145

Future works 146

Publications 149

National conferences 149

International conferences 149

References 151

A. More KPConv semantic segmentation results 161

A.1. Class scores on semantic segmentation datasets 162

A.2. Illustration of 3D scenes semantic segmentation 164

A.3. Illustration of 3D objects part segmentation 168

List of Figures

Figure 1. Illustration of the steps of a computer vision algorithm on a simple classification task. The point cloud is first transformed into a representation, which is then used in the classification process. 21
Figure 2. Illustration of the hierarchy of semantic understanding tasks on point clouds........ 23
Figure 3. The three tiers of representations illustrated with point cloud processing algorithms. In tier 1, the objects are detected as simple primitives like planes. In tier 2, the objects are described by handcrafted features. In tier 3, the objects are processed by neural networks which learn their own features. 25
Figure 4. Link between image and point clouds. A basic point cloud correspond to a binary grid in image space. To reach the descriptive level of an image, a point cloud must be associated with features...... 33
Figure 5. Photogrammetry and stereoscopy. (a) Example of stereoscopic camera. (b) Principle of photogrammetry: multiple images are used to reconstruct a building in 3D. (c) Real reconstruction example from Meshroom open source software........ 35
Figure 6. Depth cameras. (a) Example of a Kinect camera that uses structured light. (b) Principle of structured-light cameras: geometric light patterns are projected on the scene and the deformation of the pattern reveals the geometry. (c) Image of an indoor room obtained with a structured light camera. 36
Figure 7. Laser scanners. (a) Example of Velodyne lidars. (b) Laser scanners used for perception with a vertical rotation axis or for mapping with a bent rotation axis. (c) Illustration of a lidar frame, capturing a 3D scene........ 37
Figure 8. Screenshots of Paris-Lille-3D dataset. 39
Figure 9. Screenshots of Semantic3D dataset. 40
Figure 10. Screenshots of S3DIS dataset........ 41
Figure 11. Screenshots of Scannet dataset. 42
Figure 12. Screenshot of Rue Madame dataset. 43
Figure 13. Screenshot of IQmulus & TerraMobilita dataset. 43
Figure 14. Examples of objects in ModelNet40, subsampled as point clouds........ 43
Figure 15. Examples of annotated parts in ShapeNetPart dataset. 44
Figure 16. The three most used neighborhood types. 45
Figure 17. Illustration of a 3D grid around *Stanford Bunny* point cloud........ 46
Figure 18. Illustration of an octree around *Stanford Bunny* point cloud. 47
Figure 19. Example of space partitioning with a kd-tree in two dimensions. 47
Figure 20. Illustration of the density variation in a fixed laser scan from Semantic3D dataset. Red denotes higher densities (close to the scanner) and blue lower densities (far from the scanner). 48
Figure 21. Example of plane extraction in 3D point cloud using RANSAC. A random plane with a low score (left), and the best plane (right) are shown. Scene taken from Paris-Lille-3D dataset. 55
Figure 22. Illustration of crease edges and jump edges on an image of a point cloud. The jump edges only appear because of the point of view. Scene taken from Paris-Lille-3D dataset. 58
Figure 23. Standard pointwise classification framework. A set of features is computed for every point of the scene and a classifier is used to label each point independently. 60

Figure 24. Standard pipeline for the classification of segmented objects. The ground and the objects are isolated by a segmentation algorithm, then, each object is classified as a whole. 63
Figure 25. Standard pipeline for the regularization of classified point clouds. Image from (Landrieu et al., 2017). 65
Figure 26. Behavior of multiscale neighborhoods defined with KNN or with spherical neighborhoods. 69
Figure 27. Precision, Recall and F1 score on scene Lille1_1 of Paris-Lille-3D dataset during active learning convergence. 77
Figure 28. Qualitative comparison of random and active learning on an indoor scene (left) and an outdoor scene (right). Random strategy has better recalls (we see the sofa entirely detected), but active learning has better precisions (less false positives on the walls or facades). We can also see active learning is better at detecting tree trunks. 80
Figure 29. Comparison of the training points chosen with random learning (red) and active learning (blue). Only the points of the class vegetation are shown. Active learning is better at detecting tree trunks as it picks more training points in them. 81
Figure 30. Pointwise classification pipeline results on Semantic3D dataset (left) with groundtruth (right). 84
Figure 31. Pointwise classification pipeline results on S3DIS dataset (left) with groundtruth (right). 85
Figure 32. Pointwise classification pipeline results on Paris-Lille-3D dataset (left) with groundtruth (right). 85
Figure 33. Influence of the parameter ρ on the classification performances and the computation speed on folds of Paris-Lille-3D dataset 87
Figure 34. Semantic classification of 3D objects with a multi-view CNN. Image from (Su et al., 2015). 93
Figure 35. Semantic segmentation of 3D scenes with a multi-view CNN. Image from *SnapNet* (Boulch et al., 2017) 93
Figure 36. Voxel projective network. The point cloud is projected in a 3D occupancy grid, and then processed by standard 3D convolution. Image from (Maturana and Scherer, 2015). 94
Figure 37. *VoxelNet* encoding steps (Zhou and Tuzel, 2018). The points within each voxels are transformed to a learnable vector representation characterizing the shape information. 94
Figure 38. Bilateral convolution layer, with splatting/slicing process from *SplatNet* architecture (Su et al., 2018). 95
Figure 39. Architecture of a graph CNN with the four ingredients of a graph convolutional layer. Figure from (Defferrard et al., 2016) 97
Figure 40. Illustration of the patch operators from methods using polar coordinates to define their filters. Figure from (Monti et al., 2017) 98
Figure 41. Pointwise MLP block in PointNet architecture. 99
Figure 42. Illustration of the functions learned by the neurons of a MLP on 2D points. 100
Figure 43. PointNet architecture, (Qi et al., 2017) 100
Figure 44. T-Net block in PointNet architecture. d is the dimension of the input features. 101
Figure 45. PointNet++ architecture (Qi et al., 2017b). One of the first hierarchical neural networks on point clouds. 102

Figure 46. Illustration of deep parametric convolution compared to an image convolution (S. Wang et al., 2018). 104

Figure 47. Illustration of the *Probability Density Function* in Monte Carlo Convolution (Hermosilla et al., 2018). The purple disk shows the bandwidth used for density estimation and the pink neighbors are the one in a denser area. 106

Figure 48. Illustration on Pointwise convolution (Hua et al., 2018). The nearest neighbors are binned into kernel voxels, and the voxels are convolved with kernel weights. 106

Figure 49. Computational flow of the point convolution by extension (Atzmon et al., 2018). 108

Figure 50. KPConv illustrated on 2D points. Input points with a constant scalar feature (in grey) are convolved through a KPConv that is defined by a set of kernel points (in black) with filter weights on each point. 110

Figure 51. Comparison between an image convolution (left) and a KPConv (right) on 2D points for a simpler illustration. In the image, each pixel feature vector is multiplied by a weight matrix $Wkk < K$ assigned by the alignment of the kernel with the image. In KPConv, input points are not aligned with kernel points, and their number can vary. Therefore, each point feature fi is multiplied by all the kernel weight matrices, with a correlation coefficient hik depending on its relative position to kernel points. 111

Figure 52. . Illustration of the kernel points in stable dispositions. 113

Figure 53. Example of customizations of the influence area overlapping in 2D. The area of each kernel point is shown with a different color. 114

Figure 54. Example of customizations of the kernel points layout in 2D. 114

Figure 55. Example of customizations of the correlation function and aggregation mode. 115

Figure 56. Customization of the input density illustrated in 2D for clarity. 116

Figure 57. Deformable KPConv illustrated on 2D points. 117

Figure 58. Illustration of the lost kernel points effect for deformable KPConv without regularization. All the points of the deformable kernel are above or under the ground. Their influence area does not reach any input point anymore, which means the gradients of their deformations are zero. They are stuck there. 118

Figure 59. Illustration of deformed kernels (yellow) compared to the original rigid kernels of KPConv (red). With the regularization loss, the kernel points fit the point cloud geometry. 120

Figure 60. Illustration of our 2 network architectures for segmentation (top) and classification (bottom) of 3D point clouds. During a forward pass, features are transformed by consecutive operations (represented by edge colors) while points are fed to each layer as a support structure guiding the operations. 122

Figure 61. Convolutional blocks used in our architectures. Both rigid (top) and deformable (bottom) KPConv use ResNet connections, batch normalization and leaky ReLU. Optional blocks: shortcut max pooling(1) is only needed for strided KPConv, and shortcut 1×1 convolution(2) is only needed when $Din \neq 2D$. 122

Figure 62. Illustration of the parallelization of the input pipeline of KPConv Networks. We denote the batches as BX and the point clouds they comprise as pcX. 124

Figure 63. Illustration of the neighbor features gathering in KPConv, with the shadow neighbor trick. Each blue line indicates where a neighbor index points.. 125
Figure 64. Illustration of the spatially regular picking strategy with potential updates in 1D for clarity. This strategy helps picking input spheres regularly in the space, without being biased by the varying densities. 128
Figure 65. Example of objects from ShapeNetPart dataset, segmented with KP-FCNN. The different objects have varying number of parts... 131
Figure 66. Example among the worst results of our method on ShapeNetPart dataset. We can see that the errors are due to groundtruth mistakes, object misalignment or unusual shapes. 133
Figure 67. Paris-Lille-3D scene segmented with KP-FCNN. Top: predictions. Middle: Groundtruth. Bottom: Errors.. 137
Figure 68. Semantic3D scene segmented with KP-FCNN. Top: predictions. Middle: Groundtruth. Bottom: Errors.. 138
Figure 69. S3DIS scene segmented with KP-FCNN. Top: predictions. Middle: Groundtruth. Bottom: Errors.. 139
Figure 70. Multiple Scannet scenes segmented with KP-FCNN. Top: predictions. Middle: Groundtruth. Bottom: Errors. The scenes are placed together for visualization, but have been segmented independently. 140
Figure 71. Ablation study on Scannet validation set. Evolution of the mIoU when reducing the number of kernel points. 142
Figure 72. Low and high level features learned in KP-CNN. Each feature is displayed on 2 input point clouds taken from ModelNet40. High activations are in red and low activations in blue. 143
Figure 73. KPConv ERF at layer 4 of KP-FCNN, trained on Scannet. The green dots represent the ERF centers. ERF values are merged with scene colors as red intensity. The more red a point is the more influence it has on the green point features. 144
Figure 74. Illustration of a 3D scene from Paris-Lille-3D dataset segmented with KPConv. Top: predictions. Middle: Groundtruth. Bottom: Errors... 164
Figure 75. Illustration of a 3D scene from Semantic3D dataset segmented with KPConv. Top: predictions. Middle: Groundtruth. Bottom: Errors... 165
Figure 76. Illustration of a 3D scene from S3DIS dataset segmented with KPConv. Top: predictions. Middle: Groundtruth. Bottom: Errors. 166
Figure 77. Illustration of two 3D scenes from Scannet dataset segmented with KPConv. Top: predictions. Middle: Groundtruth. Bottom: Errors. 167
Figure 78. Examples of airplanes from ShapeNetPart dataset, segmented with KP-FCNN. 168
Figure 79. Examples of bags from ShapeNetPart dataset, segmented with KP-FCNN. 168
Figure 80. Examples of caps from ShapeNetPart dataset, segmented with KP-FCNN. 169
Figure 81. Examples of cars from ShapeNetPart dataset, segmented with KP-FCNN...................... 169
Figure 82. Examples of chairs from ShapeNetPart dataset, segmented with KP-FCNN.................... 170
Figure 83. Examples of earphones from ShapeNetPart dataset, segmented with KP-FCNN. 170
Figure 84. Examples of guitars from ShapeNetPart dataset, segmented with KP-FCNN. 171
Figure 85. Examples of knives from ShapeNetPart dataset, segmented with KP-FCNN. 171

Figure 86. Examples of lamps from ShapeNetPart dataset, segmented with KP-FCNN.................... 172
Figure 87. Examples of laptops from ShapeNetPart dataset, segmented with KP-FCNN.................. 172
Figure 88. Examples of motorbikes from ShapeNetPart dataset, segmented with KP-FCNN............ 173
Figure 89. Examples of mugs from ShapeNetPart dataset, segmented with KP-FCNN..................... 173
Figure 90. Examples of pistols from ShapeNetPart dataset, segmented with KP-FCNN................... 174
Figure 91. Examples of rockets from ShapeNetPart dataset, segmented with KP-FCNN.................. 174
Figure 92. Examples of skateboards from ShapeNetPart dataset, segmented with KP-FCNN. 175
Figure 93. Examples of tables from ShapeNetPart dataset, segmented with KP-FCNN.................... 175

List of Tables

Table 1. Characteristics of the semantic segmentation datasets used in our experiments. 38
Table 2. Features used for our comparative experiment. 71
Table 3. Average IoU (with standard deviation) on Rue Madame (top) and IQmulus & TerraMobilita (bottom) datasets. Previous results are converted from corresponding articles. 73
Table 4. Class IoU on the training scene of Rue Madame and IQmulus & TerraMobilita datasets, with our pointwise classification algorithm, using random point picking or active learning. 76
Table 5. Validation results averaged on the 4 folds of Paris-Lille-3D dataset, with random and active learning. 79
Table 6. Validation results averaged on the 6 folds of our modified S3DIS dataset, with random and active learning. 79
Table 7. Features used for our comparative experiment. 82
Table 8. Semantic segmentation IoU scores on S3DIS Area-5. 83
Table 9. Semantic segmentation IoU scores on Semantic3D reduced-8 challenge. 83
Table 10. Semantic segmentation IoU scores on NPM3D Paris-Lille-3D challenge. 83
Table 11. Comparison of state-of-the-art deep learning methods on point clouds. 109
Table 12. Stable dispositions of the kernel point positions when the center point is fixed. If a disposition has an axis of symmetry, we describe it by the successive groups of points sharing a plane perpendicular to this axis. 113
Table 13. Shape Classification and Segmentation results. For generalizability to real data, we only consider scores obtained without shape normals on ModelNet40 dataset. The metrics are overall accuracy (OA) for ModelNet40, class average IoU (mcIoU) and instance average IoU (mIoU) for ShapeNetPart. 132
Table 14. mIoU scores on 3D scene segmentation datasets. Scannet, Semantic3D (Sem3D) and Paris-Lille-3D (PL3D) scores are taken from their respective online benchmarks (reduced-8 challenge for Semantic3D). S3DIS scores are given for Area-5 (see appendix for k-fold). 134
Table 15. Model running statistics on 4 datasets: ModelNet40, ShapeNetPart, Scannet, Semantic3D. We give the model sizes and the training and inference speed for both rigid and deformable version...... 141
Table 16. Segmentation mIoUs for all classes of ShapeNetPart. 162
Table 17. Semantic segmentation IoU scores on S3DIS Area-5. Additionally, we give the mean class recall, a measure that some previous works call mean class accuracy. 163
Table 18. Semantic segmentation IoU scores on S3DIS k-fold. Additionally, we give the mean class recall, a measure that some previous works call mean class accuracy. 163

Chapter I

I. Introduction

French Abstract, résumé français

Les nuages de points sont des données géométriques communément utilisées dans les domaines de vision par ordinateur, infographie et robotique. Aujourd'hui, de nouvelles technologies ont permettent l'acquisition de scènes 3D volumineuses et précises. Les nouvelles applications ouvertes par ces technologies, comme les véhicules autonomes ou la maintenance d'infrastructure, reposent sur un traitement efficace des nuages de points à grande échelle.

Malgré l'avancée rapide de la recherche et l'intérêt de nombreux industriels pour ces technologies, des verrous scientifiques sont toujours présents dans l'étude des nuages de points en 3D. C'est pourquoi le CAOR (Centre de Robotique des Mines ParisTech), et le CMM (Centre de Morphologie Mathématique), fort de leur expérience de plusieurs années sur ce sujet, souhaitent développer en collaboration des recherches sur les méthodes d'acquisition, de traitement et de classification de données 3D.

Les recherches que j'ai entamées se placent dans le cadre d'une thèse de doctorat à orientation industrielle (thèse CIFRE). L'entreprise, Terra 3D, est une spin-off des travaux de recherche de MINES ParisTech sur les Systèmes Mobiles de Cartographie et les traitements de nuages de points 3D, pour des marchés liés à la gestion de réseaux de mobilité, dans les domaines urbain, routier, ferroviaire, et au développement de la voiture autonome. La start-up valorise des résultats de la recherche des laboratoires CAOR et CMM, issus de plusieurs projets de recherche dont le dernier en date était TerraMobilita. Naturellement, ce sont ces deux laboratoires qui encadrent la thèse.

Comme énoncé dans l'intitulé de la thèse, nous allons nous intéresser aux représentations de nuages de points, c'est-à-dire, les manières dont les algorithmes modélisent ces nuages de points, les transforment en informations sur l'objet ou la scène 3D. Notre but sera d'apprendre de nouvelle représentations qui permettront d'améliorer les résultats de segmentation sémantique en particulier. Cela ne nous empêchera pas de considérer d'autre taches comme la classification d'objets ou la segmentation de parties d'objets.

Nous étudierons les représentations de nuages de points à travers le prisme de leur niveau d'abstraction. Nous distinguerons trois niveaux de représentations :

niveau 1. **Les représentations modélisent des objets directement.**

niveau 2. **Les représentations modélisent des motifs géométriques. Un algorithme apprend à quels motifs correspondent les objets**

niveau 3. **Les représentions sont apprises directement par l'algorithme**

Le niveau 3 correspond aux méthodes d'apprentissages profond. Nous nous intéresserons particulièrement aux méthodes convolutives, qui ont montré des résultats exceptionnels sur une multitude de problèmes allant de la vision par ordinateur, à la médecine, en passant par les traitements linguistiques, ne peuvent pas être utilisées directement avec des nuages de points. La puissance de ces méthodes se fonde sur leur capacité à apprendre des représentations sémantiquement fortes de leurs données. Dans le cas des images, les filtres convolutifs ont permis l'apprentissage de nouvelles représentations, jusqu'alors construites « à la main » dans les méthodes de vision par ordinateur plus anciennes.

En suivant le même raisonnement, nous présentons dans cette thèse une étude des représentations construites utilisées pour le traitement des nuages de points. Certaines questions restant sans réponse dans l'état de l'art, nous proposons plusieurs contributions, qui serviront de base à la conception d'une nouvelle représentation convolutive pour le traitement du nuage de points. Parmi elles, une nouvelle définition de voisinages sphériques multi-échelles, une comparaison avec les k plus proches voisins multi-échelles, une nouvelle stratégie d'apprentissage actif, la segmentation sémantique des nuages de points à grande échelle, et une étude de l'influence de la densité dans les représentations multi-échelles.

Cette compréhension des représentations construites offre des bases solides pour la conception de représentations pouvant être apprises sous la forme d'un nouvel opérateur de convolution pour les nuages de points. Nous introduisons la « Kernel Point Convolution » (KPConv), qui utilise des voisinages sphériques et un noyau défini par des points. Ces points jouent le même rôle que les pixels du noyau des convolutions en image. Nos réseaux convolutionnels surpassent les approches de segmentation sémantique de l'état de l'art dans presque toutes les situations. En plus de ces résultats probants, nous avons conçu KPConv avec une grande flexibilité et une version déformable. Pour conclure notre réflexion, nous proposons plusieurs éclairages sur les représentations que notre méthode est capable d'apprendre.

I.1. Context

Computer vision and machine learning are currently among the most popular (and advertised) scientific fields. The recent breakthroughs related to Big Data and Deep Learning in particular, created an unprecedented enthusiasm for artificial intelligence. We are now able to design algorithms performing better than humans on several complex tasks like image recognition, strategic board games, or even driving cars.

In the meantime, new technologies allowing the acquisition of large and precise 3D point clouds have been developed. Point clouds are a type of 3D data consisting of points in 3D Euclidean space. In most cases, these points represent the surfaces of objects in 3D scenes. They are collected by 3D sensors such as LiDARs or depth cameras. This type of data have been used for a long time in the literature, but at reduced scales. With new large-scale point clouds, and the promising outlook of Deep Learning algorithms for any kind of data, many applications are on the line. For example, to create High Definition Maps or Virtual Environments for autonomous vehicles, we need the ability to scan large 3D scenes and segment semantic objects like roads or signage. For the monitoring of infrastructure, we need to segment the buildings and other relevant classes in large 3D scenes, for example to control the vegetation growing around sensible infrastructures like power lines.

This thesis was conducted in the Center For Robotics at Mines Paristech, France, jointly supervised by the Center of Mathematical Morphology, France. These labs have a wide range of research topics, including, autonomous driving, perception, control, machine learning, virtual and augmented reality, etc. Within, the NMP3D team (Nuages de Points et Modélisation 3D), researchers and PhD students are collaborating in a stimulating environment around point clouds and 3D models.

In addition, this thesis has an industrial focus, as it was funded by Terra3D, a start-up created within these two labs, to disseminate their research works. The start-up has a technical expertise on 3D point cloud processing, and aims markets related to network or infrastructure management in many different sectors.

I.2. Clarification of the subject

Our thesis subject is entitled “Learning new representations for 3D point cloud semantic segmentation”. We already introduced “3D point clouds” in the previous section and we detail their nature and their characteristics in Chapter II. In this paragraph, we elucidate the other key terms of this problem.

- **Learning representations**

Computers can handle massive numbers of mathematical operations. However, pure computational power is not enough. Resolving such complex tasks involves a certain understanding, a model of the reality. Whether they use images or point clouds as input, computer vision algorithms create representations of their data according to different models (see Figure 1). In general, we consider a representation to be a value, or a set of values that describe the data. For instance, you can be represented by your height and weight. Another example in images, an object can be represented by its main color. If it is blue, chances are it is the sky.

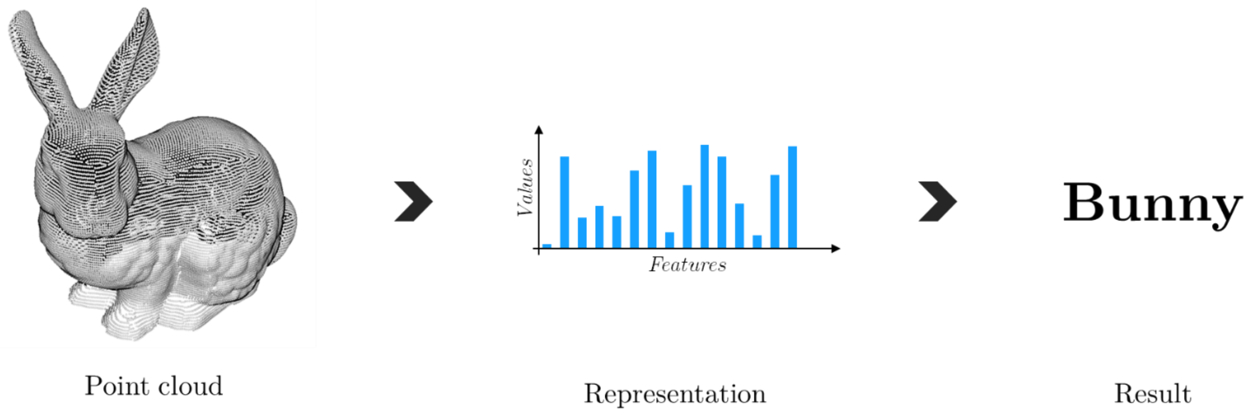


Figure 1. Illustration of the steps of a computer vision algorithm on a simple classification task. The point cloud is first transformed into a representation, which is then used in the classification process.

However, simple representations such as color or size are not elaborate enough to classify real objects, as they are rarely enough to characterize them. Through the years, the representations used in computer vision algorithms evolved and improved, but with a limit, they always were handcrafted. They consisted of what humans believed, was the best model of reality, until the emergence of Deep Learning, which uses learnable representations.

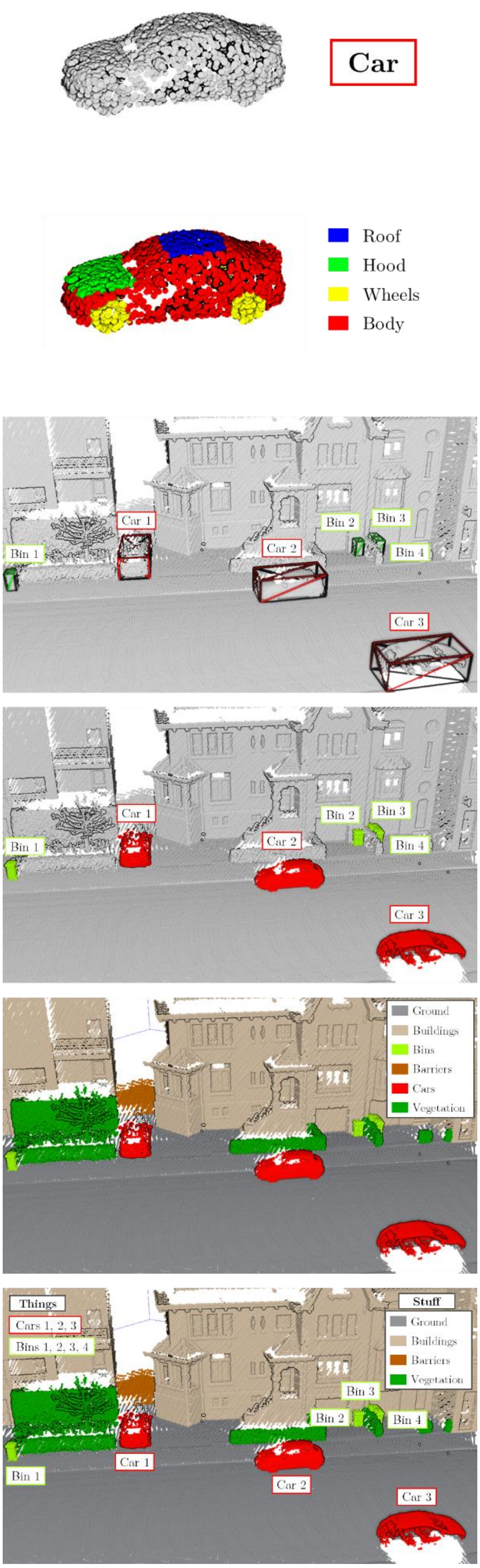
The word *representation* can have different meanings. For instance, we can say that a point cloud is a representation of a 3D scene. We can also say that this scene can be represented by a mesh or a voxel grid. To avoid any confusion, we prefer to talk about *data type* instead of *representation* in this context. In this manuscript, the word *representation* will always designate the concept of values describing the data.

- **Semantic segmentation**

These term refers to a specific computer vision problem. Semantic segmentation was first defined for images, with a simple goal: semantically understanding the role of each pixel in the image. This task naturally extends to point clouds with points replacing pixels. In the hierarchy of semantic understanding tasks, illustrated in Figure 2, it is the second most advanced behind panoptic segmentation.

In the two stages of a computer vision algorithm illustrated in Figure 1, the first one depends more on the data and the second one on the task. As we focus on the elaboration of learnable representations for point clouds, our focus is on the first stage. Tasks like instance segmentation or panoptic segmentation would require elaborate algorithms or network designs in the second stage. This is the first reason behind the choice of semantic segmentation. The second being the availability of large and diverse datasets, which are essential for the learning of point clouds representations with Deep Learning methods.

However, we also evaluate our approaches on other tasks to prove that our representation learning process generalizes to any kind of problem and is more general than a semantic segmentation algorithm.



Object classification consists of giving a label to a point cloud, generally representing a single object. This is the most simple problem, but it often relies on artificial datasets with no spatial context, which is not ideal for concrete applications.

Object part segmentation consists of giving a label to each point of an object to partition it in semantically meaningful parts. It is also far form real applications.

Object detection consists of finding the location (e.g. with a bounding box) and the label of objects in a scene. The objects are not isolated like above and have a spatial context. This task is particularly interesting when the counting and the positioning of object matters. For example, in application like automatic parking studies.

Instance segmentation consists of partitioning objects in the scene. It goes further than object detection, by finding the points belonging to each object. This task is useful for applications requiring measures that are more precise (e.g., urban furniture maintenance).

Semantic segmentation consists of giving a label to each point of a scene. It is the same task as object part segmentation, but on real scenes instead of single objects. It is thus more suited for real applications, for instance, monitoring vegetation growth.

Panoptic segmentation consists of giving a label and an instance ID to each point of a scene. The distinction is made between *stuff* and *things*. *Stuff* is uncountable, and only the *things* have an instance ID. This task offers the most advanced understanding of the scene, needed for applications like autonomous driving.

Figure 2. Illustration of the hierarchy of semantic understanding tasks on point clouds.

I.3. The three tiers of representations

Any computer vision algorithm uses representations to create an underlying model of the reality. Whether images or point clouds are used as input data, these representations can be more or less elaborate, making the underlying model more or less abstract. When looking at computer vision algorithms with this perspective, we can classify each algorithm according to the degree of abstraction of this model, which often corresponds to its degree of automation. The more abstract the model is, the more automation is needed in the algorithm. Three **tiers of representations** can be identified when classifying computer vision algorithms.

niveau 4. Modelling objects

The tier 1 representations directly describe specific objects. They are based on heuristics, and often model objects as simple primitives, e.g., “the floor is a large horizontal flat surface”. These concrete models of the reality are translated into equations and parametrized by hand with thresholds, thus, they do not involve any machine learning. For example, (Boulaassal et al., 2007) detected façades as large vertical planes with a RANSAC algorithm (more details in Section III.1.1)

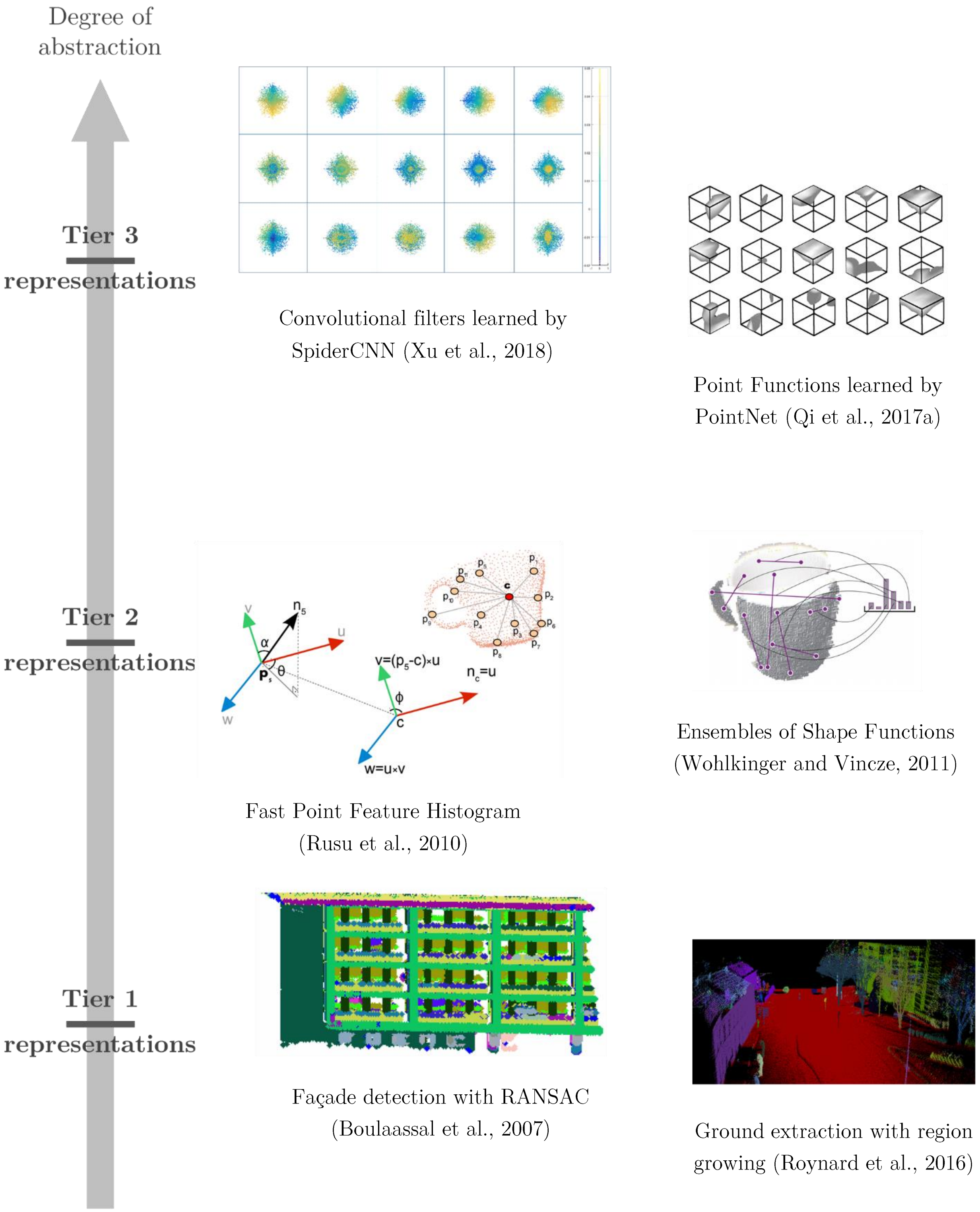
niveau 5. Modelling geometric patterns

We define the tier 2 representations as geometric patterns used to describe any objects in the scene. They consist of handcrafted features such as “This is a planar surface”, “The normal of this surface is vertical”. The underlying model is then built with a machine-learning algorithm, which finds which values of these features correspond to which semantic classes. Because the representations are more abstract than specific object descriptions, a learning process is needed. The features described by (Weinmann et al., 2013) are a perfect illustration of tier 2 representations.

niveau 6. Modelling learnable representations

The tier 3 representations are not handcrafted anymore. Deep neural networks can learn their own representations of the reality, along with how they characterize the semantic classes. However, this is not a complete automation, as we still have to specify the type of representation that the networks can learn. Instead of giving an algorithm tools to describe the data, we give it a toolkit and let it choose its own tools. The crucial part is to define the toolkit so that the network can have access to efficient tools, like PointNet (Qi et al., 2017a) did for instance.

For a better understanding, we give a few examples of algorithms for each tier of representation in Figure 3.



Degree of abstraction
Tier 3 representations
Tier 2 representations
Tier 1 representations
v
n5
α
u
p5
θ
w
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10
p11
c
v=(p5-c)×u
nc=u
ϕ
w=u×v

Convolutional filters learned by SpiderCNN (Xu et al., 2018)

Point Functions learned by PointNet (Qi et al., 2017a)

Fast Point Feature Histogram (Rusu et al., 2010)

Ensembles of Shape Functions (Wohlkinger and Vincze, 2011)

Façade detection with RANSAC (Boulaassal et al., 2007)

Ground extraction with region growing (Roynard et al., 2016)

Figure 3. The three tiers of representations illustrated with point cloud processing algorithms. In tier 1, the objects are detected as simple primitives like planes. In tier 2, the objects are described by handcrafted features. In tier 3, the objects are processed by neural networks which learn their own features.

I.4. Thesis outline and contributions

How can Deep Learning be adapted to point clouds?

As stated in the title of this thesis, we will focus on learnable representations for point clouds, or with our denomination, tier 3 representations. As explained above, deep neural networks offer the best way to learn representations. They became popular with the astonishing results of convolutional neural networks (CNNs) on images. Considering the unrivaled performances of CNNs on images, convolutional filters are promising candidates for learnable representation on point clouds.

However, adapting convolutions, or any deep learning approach, to point clouds is not straightforward. As opposed to the regular and discrete nature of images, point clouds have varying number of points, defined continuously in 3D space. Recently, there have been a lot of hype surrounding this complex problem, and many solutions have been proposed.

As, standard CNNs cannot be applied directly, some methods try to convert point clouds to other regular data types first. It can be done by projecting the points to 3D voxel grids (Maturana and Scherer, 2015; Wu et al., 2015) or to 2D images (Qi et al., 2016; Su et al., 2015). Yet, these methods are limited by the discretization of a continuous data and by their robustness to varying densities. Furthermore, these solutions do not learn representations for point clouds, but use intermediate data types. The same problem occurs with methods that processes graph structure made from point clouds (Bruna et al., 2014; Henaff et al., 2015).

PointNet (Qi et al., 2017a) is one of the first network designed to process point clouds without any intermediate structures. The particularity of this network is that it is not inspired from image CNNs. This pioneering architecture was designed considering unordered property of point clouds. Instead of convolutional filters, it uses *multi-layer perceptrons* (MLPs) to process point clouds directly. Following this work, many other architectures used MLPs have been proposed, adding hierarchical aggregation of features or combining MLPs with grid structures (J. Li et al., 2018; Qi et al., 2017b; Zhou and Tuzel, 2018). In most of these works, the learned representations have little in common with older handcrafted representation of tier 1 and 2. On the one hand, this is very innovative way to process point clouds; on the other hand, previous works also have a value. They should not be ignored just because neural network became so popular. As explained below, a lot can be learned from handcrafted representation.

More recently, many point convolutions have been proposed to adapt the concept of CNNs to point clouds (Hermosilla et al., 2018; Y. Li et al., 2018; S. Wang et al., 2018; Xu et al., 2018). In our opinion, these convolution operators are not entirely satisfying. They either are inspired by image convolutions or by MLP architectures, but do not take previous works on handcrafted features into consideration.

Why do handcrafted representations matter?

As explained above, the impact of deep learning on modern computer vision is overwhelming, but also understandable. Any computer vision or machine-learning algorithm revolves around a crucial trade-off. On the one hand, we need more elaborate representations to deal with complex tasks, with very diverse data. On the other hand, the algorithms, and their underlying model of the reality, need to be as simple as possible to be efficient.

This trade-off has existed since the premises of computer sciences and algorithmic. Indeed, during World War II, Polish and English cryptanalysts, and in particular Alan Turing, decrypted German Enigma messages with a machine called the “Bombe”. The success of the British codebreaking team heavily relied on the development of statistical procedures, helping to use the Bombes much more efficiently. Briefly, the task was extremely complex with billions of possible configurations of the Enigma machine, but, instead of searching blindly the right combination, the algorithm focused on plausible words, thus reducing the statistical complexity of the underlying model.

Today, deep learning algorithms can recognize images with the same level of precision as humans. For this task, instead of words, they deal with images. The complexity of this data, with thousands of pixels, means that more elaborate models are needed. However, according to the trade-off, deep networks should not be too complex to perform well on this task. The fundamental building block of these neural networks, image convolution, is an effective way of reducing the complexity of their underlying model. The key idea behind the success of convolutional neural networks (CNN) is to look at small image patches, instead of blindly mixing the information of the whole image. This idea comes from previous works on handcrafted computer vision algorithms. For many years, contour, corners and other image features have been computed by applying filters on small image patches. These works tell us that, like the plausible words of Enigma, real images have plausible patterns. Compared to completely random grids of pixels, the statistical distribution of real images is infinitely smaller because they represent only real object with shapes, borders, etc. This is why CNNs, which focus on detecting local patterns in images, are so efficient.

The first step, before designing a deep learning architecture for point cloud, is thus to focus on handcrafted algorithms for point cloud processing. We need to understand the underlying models that previous works have revealed, so that we can effectively translate them into efficient neural networks. This will articulate our reflection in this manuscript.

How do we propose to design new learnable representations for point clouds?

Our goal is to build a deep learning architecture that can learn representations for point clouds. The main difference between our approach and the other proposed tier 3 representations for point clouds is the particular attention we pay to previous handcrafted methods, using representations of tier 1 and 2. In this manuscript, we expose our reflection and contributions following a path in three steps:

a. Understand what point clouds are.
b. Study what hand-crafted representations can bring to a deep learning architecture
c. Design deep neural networks based on this knowledge

First, we need to understand point clouds, in practice and in theory. In Chapter II, we detail their nature, their properties and show how they are acquired. We also list all the datasets that are used in our work. This chapter also includes two notions closely related to point clouds: the neighborhoods and the sampling. It has two important purposes: introducing many notions that are used later in the document and offering a first insight of the properties that are at the center of our approaches. Like PointNet, we also need to respect and use the fundamental properties inherent to this data.

Chapter III focuses on handcrafted representations. First, we will study the representations used in various point cloud processing approaches in the literature. As some questions remain unanswered in the state of the art, we propose our own point cloud semantic segmentation pipeline with handcrafted features. The conclusions drawn from our results form the basis of our neural network architectures.

These architectures, based on a novel convolutional operator named **KPConv,** are presented in Chapter IV. Our convolutional networks outperform state-of-the-art semantic segmentation approaches in almost any situation, which justifies the steps of our approach. In addition to these strong results, KPConv has remarkable properties, unseen in other point convolutions, like a great flexibility and a deformable version. As representations are at the core of our thesis, we eventually show several insights of the representations learned by KPConv networks.

Chapter II

II. About 3D point clouds

Abstract

In this chapter, we introduce the notions related to our subject. We define point clouds and enumerate their properties. We describe how they are acquired and we present the datasets that will be used in our experiments. This chapter also details two important concepts: neighborhoods and sampling. They are essential to understand any point cloud processing approach, and will be used extensively in all our methods.

French Abstract, résumé français

Dans ce chapitre, nous introduisons les notions liées à notre sujet, et tout d'abord celle de nuage de points. Nous insistons sur leur définition, qui comprend les points mais aussi les attributs liés à ces points :

$$\mathcal{P}_{\mathcal{F}} = \{\, (x_i, f_i) \mid x_i \in \mathbb{R}^3,\ f_i \in \mathbb{R}^D \,\}_{i<N}$$

On définit donc un nuage de point comme un ensemble de paires, comprenant chacune un point et un attribut.

Les nuages de points ont aussi plusieurs propriétés qu'il est important de comprendre et de respecter quand on crée des algorithmes pour les traiter. Les nuages de points sont par nature éparse, non ordonnés, et vivent dans un espace continu. De plus dans nos applications nous traitons des nuages de point mesurés par des capteurs, et qui sont donc surfaciques. Enfin, une propriété très importante pour la suite est la localisation de l'information. Les attributs, qui constituent l'information portée par le nuage de points, sont localisés dans l'espace. Cela permet de définir des notions comme les voisinages par exemple.

Nous revenons ensuite sur les technologies qui permettent d'acquérir des nuages de points et sur les jeux de données que nous utiliserons dans la suite. De la photogrammétrie aux scanners laser en passant par les caméras de profondeurs, il existe beaucoup de méthodes pour mesurer les nuages de points. Chacune ayant ses atouts et ses défauts. On retiendra que pour des applications en extérieurs, les scanners laser sont majoritairement utilisés et pour des applications en intérieur, ce sont plutôt les caméras de profondeur. Les diffèrent datasets utilisé dans nos expériences sont récapitulés dans un tableau avec leur caractéristique. Des scènes extérieures avec Paris-Lilles-3D, Semantic3D, Paris-Rue-Madame et IQmulus & TerraMobilita, ainsi que des scènes intérieures avec S3DIS et Scannet. Ces jeux de données présentent tous des caractéristiques variées, ce qui va nous permettre de démontrer que nos méthodes se généralisent à tout type de situations.

Enfin nous définissons deux notions importantes pour les nuages de points : les voisinages et le sous-échantillonnage. Il existe plusieurs types de voisinages pouvant être utilisés sur les nuages de points, mais deux nous intéressent particulièrement. Les voisinages à rayon, qui contiennent tous les points dont la distance au centre est inférieure à un certain rayon et les k plus proches voisins, qui contiennent les k points les plus proches du centre. Les premiers ont une taille fixe, mais un nombre de points variable, et les seconds une taille variable mais un nombre de points fixe. Ces voisinages peuvent être long à calculer, mais il existe des méthodes utilisant des grilles, octrees ou KD-Tree, pour accélérer leur calcul.

La dernière notion définie est le sous échantillonnage. Cette notion est extrêmement importante car elle permet de contrôler la densité de points. À partir d'un nuage de points, les algorithmes de sous-échantillonnage en créent un nouveau moins dense. En général, ces algorithmes servent à égaliser la densité d'une scène (les nuages de points réels n'étant pas parfaits), mais ils peuvent également être utilisés pour adapter la densité de points à différentes échelles d'objets. Dans nos méthodes, nous traitons des objets à différentes échelles, nous utilisons donc plusieurs échelles d'échantillonnage.

Parmi les méthodes de sous-échantillonnage, nous distinguons les algorithmes déterministes, et les algorithmes stochastiques. Ces derniers choisissent les points à garder aléatoirement, mais peuvent être guidés par des probabilité inverse à la densité afin d’uniformiser celle-ci. Deux algorithmes déterministes sont majoritairement utilisés, la sélection de points éloignés et la sélection par grille. Le premier algorithme sélectionne les points un à un en choisissant toujours le point le plus éloigné des points actuels. Le deuxième place une grille sur le nuage de point et garde un seul point par case.

II.1. Point Cloud in theory

II.1.1. Point cloud definition

Point clouds are used in many applications and can have different natures and properties. In this first section, we define the point clouds we use in our applications. We also introduce the notations that will be used in this manuscript, and compare this data type with images.

A point cloud is a very simple data type, as it consists of a set of points in space. They are most commonly used in 3D space, but can also be defined for lower or higher dimensional spaces. In our case, we use point clouds in the 3D Euclidean space $\mathbb{R}^3$. Therefore, a point cloud $\mathcal{P}$ is defined as:

$$\mathcal{P} = \{x_i \in \mathbb{R}^3\}_{i<N}$$

Where N is the number of points in the cloud.

Although $\mathcal{P}$ is an unordered set, we often use a matrix notation to perform operations on point clouds. In the following, we use an abuse of notation and consider $\mathcal{P} \in \mathbb{R}^{N\times 3}$. We keep in mind that any permutation of this matrix along its first dimension corresponds to the same point cloud.

In some cases, point clouds are more than only points. The points in a cloud can be colored, or they can carry other types of features. These point clouds with features are often confused with basic point clouds, but they are really defined as a set of pairs:

$$\mathcal{P}_{\mathcal{F}} = \{\, (x_i, f_i) \mid x_i \in \mathbb{R}^3,\ f_i \in \mathbb{R}^D \,\}_{i<N}$$

Where D is the dimension of the features carried by the points.

We will use such point clouds and still use a matrix notation for them. Let $\mathcal{F} \in \mathbb{R}^{N\times D}$ be the matrix containing the features associated to each point of $\mathcal{P} \in \mathbb{R}^{N\times 3}$. In that case, the order of $\mathcal{P}$ and $\mathcal{F}$ need to be the same, as it represents the point-feature association.

A point cloud with features is much more descriptive than a basic point cloud. As a comparison, Figure 4 shows the equivalent of point clouds in image space. Without features, a point cloud only corresponds to a binary grid. It can be used to describe shapes, but independently, the points do not carry any information (except for their location in space).

The carried features are very important for Deep learning algorithms. In images, convolutional neural network create features maps, which are images with more than three color features. To be able to reproduce such a behavior with point clouds, we have to use features associated to each points. In that case, the point coordinates play a structural role, like the grid of pixels of an image.

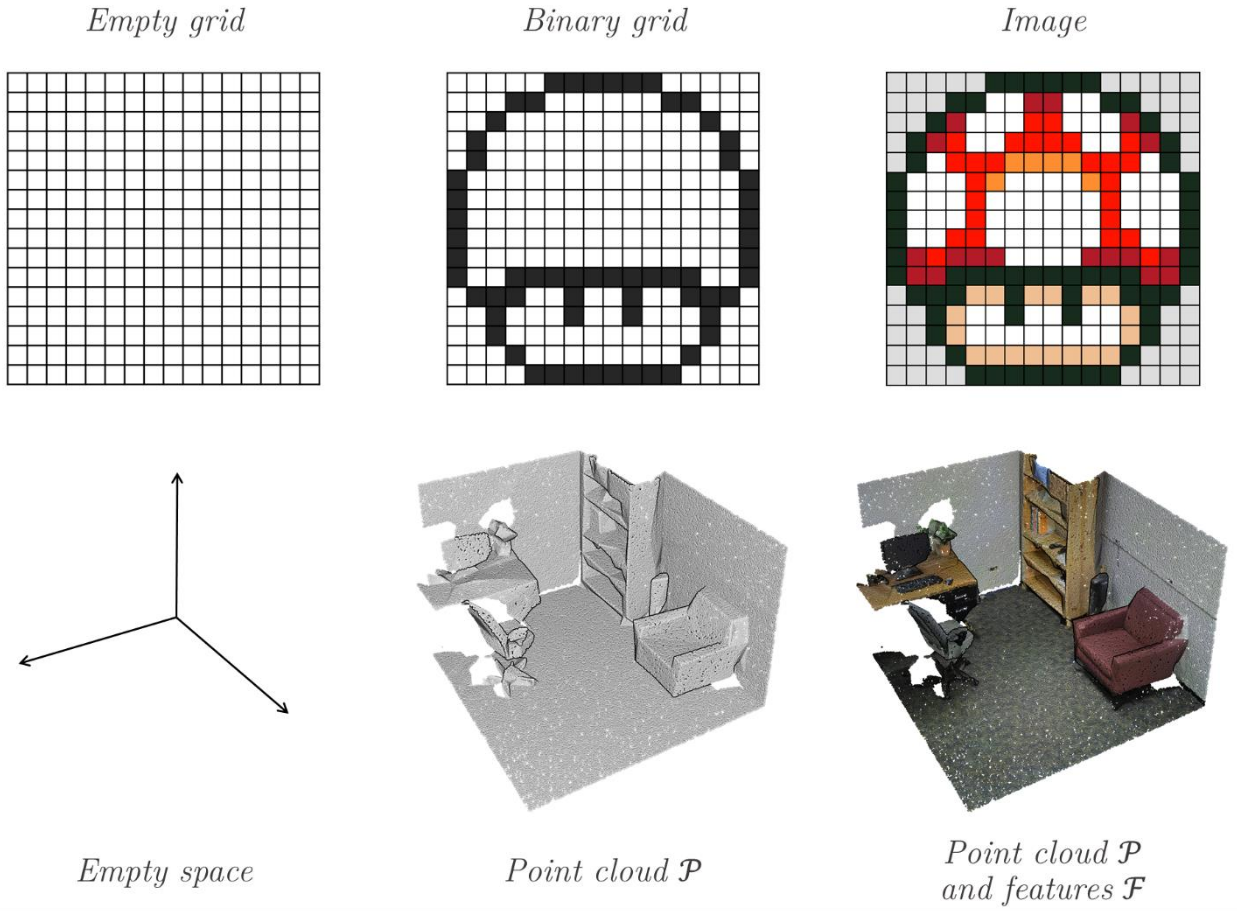

Empty grid
Binary grid
Image
Empty space
Point cloud $\mathcal{P}$
Point cloud $\mathcal{P}$ and features $\mathcal{F}$

Figure 4. Link between image and point clouds. A basic point cloud correspond to a binary grid in image space. To reach the descriptive level of an image, a point cloud must be associated with features.

II.1.2. Point cloud properties

Although a parallel can be made with images, point clouds are very different from them and from other data types. They have their own specific properties. It is crucial to understand and respect these properties when designing point cloud processing algorithms. In this section, we identify two groups of properties: the general point cloud properties, resulting from their definition, and the properties inherent to our own applications.

a. General point cloud properties

These properties are common to every point clouds. They are inherent to their nature.

Sparse. Point clouds are sparse structures. Their data cover an insignificant part of the space they live in, as points are infinitely small. They are typically rarely used for applications that use full volumes like 3D imaging in medicine.

Unordered. Point clouds are unordered sets. They are invariant to permutations and every operation used on them should respect this invariance. The use of non-symmetric functions on point clouds would result in a bad behavior: two identical point clouds would make different results.

Continuous. Point clouds are continuous data. The point coordinates are not restricted to a countable set of locations. This property makes them very different from grids, and explains why the adaptation of image CNNs to point clouds is not straightforward.

b. Properties of real 3D point clouds

These properties are not shared by all point clouds, but inherent to our applications.

Surfaces. In our case, we focus on point clouds produced by 3D scanners (See II.2.1). These scanners measure a large number of points on the external surfaces of objects around them. As a result, our point clouds describe surfaces. The volumes are not captured, as they would be in a MRI (Magnetic resonance imaging) for example.

Spatially localized. This might be the most important property for the design of point cloud processing algorithms. Although point cloud are not organized like grids, or octrees, they are still located in a space. In particular, when we use point clouds with features, the latter are located by the points, and spatial relationships can be found. This property allows to compute neighborhoods, and to combine features according to their spatial relationships.

II.2. Point cloud in practice

II.2.1. Scanning technologies

Several different technologies exist to acquire point clouds of 3D scenes. We briefly describe the most commonly used methods in this section.

a. Photogrammetry and stereoscopy

Photogrammetry is a method based solely on images. It extracts three-dimensional measurements from two-dimensional data. As shown in Figure 5, this technique uses multiple images of the same scene to reconstruct its geometry. When only two images are used for the reconstruction, we talk about stereovision. One of the advantages of this method is that it inherently have color information of the 3D scene, in addition to the geometry. Its main weakness is the computing time for the generation of large-scale datasets with a good precision. Real time stereo-cameras exist but they lack precision.

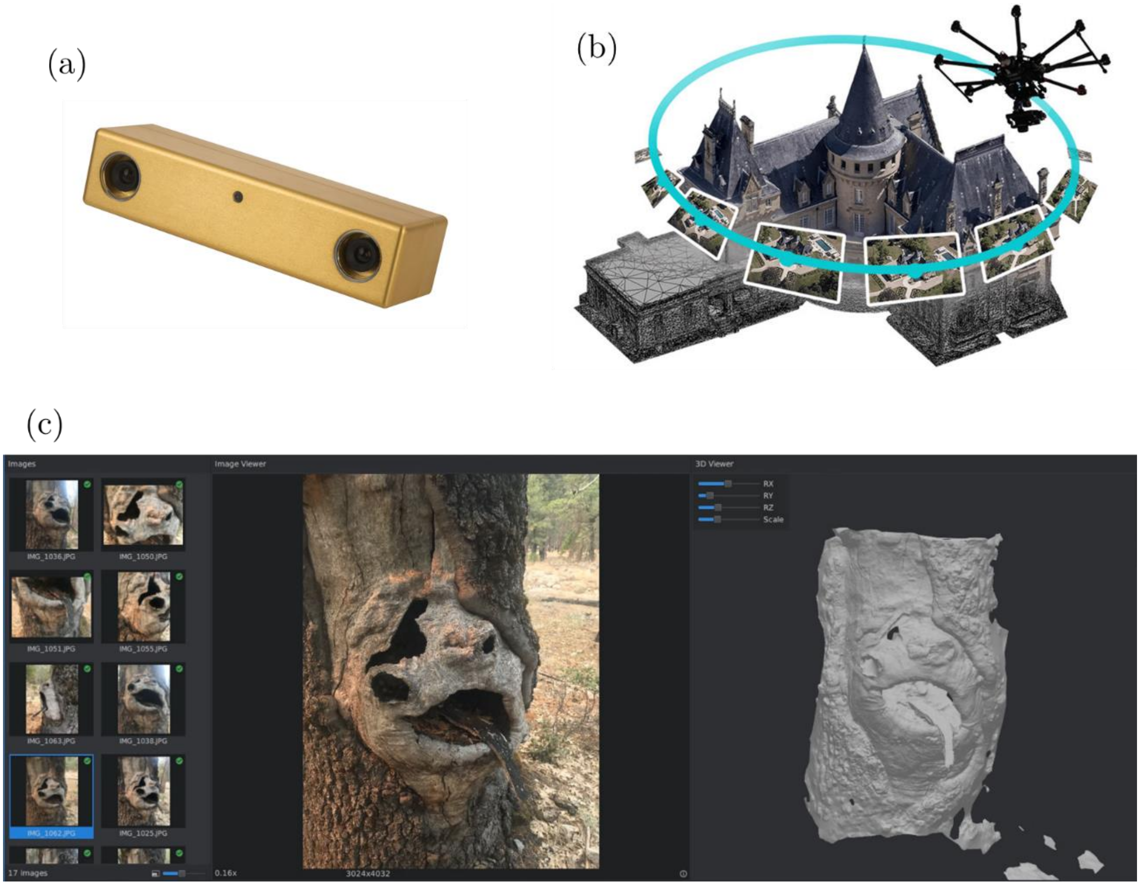

(a)
(b)
(c)
Images
Image Viewer
3D Viewer
IMG_1036.JPG
IMG_1050.JPG
IMG_1051.JPG
IMG_1055.JPG
IMG_1063.JPG
IMG_1038.JPG
IMG_1062.JPG
IMG_1025.JPG
17 images
0.16x
3024x4032
RX
RY
RZ
Scale

Figure 5. Photogrammetry and stereoscopy. (a) Example of stereoscopic camera. (b) Principle of photogrammetry: multiple images are used to reconstruct a building in 3D. (c) Real reconstruction example from Meshroom open source software.

b. Depths cameras

Point clouds can also be obtained from depth images. A 3D scene can be reconstructed from multiple depth images without the need of a sophisticated algorithm as in photogrammetry. Depth images are obtained with RGB-D cameras, which measure the depth of each pixel, in addition to their color (see Figure 6). There are many types of RGB-D cameras: structured-light cameras, stereoscopic cameras, time-of-flight cameras, and light-field cameras. This type of 3D scanner has the advantage to give the 3D information directly as an image, which is useful for many image applications. In addition, the structured-light and stereoscopic cameras are very cheap compared to other 3D scanners. However, the current technologies struggle in outdoor scenarios and lack precision. They are thus particularly useful to create point clouds of indoor scenes, where the distances are much smaller, and the sunlight does not affect them.

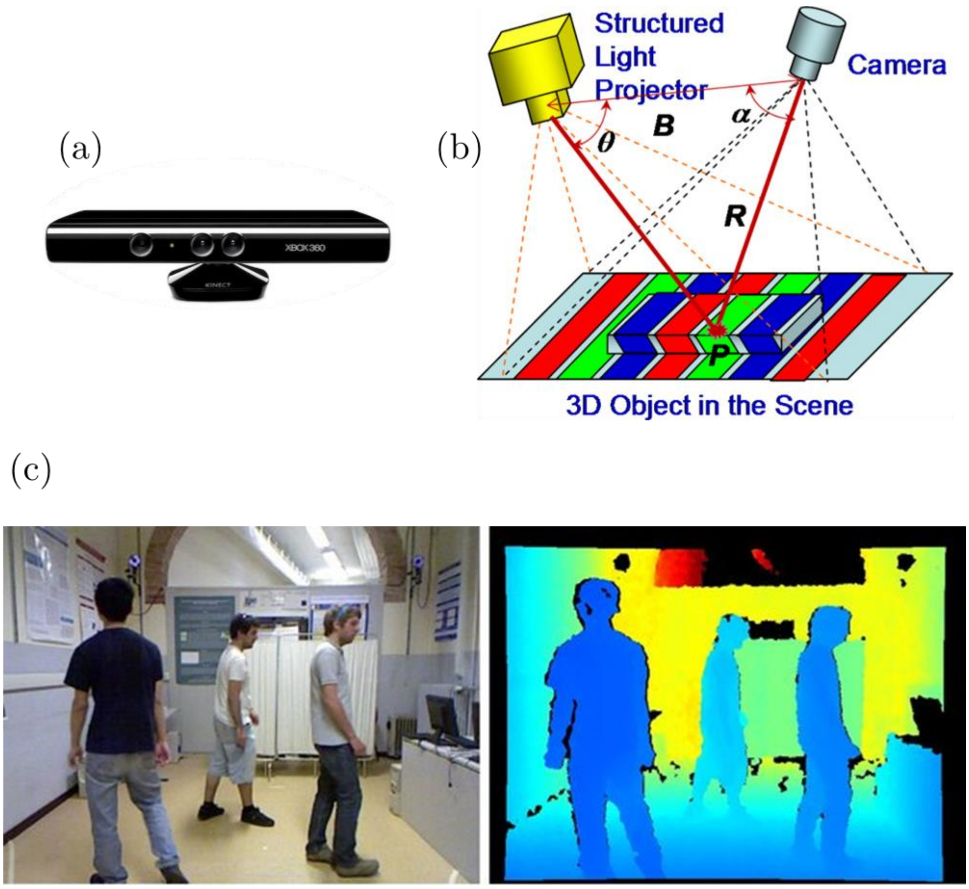


Figure 6. Depth cameras. (a) Example of a Kinect camera that uses structured light. (b) Principle of structured-light cameras: geometric light patterns are projected on the scene and the deformation of the pattern reveals the geometry. (c) Image of an indoor room obtained with a structured light camera.

c. Laser scanners

Laser scanners are the most accurate 3D sensors, thanks to the use of light detection and ranging (lidar) technology. Like radars, lidars use a "time of flight" principle to measure distances travelled by laser rays. Compared to time-of-flight cameras, lidars use lasers, which are thin rays of coherent light; therefore, lidars are much more precise. They offer 3D measurements with a centimeter precision and range of more than a hundred meters. Lidar scanners use mirrors to deviate the rays and obtain a 360° scan of the surrounding scene. They thus have a rotation axis that can be oriented differently depending on the application. Different types of lidars exist, fixed or mobile, with a single rotating laser, or multiple lasers. Mobile lidars can be combined with a high-precision location system (Inertial measurement unit and GPS) to create mobile mapping systems, or used as perception tools for autonomous vehicles. As shown in Figure 7, a vertical axis is more adapted for perception in autonomous driving applications, and a bent axis offers a better scan of the scene for mobile mapping systems.

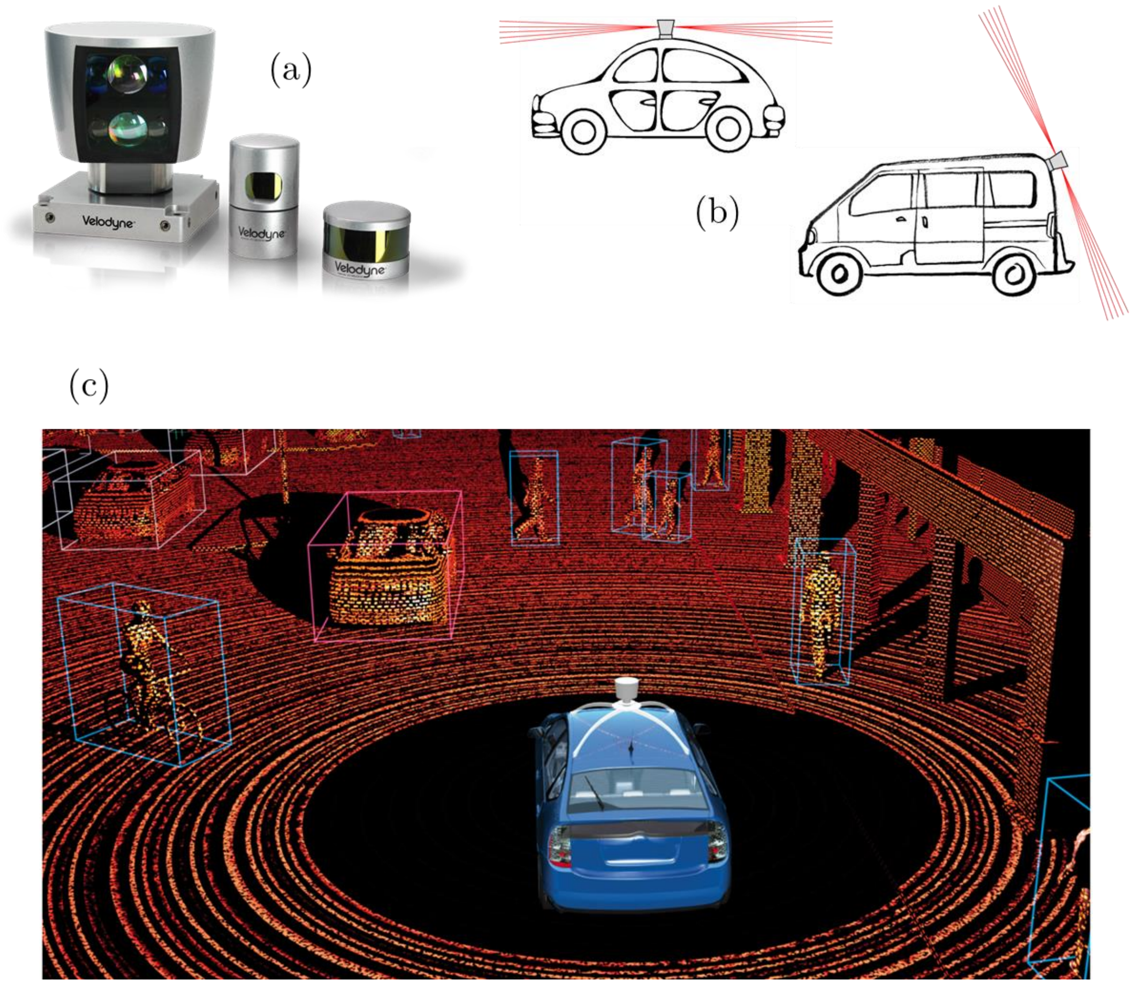


Figure 7. Laser scanners. (a) Example of Velodyne lidars. (b) Laser scanners used for perception with a vertical rotation axis or for mapping with a bent rotation axis. (c) Illustration of a lidar frame, capturing a 3D scene.

II.2.2. Point cloud datasets

In the section, we present the datasets that will be used in our experiments. As explained in the introduction, we focus on the semantic segmentation task, particularly on large-scale point clouds of real scenes. We choose several datasets to show that our algorithms can adapt to indoor or outdoor scenarios, large spaces or cluttered scenes, fixed or mobile scans. In addition, we use other types of datasets, mainly to compare our approaches with the state of the art, but also to show that they can address different tasks on point clouds. We thus present experiments on two other tasks: object classification and object part segmentation, and on smaller point clouds. Each dataset is presented individually and Table 1 summarizes some characteristics of the semantic segmentation datasets.

As points are infinitely small, the covered area was computed by projecting the point of each dataset on a horizontal 2D grid with 10cm cells. We define the covered area as the summed area of the cells that contain at least one point. This measure is a better indicator of a dataset size than the number of points, which depends on the density. A floor area is given in the original Scannet paper, but only for the training set. For a fair comparison, we use the covered area of our method for this dataset too.

Table 1. Characteristics of the semantic segmentation datasets used in our experiments.

				Points		**Covered Area**	
Name	Environment	Acquisition	Colors	Training	Test	Training	Test
Paris-Lille-3D	Outdoor	Mobile lidar	No	140 M	30 M	53,357 m^2	9,671 m^2
Semantic3D	Outdoor	Fixed lidar	Yes	1,660 M	2,349 M	58,335 m^2	52,869 m^2
S3DIS	Indoor	Depth Cameras	Yes	273 M	-	6,066 m^2	-
Scannet	Indoor	Depth Cameras	Yes	5,521 M	449 M	27,366 m^2	1,988 m^2
Rue Madame	Outdoor	Mobile lidar	No	20 M	-	2,964 m^2	-
IQmulus & TerraMobilita	Outdoor	Mobile lidar	No	12 M	-	3,121 m^2	-

Paris-Lille-3D

Paris-Lille-3D (Roynard et al., 2018a) is a recent large scale outdoor dataset for semantic segmentation. It was acquired with a mobile mapping system in four different French cities. The training set, presented in the article, was acquired in Lille and Paris. A test set was added later in two other cities: Dijon and Ajaccio. This dataset thus offer the possibility to evaluate how semantic segmentation methods can adapt to cities with different architecture styles.

Overall, the scans contain more than 170 million points on 2km of streets. The points are not colored, but the reflectance value of the lidar is available. The area covered by this dataset represents more than 60,000 m^2, which is much bigger than other mobile mapping datasets like Rue Madame or IQmulus & TerraMobilita. The original dataset was fully annotated by hand, with 50 classes unequally distributed in three scenes Lille1, Lille2, and Paris. The test set was added to create an online benchmark and only use 10 coarser classes listed in Figure 8, with screenshots of this dataset.

Website: http://npm3d.fr/paris-lille-3d

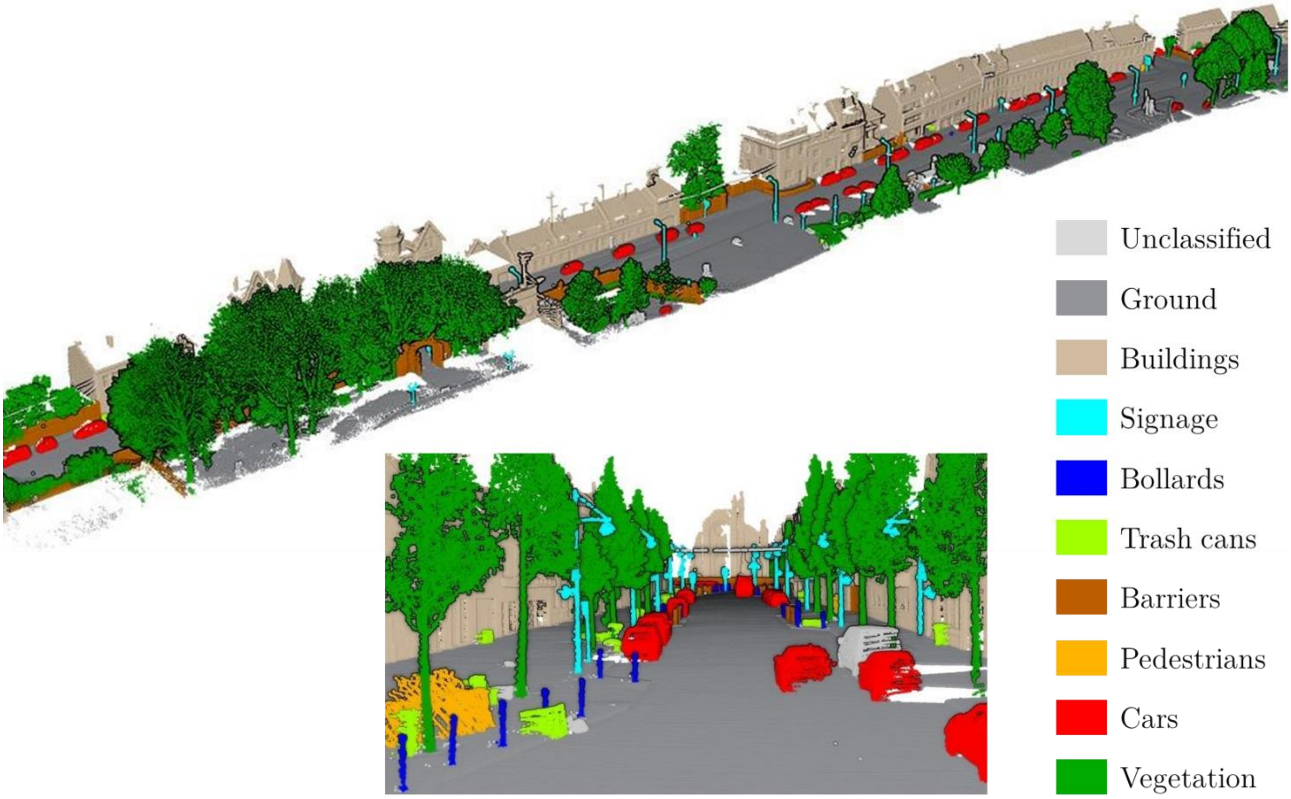


Figure 8. Screenshots of Paris-Lille-3D dataset.

Semantic3D

Semantic3D (Hackel et al., 2017) is a semantic segmentation dataset containing several point clouds from fixed lidar scans. This large-scale outdoor dataset comprises more than 4 billion points, with colors, and annotated with 8 classes. It covers diverse urban and rural scenes. Although it contains many points, it covers an area about the same size as Paris-Lille-3D.

This dataset is an online benchmark, and offers two different test sets. The first one called *semantic-8* contains the original scans of 15 scenes and the second one, *reduced-8*, contains only four scenes subsampled to reduce the number of points. We favor the *reduced-8* challenge in all our experiments because it is less biased by the objects close to the scanner. Some of the training scenes are shown in Figure 9.

Website: http://www.semantic3d.net

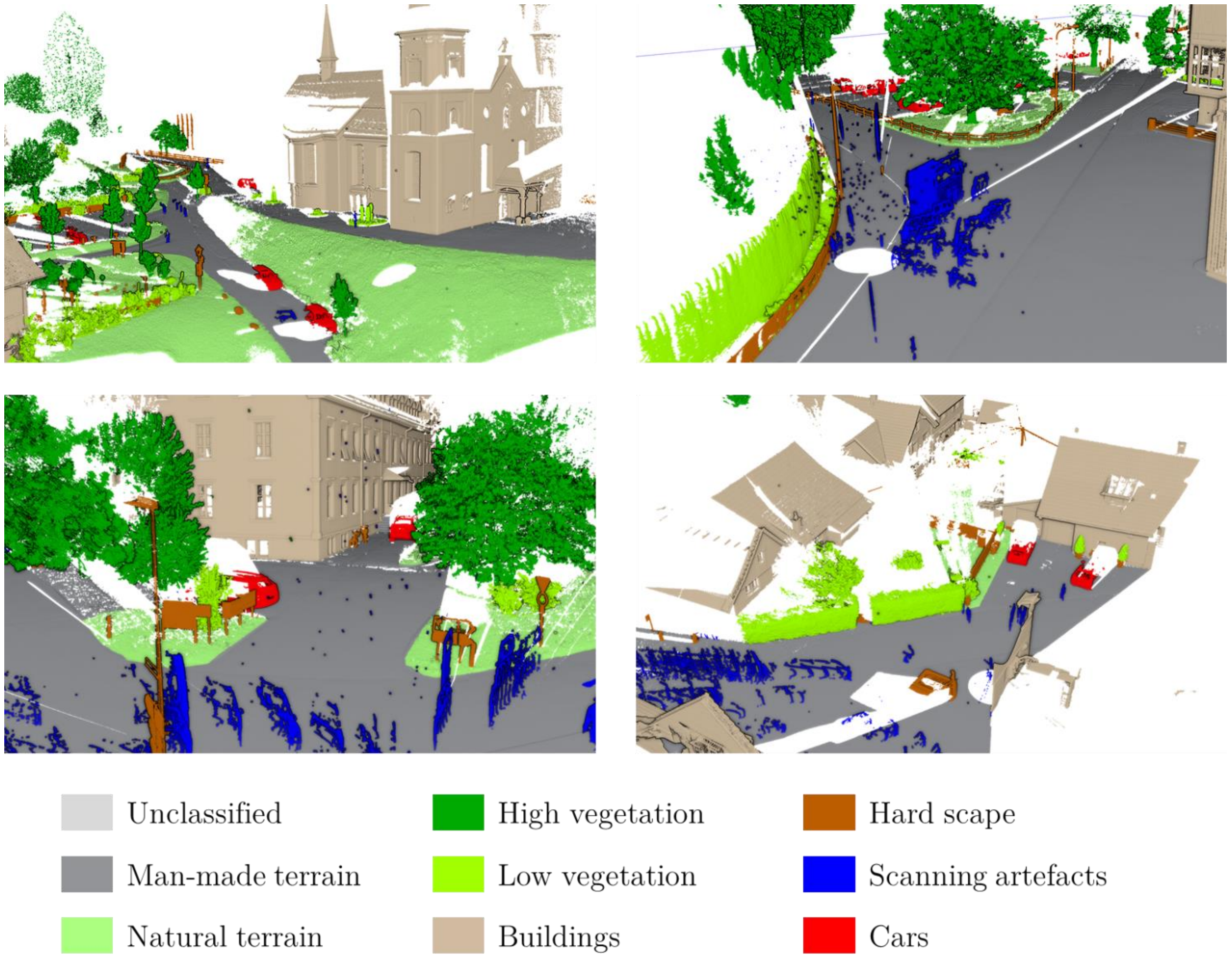

Unclassified
High vegetation
Hard scape
Man-made terrain
Low vegetation
Scanning artefacts
Natural terrain
Buildings
Cars

Figure 9. Screenshots of Semantic3D dataset.

S3DIS

S3DIS (Armeni et al., 2016), also semantic segmentation dataset, covers six large-scale indoor areas from three different buildings for a total of 273 million points annotated with 13 classes. The scenes are from buildings of mainly educational and office use and include colors, as shown in Figure 10. The point clouds were scanned with depth cameras, thus are very different from the previous datasets. They have a much more consistent density across the scenes but also more noise, as the depth cameras are not as precise as laser scanners.

This dataset does not have an online benchmark website, but in a later paper (Tchapmi et al., 2017), the authors advocate the use of Area-5 as test scene. As it was scanned in a different building from the other ones, it offers a better measure of the generalization ability of semantic segmentation methods. We follow this recommendation in our experiments.

Website: http://buildingparser.stanford.edu

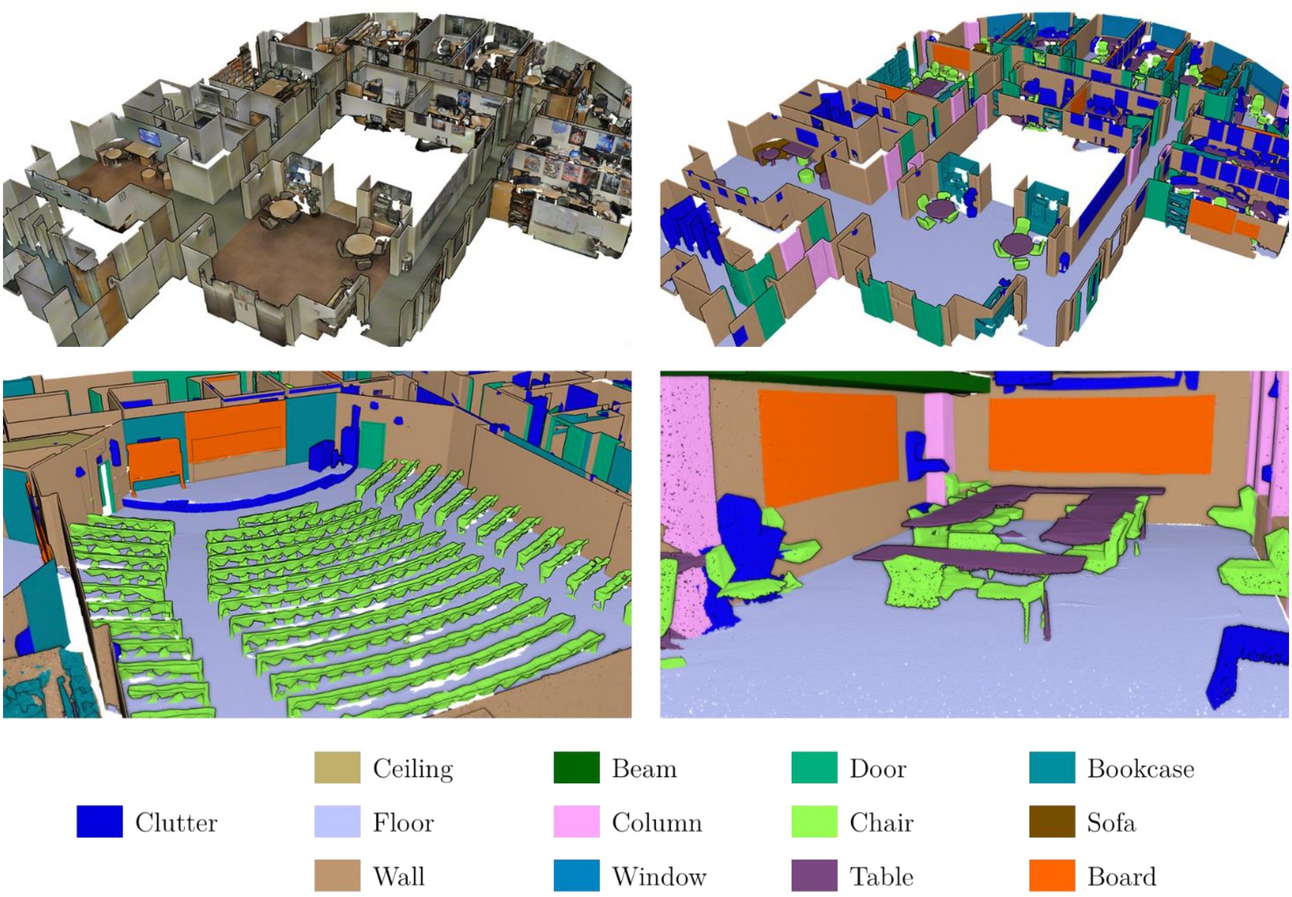


Figure 10. Screenshots of S3DIS dataset.

Scannet

The last large-scale semantic segmentation dataset we use is Scannet (A. Dai et al., 2017). In its original version, it contained 1,513 small training scenes of cluttered indoor spaces. The annotations, comprising 20 semantic classes, were crowdsourced on Amazon Mechanical Turk. The scenes were acquired with depth cameras, and colors are available. Some of them are shown in Figure 11.

The nature of this dataset is very different from the previous ones as the scenes are very small, with only one room. They are not large continuous spaces. However, the dataset has many different scenes, and is the largest of the ones we used in terms of number of points, with more than 5 billion points. Later on, the dataset was updated to a second version, with better annotations, and 100 additional test scenes for online benchmarking.

Website: http://www.scan-net.org/

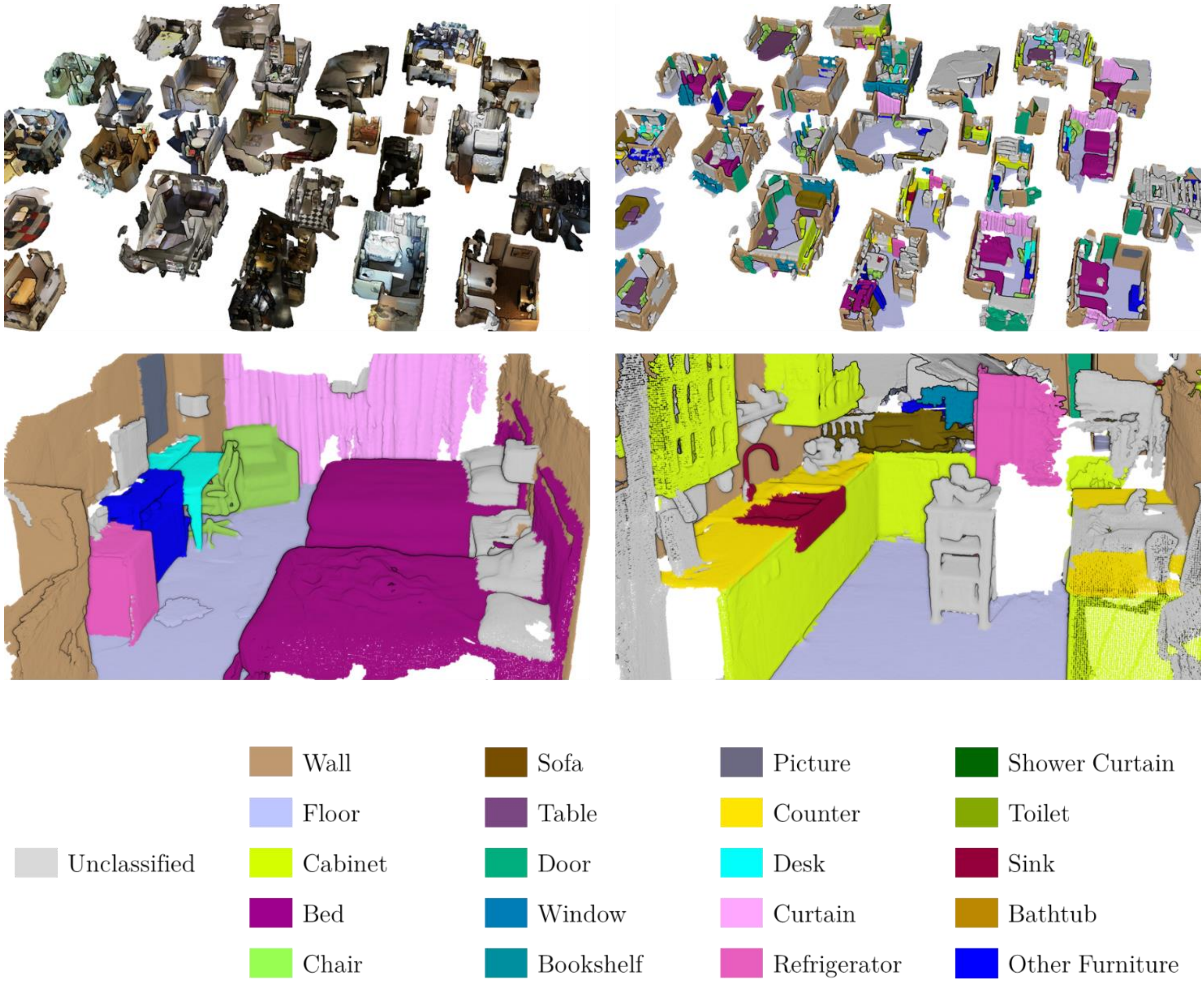

Unclassified
Wall
Floor
Cabinet
Bed
Chair
Sofa
Table
Door
Window
Bookshelf
Picture
Counter
Desk
Curtain
Refrigerator
Shower Curtain
Toilet
Sink
Bathtub
Other Furniture

Figure 11. Screenshots of Scannet dataset.

Rue Madame

Paris-Rue-Madame dataset (Serna et al., 2014), is a 160-meter street scan containing 20 million points (see Figure 12). It does not have colors and offers only one scene, with only a few instances for some of the six classes. Because it is very small, we only use it to compare with other state-of-the-art methods that were not evaluated on other datasets.

Website: http://cmm.ensmp.fr/~serna/rueMadameDataset.html

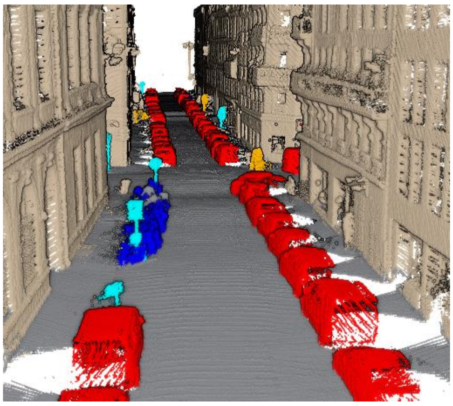


Figure 12. Screenshot of Rue Madame dataset.

IQmulus & TerraMobilita

IQmulus & TerraMobilita (Vallet et al., 2015), is a 200-meter street scan containing 12 million points (see Figure 13). Like Rue Madame, it does not have colors and offers only one scene. For the same reasons, it is only used for comparison purposes.

Website: http://data.ign.fr/benchmarks/UrbanAnalysis/

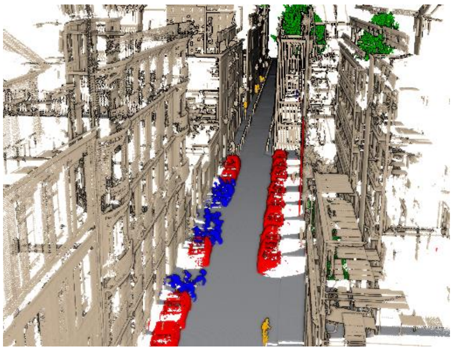


Figure 13. Screenshot of IQmulus & TerraMobilita dataset.

ModelNet40

ModelNet40 (Wu et al., 2015) is a 3D object classification dataset. It contains 12,311 meshed CAD models from 40 categories. The models are 3D meshes and have to be sampled as point clouds to be used in our methods (see Figure 14). For benchmarking purposes, we use data provided by (Qi et al., 2017b), and follow the official split with 9,843 shapes for training and 2,468 for testing. All the models are rescaled to fit into a unit sphere for our experiments. Because the objects are artificial, surface normals are available in addition to the points, but we choose to ignore them, as they are not available in datasets of real scenes.

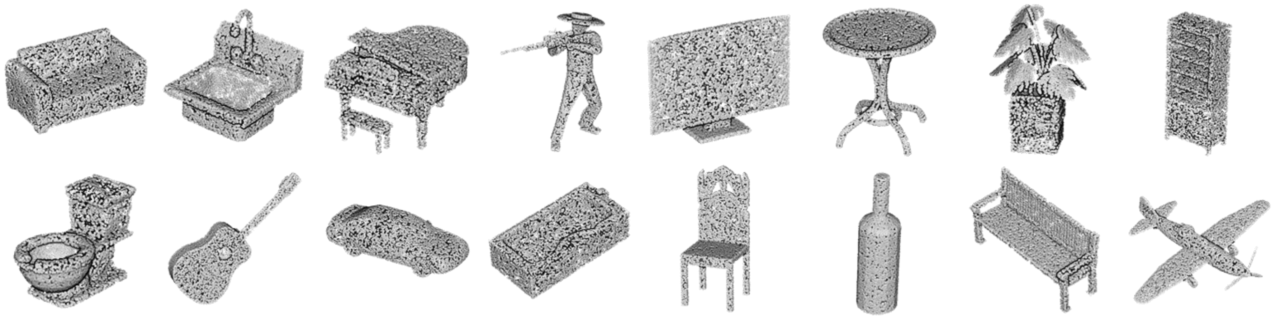


Figure 14. Examples of objects in ModelNet40, subsampled as point clouds.

ShapeNetPart

ShapeNetPart (Yi et al., 2016) is an object part segmentation dataset that was built on top of the ShapeNet dataset (Chang et al., 2015), a large-scale 3D shape repository. It contains 16,881 shapes from 16 of the ShapeNet classes, annotated with 50 parts in total. The object models are already sampled as point clouds and we use the official training and test splits. For the same reason as ModelNet40, we ignore the normals, available in this dataset, and rescale the point clouds to fit in a unit sphere. Example of annotated parts are shown in Figure 15.

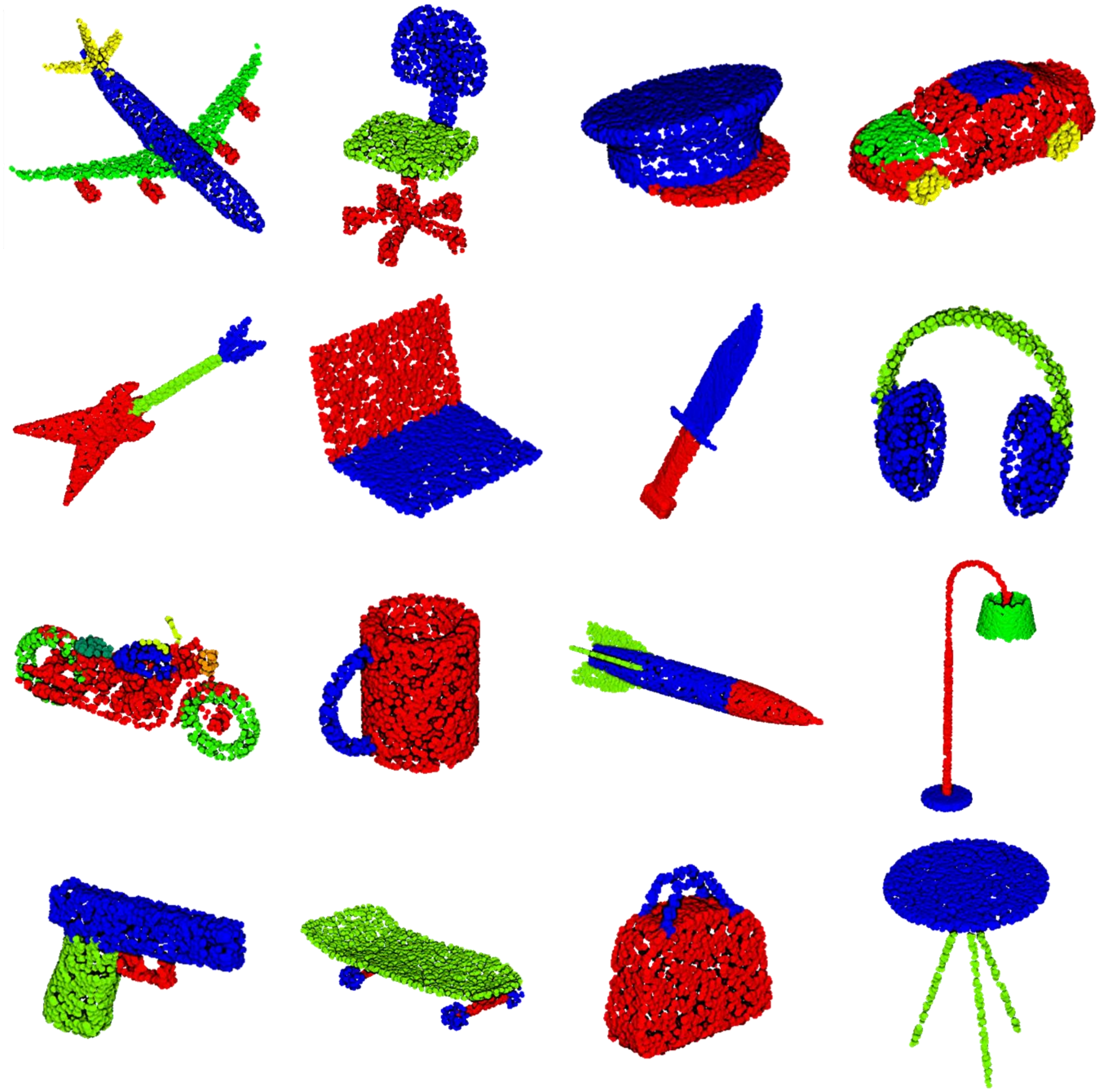


Figure 15. Examples of annotated parts in ShapeNetPart dataset.

II.3. Point neighborhoods

As explained in section II.1.2, point clouds are spatially localized data. More than a list of unrelated points, it is a list of locations, with spatial relationships. The simplest and most used spatial relation between points are neighborhoods. A neighborhood of a point $y_0 \in \mathbb{R}^3$, from a cloud $\mathcal{P} = \{x_i \in \mathbb{R}^3\}_{i<N}$ is a set containing the closest points x_i to y_0, according to a certain distance criterion.

II.3.1. Neighborhood types

Different types of neighborhoods can be used, depending on the chosen distance criterion. On 3D point clouds, the three following types are commonly used.

The **Radius neighbors** or spherical neighborhoods are defined with a radius criterion. The radius neighbors of y_0 are the points x_i whose distances to y_0 are less than a radius $r \in \mathbb{R}$. They are located inside a sphere of radius r centered in y_0.

$$\mathcal{N}_{radius}(y_0, \mathcal{P}, r) = \{\, x_i \in \mathcal{P} \;\mid\; \|y_0 - x_i\| \le r \,\} \qquad \text{(II-1)}$$

The **k-nearest-neighbors** (KNN) are defined with a number criterion. As their name suggests, the k-nearest-neighbors of y_0 are the set of k points that are the closest to y_0.

$$\mathcal{N}_{KNN}(y_0, \mathcal{P}, k) = \{x_i \in \mathcal{P} \mid \|y_0 - x_i\| \le \|y_0 - x_{i+1}\|,\ i < k\,\} \qquad \text{(II-2)}$$

The **cylindrical neighborhoods** are close to the radius neighbors, as they are also defined with a radius. However, they are defined as the points x_i whose distances to y_0 are less than a radius $r \in \mathbb{R}$ in a projective plane H (generally the horizontal plane). Let P_H be the projection on plane H.

$$\mathcal{N}_{cyl}(y_0, \mathcal{P}, r, H) = \{\, x_i \in \mathcal{P} \;\mid\; \|P_H(y_0) - P_H(x_i)\| \le r \,\} \qquad \text{(II-3)}$$

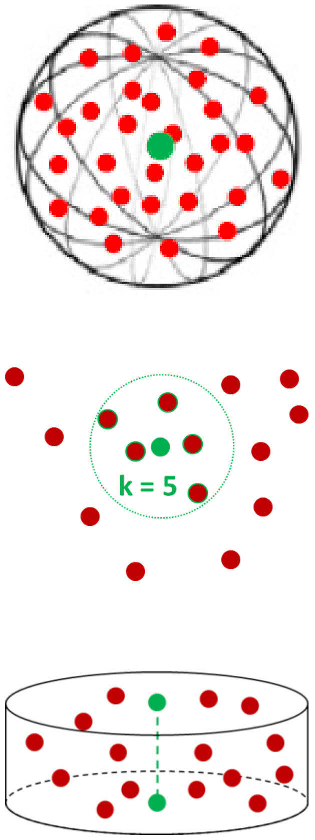


Figure 16. The three most used neighborhood types.

Each type has its own advantages and drawbacks. The spherical neighborhoods cover a fixed geometric volume, defined by the chosen radius, while the KNN can be located anywhere in space. The first are preferable for applications where geometrical consistency matters. In return, KNN have the advantage to contain a fixed number of neighbors, while the number of radius neighbors is variable. Some spherical neighborhoods can be empty, while others can be full of points. Using KNN is mostly advantageous for computing reasons; it ensures a steady computing speed and fixed matrices sizes for GPUs. Eventually cylindrical neighborhoods are often used on aerial point clouds, because the points are already close to a horizontal plane. In some cases, they are used to compute height distributions in vertical columns.

With neighborhoods, we can introduce the notion of adjacency graphs. A graph consists of a set of vertices (the points) and a set of edges (link between points). The edges are defined as pair of vertices and can be oriented or not. From a point cloud, we can build an adjacency graph with any neighborhood definition. An edge will be defined between two points if they are neighbors. In our methods, we use neighbors without defining adjacency graphs, but concurrent methods do. We highlight the fact that adjacency graphs and neighborhood lists are two formulations for an equivalent notion. Other types of graphs exist, not based on neighborhoods and point clouds, but on surfaces and manifolds. The methods using these types of graphs are different from ours in their nature, because they deal with a different type of data.

II.3.2. Accelerating structures

We use neighborhoods extensively in our methods and need to compute them efficiently. The naïve way to compute a neighborhood is to check the distance of every point of the cloud to our target. However, this method called brute force is highly inefficient. To compute many neighborhoods in a single point cloud, we often use intermediates structures that reduce the complexity of the neighbor search. Most of these structures have other uses, like point sampling (see section II.4).

Regular grids are the most basic structures that can be used to accelerate neighborhood computations. They consist of partitioning the space in a three dimensional voxel grid. In general, the voxels are cubes, but their size can also be different in one or two of the dimensions. When designing a grid structure, one can decide to store different levels of information. The simplest grid is the occupation grid and consists of a binary matrix. More complex grids store a point or a list of points in each voxel. To compute neighborhoods, we need the list of every point that lies in each voxel. Therefore, only the points of the neighboring voxels have to be checked and the computations are drastically reduced, especially on large-scale point clouds. The implementation of such grid is more complex than binary matrices. The number of points in each voxel is different, and we do not want to waste memory for empty voxels. For example, spatial hash table (Lefebvre and Hoppe, 2006) or dedicated hierarchical structures (Museth, 2013) can be used. Regular grids can be optimized to be very fast for neighborhood computations, but they lack flexibility. Indeed, the size of the voxels is fixed, and therefore, optimized only for one scale of neighborhoods. When searching KNNs or spherical neighborhoods with varying radiuses, other types of structures are preferable.

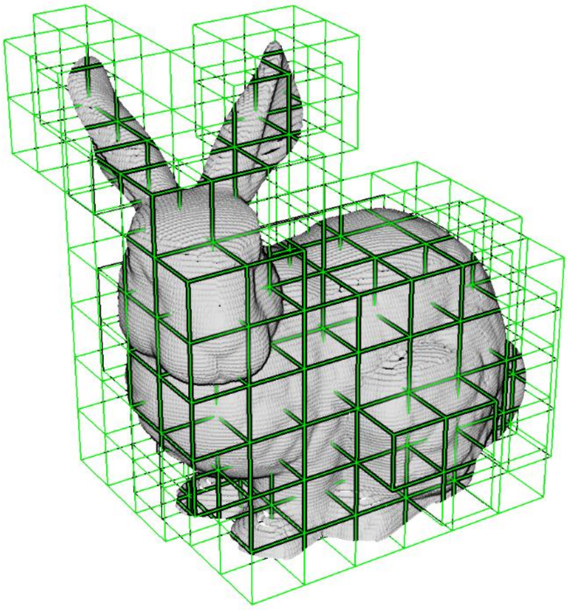


Figure 17. Illustration of a 3D grid around *Stanford Bunny* point cloud.

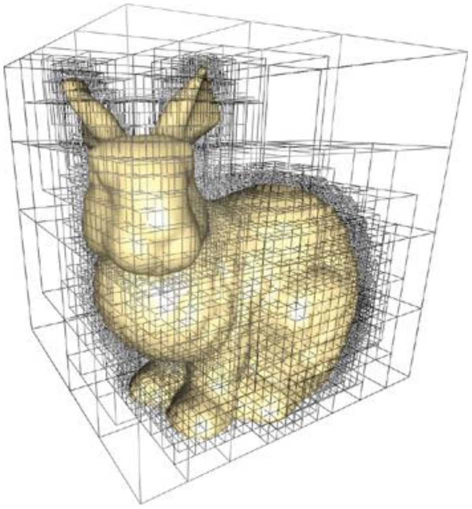


Figure 18. Illustration of an octree around *Stanford Bunny* point cloud.

Octrees are similar to grids, but extend their concept to multiple scales. They keep the idea of partitioning the space with voxels, but they use a hierarchical tree instead of a complete grid. The tree root is a cube containing the whole point cloud. This cube is divided in eight sub-cubes, two times smaller than the original. These sub cubes are the children of the tree root. Each of the children can be divided like this iteratively, expanding the tree. When a cube reaches a certain size or a certain number of points (for example smaller than 10 points), the division is stopped and it becomes a leaf (tree node with no children). Like grid voxels, octree leaves can store any information, but in our case, we need to keep the list of all points located in their cubes. Therefore, neighbors can be searched only in the neighboring leaves in the tree structure. This structure is particularly adapted to the processing of geometric shapes, and has a low memory consumption for sparse data. As a result, it is widely used as a data type for the processing of large-scale 3D scenes (Lalonde et al., 2007; Meagher, 1982; Riegler et al., 2017; Wurm et al., 2010). The weakness of octrees is to extend poorly to greater dimension, their 2D version quad-trees are commonly used, but in four or more dimensions, they are not used a lot.

Kd-trees are hierarchical tree structures like octrees, but with a different principle. They are designed to partition any kind of data in any space efficiently, while octrees were optimized for 3D Euclidean space. Their principle is to define half-spaces with hyperplanes to partition the data points. Kd-trees thus are binary trees, each node having 2 children. Like octrees, they are efficient for any neighborhood scale, and compact in memory. Both structures offer similar performances for neighbor search in theory, but in practice, highly optimized open-source kd-trees implementations are available, making them more easy to use. In our case we will use nanoflann library (Blanco and Rai, 2014), based on the older FLANN method (Muja and Lowe, 2009).

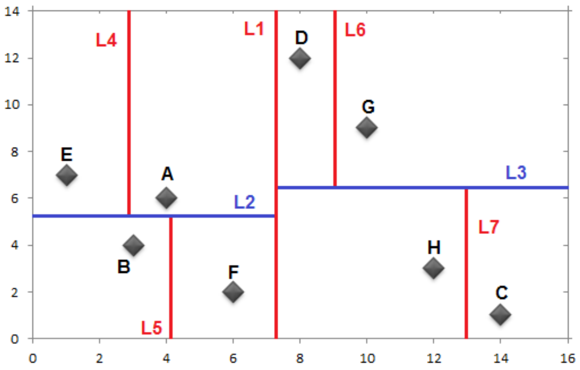


Figure 19. Example of space partitioning with a kd-tree in two dimensions.

II.4. Point Cloud Sampling

In this last section of the chapter, we introduce the notion of point cloud sampling and present some methods commonly used to sample point clouds. We will use subsampling strategies in our methods in the next chapters.

II.4.1. Principle

Point clouds are measures of real data, in our case, 3D scenes or objects. To be even more specific, they are measures of the exterior surfaces of 3D scenes, for real datasets. This is only true for point clouds of real scenes; point clouds of artificial objects can also describe the interior of these objects. Unlike surfaces, points are not continuous, they are infinitely small and the measure they offer is thus sparse. The point sampling, which consists of their disposition and density, is extremely important in point cloud processing, as it can influence how the real shapes are characterized.

The original sampling of a point cloud depends on the method used for its acquisition. We already talked about the different scanning technologies and their precision, but the method also influences the quality of the sampling. Fixed laser scans often suffer from bad samplings, with a huge number of points close to the scanner, and very few points far from it. Mobile mapping systems compensate the density irregularity by their movement. The vehicle can pass again in an area that has a low density. However, there are limited by roads in case of system mounted on cars for example. In general, point clouds of real scenes are not perfect and present varying densities.

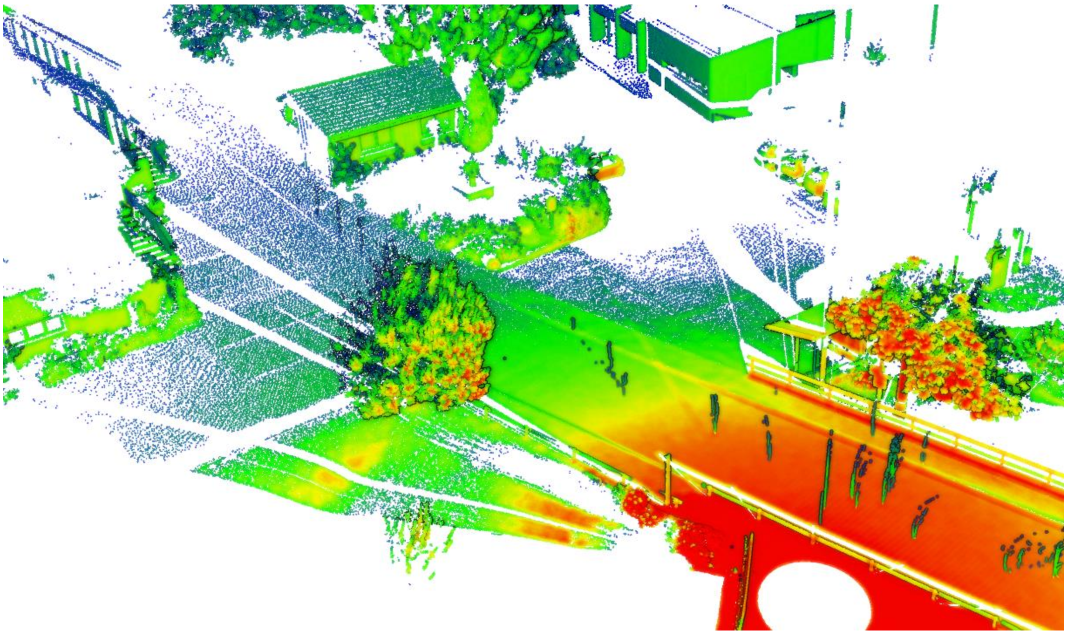


Figure 20. Illustration of the density variation in a fixed laser scan from Semantic3D dataset. Red denotes higher densities (close to the scanner) and blue lower densities (far from the scanner).

Varying densities can affect the results and the efficiency of point cloud processing methods. If the density is too low, it can be harder to distinguish shapes, and if the density is too high, the computational time can be too long. This trade-off between performances and efficiency depends on the scale of the objects and the shape details that the methods need. On very large objects with few details like roads for example, the density does not need to be very high. On smaller objects with more complex geometry, we need more points to differentiate the details. The density of points can be adapted to different scales with subsampling algorithms. From a point cloud, these algorithms create a new one with a lower density. In general, these algorithms are used to equalize the density across a scene (as real point clouds are not perfect), but they can also be used to adapt the point sampling to different scales of objects. In our methods, we deal with objects of various scales, so we use multiple sampling scales.

II.4.2. Deterministic versus stochastic subsampling methods

There are many subsampling methods. Here, we describe the most commonly used for point clouds. Some of them are deterministic and other are stochastic. In our case, we need to keep control on the density of points in our point clouds. We will thus use deterministic methods.

Random Decimation is a stochastic method as its name suggests. This is the simplest and fastest subsampling method. It consists of picking points randomly in the point cloud. It is very efficient as it only involves a random permutation, but it does not offer any control over the result. It is completely biased by the density, because high-density areas are more likely to have their points picked than low-density areas. Therefore, they will remain with higher densities after the subsampling. This method does not help to equalize density.

Inverse density importance sub-sampling is a derivate from random decimation. This stochastic method consists of picking points randomly according to a non-uniform importance. Each point is given an importance equal to the inverse of the density at its location. Therefore, the points from high-density areas are less likely to be picked. This strategy helps equalizing densities, but we have no insurance that the low-density region will be kept intact, as the picking remains random.

Farthest point sampling is a deterministic method that samples points from a point cloud only using a distance criterion. The process is iterative and begins with the choice of a random point in the cloud. At each step, the farthest point from the currently sampled points is picked. The process goes on until a certain distance threshold is reached. This strategy helps picking points regularly across the whole point cloud and ensures a minimal distance between them. Its major drawback is its computational cost. It has a complexity of $O(n^2)$.

Grid subsampling is another deterministic method, which uses grid as a tool for subsampling. It projects the cloud into a grid and keep only one point per voxel. This method has many advantages. It equalizes the density of the point clouds, with an easy parameter to control the scale (the voxel size) and it is very fast. Furthermore, we can keep any point in each cell, for example, we can keep the barycenter of the points inside it. Keeping the barycenter has the advantage to remain more faithful to the original shape, than keeping one of the points. This is why we choose to use this strategy in our methods

Chapter III

III. Hand-crafted approaches for point cloud processing

Abstract

Now that all necessary notions and definitions are established, this chapter introduces our first works and contributions revolving around handcrafted approaches for point cloud processing. As part of our line of thought, we need to understand how these handcrafted methods create representations for point clouds, before designing more elaborate learnable representations. We first review a wide range of handcrafted point cloud processing methods, and handcrafted features used for point cloud semantic segmentation. From this review, we draw conclusions about the design of our deep learning framework for point clouds but some unresolved issues remain. We thus propose a new pointwise classification pipeline and new experiments to offer answers to these issues. This new pipeline revolves around two main contributions: a definition of multiscale spherical neighborhoods and an active learning strategy.

French Abstract, résumé français

Maintenant que toutes les notions et définitions nécessaires sont établies, ce chapitre présente nos premiers travaux et contributions axés sur des approches « à la main » du traitement de nuages de points. Nous cherchons ici à comprendre comment ces méthodes créent des représentations pour les nuages de points, avant de nous lancer dans la conception de représentations plus élaborées et pouvant être apprises.

Nous passons d'abord en revue un large éventail de méthodes de traitement de nuages de points. Des méthodes sans apprentissage qui utilisent des représentations de niveau 1, puis des méthodes avec apprentissage qui utilisent des représentations de niveaux 2. L'analyse de toutes ces méthodes à travers les prismes des représentations qu'elles utilisent nous permet de tirer plusieurs conclusions sur les meilleures représentations, et la meilleure manière de les apprendre.

Ces conclusions sont conformes aux conclusions tirées en traitement d'image qui ont abouties à la conception de réseaux de neurones à convolution. Nous retrouvons donc les idées que les caractéristiques locales sont prédominantes, les méthodes multi-échelles fonctionnent le mieux, et l'agrégation des informations environnantes est très prometteuse. Par conséquent, il semble prometteur de concevoir nos représentations pouvant être apprises sous forme de réseaux de neurones à convolution (CNN).

Nous développerons cette conception dans le chapitre suivant, mais dans un premier temps, nous cherchons à élucider certaines questions auxquelles l'état de l'art ne répond pas. Nous proposons des réponses grâce à de nouvelles contributions et expériences.

Notre première contribution est un nouveau pipeline de classification par points utilisant des voisinages sphériques multi-échelles. Dans une première expérience nous montrons que ces voisinages sphériques sont bien supérieurs aux plus proches voisins dans les mêmes conditions expérimentales.

Grace à une nouvel méthode d'apprentissage actif, nous montrons ensuite l'importance de tester les performances sur un jeu de test diffèrent du jeu d'apprentissage, et nous établissons une nouvelle base de référence à laquelle nous pourront comparer des méthodes d'apprentissage profond. En effet notre méthode présente des résultats très compétitifs et surpassant même plusieurs approches d'apprentissage profond, malgré la simplicité de l'algorithme utilisé.

Enfin nous étudions l'influence de la densité de points dans nos voisinages. Cette expérience montre qu'il n'est pas nécessaire d'avoir beaucoup de points pour obtenir des représentations performantes. Réduire le nombre de points permet d'économiser du cout en mémoire et calculs, ce qui sera d'une importance cruciale dans l'optique d'un réseau de neurones très profond.

III.1. Overview of non-learning approaches

In this section, we review point cloud processing methods that do not use any learning mechanism. Their underlying model is thus built with tier 1 representations. In particular, we focus on algorithms that detect geometric patterns in point clouds.

Among these methods, a first category focuses on primitive shapes, like planes or spheres. Another group aims for contours, i.e. lines along which the surface orientation sharply changes. Eventually, some try to segment real objects, which requires underlying model that are more sophisticated.

Whether these patterns are used for semantic classification, or for other tasks like point cloud registration, they can be relevant to our study. Keeping in mind our goal to design new learnable representations for point clouds, we will analyze the underlying models in these methods and their link with representations that could be learned with deep neural networks.

III.1.1. Primitive extraction

Primitive extraction in 3D point clouds is a long known problem, crucial for many robotics applications. Primitives are standard geometric shapes that can be found in nature, but mostly in man-made environments. The most commonly used primitives are planes, spheres, cylinders, cones and tori. Planes are the most important primitives, because of their simplicity, and because in a man-made environment, most of the surfaces consist of planes.

Most of the primitive extraction methods fall into one of the two following categories:

- The approaches based on Random Sample Consensus algorithm (RANSAC), which directly estimate surface parameters by finding local maxima in the parameter space.
- The approaches based on region growing, which segment surfaces using local proximity of points and similarity criteria.

We will focus on these two types of methods and on what they can bring to our representation learning.

a. RANSAC

RANSAC is the mathematical translation of the “trial and error” method. This algorithm, initially published by (Fischler and Bolles, 1981), aims at estimating the parameters of a mathematical model from an observed data set that contains outliers. Under some theoretical assumptions, RANSAC ensures that there is a high probability to find a good estimate of the model, simply by trying out possible sets of parameters a certain number of times.

As an illustration, let us take the example of plane extraction in a 3D point cloud. The first assumption is that the data contains “inliers”, i.e. points that are located in the searched plane, and “outliers”, which are not located in this plane. The inliers may be subject to noise but will remain close to the model. The second assumption is that the searched model can be defined with a small number k of points. In our case, a plane can be defined with $k = 3$ points.

The plane thus can be estimated by iterating the following steps T times:

a. Randomly sample 3 points from the cloud.
b. Define the plane passing through these points.
c. Compute the score of this plane.

The plane that has the highest score after T iterations is kept as the prominent plane. The score is usually the number of data points that fit the model, so that are in a range σ of the plane.

Although this algorithm is not deterministic, we can ensure that the result will be found with a certain probability. Let n be the number of points in the plane and N the number of points in the whole cloud. The probability of picking an inlier each time a random point is selected is $P(inlier) = n/N$. The probability that we never picked three inliers in T iteration thus is $P(always\,false) = \left(1 - (n/N)^k\right)^T$. Thus the probability that we found the plane is:

$$P(sucessful\,ransac) = 1 - \left(1 - (n/N)^k\right)^T$$

Using this equation, we can determine the number of iterations T_{min} needed to detect a certain plane with the probability p:

$$T_{min} = \frac{\log(1-p)}{\log(1-(n/N)^k)}$$

For example, the point cloud in Figure 21 contains 160,000 points and we can assume that the ground plane roughly contains half of them. To ensure the ground will be found with more than 99% chance, the minimal number of iterations is:

$$T_{min} = \frac{\log(1-0.99)}{\log(1-0.5^3)} = 35$$

This shows how efficient this algorithm can be, under the right conditions. As shown in Figure 21, the ground plane is the one with the highest score and has been found in a few milliseconds.

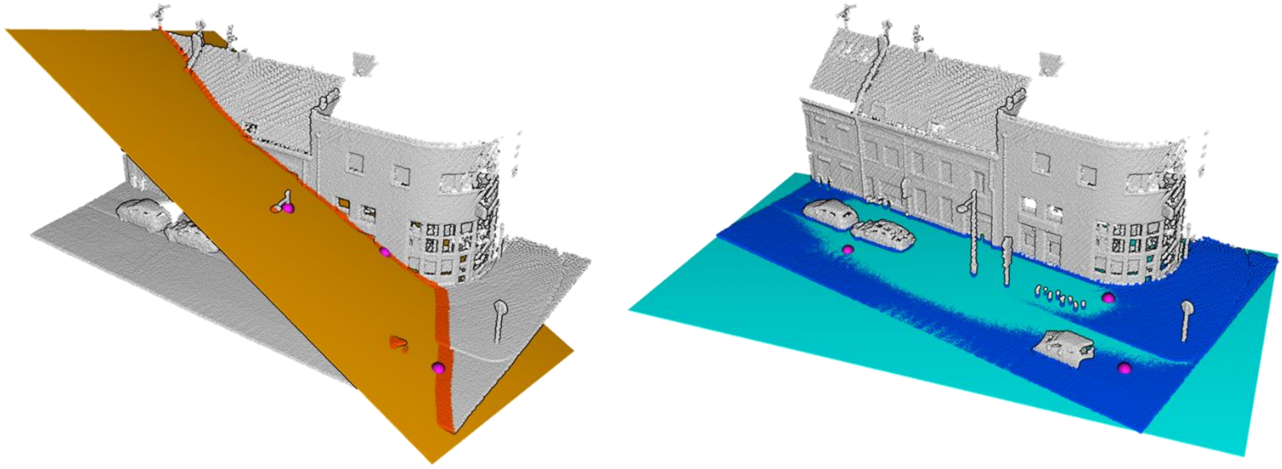


Figure 21. Example of plane extraction in 3D point cloud using RANSAC. A random plane with a low score (left), and the best plane (right) are shown. Scene taken from Paris-Lille-3D dataset.

RANSAC is robust to outliers and very efficient to detect large planes in point clouds. Indeed, if $n \to N$, then $T_{min} = 1$. However, for small planes, or more complex models with $k > 3$, the minimum number of iterations explodes. When $n \ll N$,

$$T_{min} \sim \log\left(\frac{1}{1-p}\right)\left(\frac{N}{n}\right)^k \sim O\left(\left(\frac{N}{n}\right)^k\right) \tag{III-1}$$

However, for concrete applications like primitive detection in 3D point clouds, RANSAC can be improved with heuristics. Many works have used RANSAC for different applications including façade detection (Boulaassal et al., 2007) or roof detection (Tarsha-Kurdi et al., 2007). Two optimizations of the algorithm have been proposed by (Schnabel et al., 2007) for 3D point clouds. First, they enforce a locality for the selected candidates, by picking them in an octree node of random depth. Then, they only compute an approximate score for each plane, on a subsampled version of the point cloud.

From our perspective, it is difficult to see how RANSAC could be adapted to a representation learning neural network. The key idea of RANSAC is at the opposite of our objective. It tries to estimate a simple and global model explaining the data, while our goal is to estimate a complex model built on various local representations.

b. Region growing

The name of this algorithm clearly indicates its principle: starting from a seed, a region expands to its neighborhood according to some conditions. In this formulation, we see the three crucial parts of this algorithm, the choice of a good seed, the notion of neighborhood and the definition of suitable growing condition. This algorithm is often used with depth images, where a neighborhood is already defined by pixel coordinates (Feng et al., 2014; Poppinga et al., 2008). In the case of 3D point cloud, the neighborhoods can be defined as KNN or radius neighborhoods, which means additional computations, even if they are accelerated with structures like Kd-Trees. Furthermore, the growing criteria often involve point normals, which are computed beforehand. In general, region growing is thus slower than RANSAC.

However, with efficient growing strategies, it can be very fast, especially for small planes. (Rabbani et al., 2006) developed a region growing method with a $O\big(N + nlog(N)\big)$ complexity. They first compute the point normal with a PCA method (Hoppe et al., 1992), and choose their seed as the point with the minimum residual. Then, they use the angle between the plane normal and the candidate normals as growing criterion with KNN neighborhoods. This complexity is low for small planes, but, for large planes, with $n \to N$, it becomes $O\big(Nlog(N)\big)$. (Deschaud and Goulette, 2010) enhanced the seed detection by choosing seed planes instead of seed points. They also use a voxel growing method, which has two crucial advantages:

1. The neighborhood are already defined,
2. The points are added to the region as groups and not individuals

With a complexity in $O(N)$, this method is fast and accurate and is able to detect small and large planes in point cloud datasets. The weakness of this algorithm, which is common with all other region-growing methods, is that it is highly dependent on the parameters. The criteria need to be chosen very carefully and the accuracy of the detection is unstable when they are modified. A single region can be separated or two regions can be merged together.

The idea behind region growing is really different from RANSAC and closer to our perspective. Region growing assumes that the global model (here, the plane) can be found by aggregating local information. However, the different criteria defined in these works are too simple to inspire new representations for neural networks.

We can still draw some conclusions from these region-growing algorithms. First, the type of neighborhood used is crucial for both the accuracy and the efficiency of the methods. As we seek to define local representations, neighborhood definition will also be critical for our algorithms and we will see that a simple choice between KNN and radius neighbors can be decisive for the performances. The other conclusion comes from the weakness of the method. The fact that these simple criteria are unstable indicates that the surfaces described by point clouds are more complex than they seem. Handcrafted representations like planes and normal may not be sufficient to describe every pattern in real point clouds. Adding more features and having learnable representations could help building more elaborate models.

III.1.2. Contour extraction

3D contour extraction in point cloud is a surprisingly difficult task. As opposed to images, where contour detection is a well known subject with many solutions, point clouds are unstructured and often inhomogeneous. The first works on 3D contour extraction actually concerned depth images, where the data is more organized (Sappa and Devy, 2001; Wani and Arabnia, 2003). However, in depth images, the definition of a contour is not exactly the same as in a point cloud. Because a depth image is taken from a certain point of view, the occlusions create *jump edges*, which are located where a pixel is between two dissociate surfaces, one in front of the other. In point clouds, there is no point of view and we only consider *crease edges*, which are located where a discontinuity in the surface orientation appears. The difference between *jump edges* and *crease edges* is illustrated in Figure 22.

In 3D point clouds, two levels of 3D contours can be detected: the points that are located near sharp edges and the straight lines formed by these points. Using only the first level is simpler as we only need a local measure of the sharpness, and already offers some applications like point cloud simplification (Song and Feng, 2009) or object detection (Rusu et al., 2008). This measure is often derived from normal vectors (Song et al., 2008) or local surface approximations (Fleishman et al., 2005; Öztireli et al., 2009). However, this first level only works locally and the found sharp edges do not always correspond to what we would call contours. This is particularly true in outdoor scenes with vegetation, where the point clouds can present whole areas with high surface roughness. The second level of 3D contour is more suitable for such scenarios.

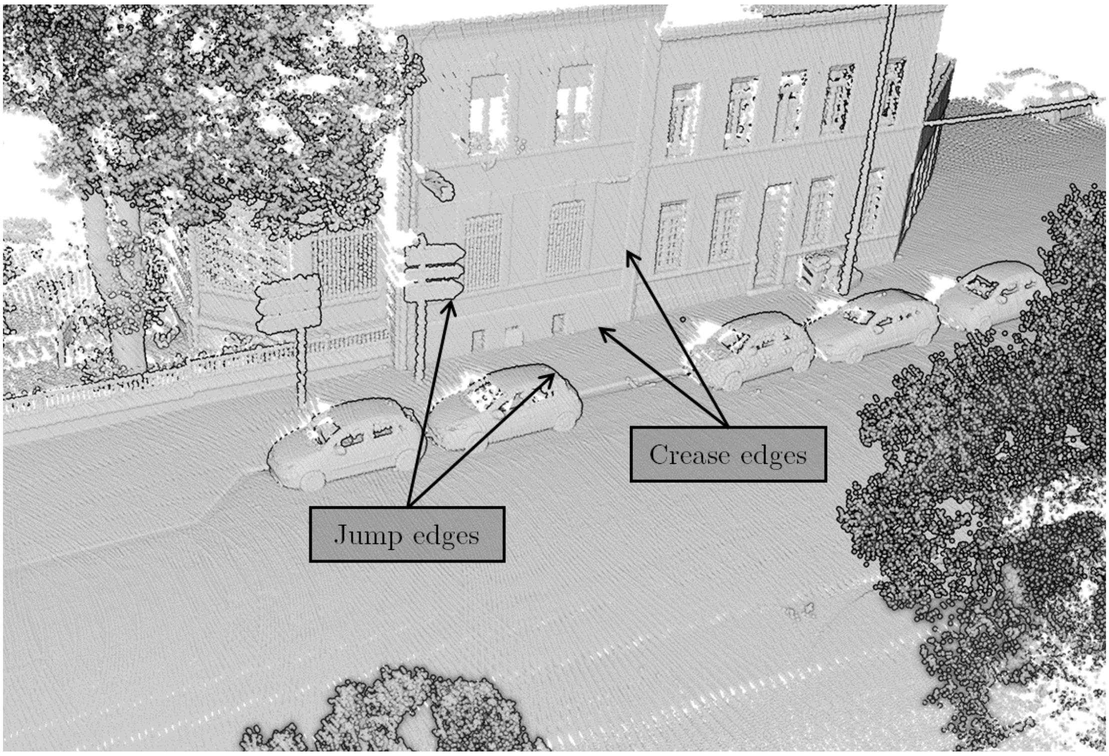


Figure 22. Illustration of crease edges and jump edges on an image of a point cloud. The jump edges only appear because of the point of view. Scene taken from Paris-Lille-3D dataset.

This second level, which considers contour as long-range line structures, is more elaborate and a simple threshold on local measure of sharpness is not sufficient anymore. (Hackel et al., 2016a) recently proposed a solution to tackle the complex task of detecting such contours in large point clouds. Interestingly, their method uses intermediate features to describe the point cloud and a learning mechanism to propose relevant line candidates. They do not consider contours as simple local patterns, but as an elaborate model to be found as the result of the algorithm.

As a first conclusion, the local sharpness features are interesting for our perspective as they also offer a local representation of point clouds. The example of the vegetation, which is not a contour but still presents sharpness locally, shows the high variability of local point cloud patterns. It confirms the conclusion drawn from region growing algorithms: one type of local features is not enough to describe all possible patterns in a real scenario; we need a larger collection of features and learnable representations.

Last but not least, we can see that even simple patterns, like structured lines, can be hard to detect in point clouds. They already are more elaborate representations than normals, sharpness or planarity measures. To go further and model real objects, our representations thus need to be built hierarchically, and aggregate simple features into more complex information. This is exactly what neural networks can do, with several layers of features.

III.1.3. Point cloud segmentation

Segmenting point clouds in rational components is not a specific, well-defined task. There are many ways to consider what rational components are in point clouds, and thus, several different strategies to segment them. We will try to cover most of the works that have been conducted with the objective to segment a point cloud in one way or another.

- **Primitive-based segmentation:** extracting primitives can be seen as a way to segment point clouds, or as a step in a segmentation process. (Vosselman et al., 2004) proposed a method based on multiple primitive extraction to segment point clouds. Another common segmentation scheme in urban environment consists of extracting connected component after removing the ground with a region growing or a similar algorithm (Golovinskiy et al., 2009; Hernández and Marcotegui, 2009; Roynard et al., 2016).
- **Contour-based segmentation:** the combination of an adjacency matrix and point cloud contours can be used to segment smooth pieces of surface (Castillo et al., 2013). This method is heavily based on the variation of normal vector, thus only viable with artificial point clouds.
- **Graph-based segmentation:** after an adjacency graph has been built on a point cloud, it can be segmented with graph cutting techniques. These methods consist of searching for and removing the least significant edges of a graph. After detecting an approximate position of objects, (Golovinskiy and Funkhouser, 2009) segment them from the background using a Min-Cut algorithm.
- **Morphology-based segmentation:** (Serna and Marcotegui, 2014) extract the objects from urban scenes in bird-eye view images of the point clouds. These bird-eye view images can be accumulation or elevation images, and are processed with mathematical morphology operations.
- **Super-voxel segmentation:** These methods first project the points into voxels, and then create super-voxels, by grouping adjacent voxels with similar properties (Aijazi et al., 2013; Papon et al., 2013). They are often used as an intermediary steps, as it reduces the number of elements to be segmented afterwards. (Christoph Stein et al., 2014; Schoeler et al., 2015) use connectivity criteria and local convexity to build super-voxels that respect the actual object boundaries.

These segmentation methods, whether they produce segmented objet, or segmented part of objects, often rely on more elaborate model of the reality than primitive or contour extraction. In particular, they include spatial context into their representations. For example, in urban environment, they rely on the assumption that objects are isolated, only surrounded by ground. To include such representations, the successive layers of our neural network have to aggregate local features into a more global information that has a certain context awareness. In the last layers, each feature need to account for a large part of the space, this is what we will call the receptive field. We will come back to this notion in the Chapter IV of this manuscript.

In most cases, these segmentation methods are followed by learning based classification schemes to identify the type of each segmented object. We will develop them in the Sections III.2.2 and III.2.3.

III.2. Learning-based methods using hand-crafted features

This section describes learning-based methods for point cloud processing. In particular, we focus on supervised learning strategies, which use tier 2 representations. They do not use explicit descriptions of the appearance of specific objects anymore. Instead, they use handcrafted descriptive tools, the features, and rely on previous examples that are given to learn how each object looks like according to these features.

The handcrafted features give a description of the object, its appearance, its characteristics, as a list of values. They are of the same nature than the values seen in the last sections, like sharpness that was used for contour detection. However, we do not fix rigid thresholds on these values to describe the object anymore, but we let the algorithms learn them thanks to the second crucial element, the database.

In order to learn which combination of feature values corresponds to which object, the algorithms need to see some examples. These examples are to be seen as samples of random distributions. In reality, objects are diverse, and even though we can make the difference between a car and a tree, not all trees look the same. The ensemble of possible trees is a probabilistic distribution, and the goal of the learning process, is to let the algorithm estimate the parameters of this distribution, with a few samples.

We will focus on semantic segmentation algorithms, and, to a lesser extent, instances segmentation algorithms. We seek to understand what kind of handcrafted representations they use, and draw conclusion for which ideas should be kept in our neural network representations. Among the ways to build handcrafted point cloud representations, two paradigms stand out. In the first instance, each point is considered individually and is given a semantic label or class probabilities (see Figure 23). In the second instance, the point cloud is segmented and then a label is given to each segment (see Figure 24). From these two paradigms, hybrid methods emerged using graph partitioning algorithms.

III.2.1. Pointwise classification

The simplest form of point cloud semantic segmentation algorithm is a pointwise classification framework described in Figure 23. This is a naïve, but efficient, solution to this complex task. It starts from the typical supervised learning approach: we have a set of elements that we want to classify: the points, then we just need to describe each point individually with features, and let a standard machine-learning algorithm do the labeling.

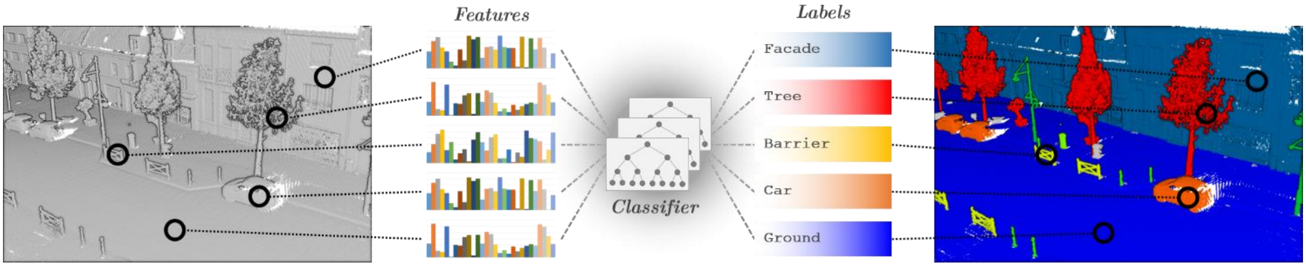


Figure 23. Standard pointwise classification framework. A set of features is computed for every point of the scene and a classifier is used to label each point independently.

Without any prior segmentation, classifying 3D points only relies on the appearance of point neighborhoods. Thus, we need a set of expressive features to describe the geometry of a local group of points. Several types of features have been proposed in the literature, and the idea of using covariance based features to describe the dimensionality of a local group of points arose with (Lalonde et al., 2006). This covariance features proved to be very discriminant, despite their simple definition (Demantke et al., 2011; Douillard et al., 2011). To complement this local shape description, (Weinmann et al., 2013) added measures of verticality and height distribution.

We can find more complex descriptors in the literature, like *Spin Images* (Johnson and Hebert, 1999), *Fast Point Feature Histograms* (Rusu et al., 2009), *Viewpoint Feature Histograms* (Rusu et al., 2010), or *Radius-based Shape Descriptors* (Marton et al., 2010a). However, these features are quite heavy to compute for a very low added value. In the context of large-scale point cloud processing, they are therefore difficult to exploit. Recently, (Weinmann et al., 2015a) and (Hackel et al., 2016b) proved that simple covariance features already offer a very expressive description, while maintaining fast computations.

Whatever feature is chosen, it will depend on the type of point neighborhood used. In the case of 3D point classification, the two most commonly used neighborhood definitions are the radius neighborhood (Brodu and Lague, 2012) and the KNN (Weinmann et al., 2013). Some works used the cylindrical neighborhood, but mostly for airborne lidar data (Chehata et al., 2009; Niemeyer et al., 2014). The scale of the neighborhood plays a crucial role for the descriptors to be expressive. Using a fixed scale across the scene is inadequate because most scenes contain objects of various sizes. (Demantke et al., 2011) explored a way to adapt the scale to each point of the cloud, and (Weinmann et al., 2015a) improved the idea and called it *adaptive neighborhood*. However, using a multiscale approach has proven to be more effective, whether it is used with KNN (Hackel et al., 2016b; Pauly et al., 2003), with spherical/cylindrical neighborhoods (Brodu and Lague, 2012; Niemeyer et al., 2014) or with a combination of all neighborhood types (Blomley et al., 2016). The major drawback of the multiscale neighborhoods is their computational time. (Hackel et al., 2016b) suggested a simple and efficient solution to implement them, based on iterative subsampling of the cloud.

The last cornerstone of the framework is the classification algorithm. Standard machine learning techniques are usually chosen to classify 3D points described by geometric features. According to (Weinmann et al., 2015a) extensive work, *Random Forest* is the most suitable classifier. But we can see that this part of the framework is the least suited for point cloud processing. The neighborhoods and features are designed accordingly to the nature of the data, while this classifier processes points independently, ignoring spatial relations between points. We will see in Section III.2.3, that other types of classifiers, involving graph structures can be used to ensure spatial coherence in the classification.

Many interesting conclusions can be drawn from this pointwise classification approaches. First, we see that building complex representations does not lead to better performances than simple representations, as long as the latter are used on multiple scales. The covariance features show very interesting properties, including an invariance by rotation. This invariance is beneficial, as we want our method to be invariant by rotation with the entry. However, we also noticed the importance of adding verticality features, which highlights the specificity of the vertical direction on real scenes. In the end, the whole set of features is only invariant by rotation around the vertical axis. In our representations, it would be desirable to have an invariance around this axis, and we will see that PCA is not the only way to build it in a neural network.

In addition, we can draw an inference concerning the idea of multiscale neighborhoods. Using multiple scales of features gives more expressivity to the final description as it includes more details. It is also a way to incorporate the spatial context (with the bigger scales), but it is a very limited representation of the surrounding of a point. Instead of computing a single description of a large area around a point, it would be better to aggregate many local descriptions of the surrounding of the point, which is something that we can do with neural network representations.

Another significant idea is the subsampling scheme used by (Hackel et al., 2016b). In their multi-scale approach, the points used to compute each neighborhood are subsampled by a factor proportional to the scale of the neighborhood. Indeed, each neighborhood is a local group of points that is described as a whole by the features. The measure of the global aspect of a group of points does not require fine details, a few points are sufficient to get accurate features. This is something that we have to take into account when building neural network representations. With several layers, which correspond to several scales of features, a neural network also needs a proportional subsampling strategy. This is the role of the max-pooling layers in image CNN for example.

Eventually, there is a question that all these methods raised but did not answer. Is it preferable to use KNN or radius neighborhoods? Some works tried to compare them, but became irrelevant with the rise of the multiscale features. The results obtained by (Hackel et al., 2016b) with KNN outperformed any competition by a large margin, but it is their multiscale approach that really made the difference. How would radius neighborhood perform with such a multiscale definition? This question is essential, because the point cloud representations that we want to build also needs a neighborhood definition, and will use multiple scales. As our goal is to describe the geometry around points, we can guess that spherical neighborhoods would give a better spatial consistency to our features. Our first contributions, described in Section III.3, aim at investigating this question and finding if this intuition is indeed true.

III.2.2. Segmented object classification

Pointwise classification approaches do not consider objects in the scene; they only process points according to what their neighborhoods look like. As we saw in Section III.1.3, there are ways to segment the object of a scene with non-learning strategies. Once segmented, these objects can be described by a set of global features and then classified by a standard classifier. This second framework, proposed by (Golovinskiy et al., 2009) and illustrated in Figure 24 can be used for the semantic segmentation of a point cloud, or for instances segmentation in a point cloud.

The particularity of these methods is that they use global object features. These features need to describe the different objects that are bigger groups of points than local neighborhoods. If we used the local covariance features naively on these objects, they would not offer an expressive description, as the

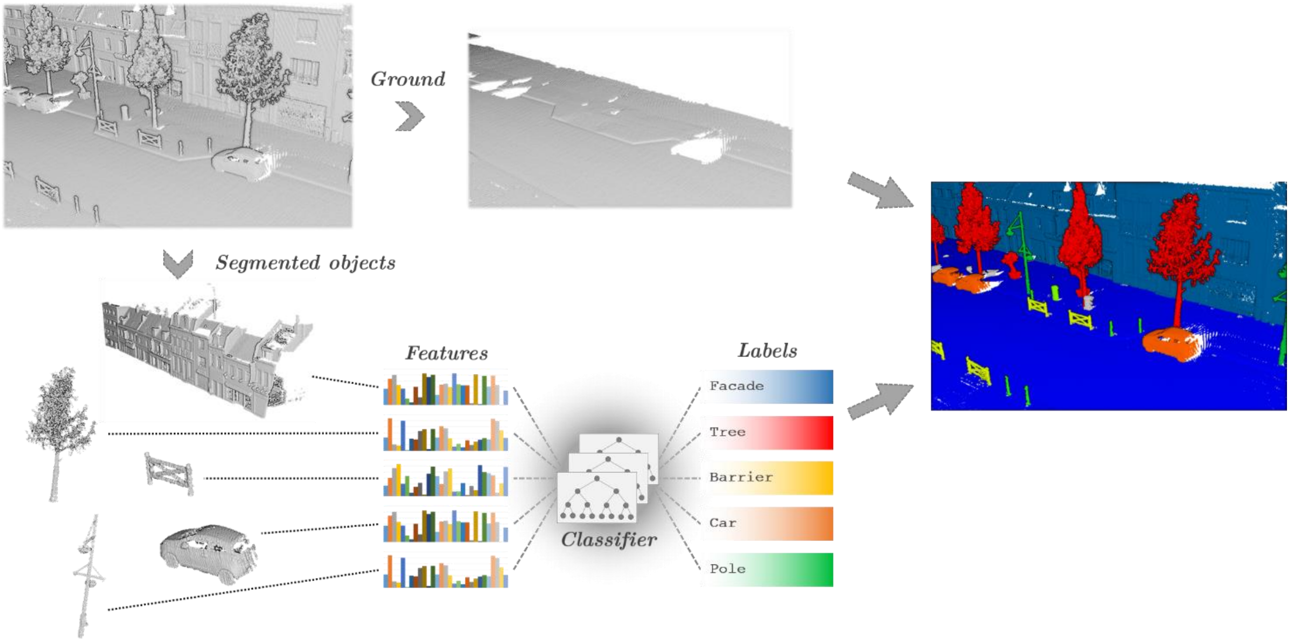


Figure 24. Standard pipeline for the classification of segmented objects. The ground and the objects are isolated by a segmentation algorithm, then, each object is classified as a whole.

objects often have complex shapes that need details to be described. The global features used by these methods thus are more complicated than the local features we just saw.

Several geometric descriptors based on elevation images have been proposed by (Hernández and Marcotegui, 2009), they include the area, perimeter and volume of the object, as well as the minimum value, the maximum value and the standard deviation of the object height. These descriptors are simple but already offer strong discriminative power, especially in urban scenes, where the different types of objects have various geometries. Other simple features have been used from time to time in the literature, like the number of points, the length and the width of the object (in the main component reference), and the eigenvalues of the covariance matrix (same idea as the local covariance features). In particular, (Roynard et al., 2016) added these features to other known descriptors to help their segmentation pipeline.

For a stronger discriminative power, more complex features have been developed, often in the form of histograms. The *Viewpoint Feature Histograms* introduced by (Rusu et al., 2010) are based on the angles of the normals from a certain point of view, over the entire surface of the object. The *Clustered Viewpoint Feature Histograms* (Aldoma et al., 2011) improved the latter by segmenting the object into surfaces with coherent normals, and by using these surfaces instead of the entire object. An even simpler description scheme called *Ensembles of Shape Functions* was proposed by (Wohlkinger and Vincze, 2011), and consists of evaluating the distributions of some shape functions (like distance between two points, or angle between three points) in the object. *Ensembles of Shape Functions* are very fast to compute as they only involve random point sampling and evaluation of these simple functions.

Eventually some global descriptors are built using histograms of local descriptors. For example the *Radius-based Shape Descriptors* become the *Global Radius-based Shape Descriptors* (Marton et al., 2010b), and the *Fast Point Feature Histograms* become the *Global Fast Point Feature Histograms* (Rusu et al., 2009).

The main weakness of these methods is that they rely on the first segmentation, which is unreliable when the objects are partially occluded or very close to one another. In the first case, there is a risk that multiple close objects are segmented together (sub-segmentation); in the second, the objects can end up being segmented into multiple pieces (over-segmentation). These mistakes are not corrected during the classification part of the standard pipeline, but (Wang et al., 2015) proposed a solution using super-voxels. The idea is to over-segment the object in super-voxels, describe these patches with local features, and use a *Random Forest* to decide which super-voxel effectively belongs to the object. This idea of super-voxel over-segmentation is even more effective when combined with graph-based methods, as we will see in Section III.2.3.

The definition of global features is a valuable idea for our neural network representations. Compared to the multiscale pointwise features, the global descriptors are defined as an aggregation of local measures. However, these descriptors are not satisfactory, as they only include one level of aggregation. The use of histograms to aggregate local features whole objects is too harsh. Our neural network representations need to be aggregated gradually, at several scales, which can be viewed as a mix between the local and global features concepts.

The lack of context is another weakness that these global features share with the local pointwise features. We only use the points of the object to classify them, while the spatial context could also be meaningful. Like local features, some kind of context can be acquired when using features involving neighborhoods, but it remains limited. This issue can be solved with neural networks, as they segment a whole scene at once. They use the spatial context around an object as well as its own geometry, to recognize it.

III.2.3. Graph-based classification

Both point-wise and object-wise classification approaches share the lack of spatial context in their classification process. They classify every element of a scene independently, without considering the surrounding elements and their possible relationships. Many methods have been proposed to help in this matter, and they nearly always involve graph structures.

Whether the adjacency graph is built on the points themselves, or on segmented components, the edges will encode the spatial relationship between scene elements. With a good graph optimization strategy, it is thus possible to perform a semantic segmentation with a sense of spatial context. To replace standard classifiers like *Random Forest* in the pointwise classification framework, (Munoz et al., 2009) proposed to use *Markov Random Fields* (MRF), while (Niemeyer et al., 2011) proposed to use *Conditional Random Fields* (CRF), the deterministic version of MRF. These probabilistic models allow to define pairwise interactions (for example, two adjacent points should more likely have the same label) and unary potentials (label of a point should depend on the features at this point).

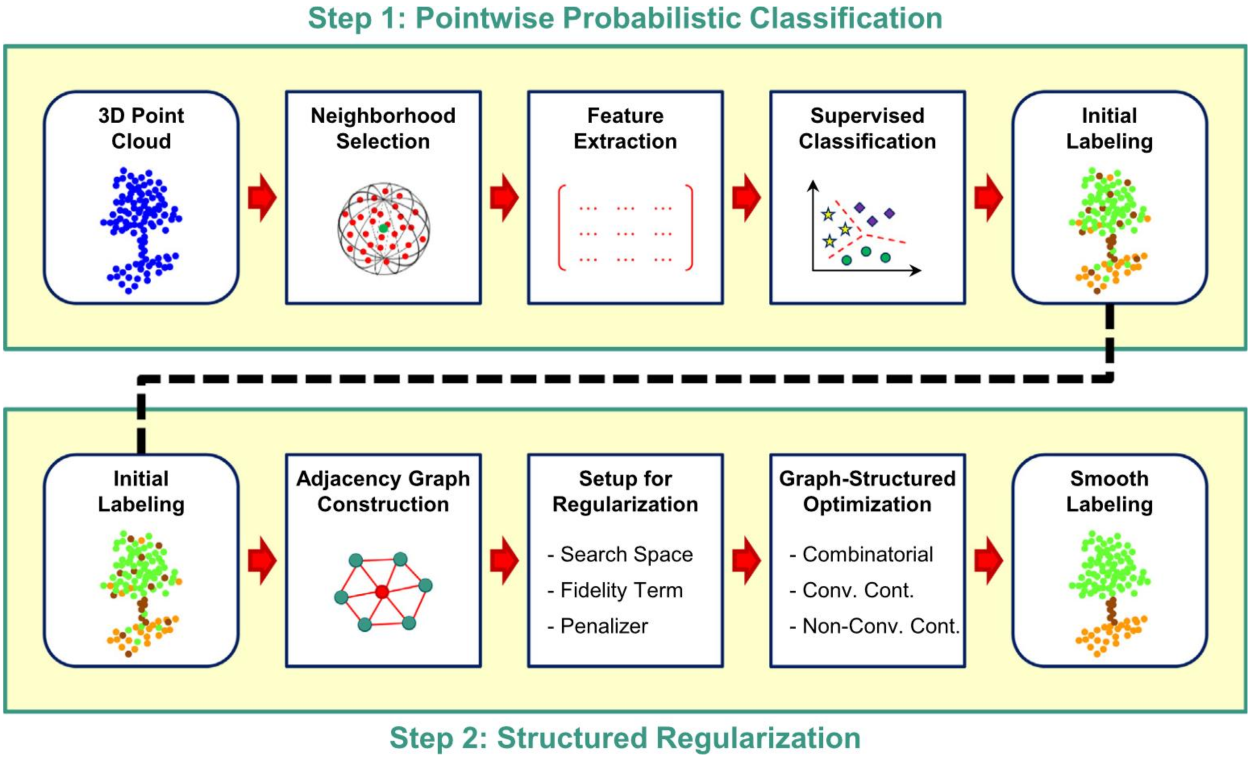

Step 1: Pointwise Probabilistic Classification
3D Point Cloud
Neighborhood Selection
Feature Extraction
Supervised Classification
Initial Labeling
Initial Labeling
Adjacency Graph Construction
Setup for Regularization
- Search Space
- Fidelity Term
- Penalizer
Graph-Structured Optimization
- Combinatorial
- Conv. Cont.
- Non-Conv. Cont.
Smooth Labeling
Step 2: Structured Regularization

Figure 25. Standard pipeline for the regularization of classified point clouds. Image from (Landrieu et al., 2017).

Although this kind of classifiers are good at creating a spatial consistency in the segmentation, they are not known for their discriminative power on complex features. Therefore, a better classification scheme consists of using the label probabilities given by a standard classifier as unary potentials instead of the features themselves (Najafi et al., 2014; Shapovalov et al., 2010; Weinmann et al., 2015b). In that case, the graph method can be seen as a regularization method for the initial labeling of the standard classifier. More recently, (Landrieu et al., 2017) used this regularization approach and compared several graph optimization techniques. The steps of their approach are illustrated in Figure 25.

As we already explained, this kind of method can be used on segmented components instead of points. The idea is to first segment the scene with *super-voxels* or *super-points*, and then to use these patches as graph nodes instead of the points. (Wolf et al., 2015) used this super-voxel approach, classifying each segment with a *Random Forest*, and regularizing the whole scene with a CRF. More recently, (Landrieu and Simonovsky, 2017) created a segmentation pipeline using super-points, however, their classification and graph regularization is formulated as a single framework using deep learning architectures.

When using super-voxels or super-points, the initial mistakes in the over-segmentation remain. If the segmented patches do not respect the actual object boundaries, it will lead to errors in the results. If the computational cost of the method is not critical for the application, a pointwise approach should be preferred.

The graph structure thus helps getting a sense of spatial context in the classification process, which is very valuable in a semantic segmentation pipeline. However, it is often used as an additional regularization bloc in the framework. In that case, it is not very useful for our representations, as it only tries to regularize semantic labels that have already been found by the neural network. It helps the whole segmentation results, but does not affect the representations used in the early stages. In fact, it is not obvious that we need such graph regularization techniques with deep learning approaches. As explained in the previous sections, the spatial context is already encoded because the network processes a whole scene at once, and the receptive fields of the last layers (areas that influence the value of each feature) cover large parts of the space (c.f. Section IV.3.7).

Graphs can also be used directly as the support structure for convolutions. However, these graph convolution approaches do not share much more than the graph structure with the methods we described here. We will describe them in details in Section IV.1.2.

III.3. Proposed pointwise classification method

The conclusions drawn from the two previous sections are in accordance with conclusions made in image computer vision and leading to the design of convolutional neural networks. Among the ways to process point clouds, the local features are predominant, and the schemes including multiple scales or aggregation of the surrounding information are very promising. It is not surprising as point cloud are spatialized data. Like in images, notions like neighborhoods or spatial relationships are crucial to characterize patterns. Therefore, it seems promising to design our learnable representations as convolutional neural networks (CNN).

We will develop this design in the next chapter, but we first need to elucidate some questions that have not been answered in the state of the art. In this section, we are going to investigate these unresolved issues and propose some answers thanks to new contributions and experiments.

III.3.1. Unresolved issues from the state of the art

Our goal is to build a convolutional architecture for point clouds that can learn its own tier 3 representations and reach a higher degree of abstraction. In order to achieve this goal, we need some essential elements:

- A neighborhood definition.
- A convolutional operator.
- A subsampling scheme.
- Large datasets for training.

Let us forget the convolutional operator for now, and focus on the other elements. In a CNN, there are several layers with multiple scales; therefore, we are very interested in the definition of multiscale neighborhoods. These approaches have a strong potential to be adapted to deep neural networks. In particular, the proportional subsampling scheme proposed by (Hackel et al., 2016b) can be used to define consecutive layers in a CNN, with a straightforward control over the range of the convolution at each layer. The CNN design will bring the spatial aggregation that these features are missing, and the overall architecture should present a very strong descriptive power. However, this adaptation raises two issues that had no answer in the state of the art before this thesis.

issue 1. As we saw in the state of the art, **the type of neighborhood** can have a decisive impact on the performance of a method. In previous work, none of the KNN and radius neighborhoods definitions have convincingly outperformed the other. The latest works seem to prefer KNN, and the multiscale KNN features have the most promising performances for pointwise classification. However, radius neighborhoods have not been compared to KNN in a multiscale scheme like (Hackel et al., 2016b). We need to assess their performances in the same experimental conditions.

issue 2. In the multiscale scheme, it is important to understand **the effect of the density**. In (Hackel et al., 2016b), the same number of nearest neighbors is used at each scale, only the size of the subsampling grid is growing. As a result, the sampled neighbors cover a larger and larger space. With radius neighbors, the same behavior is obtained by growing the radius proportionally to the subsampling grid size. Then, the ratio between the neighborhood radius and the grid size (corresponding to the number of neighbors of KNN) controls the amount of details that the neighborhoods are allowed to see. More neighbors means better descriptions, but also slower computations. We need to know how many neighbors are sufficient to get expressive features.

Even though it is not a part of the network design, the last point is quite important. We know from image computer vision that deep learning algorithms need large amount of data, with diversity, to become relevant and learn meaningful representations. As 3D point cloud scanning technologies are recent, there is no huge standard dataset, like ImageNet (Deng et al., 2009), to train our networks. Two issues concerning 3D point cloud datasets should also be answered by new experiments.

issue 3. In most of the state of the art approaches published before 2016, which is the case for almost every work on handcrafted features, there was a lack of dataset to conduct large-scale experiments. The classifiers were trained with points of the same scene they were supposed to label. The validity of such experiments need to be verified. We need to be careful about the conclusions we draw from them.

issue 4. At the time of the development of this work, very few deep learning architectures had been proposed and their performances were not cutting-edge. As very few handcrafted method had been tested on new large-scale point cloud datasets, we were not even sure that these first deep learning approaches outperformed simple handcrafted classification pipelines. We would like to establish a strong baseline on several new large-scale datasets with handcrafted features, to evaluate the performances of deep learning methods.

Our contributions, presented in next sections, address each of these issues. Most of them were published in our first conference paper (Thomas et al., 2018), but we also add some insightful experiments that were conducted earlier and could not fit in this publication.

III.3.2. New multiscale spherical neighborhoods

Our first contribution is the definition of new multiscale spherical neighborhoods. This definition, inspired by the multiscale KNN of (Hackel et al., 2016b), will be used to solve issue 1 and issue 2. This section highlights the differences and the theoretical advantages of our definition over multiscale KNN. The experimental comparison will be detailed in the next section.

Let $C \subset \mathbb{R}^3$ be a point cloud. The radius neighborhood of point $\mathbf{p_0} \in \mathbb{R}^3$ in C with radius $r \in \mathbb{R}$ is defined by:

$$\mathcal{N}(\mathbf{p_0}, C, r) = \{\mathbf{p} \in C \mid \|\mathbf{p_0} - \mathbf{p}\| \le r\} \qquad \text{(III-2)}$$

Unlike KNN, this neighborhood corresponds to a fixed part of the space as shown in Figure 26. This property is the key to give a more consistent geometrical meaning to the features. However, in that fixed part, the number of points can vary according to the cloud density. As (Hackel et al., 2016b) explained, radius search should be the correct procedure from a purely conceptual viewpoint but it is impractical if the point density exhibits strong variations. Two phenomena appear in particular:

- Having too many points when the neighborhood scale is too big or the density too high
- Having too few points when the neighborhood scale is too small or the density too low

The first phenomenon has computational consequences as getting a large number of neighbors in a larger set of points takes a lot of time. The computational cost of multiscale features does not come from the fact that we have to compute the features S times, where S is the number of scales. The real limiting

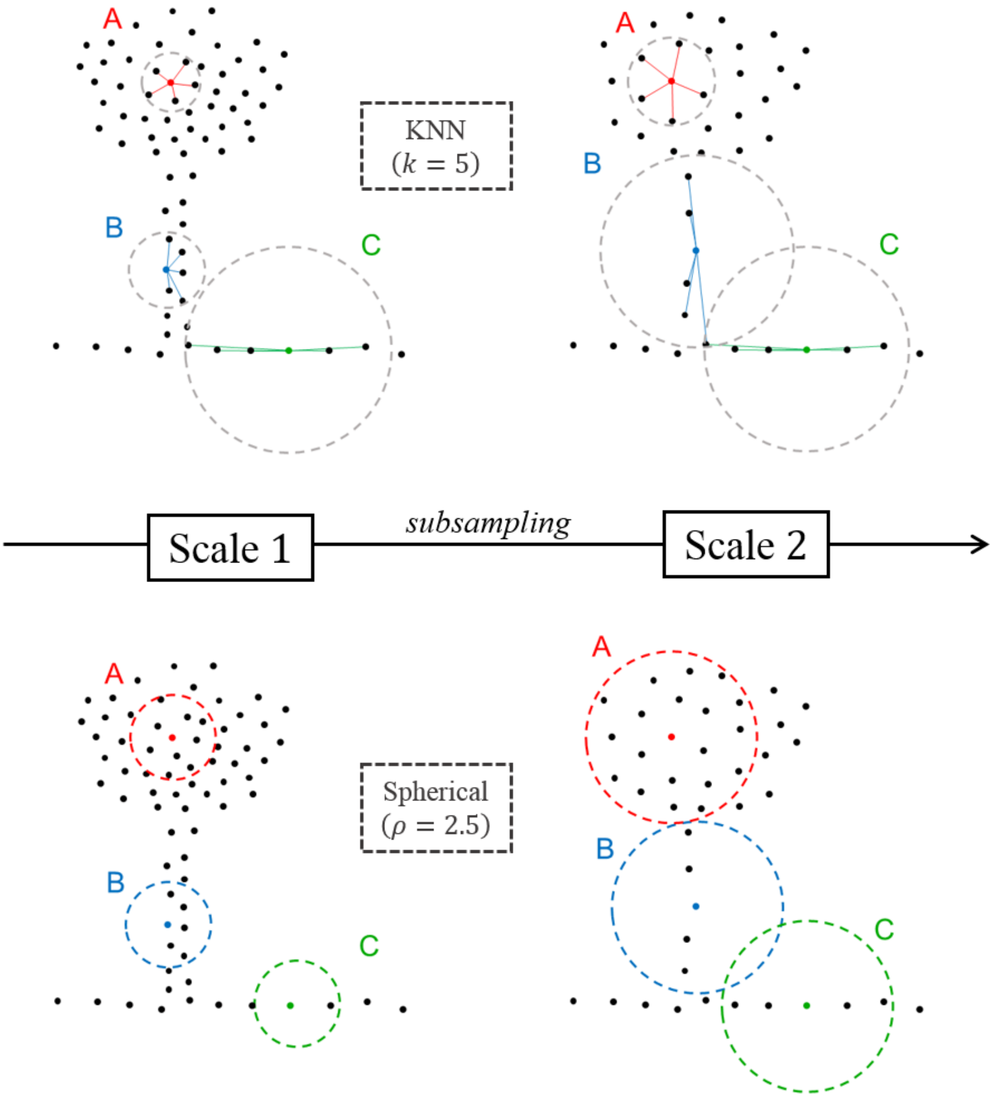


Figure 26. Behavior of multiscale neighborhoods defined with KNN or with spherical neighborhoods.

factor is the number of points contained in the biggest scales. Furthermore, not all these points are required to compute relevant features. Our features capture a global shape in the neighborhood and do not need fine details. The solution proposed by (Hackel et al., 2016b) to subsample the cloud proportionally to the scale of the neighborhood can be adapted to the spherical definition. This solution better suits the spherical neighborhoods than the KNN. With a uniform density, the number of points in the neighborhood becomes a feature itself, describing the neighborhood occupancy rate. We choose to subsample the cloud with a grid, by keeping the barycenter of the points comprised in each cell. Let $l \in \mathbb{R}$ be the size of the grid cells, for any radius of a neighborhood $r \in \mathbb{R}$, we can control the maximum number of points in our neighborhood with the parameter $\rho = \frac{r}{l}$. If ρ is too low, we will not have enough points and the features will not be discriminant, but the higher its value is, the longer computations are.

The impact of the second phenomenon, caused by low densities, should be limited by the use of multiple scales. As illustrated in Figure 26, if the density is too low, the points will remain the same after subsampling between two consecutive scales (neighborhood C in green). With spherical neighborhoods, the smallest scales might not contain enough points for a good description, but the larger scales will deliver the information. In the same case, the KNN behave differently, giving exactly the same information at both scales. Regardless of the neighborhoods, there is no information to get from the data at the smaller scale. However, the KNN give a false description of the smaller scale without any measure of its reliability, whereas spherical neighborhoods give the number of points, which is an indication of the robustness of the description.

The scales are defined by three parameters: the radius of the smallest neighborhood r_0, the number of scales S, and the ratio between the radiuses of consecutive neighborhoods φ. We can then define the neighborhood at each scale $s \in \{0, \dots, S-1\}$ around point $\mathbf{p_0} \in \mathbb{R}^3$ as:

$$\mathcal{N}_s(\mathbf{p_0}) = \mathcal{N}(\mathbf{p_0}, C_s, r_s) \tag{III-3}$$

With $r_s = r_0 \times \varphi^s$ being the radius at scale s and C_s being the cloud subsampled with a grid size of r_s/ρ.

Despite its similarity with the definition proposed by (Hackel et al., 2016b), our multiscale neighborhood definition stands out with its geometrical meaning. With spherical neighborhoods instead of KNN, the features always describe a part of the space of the same size at each scale. Moreover, the number of points in the neighborhood is now a feature itself adding even more value to this definition. More than a theoretical good behavior, this leads to better feature performances, as shown in Section III.3.3.

III.3.3. Comparison with multiscale KNN

In this section we evaluate the performances of our multiscale spherical neighborhoods, in comparison with multiscale KNN, and adaptive scale KNN (Hackel et al., 2016b; Weinmann et al., 2015a). We first describe the experimental conditions and then present the results.

a. Experimental conditions

For benchmarking purposes, we use a first set of features, described in Table 2, which does not use any additional information like intensity, color, or multispectral measure. These features are slightly different than those used in (Hackel et al., 2016b; Weinmann et al., 2015a), because we want to establish new standard, for the choice of features in the future. Nevertheless, the changes are minor and the crucial difference remains the neighborhood definition. We use covariance-based features that simply derive from the eigenvalues $\lambda_1 > \lambda_2 > \lambda_3 \in \mathbb{R}$ and corresponding eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \in \mathbb{R}^3$ of the neighborhood covariance matrix defined by

$$\mathrm{cov}(\mathcal{N}) = \frac{1}{|\mathcal{N}|}\sum_{\mathbf{p}\in\mathcal{N}} (\mathbf{p}-\overline{\mathbf{p}})(\mathbf{p}-\overline{\mathbf{p}})^T \tag{III-4}$$

Table 2. Features used for our comparative experiment.

Features	Definitions
Sum of eigenvalues	$\sum \lambda_i$
Omnivariance	$\left(\prod \lambda_i\right)^{1/3}$
Eigenentropy	$-\sum \lambda_i \ln(\lambda_i)$
Linearity	$(\lambda_1 - \lambda_2)/\lambda_1$
Planarity	$(\lambda_2 - \lambda_3)/\lambda_1$
Sphericity	λ_3/λ_1
Change of curvature	$\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
Verticality (× 2)	$\left\|\frac{\pi}{2} - \mathrm{angle}(\mathbf{e}_i - \mathbf{e}_z)\right\|_{i\in[0,2]}$
Absolute moment (× 6)	$\frac{1}{\|\mathcal{N}\|}\left\|\sum \langle \mathbf{p} - \mathbf{p_0} \| \mathbf{e}_i \rangle^k\right\|_{i\in[0,1,2]}^{k\in[1,2]}$
Vertical moment (× 2)	$\frac{1}{\|\mathcal{N}\|}\sum \langle \mathbf{p} - \mathbf{p_0} \| \mathbf{e}_i \rangle^k_{k\in[1,2]}$
Number of points	$\|\mathcal{N}\|$

Where $\overline{\mathbf{p}}$ is the centroid of the neighborhood $\mathcal{N}$. From the eigenvalues, we can compute several features: *sum of eigenvalues*, *omnivariance*, *eigenentropy*, *linearity*, *planarity*, *sphericity*, *anisotropy*, and *change of curvature*. However, among those commonly used features, we eliminate *anisotropy* defined by $(\lambda_1 - \lambda_3)/\lambda_1$ as it is strictly equivalent to *sphericity*. We can notice that, thanks to the nature of our neighborhoods, we do not need to normalize the eigenvalues as in previous works. Their values do not vary with the original point cloud density, which means the features that do not involve ratios (e.g. *sum of eigenvalues*, *omnivariance*, and *eigenentropy*) make more sense. Our feature set is completed by verticality that we redefined as $|\pi/2 - \text{angle}(\mathbf{e}_i - \mathbf{e}_z)|$. Unlike (Hackel et al., 2016b), we keep the verticality for the first and the last eigenvectors. The first one encodes the verticality of linear objects and the last one the verticality of the normal vector of planar objects.

We also use first and second order moments around all three eigenvectors, but in absolute value as the eigenvectors have undefined orientations. Following our assumption that vertical direction plays an important role, additional vertical moments are computed around the vertical vector $\mathbf{e}_z$ in relative value as the upward direction is always the same. Eventually, as explained in Section III.3.2, the number of points in a neighborhood completes our first set of features, which contains 18 values at each scale.

We use the Rue Madame dataset (Serna et al., 2014), and the IQmulus & TerraMobilita dataset (Vallet et al., 2015), presented in Section II.2.2. Like previous works, we use a random forest classifier trained on 1000 random points per class for each dataset. We choose the following parameters

$$S = 8 \qquad r_0 = 0.1m$$

$$\varphi = 2 \qquad \rho = 5$$

The first three parameters S, r_0, φ were chosen so that the scales of our neighborhoods cover a range from the smallest object size to the order of magnitude of a facade and the last parameter ρ was chosen empirically (see Section III.3.6). With an average personal computer setup (32 GB RAM, Intel Core i7-3770; 3.4 GHz), our feature extraction took about 319 seconds on IQmulus & TerraMobilita, which is the same order of magnitude as (Hackel et al., 2016b) (191s) and way faster than (Weinmann et al., 2015a) (23,000s).

b. Results

For consistency, we choose to use the classes "Intersection over Union" (IoU) metric in all our experiments. Thus, we convert the class F_1 scores given in (Hackel et al., 2016b; Weinmann et al., 2015a) with the equations :

$$F_1 = \frac{2TP}{2TP + FP + FN} \tag{III-5}$$

$$\text{IoU} = \frac{TP}{TP + FP + FN} = \frac{F_1}{2 - F_1} \tag{III-6}$$

Where TP, FP, and FN respectively denote true positives, false positives, and false negatives for each class. In Table 3, we report the average class IoU, to compare with previous results and the standard deviations to prove the consistency of the classification. As will be explained in Section III.3.4, we reproduced our results 500 times to ensure the validity of the comparison despite the random factor in the choice of the training set. The performances of our multiscale features exceed previous results by 24 mean IoU points on Rue Madame and 11 mean IoU points on IQmulus & TerraMobilita. We can also note that our results do not vary much, the standard deviations being limited to a few percents even for the hardest classes with fewer points.

Table 3. Average IoU (with standard deviation) on Rue Madame (top) and IQmulus & TerraMobilita (bottom) datasets. Previous results are converted from corresponding articles.

Class	Ours	(Hackel et al., 2016b)	(Weinmann et al., 2015a)
Façade	**98,22%** (± 0,11)	97,06%	91,81%
Ground	**96,62%** (± 0,18)	96,29%	84,88%
Cars	**95,37%** (± 0.43)	89,09%	55,48%
Motorcycles	**61,55%** (± 2.22)	47,44%	9,44%
Traffic Signs	**67,43%** (± 8.67)	33,96%	4,90%
Pedestrians	**77,86%** (± 3.32)	24,13%	1,63%
Mean	**82,84 %**	58,89 %	31,68%

Façade	**97,27%** (± 0,20)	93,89%	86,65%
Ground	**97,77%** (± 0,20)	96,99%	95,75%
Cars	**84,94%** (± 1,54)	80,88%	47,31%
Motorcycles	**58,99%** (± 3,27)	51,33%	17,12%
Traffic Signs	12,71% (± 3,81)	**18,58%**	14,29%
Pedestrians	**35,31%** (± 3,80)	24,69%	9,06%
Vegetation	**71,48%** (± 1,94)	51,40%	24,63%
Mean	**65,50 %**	54,08%	35,30%

As a conclusion, the low standard deviations validate the random selection of the training set and legitimates the comparison of the different sets of features. Our multiscale features thus proved to be superior to state-of-the-art features. The difference between (Hackel et al., 2016b) multiscale features and ours may seem like an implementation detail, with radius neighborhoods instead of KNN. However, the results prove that the type of local neighborhood definition has a great impact on the robustness of the features, and that the spatial consistency of radius neighborhoods make them superior to KNN. We use these results to justify the choice of radius neighborhoods in our point convolution definition.

III.3.4. Learning strategy with a limited amount of points

In this section, we investigate the issue 3, and question the training and test strategies used in previous work. We design new “active” learning strategy that helps learning more details from a scene with a limited amount of training points. This strategy also highlights the flaws of previous experimental conditions.

a. A new active learning strategy

The choices we made for our learning strategy are based on the specificity of a pointwise classification problem. There are two setbacks when classifying a point cloud. First, its size is generally huge; second, the classes are heavily unbalanced. We choose the mean IoU as our metric, because it mitigates the distortion dued to the class imbalance. However, the training strategy also needs to be adapted to the problem. In the literature, the standard approach, that we used in Section III.3.3, consists of taking a subset of the training data, small enough to allow reasonable training times, and balance the classes in that subset. The advantage of this strategy is that it alleviates the class imbalance. Furthermore, the majority classes are often not very diversified.

Let us take the example of an urban scene. The largest class is the ground, and it rarely differs from a horizontal plane. The algorithm does not need a thousand samples of horizontal planes to understand that it is the shape of the ground.

Nevertheless, this strategy tends to create a lot of false positive for the minority classes. This phenomenon, which we will confirm quantitatively in Section III.3.5, can easily be understood in our example.

The training points of the ground class are chosen randomly, and the majority of ground points are on horizontal planes, so it is likely that the algorithm only sees horizontal plane examples for the ground class. During the test, the small parts of the ground that are not horizontal, like steps or curbs, will thus be misclassified. Although it is negligible compared to the majority of the ground points, these errors represent many points compared to the small classes. Therefore, these false positives will drastically reduce the precision of the small classes.

We need to help the algorithm to choose better training points. In our ground example, it should choose points in order to learn every different details of the ground class, even those in minority. We design a simple iterative trial and error procedure to solve this problem. A classifier is trained on a set of points $\mathcal{T}$ from the training clouds $\mathcal{U}$, then, the classifier is tested on $\mathcal{U}$, and we randomly add some of the misclassified points to $\mathcal{T}$. After some iterations, the classifier is used on the test scenes. This strategy, inspired from the theory of *active learning* (Settles, 2009), only consists of a smart choice of the training points.

Algorithm 1 describes the pseudo-code of this strategy. The crucial step in this code is the choice of the new training points among the errors, at line 14. Instead of a random picking among the errors, which would be biased by the class imbalance, we choose the same number of errors in each cell of the confusion matrix. Hence, the false positives and the false negatives of each class are reduced smoothly.

We compare this strategy to the random choice of the training points in the experiments of Section III.3.5. In addition to the better performances, we also show that active learning helps getting better precision for the minority classes, as explained above.

Algorithm 1: Active learning strategy for the training of point cloud classifiers.

```
Parameters
$n_0$: initial number of points per class
$n_{inc}$: number of points added at each step
$i_{max}$: number of iterations
Inputs
$\mathcal{P}$: points of the cloud
$C$: number of classes
$\mathcal{P}_c$ : points of the class $c$
Start
    Random pick of the first training points: $\mathcal{T}_0 = \bigcup_{c<C}\left\{(p_j)_{j<n_0} \middle| p_j \in \mathcal{P}_c\right\}$
    For $i = 0$ to $i_{max}$
        Train classifier on the current training points $\mathcal{T}_i$
        Test classifier on $\mathcal{U}_i = \mathcal{P} \setminus \mathcal{T}_i$
        Get the prediction errors $\mathcal{E}$.
        Pick new training points among errors: $\mathcal{T}_{i+1} = \mathcal{T}_i \cup \left\{(p_j)_{j<n_{inc}} \middle| p_j \in \mathcal{E}\right\}$
    End
End
```

b. Overfitting of the training scene

Our active learning strategy helps the classifier to choose a limited amount of points that will represent every details of the training scene. In the case of the small datasets used in Section III.3.3, it can be used to know how many points of a scene are needed to achieve a nearly perfect classification.

The results from the experiments Section III.3.3 are questionable as a measure of the classification performances, because the training and test points are picked in the same scene and the features are computed on neighborhoods. There is a considerable overlap between training and test features. As a proof of this behavior, we can use the active learning strategy to show that with the same amount of training points used in the experiments of Section III.3.3, we can achieve nearly perfect classification of the scene. On both datasets, we choose the following parameters:

$$n_0 = 100 \qquad n_{inc} = 300$$

$$i_{max} = 18$$

Using 18 iterations allows us to reach 6,000 points on Rue Madame (same as before) and 6,100 points on IQmulus & TerraMobilita (less than before). The rest of the classification pipeline remains the same. Table 4 shows the results in comparison with the scores obtained with random choice of training points. The active learning strategy shows that it is possible to find a set of points that allows a training error close to zero.

Table 4. Class IoU on the training scene of Rue Madame and IQmulus & TerraMobilita datasets, with our pointwise classification algorithm, using random point picking or active learning.

Rue Madame			**IQmulus & TerraMobilita**	
Random	Active		Random	Active
98,22%	**99.39%**	Façade	97,27%	**99.90%**
96,62%	**99.04%**	Ground	97,77%	**99.82%**
95,37%	**99.31%**	Cars	84,94%	**99.29%**
61,55%	**96.79%**	Motorcycles	58,99%	**97.74%**
67,43%	**99.93%**	Traffic Signs	12,71%	**99.93%**
77,86%	**99.99%**	Pedestrians	35,31%	**98.79%**
–	–	Vegetation	71,48%	**99.25%**
82,84 %	**99.08%**	**Mean**	65,50 %	**99.25%**

Even though this set of points has a very small probability to be picked by a random procedure, the scores obtained when picking training and test points in the same scene do not reflect the real classification performances of a method. To assess the real classification performances, we need bigger datasets to be able to make the difference between training set and test set.

However, in the experiments of Section III.3.3, we showed that the variations of the score with random picking are small. Provided that we compute the result several times and keep the average, they still may be used to compare the descriptive power of different features relatively. The validity of the comparison is ensured by the large number of trials.

III.3.5. Performances on large scale datasets

The recent appearance of bigger point cloud datasets allowed the separation of the training set and the test set. With such point clouds, it is possible to get a relevant measure of how well the classification generalizes to unseen data. In this section, we confirm that our active learning strategy helps the precision of the minority classes for better performances overall. With our best setup, we also address issue 4, and set a strong baseline that outperforms several deep learning strategies. We choose three large-scale datasets from different environments and acquisition methods for this experiment: Paris-Lille-3D, Semantic3D and S3DIS.

We divide this section in four parts. In the first one, we analyze the convergence of the scores per class on Paris-Lille-3D dataset, during the active learning process. In the second one, we will use the training/validation splits of two datasets to compare the performances of random and active learning strategies. Then we will evaluate our best setup on the test sets of the three different datasets. Eventually we briefly explain how our algorithm was used by Terra3D for industrial applications.

a. Convergence during active learning

In our first experiment, we want to understand the impact of active learning on the scores of different classes in an urban environment. We thus focus on Paris-Lille-3D dataset and use the scene called “Lille1_1” as our validation split. Because it is an outdoor dataset, we keep the parameters used in Section III.3.3, but changed the active learning point picking. We pick more points at each iteration as the dataset is bigger and has more classes. Furthermore, we iterate more to see how many epochs are needed for convergence:

$$n_0 = 200 \qquad\qquad n_{inc} = 1{,}800$$

$$i_{max} = 50$$

Figure 27 shows the convergence of the validation results with precision, recall and F_1 scores. Before any comparison, we can already note that the learning needs about 20 epochs to converge. In the following experiments, we will limit the number of epochs to save training time.

We can also see that, during the learning phase, the precisions rise (except for Vegetation) and some recalls (Signage, Trash cans, Bollards, Pedestrians) decrease. However, F_1 score increases for all classes. This trend confirms our claim that the smart picking of new points during active learning should overall help all classes. It is also not surprising that the precision of minority classes increase, as explained in the theoretical part of Section III.3.4.

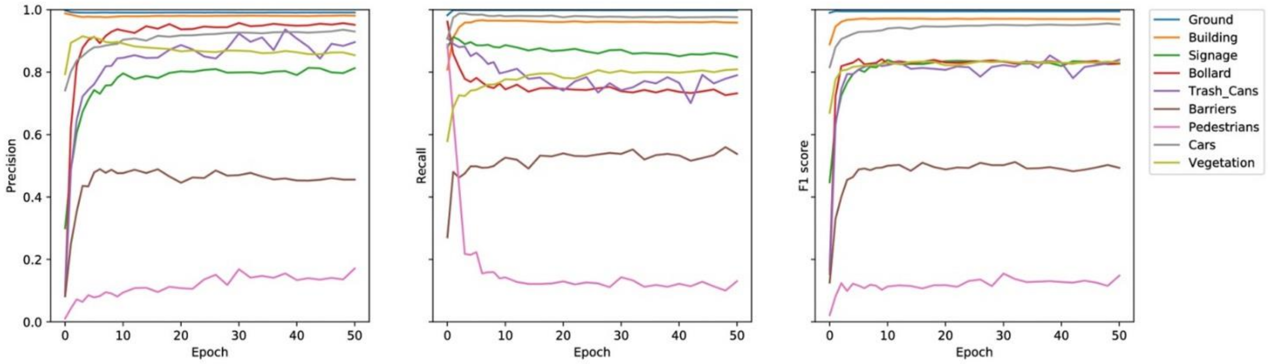


Figure 27. Precision, Recall and F1 score on scene Lille1_1 of Paris-Lille-3D dataset during active learning convergence.

b. Comparison between random and active learning

The results shown in Table 4 are not a valid comparison of the performances of random and active learning. They only represent the training errors of the two learning processes. It is common that a classifier with perfect training scores does not generalize well to the test scene. This phenomenon is called overfitting, and could happen with active learning. This is why it is important to compare the two learning processes on a test or validation set.

This experiment was conducted for an earlier version of our article (Thomas et al., 2018), and uses a different version of the S3DIS dataset. As we did not include color features at the time, we decided to reduce to seven coarser classes because from a purely geometric point of view, some classes were not distinguishable from the walls. Thus in the results of this experiment only, the classes Beam, Column, Window, Door and Board from the original dataset, were merged with the class Wall. The dataset Paris-Lille-3D was not changed and we did not use Semantic3D.

Therefore, we used a k-fold validation strategy on the training scenes of these two datasets. They consist of 4 and 6 scenes respectively, and each scene was considered as one fold. For Paris-Lille-3D, which is outdoors, we used the parameters used in Section III.3.3, while we chose the following parameters for S3DIS:

$$S = 8 \qquad r_0 = 0.05m$$

$$\varphi = 2 \qquad \rho = 5$$

We could adapt the parameters thanks to the geometrical meaning of our features. We reduced the radiuses of the neighborhoods as the scales of indoor objects and rooms are smaller than the scales of outdoor objects and streets.

For both datasets, we trained our random learning pipeline with 10,000 points per class and used an active learning strategy with the following parameters:

$$n_0 = 1{,}000 \qquad n_{inc} = 2{,}000$$

$$i_{max} = 20$$

In both cases, this corresponds to approximatively 50,000 training points, which is far less than the random strategy. The results shown in Table 5 and Table 6 confirm our hypothesis that *active learning* has better precisions and *random picking* has better recalls overall. They also show that in terms of F_1 score/IoU, active learning outperforms random learning for nearly every class.

We observe that the classes that contain a large amount of points and a lower diversity (Ground outdoors; Ceiling, Floor and Wall indoors), have better precisions with random learning and the better recall with active learning. It is the opposite for the minority classes with more diversity (Signage, Bollard, Trash Cans and Pedestrians outdoors; Sofa and Bookcase indoors), with active learning, they get better precisions at the price of a lower recall. For the other classes, active learning is better in terms of precision and recall. These findings are in accordance with the expectations we formulated in Section III.3.4.

We also provide a qualitative comparison in Figure 28, where the same conclusions appear. For example, in the indoor scene, we see many false positives of the sofa and bookcase classes (in orange and red respectively) with random learning. These false positives are greatly reduced in the active learning classification. In the outdoor scene, we see several barrier false positives in the buildings with random learning, and the actual barriers are not very well classified. With active learning, the barrier class seems much better classified, with fewer false positives. Eventually, another example illustrates the role of active learning in the outdoor vegetation class. If we focus on the tree trunks, we see that random learning does not manage to classify them as vegetation. This is not surprising as the vegetation points are in great majority in the leaves, which are more dense areas. The number of points in the trunks being small, they have a lower probability to be picked as training points with the random strategy. When using active learning, more points are picked in the tree trunks and their classification is better even if it is not perfect. To support this claim, we show the points that were chosen for the training of this particular experiment in Figure 29. We can indeed see more blue points in the tree trunks.

As a conclusion for this experiment, we showed the opposite behaviors of random and active learning in terms of precision and recall, and the superior performances of active learning. In the following, we favor the active learning strategy, which has better performances, to establish the strongest baseline.

Table 5. Validation results averaged on the 4 folds of Paris-Lille-3D dataset, with random and active learning.

	Precision		Recall		F_1 scores	
	Random	Active	Random	Active	Random	Active
Ground	**99.79%**	99.23%	98.09%	**99.80%**	98.93%	**99.52%**
Building	92.08%	**94.87%**	91.41%	**97.24%**	91.24%	**96.00%**
Signage	66.51%	**83.03%**	**75.57%**	72.59%	70.29%	**77.24%**
Bollard	16.57%	**84.52%**	**91.55%**	74.13%	26.70%	**76.94%**
Trash Cans	21.15%	**55.26%**	**48.97%**	27.67%	24.67%	**33.46%**
Barriers	31.51%	**62.95%**	49.34%	**52.34%**	28.67%	**52.25%**
Pedestrians	21.29%	**66.82%**	**58.99%**	12.41%	**21.42%**	18.03%
Cars	88.08%	**96.29%**	92.09%	**93.55%**	90.01%	**94.76%**
Vegetation	76.78%	**82.86%**	81.55%	**84.44%**	79.08%	**83.42%**
Mean	57.08%	**80.65%**	**76.40%**	68.24%	59.00%	**70.18%**

Table 6. Validation results averaged on the 6 folds of our modified S3DIS dataset, with random and active learning.

	Precision		Recall		F_1 scores	
	Random	Active	Random	Active	Random	Active
Ceiling	**97.07%**	95.86%	97.55%	**98.25%**	97.16%	**97.31%**
Floor	**98.31%**	97.33%	98.60%	**99.42%**	98.25%	**98.36%**
Wall (merged)	**93.24%**	92.36%	85.73%	**93.80%**	89.18%	**93.02%**
Table	**67.85%**	66.57%	62.14%	**73.78%**	64.46%	**69.14%**
Chair	77.70%	**79.13%**	68.02%	**82.64%**	68.92%	**80.36%**
Sofa	17.67%	**62.06%**	**64.65%**	24.38%	26.43%	**33.98%**
Bookcase	41.42%	**61.89%**	**55.66%**	47.89%	45.01%	**53.52%**
Mean	70.47%	**79.32%**	**76.05%**	74.31%	69.96%	**75.06%**

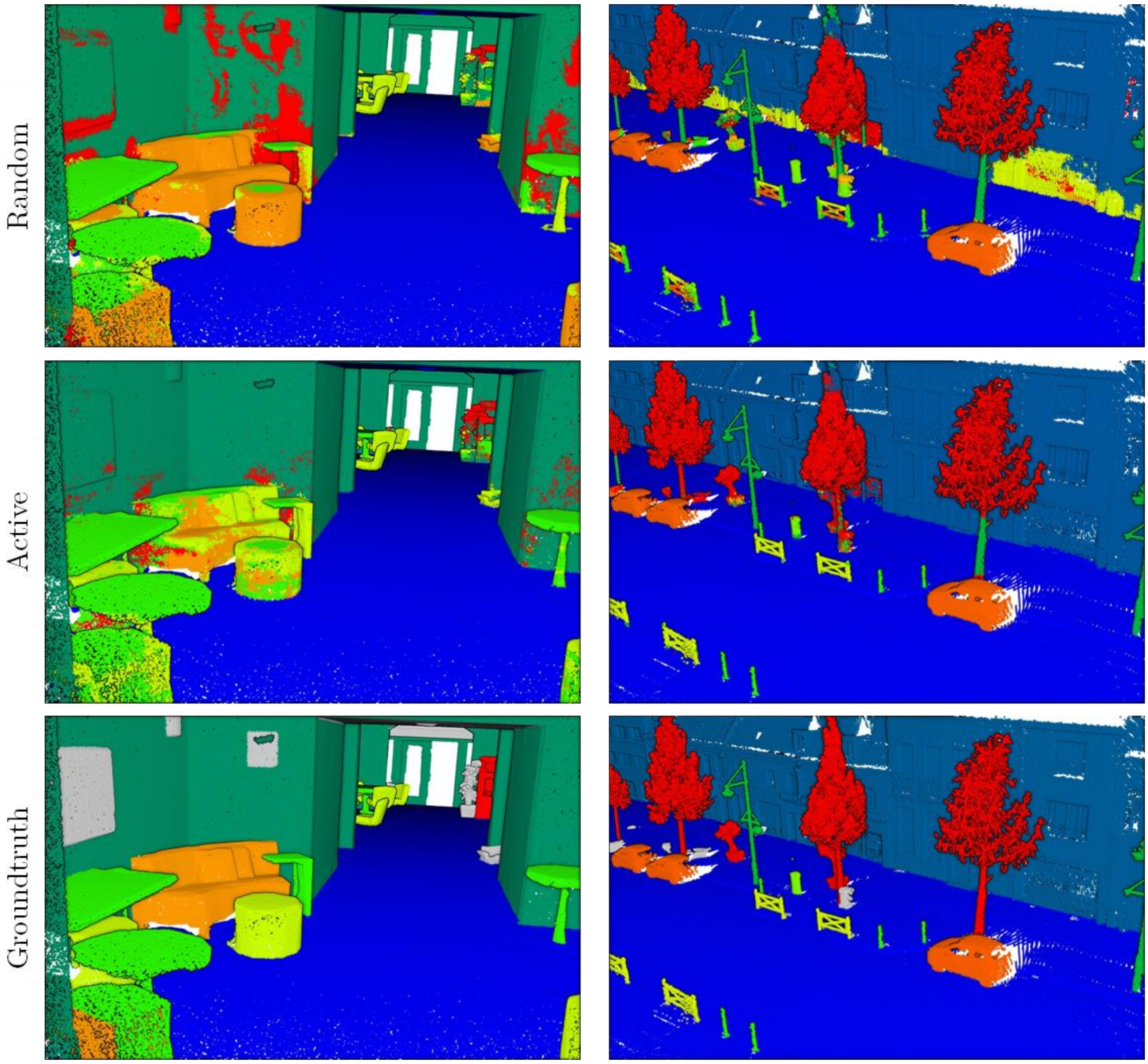


Figure 28. Qualitative comparison of random and active learning on an indoor scene (left) and an outdoor scene (right). Random strategy has better recalls (we see the sofa entirely detected), but active learning has better precisions (less false positives on the walls or facades). We can also see active learning is better at detecting tree trunks.

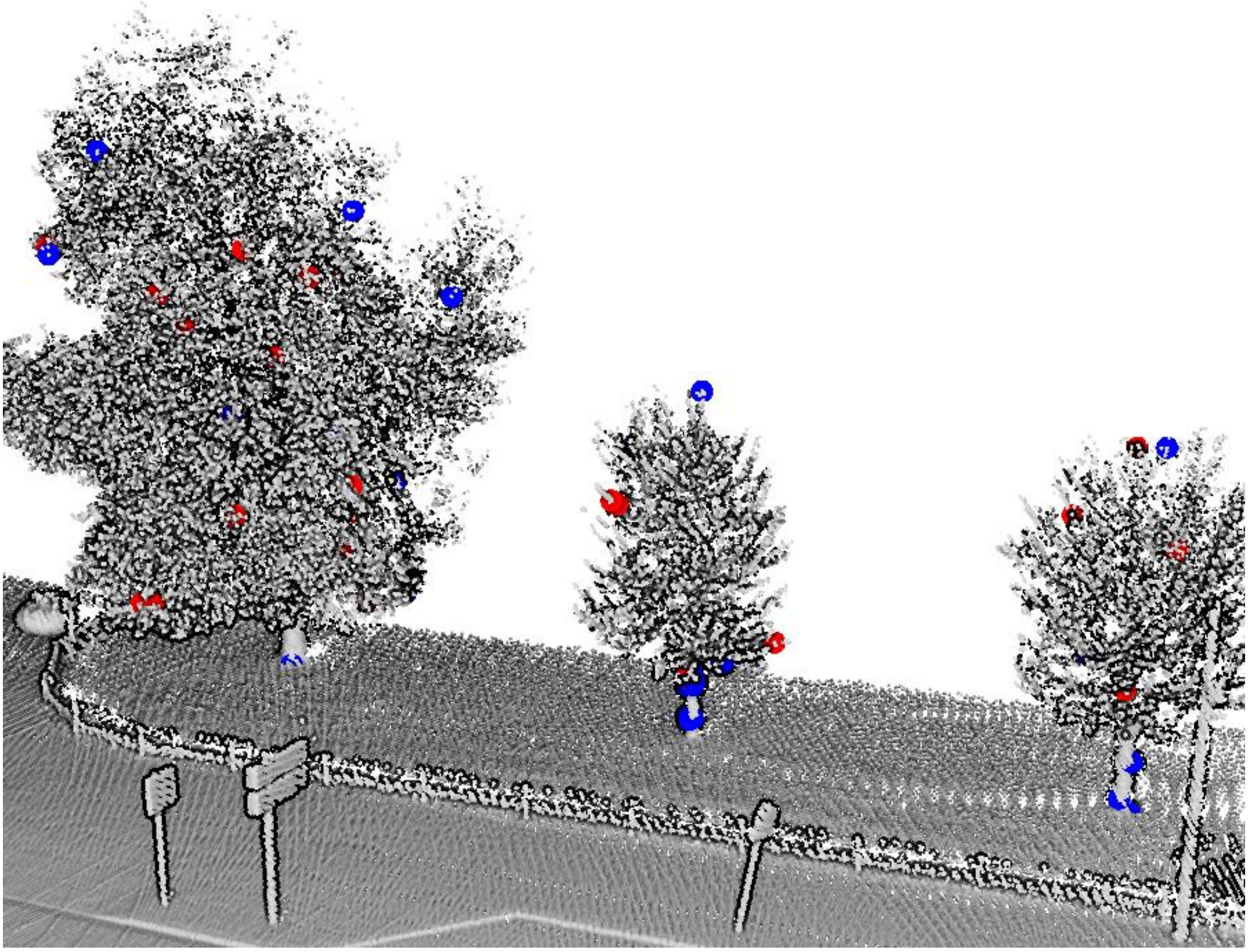


Figure 29. Comparison of the training points chosen with random learning (red) and active learning (blue). Only the points of the class vegetation are shown. Active learning is better at detecting tree trunks as it picks more training points in them.

c. Performances of our best classification pipeline

Now that we have found the best configuration for our classification pipeline, using multiscale spherical neighborhood and active learning strategy, with a random forest classifier, we show how it generalizes to multiple datasets.

We notice that some of these datasets include color as additional information. When possible, we will add the color features described in Table 7 at each scale, in order to compete with Deep Learning methods that use the colors.

Table 7. Features used for our comparative experiment.

Color Features	Definitions
Average color (× 3)	$\frac{1}{\lvert\mathcal{N}\rvert}\sum c$
Color variance (× 3)	$\frac{1}{\lvert\mathcal{N}\rvert - 1}\sum (c-\bar{c})^2$

We will test our pipeline on the three datasets listed above, and call it RF_MSSF for "Random Forest with Multi-Scale Spherical Features". On S3DIS dataset, we keep the same experimental conditions as (Tchapmi et al., 2017), and use the fifth area as the test set and train on the rest of the data. In this experiment, we use the 13 original classes for benchmarking purposes. We keep the indoor settings previously used:

$$S = 8 \qquad r_0 = 0.05m$$

$$\varphi = 2 \qquad \rho = 5$$

The two other datasets are online benchmarks with a separate test set. We choose the *reduced-8* challenge on Semantic3D. As they are both outdoor datasets, we use the previous outdoor settings:

$$S = 8 \qquad r_0 = 0.1m$$

$$\varphi = 2 \qquad \rho = 5$$

The training on each dataset is conducted with the following active learning settings:

$$n_0 = 1{,}000 \qquad n_{inc} = 2{,}000$$

$$i_{max} = 20$$

The results are compiled in Table 8, Table 9, and Table 10, against competing methods at the time of the submission of our article. Despite its simplicity, our classification pipeline ranked second on S3DIS and Semantic3D. At this time, only a few 3D semantic segmentation methods were competing, but among them, we outperformed, *PointNet* (Qi et al., 2017a), a groundbreaking deep learning architecture for point clouds, and *SEGCloud* (Tchapmi et al., 2017), an elaborate 3D segmentation pipeline using 3D voxel CNN. Even if the cutting edge algorithm of (Landrieu and Simonovsky, 2017) outperforms our method by a large margin, this is still a serious baseline that shows that the early deep learning works in 3D semantic segmentation are not at the level of their image counterpart.

We will see in the Chapter IV that many methods, including our own deep learning pipeline, have been published since then, and outperform our 3D semantic segmentation baseline. This is not surprising, as our method was not designed to remain at the top of the ranking, it was meant to establish where handcrafted features stand against deep learning methods.

Table 8. Semantic segmentation IoU scores on S3DIS Area-5.

Method	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	mean
PointNet (Qi et al., 2017a)	88.8	**97.3**	69.8	**0.1**	3.9	46.3	10.8	52.6	58.9	40.3	5.9	**26.4**	41.7
SEGCloud (Tchapmi et al., 2017)	90.1	96.1	69.9	0.0	18.4	38.4	23.1	75.9	70.4	58.4	40.9	13.0	49.5
SPGraph (Landrieu and Simonovsky, 2017)	89.4	96.9	**78.1**	0.0	**42.8**	**48.9**	**61.6**	**84.7**	**75.4**	**69.8**	**52.6**	2.1	**58.5**
RF_MSSF (ours)	**95.9**	96.4	67.6	0.0	11.9	48.3	28.8	64.4	68.9	58.6	33.9	22.3	49.8

Table 9. Semantic segmentation IoU scores on Semantic3D reduced-8 challenge.

Method	man-made terrain	natural terrain	high vegetation	low vegetation	buildings	hard scape	scanning artefacts	cars	mean
TMLC-MSR (Hackel et al., 2016b)	89.8	74.5	53.7	26.8	88.8	18.9	36.4	44.7	54.2
DeePr3SS (Lawin et al., 2017)	85.6	83.2	74.2	32.4	89.7	18.5	25.1	59.2	58.5
SnapNet (Boulch et al., 2017)	82.0	77.3	79.7	22.9	91.1	18.4	37.3	64.4	59.1
SEGCloud (Tchapmi et al., 2017)	83.9	66.0	86.0	40.5	91.1	30.9	27.5	64.3	61.3
SPGraph (Landrieu and Simonovsky, 2017)	**97.4**	**92.6**	**87.9**	**44.0**	**93.2**	**31.0**	**63.5**	**76.2**	**73.2**
RF_MSSF (ours)	87.6	80.3	81.8	36.4	92.2	24.1	42.6	56.6	62.7

Table 10. Semantic segmentation IoU scores on NPM3D Paris-Lille-3D challenge.

Method	Ground	Building	Pole	Bollard	Trash can	Barrier	Pedestrian	Car	Natural	Mean
RF_MSSF (ours)	99.3	88.6	47.8	67.3	2.3	27.1	20.6	74.8	78.8	56.3

We can notice that the method of (Hackel et al., 2016b), using multiscale KNN competes in the Semantic3D benchmark. As expected, our results exceed theirs by a large margin. It is not only due to the use of spherical neighborhoods, but also to the active learning strategy.

Figure 30, Figure 31 and Figure 32 show some examples of classified scenes. First, we can notice that the classification has no object coherence as some unstructured patches appear, for example on the columns in Figure 31 or on the facades in Figure 32. This highlights the particularity of our method to focus on points independently, not using any segmentation scheme. Another very interesting pattern appears on the second scene in Figure 32: when a car is close to a tree, it is misclassified and we can actually see the influence area of the tree on the car. We can assume that the classifier relies more on the large scales to distinguish those two particular classes.

Overall, our classification algorithm ranked among the best approaches at the time of its submission, beating nearly every other elaborate method apart from Super-point Graphs (Landrieu and Simonovsky, 2017) on three large scale datasets in different environments. However, this has to be considered in light of the fact that we did not use any segmentation or regularization process and only focus on the descriptive power of our features. We proved that our features beat state-of-the-art features in terms of classification performances, and that they could, alone, compete with complex classification schemes, including deep learning methods.

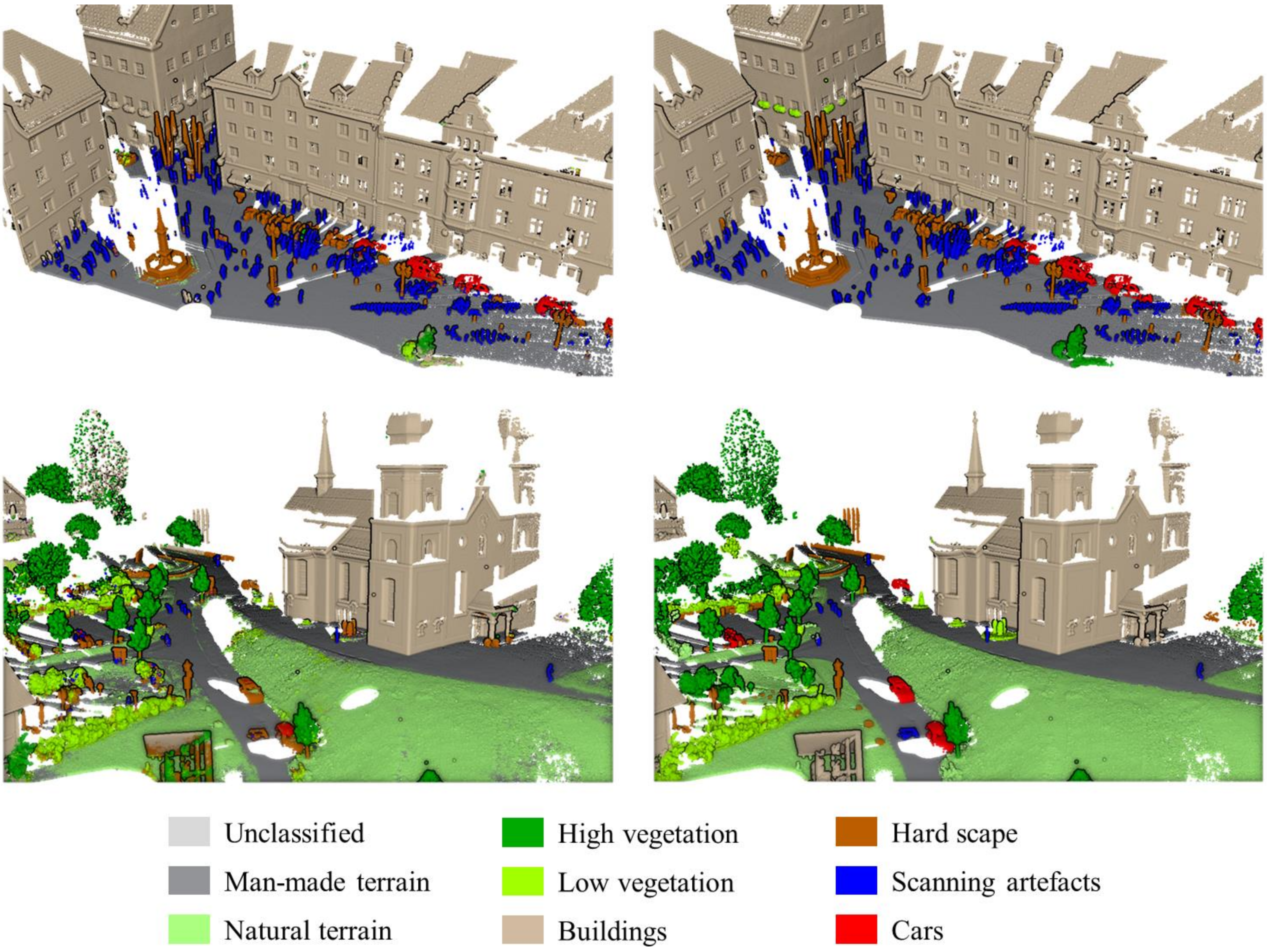


Figure 30. Pointwise classification pipeline results on Semantic3D dataset (left) with groundtruth (right).

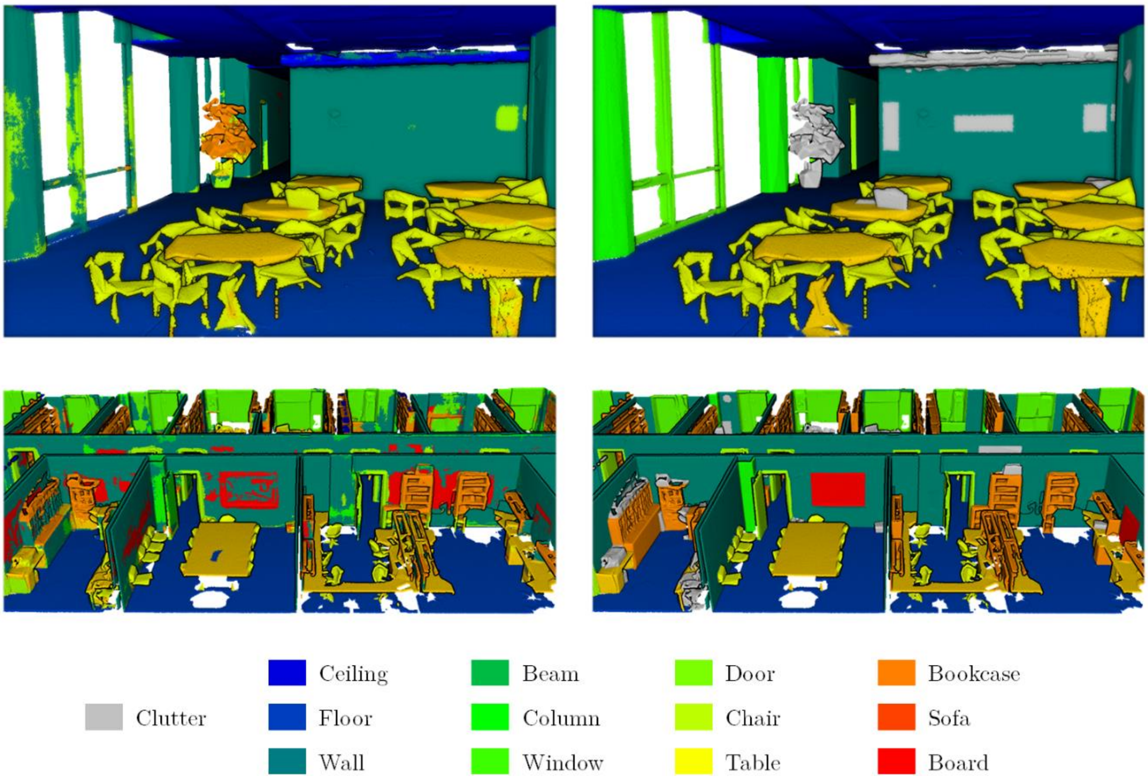

Clutter
Ceiling
Floor
Wall
Beam
Column
Window
Door
Chair
Table
Bookcase
Sofa
Board

Figure 31. Pointwise classification pipeline results on S3DIS dataset (left) with groundtruth (right).

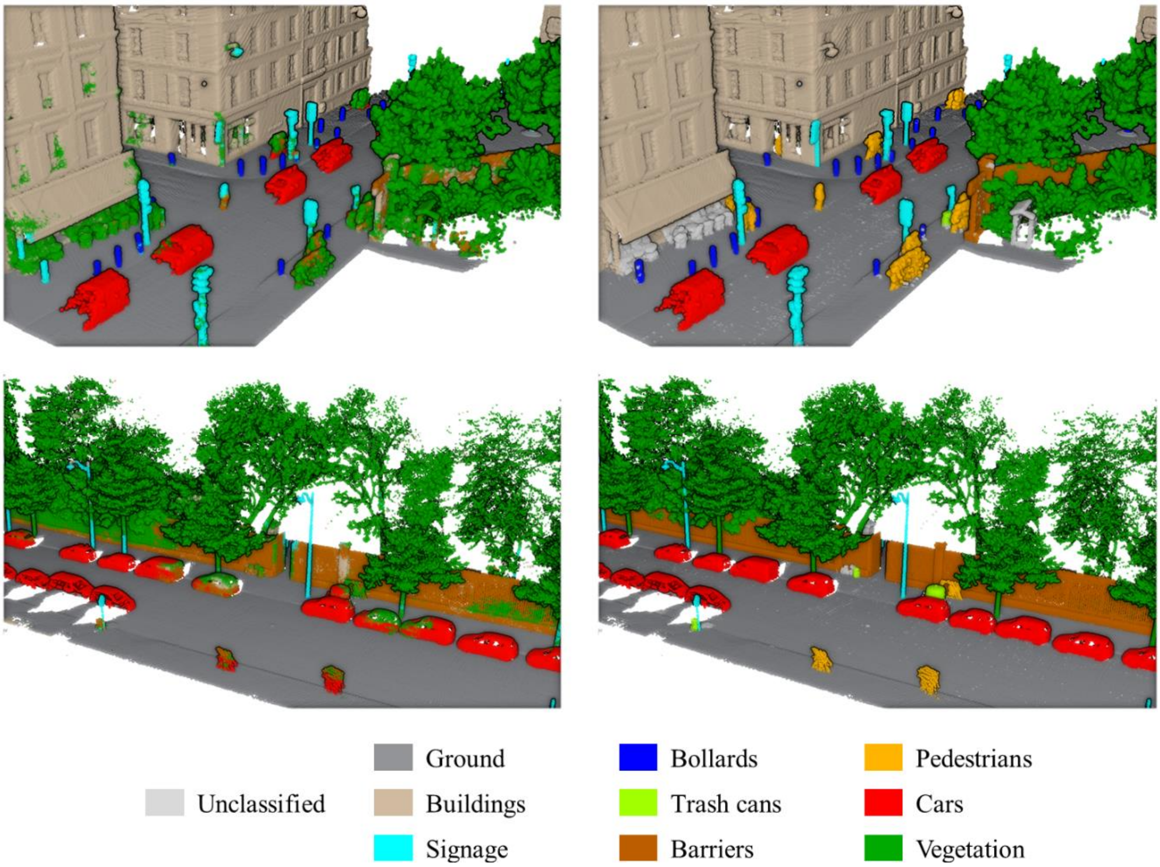

Unclassified
Ground
Buildings
Signage
Bollards
Trash cans
Barriers
Pedestrians
Cars
Vegetation

Figure 32. Pointwise classification pipeline results on Paris-Lille-3D dataset (left) with groundtruth (right).

d. Industrial applications

The very strong performances of the multiscale features in our classification pipeline were not only proved on public datasets for academic applications. They were also tested on industrial applications by Terra3D, the start-up financing this thesis. Terra3D eventually adopted our multiscale features as part of their point cloud segmentation pipeline. In this paragraph, we discuss the differences between academic and industrial concerns that are illustrated by the usage of our multiscale features.

In the previous paragraphs, we insisted on metrics like precision and recall. These metrics are used in academic research as they offer a way to understand the behavior of different algorithms, in addition to measuring their performances. Although we offer a qualitative analysis of our method, quantitative results generally prevail in academic works as they offer an impartial comparison of performances. As a result, we provided several numbers of precision, recall and mIoU for every class on different datasets.

However, these numbers do not really matter in industrial applications. The qualitative analyses are more important for client-oriented solutions as they represent what is visible in the results. For example, if you consider the tree trunk mistakes in Figure 28, they could be critical in urban planning applications, where the exact position of the trunk is needed, and the leaves do not really matter. Although, our algorithm has good performances in term of vegetation mIoU, it would be useless in that scenario. The notion of critical mistakes is crucial in industrial applications and drives the design of algorithms. In our case, the multiscale features are very descriptive, and valuable in a 3D processing algorithm, but the pointwise pipeline is not adapted to avoid critical mistakes. Therefore, the multiscale features were adapted to fit in an object segmentation algorithm that has theoretically worse performances than our pointwise pipeline, but that makes less critical mistakes. Furthermore, it is not used alone but in combination with other methods, to enhance the robustness of the overall segmentation.

If the results were only quantitative, there would be no way to measure the impact of our multiscale features on Terra3D's segmentation pipeline. However, industrial applications also have their own metrics to measure the efficiency of semantic segmentation algorithms. The most meaningful metric in a client-oriented solution is the fixing time. It represents the time for the client to correct the critical mistakes in the segmentation provided by the algorithm. Ideally, this time should be equal to zero, but in practice, algorithms always make mistakes at one point or another, especially for large-scale applications. In the case of our multiscale features, Terra3D noticed a significant reduction of the fixing time after their introduction into the segmentation pipeline.

III.3.6. Study of the influence of density in our multiscale neighborhood

The last remaining issue 2 has not been addressed yet. We have defined the parameter ρ for our multiscale neighborhoods, which controls the number of subsampled points that a neighborhood can contain. With this parameter, the density of points at each scale is fixed proportionally to the radius. We now need to study the effect of this parameter, and in particular, to find how many neighbors are sufficient to get expressive features. In this experiment, we choose to use the Paris-Lille-3D dataset for two reasons. First, we want to focus on the 3D descriptors and, thus, do not need color information. Then, the results generalize well because they are cross-city, tested on Paris after being trained on Lille. With the parameters previously used on this dataset, we compute average IoU scores across all classes for different values of ρ. Figure 33 shows the evolution of the results along with the features computation speed for different values of ρ. We can note that average IoU scores rise quickly up to $\rho = 3$ and do not increase a lot for higher values of ρ. Depending on the application, one can choose to optimize the results or the processing speed with this parameter. Although our performances could be slightly increased with a higher ρ value, we choose to keep $\rho = 5$ in our work because it is a trade-off between performance and computation speed.

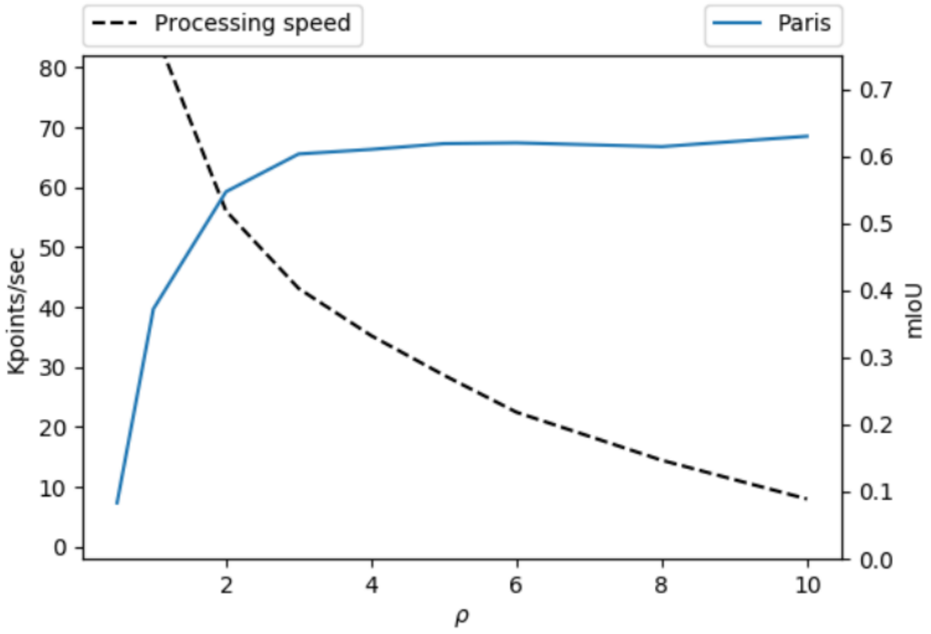


Figure 33. Influence of the parameter ρ on the classification performances and the computation speed on folds of Paris-Lille-3D dataset.

Concerning the design of a CNN architecture however, the value of ρ should not be chosen in the same way. Indeed, ρ controls the number of input points that a convolution kernel will see. In image convolution, this is the effect of the kernel size, and (Simonyan and Zisserman, 2014) showed that using smaller 3×3 kernels, with more layers, lead to significant improvement of the performances. This is why we will reduce the value of ρ in our convolution design.

III.3.7. Conclusion for the design of neural network representations

In the previous sections, we have collected several elements and results that answer the unresolved issues of the state of the art, and allow us to design a deep learning architecture.

First, we have a neighborhood definition. Our multiscale spherical neighborhoods proved to be superior to a multiscale KNN, and it can easily be adapted to a neural network design. The multiple scales will correspond to the successive layers, it is very fast, and the density of points can be controlled with a single parameter across the network. The drawback of spherical neighborhoods is that they contain a varying number of points, which is not convenient when computing parallel operations on a GPU. This is why many works still prefer KNN in their deep learning frameworks. Hoverer, with similar arguments as in our analysis of Figure 26, (Hermosilla et al., 2018) also highlights that KNN are not robust in non-uniform sampling settings. We keep our radius neighborhoods for their geometrical consistency and find implementation tricks to deal with the varying number of neighbors.

We also have a subsampling scheme. We will keep a grid subsampling as it offers a very simple way to control the density at each scale with parameter ρ. Compared to other subsampling methods described in Section II.4, it has the advantages of being very fast and spatially consistent, and thus, pairs well with spherical neighborhoods.

Concerning the datasets, new benchmarks have been published in the recent years. Their large-scale point clouds allow the training of deep neural networks. We also have a good idea of the capabilities of handcrafted features on such datasets, and can compare our deep learning network to this baseline.

Eventually, we still need a convolutional operator on point cloud. This is the most important element of our deep learning framework, defining how it will behave and create local learnable representations. In the next chapter, we will define our own convolution operator for point cloud, using the knowledge we gathered on handcrafted features in this chapter.

Chapter IV

IV. Deep learning architectures for point cloud processing

Abstract

The study of handcrafted representations in the previous chapter, offers strong foundations for the design of learnable representations in the form of a new convolutional operator for point clouds. In this chapter, we introduce the Kernel Point Convolution (KPConv), which uses radius neighborhoods and a set of kernel points to play the role of the kernel pixels in image convolution. From this operator, we build convolutional networks that outperform state-of-the-art semantic segmentation approaches in almost any situation. In addition to these strong results, we designed KPConv with a great flexibility and a deformable version. To conclude our argumentation, we propose several insights on the representations that our method is able to learn.

French Abstract, résumé français

L'étude des représentations « à la main » du chapitre précédent offre des bases solides pour la conception de représentations pouvant être apprises sous la forme d'un nouvel opérateur de convolution pour les nuages de points.

Nous commençons donc ce chapitre par une revue des différentes méthodes de traitement de nuage de point utilisant de l'apprentissage profond. Nous pouvons classer ces approches en quatre groupes. D'abord, les réseaux de neurones utilisant une projection. Ces réseaux transforment donc, dans un premier temps, les nuages de points en images, grilles 3D, ou autres structures régulières. Ainsi ils sont en mesure de reproduire facilement les schémas qui fonctionnent en traitement d'image, sans s'adapter réellement à la nature des nuages de points. Viennent ensuite les réseaux sur graphes, qui comme leur nom l'indique, profitent de la structure de graphe (nuage de point relié par des arrêtes) pour définir des convolutions. La grande différence avec les réseaux à point vient du fait qu'ils définissent le plus souvent une convolution par les arrêtes, et se concentre donc sur l'information topologique, plutôt que sur l'information géométrique de la forme. Puis nous distinguons les réseaux de neurones point par point, qui utilise des *Multi-Layer Perceptrons* (MLP), réseaux de neurones complètement connectés, pour encoder la forme du nuage. Ces réseaux, reprenant l'idée présentée par PointNet (Qi et al., 2017a), se base sur le théorème d'approximation universelle, stipulant qu'un MLP peut approximer n'importe quelle fonction continue de l'espace. Enfin, le groupe le plus intéressant pour nous est celui des convolutions sur nuage de points. Ces méthodes définissent des noyaux de convolution directement sur les points sans transformations intermédiaires. Nous montrons que parmi toutes ces méthodes, aucune ne fournit une définition de convolution entièrement satisfaisante à nos yeux. C'est pourquoi nous allons proposer notre propre definition.

Nous présentons donc la « Kernel Point Convolution » (KPConv), qui utilise des voisinages sphériques et un noyau défini par des points qui jouent le rôle de pixels du noyau de la convolution d'image. Cette définition se base à la fois sur nos conclusions de la partie précédentes et sur les intuitions derrière la convolution en image. À partir de cet opérateur extrêmement flexible, nous construisons des réseaux de neurones convolutifs profonds, pour la segmentation et la classification de nuages de points. De plus, nous étendons le concept de convolution déformable à KPConv, avec une nouvelle forme de régularisation nécessaire pour le faire fonctionner sur des nuages de points.

Plusieurs expériences viennent confirmer la supériorité de KPConv par rapport aux autres convolution sur points de l'état de l'art, que ce soit en classification d'objets, segmentation de parties d'objets, ou segmentation sémantique de scènes. Nos réseaux se classent premier dans quasiment tous les benchmarks sur lesquels ils ont été testés.

En plus de ces résultats probants, nous proposons plusieurs expériences permettant de mieux comprendre les représentations que notre méthode est capable d'apprendre, ainsi que les spécificités de la version déformable. Ces expériences incluent, une comparaison des temps de calculs avec d'autres méthodes similaires, une étude de l'influence du nombre de points dans le noyau, la visualisation des représentations apprises à différents niveaux de nos réseaux, ainsi qu'une évaluation de champs réceptif effectif de nos convolutions.

IV.1. Overview of 3D deep learning approaches

In this section, we review several 3D deep learning approaches, and in particular, the methods processing point clouds. At the beginning of this thesis, in October 2016, hardly any paper on deep learning and point cloud had been published. It was a nearly empty field, but very promising, with the beginning of 3D deep architectures like VoxNet (Maturana and Scherer, 2015). Since then, hundreds of papers have been published on the subject presenting new types of architectures, new convolution definitions, new tasks solved with deep learning, etc. Here are some examples of the variety of tasks solved with deep learning methods:

- Scene flow estimation (Gu et al., 2019; Liu et al., 2018)
- Cloud registration (Elbaz et al., 2017; Wang and Solomon, 2019)
- Normal estimation (Boulch and Marlet, 2016; Qi et al., 2017a)
- Curvature estimation (Guerrero et al., 2018)
- Primitive fitting (L. Li et al., 2018)
- Denoising (Duan et al., 2019; Rakotosaona et al., 2019)
- Contour extraction and consolidation (Roveri et al., 2018; Yu et al., 2018)

Our work was conducted in parallel with this surge of new ideas, and our own deep learning architecture for point cloud arrived a bit late in this crowded field. However, it stands out for several reasons explained in this third chapter. In this section, we focus on previous works, detail the most remarkable methods, and expose their limitations.

Various deep learning approaches have been proposed to handle point clouds, and can be grouped into different categories. Several methods, following *VoxNet* (Maturana and Scherer, 2015), fall into the grid-based or "projective" category, whose principle is to project the sparse 3D data on a regular structure where a convolution operation can be defined more easily. Some works, after (Bruna et al., 2014; Henaff et al., 2015) tried to define convolutions on less regular structures: graphs. Other approaches use *multi-layer perceptrons* (MLPs) to process point clouds directly, following the idea proposed by (Qi et al., 2017a). More recently, some attempts have been made to design a convolution that operates directly on 3D points. These methods use the spatial localization property of a point cloud to define point convolutions with spatial kernels. They share the idea that a convolution should define a set of customizable spatial filters applied locally in the point cloud.

IV.1.1. Projective networks

Several methods project points to an intermediate grid structure. This is a straightforward technique, as convolution is well defined for images, which are 2D grids. The simplest form of projection thus consists of using images of the point cloud. Image-based networks are often multi-view, using a set of 2D images rendered from the point cloud at different viewpoints (Qi et al., 2016; Su et al., 2015). As shown in Figure 34, this kind of approach works well for isolated object models, as the viewpoints can be generated easily.

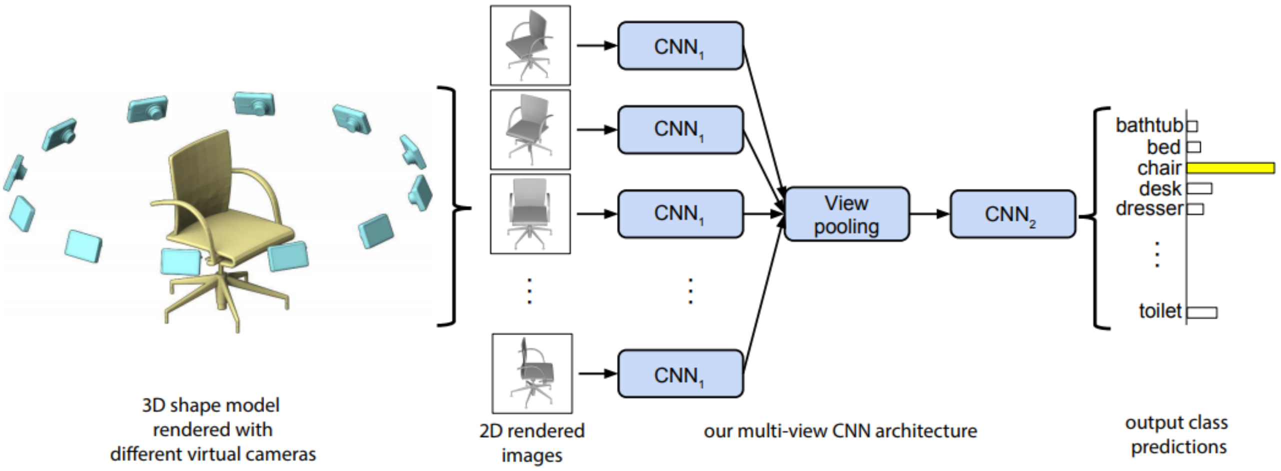

CNN$_1$
View pooling
CNN$_2$
bathtub
bed
chair
desk
dresser
toilet
3D shape model rendered with different virtual cameras
2D rendered images
our multi-view CNN architecture
output class predictions

Figure 34. Semantic classification of 3D objects with a multi-view CNN. Image from (Su et al., 2015).

From this simple principle, a few point cloud semantic segmentation pipelines have been designed, using images generated from several view points in the scene (Boulch et al., 2017; Lawin et al., 2017). These methods heavily rely on the initial and final steps shown in Figure 35, respectively, the viewpoint selection, and the segmentation back-projection. These steps are not very robust and the whole pipeline thus suffers from occluded surfaces and density variations.

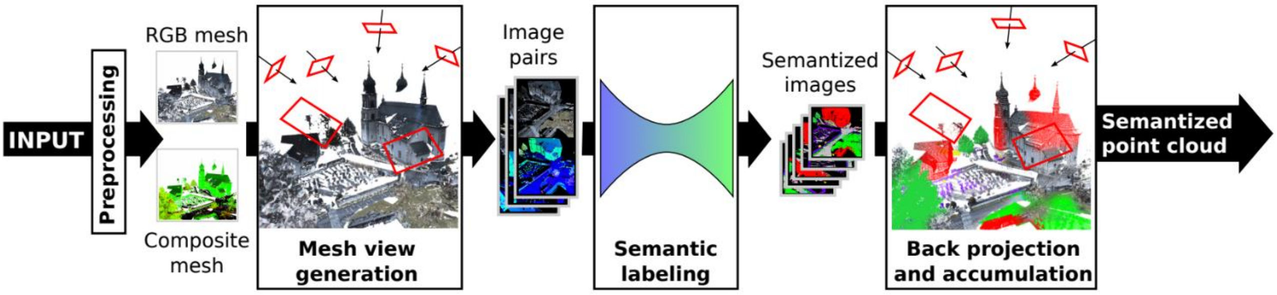

INPUT
Preprocessing
RGB mesh
Composite mesh
Mesh view generation
Image pairs
Semantic labeling
Semantized images
Back projection and accumulation
Semantized point cloud

Figure 35. Semantic segmentation of 3D scenes with a multi-view CNN. Image from *SnapNet* (Boulch et al., 2017).

Instead of choosing global projection viewpoints, (Tatarchenko et al., 2018) proposed projecting local neighborhoods to local tangent planes and processing them with 2D convolutions. However, this method relies heavily on tangent estimation.

To remain closer to the nature of 3D point clouds, a second form of projection, on 3D grids, is preferable. The definition of image convolutions is not limited to two dimensions and can easily be extended to three dimensions, to design voxel-based networks (Maturana and Scherer, 2015; Wu et al., 2015). An example of 3D voxel architecture is represented in Figure 36. We can notice that the size of the input grid is quite small, with only 32^3 voxels. This is a common issue for voxel networks, as the complexity

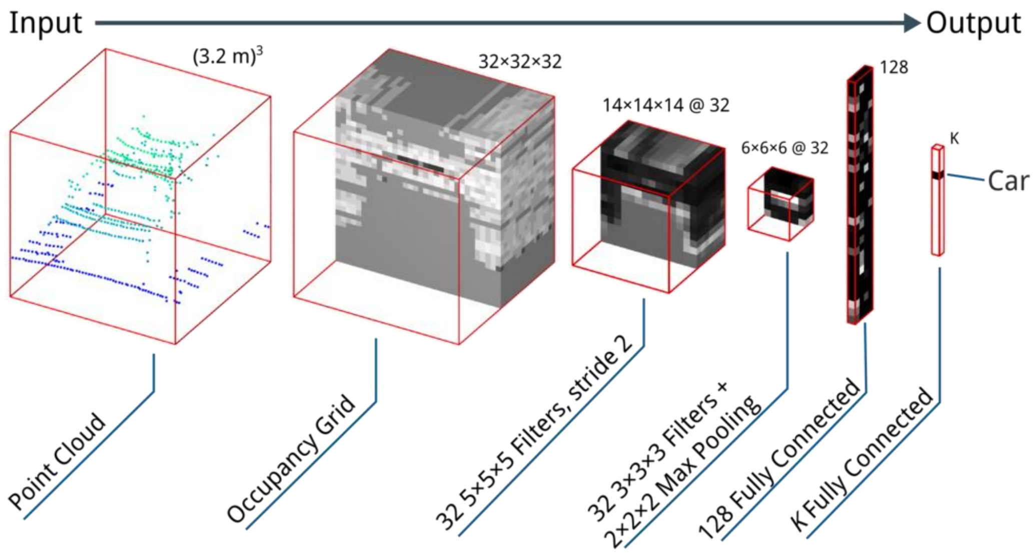

Input
Output
(3.2 m)^3
32×32×32
14×14×14 @ 32
6×6×6 @ 32
128
K
Car
Point Cloud
Occupancy Grid
32 5×5×5 Filters, stride 2
32 3×3×3 Filters +
2×2×2 Max Pooling
128 Fully Connected
K Fully Connected

Figure 36. Voxel projective network. The point cloud is projected in a 3D occupancy grid, and then processed by standard 3D convolution. Image from (Maturana and Scherer, 2015).

becomes cubic with the size of the input, instead of quadratic. This is a problem for the memory consumption of these methods, which is limited by the capacities of GPUs.

Two types of solutions have been proposed in the literature to deal with the memory issue of voxel networks. First, one can circumvent the problem and design algorithms that do not need a network with larger dimension. For example, (Ben-Shabat et al., 2018) first converts the input point cloud to a *Fisher vector* representation on a grid, and then use a voxel architecture on this grid. We can consider that the first representations replace the first layers of the convolutional architecture, but they are not learnable representations, which limits the descriptive capabilities of the algorithm. Instead of hand-crafted features, (Zhou and Tuzel, 2018) proposed to use neural network encodings as input in each voxel as shown in Figure 37. In this hybrid pipeline, combining voxel networks and MLP architectures presented in Section IV.1.3, the initial representations are learnable. However, they do not consist of convolutional

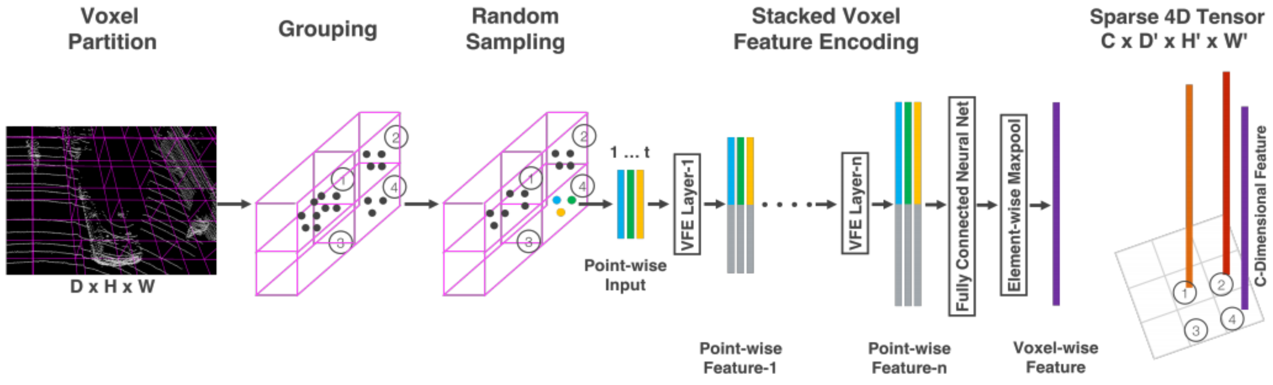

Voxel Partition
Grouping
Random Sampling
Stacked Voxel Feature Encoding
Sparse 4D Tensor
C x D' x H' x W'
D x H x W
1
2
3
4
1 ... t
Point-wise Input
VFE Layer-1
VFE Layer-n
Fully Connected Neural Net
Element-wise Maxpool
Point-wise Feature-1
Point-wise Feature-n
Voxel-wise Feature
C-Dimensional Feature

Figure 37. *VoxelNet* encoding steps (Zhou and Tuzel, 2018). The points within each voxels are transformed to a learnable vector representation characterizing the shape information.

layers, and thus, depart from the type of representation that we seek. The success of image convolutions, and the conclusions of our study on handcrafted methods, suggest that the first layers should aggregate the results of local learnable spatial filters.

With the same idea of circumventing the problem, (Roynard et al., 2018b) proposed a different strategy, with multiscale voxel networks. If voxel network dimensions are limited, using them on multiple scales of input is a way to add the information of larger scale inputs. The weakness of this method is that the backbone remains a small dimension network. Even though the large scales bring contextual information while the small scales bring details, the representation learned by the network will not be as elaborate as with larger and deeper networks.

The other strategy thus aims at enlarging the network, with the help of sparse structures. The main assumption of these methods is that point clouds are sparse. Therefore, in a grid, only a minority of voxels contain points and should thus be concerned by the convolution operations. Among the possible sparse structures that can encode grid data, octrees were the first to be proposed (Riegler et al., 2017; Wang et al., 2017). Thanks to this structure, input grids of 256^3 voxels can be processed on modern GPUs. In the same manner (Graham et al., 2018) proposed to use hash maps to limit the memory consumption of the grid and the locations where convolutions are computed. This ability to use larger dimensions and to force the convolutions to happen only where points exist, gives better capabilities to the networks, and they outperform their dense counterparts significantly. However, the complexity of the convolution operation remains fixed with the kernel size of $3^3 = 27$ parameters, and the nature of data is changed, from a continuous point cloud, to a discrete grid.

These two issues were addressed by (Su et al., 2018), with their *SplatNet* architecture. First, they use a permutohedral lattice instead of a Euclidean grid to reduce the kernel to 15 lattices instead of 27 voxels. Then, they use a splatting/slicing process described in Figure 38, which projects the features back and forth from the points, which really carry the data, to the grid structure, where the convolutions happen. In the end, the results are held by points and not grid voxels. However, this definition is still limited by the grid structure that holds the convolution operation, and lacks flexibility. Even though it is smaller, the number of parameters in the convolution kernels is still constrained. Our design allows any number of kernel points. Moreover, avoiding intermediate structures should make the design of more complex architectures, like instance mask detector or generative models, more straightforward in future works.

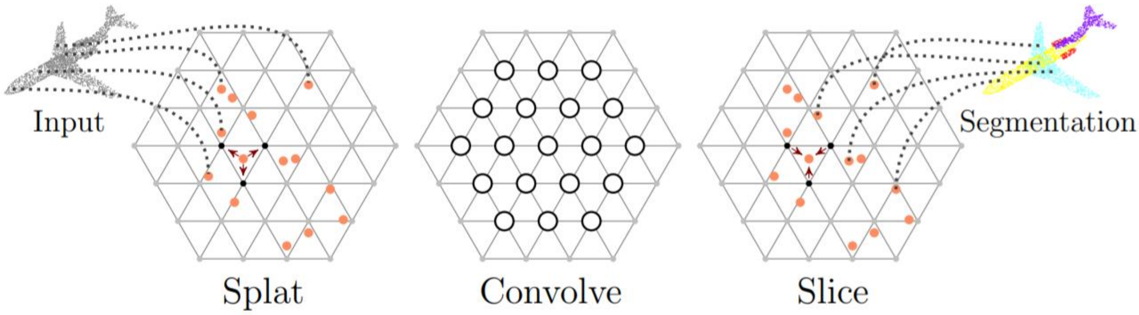


Figure 38. Bilateral convolution layer, with splatting/slicing process from *SplatNet* architecture (Su et al., 2018).

IV.1.2. Graph convolution networks

Two types of graph convolution operators have been proposed, the spectral-based and the spatial-based. Following the theories of graph signal processing, spectral-based approaches define graph convolutions by introducing filters on the spectral representation of the graph. Spatial-based approaches focus on the surface represented by the graph and aggregate features from neighboring edges.

a. Spectral graph convolution

Spectral-based convolutions are based on the property of the Fourier transform, which converts convolution to multiplication. As explained thoroughly by (Simonovsky, 2018), there is a generalized notion of Laplacian operator on a graph or a Riemannian manifold, which makes it possible to define a graph convolution in its Fourier domain. Let F and g be two functions on the graph nodes, then:

$$F * g = \mathcal{F}^{-1}\big(\mathcal{F}(F) \odot \mathcal{F}(g)\big) \tag{IV-1}$$

Where $\odot$ is the element-wise matrix multiplication (Hadamard product), $*$ the convolution operator, and $\mathcal{F}$ the Fourier transform. If we call U the matrix of the eigenvectors of the Laplacian matrix of the graph, then the graph Fourier transform of a signal x on the graph nodes is:

$$\hat{x} = \mathcal{F}(x) = U^T x \tag{IV-2}$$

And the inverse graph Fourier transform is defined as:

$$x = \mathcal{F}^{-1}(\hat{x}) = U\hat{x} \tag{IV-3}$$

Therefore, a graph convolution operator can be defined as:

$$F * g = U(U^T F \odot U^T g) \tag{IV-4}$$

We can see that this operator is parametrized by the choice of $U^T g$, the kernel function.

In their pioneering work, (Bruna et al., 2014) proposed the first graph convolution operator by simplifying the choice of $U^T g$ to a diagonal matrix Θ. With this simplification, the formulation of the convolution becomes:

$$F * g = U\Theta U^T F \tag{IV-5}$$

They chose B-splines for their spectral filters Θ, which are smooth functions, and therefore correspond to spatially compact kernels.

(Defferrard et al., 2016) later proposed to define the filters as Chebyshev polynomials of the diagonal matrix of eigenvalues. *ChebNet* and its first order approximation *1stChebNet*, proposed by (Kipf and Welling, 2016), reduce the complexity of filtering, because they avoid the computation of the graph Fourier basis.

Because they rely on the graph Laplacian, these spectral methods may not generalize to multiple graph structures and shapes. Any perturbation to a graph results in a change of eigenbasis, and, the learned filters are domain dependent, meaning they cannot be applied to a graph with a different structure.

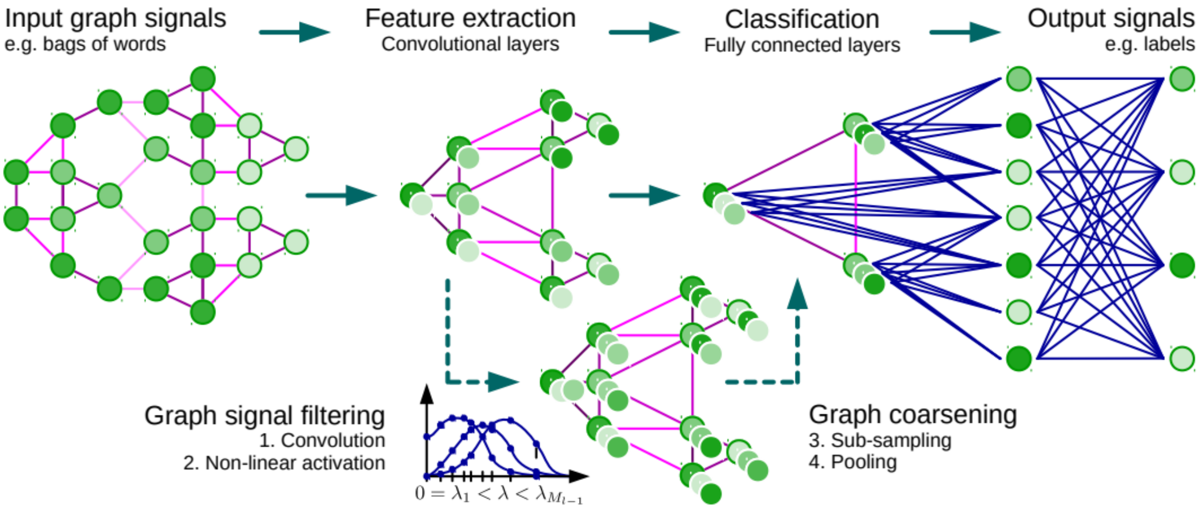

Input graph signals
e.g. bags of words
Feature extraction
Convolutional layers
Classification
Fully connected layers
Output signals
e.g. labels
Graph signal filtering
1. Convolution
2. Non-linear activation
$0 = \lambda_1 < \lambda < \lambda_{M_l-1}$
Graph coarsening
3. Sub-sampling
4. Pooling

Figure 39. Architecture of a graph CNN with the four ingredients of a graph convolutional layer. Figure from (Defferrard et al., 2016).

SyncSpecCNN (Yi et al., 2017) addressed this issue in the case of 3D shapes meshes, by mapping the spectral representation of signal on an input graph to a predefined canonical graph. This weight sharing scheme to align spectral domains uses functional maps (Ovsjanikov et al., 2012).

To explore hidden structural relations unspecified by the graph Laplacian matrix, (R. Li et al., 2018) proposed the adaptive graph convolution network. They augment a graph with a residual graph, which is constructed by computing a pairwise distance of nodes.

Despite these recent tricks to circumvent the problem, spectral methods share a common weakness. They need to load the whole graph into the memory to perform graph convolutions, which is not efficient on big graphs.

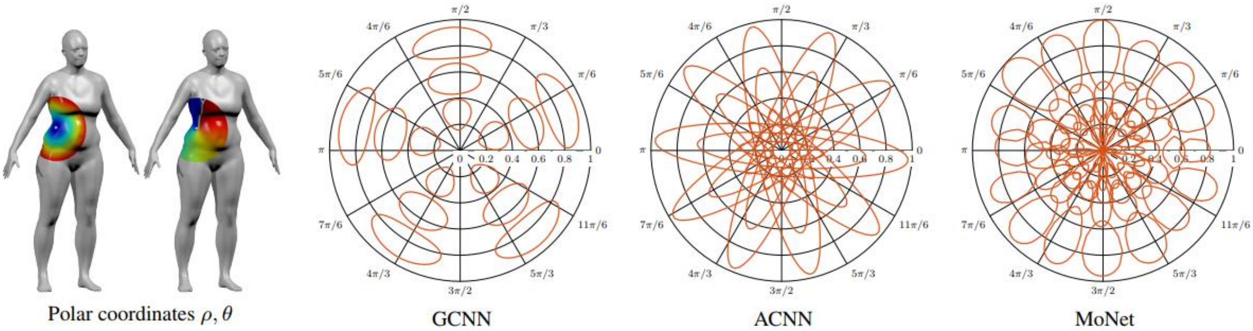
b. Spatial graph convolution

Spatial graph convolutions use the edges of the graph to define neighborhoods between nodes. They are quite similar to point convolution that would be applied to the nodes with the edges as the neighborhood definition. However, for most works on graph convolutions, the graph represents a Riemannian manifold or at least a surface and the operator is designed to be invariant to the surface variations.

The first attempt at a spatial graph convolution, *Geodesic CNN* (Masci et al., 2015), extracted local patches on meshes, and convolved them with filters expressed in polar coordinates. The orientation ambiguity was solved by applying filters in all possible orientations, and retaining the maximum responses.

(Boscaini et al., 2016) proposed the *Anisotropic CNN* model, which extends the *Geodesic CNN* model with anisotropic filters. They exploit the maximum curvature directions to orient the patches they extract.

Using the same local polar coordinates, (Monti et al., 2017) proposed *MoNet*, a model introducing pseudo-coordinates and weight functions to let the weight of a node's neighbor be determined by the relative position (pseudo-coordinates) between the node and its neighbor. They learn filter shapes by estimating the means and variances of Gaussians that associate filter weights to the local pseudo-coordinates. These three graph convolution definitions are illustrated in Figure 40.



Polar coordinates ρ, θ GCNN ACNN MoNet

Figure 40. Illustration of the patch operators from methods using polar coordinates to define their filters. Figure from (Monti et al., 2017).

(Simonovsky and Komodakis, 2017) use edge labels, which play a similar role as the local pseudo-coordinates, as input to a filter-generating subnetwork. With the same idea of generating filter weights, *FeastNet* (Verma et al., 2018) does not rely on hand-designed local pseudo-coordinates, but learns the mapping between local graph patches and filter weights using the features in the previous network layer.

Closer to our problem, (Y. Wang et al., 2018) designed a graph convolution especially for the processing of point clouds. Their key idea is to update dynamically the graph after each layer of the network. The graph edges are chosen as the KNN of the points in the features space. The proximity between point is thus redefined dynamically by the network, which allows nonlocal diffusion of information throughout the point cloud.

Another work applying graph convolutions to point clouds is the *SuperPoint Graph* method (Landrieu and Simonovsky, 2017). Instead of simple points, they use super-points (see Section III.2.3) to build their graph and use recurrent graph convolutions to create a graph segmentation. This pipeline is more elaborate than a simple graph CNN. They combine MLP feature encoders in the style of PointNet (Qi et al., 2017a), Gated Graph Neural Networks (Li et al., 2015) and Edge-Conditioned graph convolutions (Simonovsky and Komodakis, 2017), to refine the super-points embedding according to pieces of information passed along super-edges.

If we focus on the most recent spatial graph convolutions, which are the closest to our work, we notice that they combine features on local surface patches, while being invariant to the deformations of those patches in Euclidean space. In contrast, a point convolution combines features locally according to the 3D geometry, thus capturing the deformations of the surfaces.

IV.1.3. Pointwise multi-layer perceptrons

In the end of 2016, just a few months after the beginning of this thesis, a new family of neural networks for point cloud processing appeared. Instead of focusing on what worked in images, and what could be learned from handcrafted features, some researchers chose to focus on the nature of point clouds, and designed innovative network architectures. Two papers originated this new family of neural network, (Qi et al., 2017a; Ravanbakhsh et al., 2016). However, (Qi et al., 2017a) is considered as the major precursor of this kind of networks. We first explain this method and then enumerate some of the approaches that used the same principle later.

a. Pointnet, a deep learning framework without convolutions

PointNet (Qi et al., 2017a) is considered a milestone in point cloud deep learning, because it introduced a new type of network architecture for point cloud processing, that does not use convolutions. This network is not inspired from image CNNs, but uses the characteristics of point clouds; in particular, the fact that they are unordered sets. Before explaining the full PointNet architecture, we need to understand its main brick, the pointwise MLP shown in Figure 41.

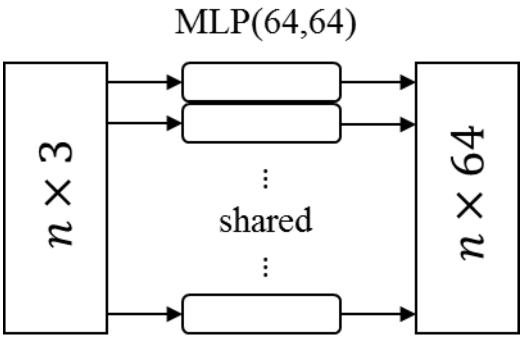


Figure 41. Pointwise MLP block in PointNet architecture.

A *multi-layer perceptron* is a simple form of neural network, consisting of a succession of fully connected layers. We can describe a MLP architecture with the notation $MLP(h_1, \ldots, h_l)$, where l is the number of layers and h_i the number of neurons of layer i. For example, the first MLP used in PointNet architecture, illustrated in Figure 41, has 2 layers of 64 neurons. When used as a standard machine learning classifier like in Figure 42, these layers are followed by an output layer, generally a softmax, which returns the class scores. In PointNet, this MLP block is just used as a feature encoder and is applied to each input point in parallel, hence its name, pointwise MLP. The 1×1 convolution in image is the equivalent of this block. It processes each pixel of the image independently, without neighbor information.

For a better understanding of the features that are learned by such a block, we can show them in a 2D Euclidean space with the open source visualization tool http://playground.tensorflow.org. Figure 42 illustrates the classification of two groups of points (orange and blue) in a 2D space with a $MLP(6,6)$. Many details are illustrated in this scheme. They allow a deep understanding of the mechanisms involved. First, on the right, we see the two groups of points in their 2D space and the output of the

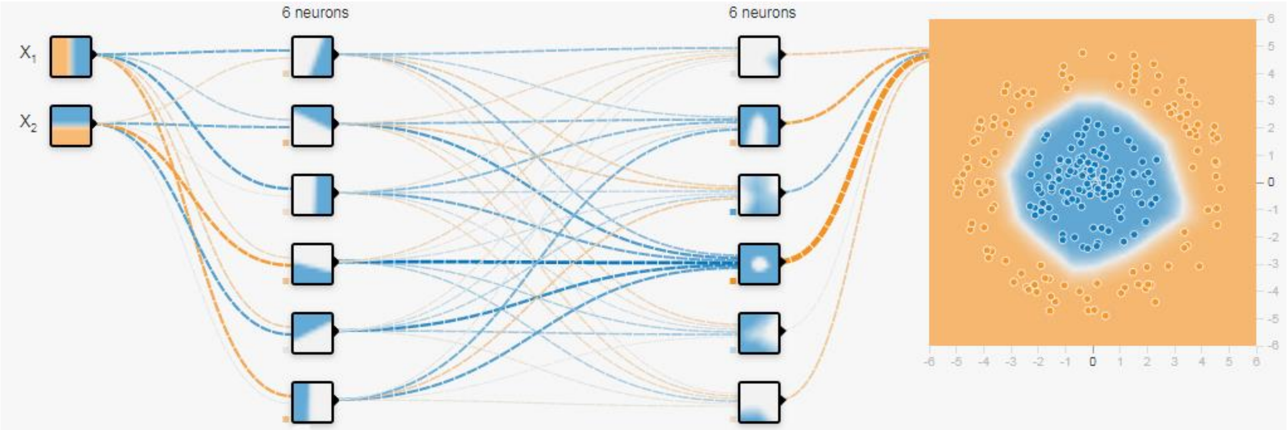

6 neurons
6 neurons
X_1
X_2

Figure 42. Illustration of the functions learned by the neurons of a MLP on 2D points.

classifier in the whole space (background color). It is clear that the MLP managed to learn a function of $\mathbb{R}^2$ that separated the two groups. In the middle, we can see the neurons and their activation functions. The value of each neuron's activation depends on the coordinates of the input point. Thus, they also are functions of $\mathbb{R}^2$, and are displayed in the small cells representing the neurons. We can notice that the functions learned in the first layer are very simple. This is normal, as they are simple linear combinations of the input coordinates, followed by an activation function, here a *Rectified Linear Unit* (ReLU). In the second layer, the neurons learn more complex space functions, which are linear combinations of the previous neuron activations. Therefore, the pointwise MLP blocks of PointNet learn functions of the space ($\mathbb{R}^3$ for the first layer $\mathbb{R}^n$ afterwards), that we call spatial encodings. They have higher or lower responses for different parts of the space.

The whole PointNet architecture, shown in Figure 43, is more than a simple spatial encoder. The final 1024 responses are not only spatial encodings; they also depend on the input shape globally. A point with the same coordinates in two different shapes will have different responses. At the end of the architecture, a *maxpooling* operation aggregates these responses into a feature vector characterizing the input shape. The final classification scores are obtained with a standard MLP on these 1024 features.

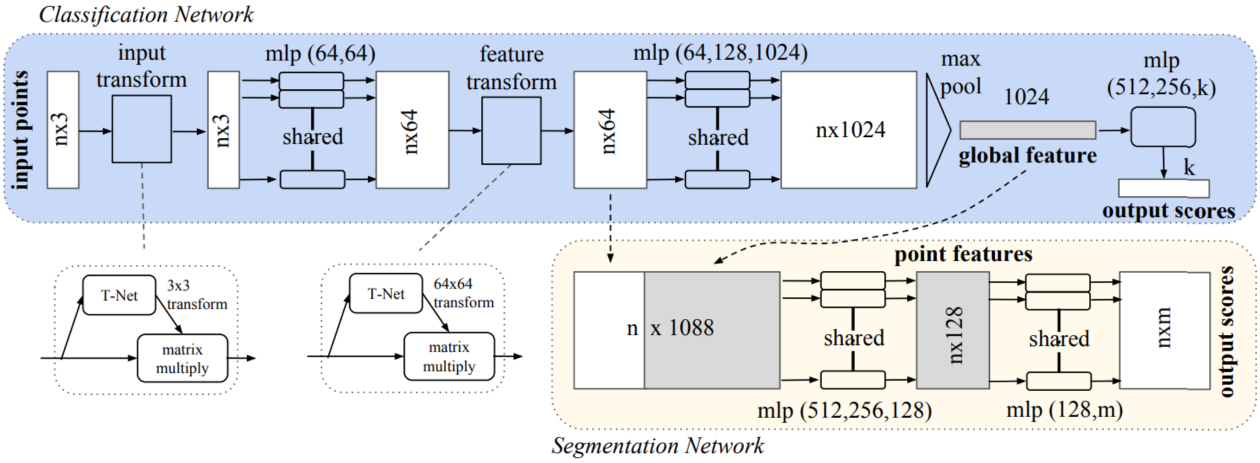

Classification Network
input points
nx3
input transform
nx3
mlp (64,64)
shared
nx64
feature transform
nx64
mlp (64,128,1024)
shared
nx1024
max pool
1024
global feature
mlp (512,256,k)
k
output scores
T-Net
3x3 transform
matrix multiply
T-Net
64x64 transform
matrix multiply
point features
n x 1088
shared
nx128
shared
nxm
output scores
mlp (512,256,128)
mlp (128,m)
Segmentation Network

Figure 43. PointNet architecture, (Qi et al., 2017).

The authors also propose a segmentation head to their architecture, which combines the pointwise features and the global shape signature to output scores for each point.

The most remarkable part of the architecture, though, is before the maxpooling. As we notice in Figure 43, a simple pointwise MLP is not sufficient to learn the descriptive features. PointNet uses two MLPs, each preceded by a transformer. Before the first MLP spatial encoding, the input points are transformed by a transformer network (T-Net). The T-Net, described in Figure 44, learns and returns a 3 by 3 matrix that is used to realign the points in a canonical orientation. This network is simply based on a MLP spatial encoder, followed by a maxpooling that aggregates the point features. We can consider it as a simplified PointNet, but with only spatial encodings and not shape dependent features. The shape signature used to get the canonical orientation is computed as the maximal response among all the points for each encoding. A simple way to understand this signature is to look at each of the 1024 spatial encodings individually. For each one of them, if some of the points of the shape are located in a part of the space with high responses, then the corresponding feature has a high value. Together, the features encode the presence or the absence of points in different parts of the space, effectively describing the shape.

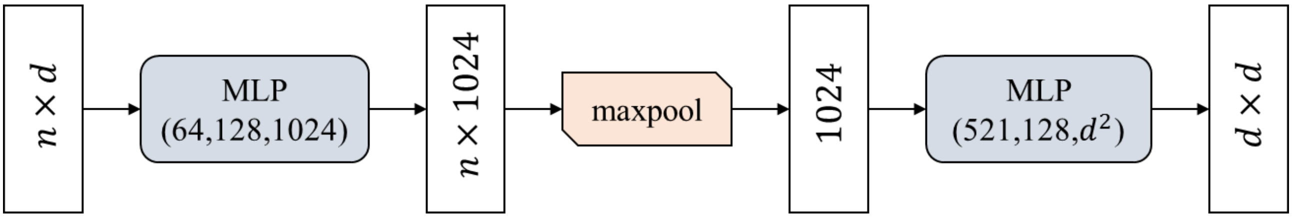


Figure 44. T-Net block in PointNet architecture. d is the dimension of the input features.

After the learned alignment, new spatial encodings are learned by the network. At this point the features are shape dependent, because two shapes would not have been oriented in the same direction. We can understand why the transformer was crucial, by considering two objects with the same shape, but different orientations. If the two objects were not realigned beforehand, the responses by spatial encoding would be very different. To be able to classify different shapes, the network should have learned spatial encodings that describe all the shapes in all their orientations. With a first block that realigns the shapes, the diversity of spatial encodings that have to be learned is reduced. This is exactly what we explained in the introduction, the complexity of the underlying model is reduced, and the network can learn better representations.

A second T-Net is used to realign the features from the first spatial encodings. It is harder to grasp the meaning of this transformer, as it returns a 64×64 orthogonal matrix, that we cannot represent like a 3×3 rotation matrix. The idea is the same as the first T-Net, it places the features in a canonical space orientation, so that the following spatial encodings are easier to learn by the MLP.

As a conclusion, this architecture created a small revolution in the point cloud community, because of its originality and innovativity. Although it did not take into account what could be learned from handcrafted features, it created its own type of representations for point clouds, with this succession of spatial encodings and found ways to reduce the complexity of the underlying model without convolutions. We see that these pointwise MLPs do not have a high descriptive power, and need to be helped by transformer networks to be able to learn complex shapes. However if we need to learn simple geometric representations, they can be used alone. Later works explore this idea, and we present them in the next paragraph.

b. Hierarchical MLP architectures

Despite being a groundbreaking work, PointNet is still limited because it does not consider local spatial relationships in the data. Following PointNet, some approaches reproduced the idea to use MLP-based architectures for point cloud processing, with various modifications to include spatial relationships. We already talked about (Zhou and Tuzel, 2018), who used small MLP encoders inside a voxel-based architecture. (J. Li et al., 2018) proposed SO-Net, an architecture combining MLP encoders with *self-organization maps* structures, that take care of the hierarchical aggregation of features. The authors of PointNet also proposed a hierarchical architecture called PointNet++ (Qi et al., 2017b), which hierarchically samples and groups point features, by applying MLP encoders at multiple scales. We are more interested in this kind of networks, which start to look like a convolutional architecture on point clouds.

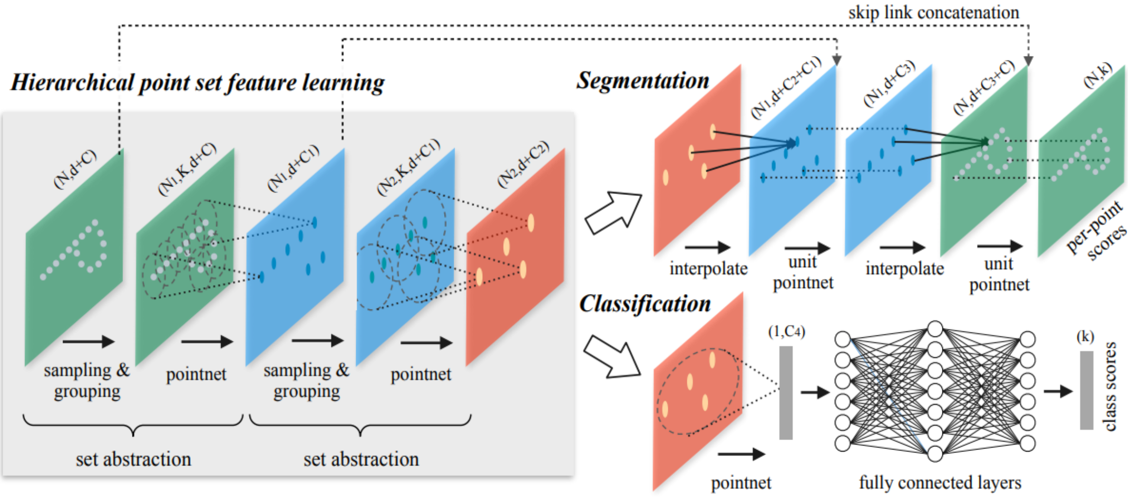

skip link concatenation
Hierarchical point set feature learning
(N,d+C)
(N1,K,d+C)
(N1,d+C1)
(N2,K,d+C1)
(N2,d+C2)
sampling & grouping
pointnet
sampling & grouping
pointnet
set abstraction
set abstraction
Segmentation
(N1,d+C2+C1)
(N1,d+C3)
(N,d+C3+C)
(N,k)
interpolate
unit pointnet
interpolate
unit pointnet
per-point scores
Classification
(1,C4)
(k)
pointnet
fully connected layers
class scores

Figure 45. PointNet++ architecture (Qi et al., 2017b). One of the first hierarchical neural networks on point clouds.

After PointNet++, many works developed the idea to aggregate local neighborhood information with MLPs. The idea behind these architectures is the following. Although (Qi et al., 2017a) proved a universal approximation theorem, stating that their pointwise MLPs can approximate any continuous functions, they also showed that these MLPs cannot learn very complex representations. However, we know that the representations learned by convolution kernels are very simple, and thus can be learned with MLPs. This is very convenient, because, as we show in the next section, the formulation of a convolution kernel on point clouds is not obvious. Instead of finding a way to define it, one can simply let MLPs do the job. We give more details about the MLP-based point convolutions in the next section, with other point convolution approaches.

IV.1.4. Point convolution networks

In this section, we review the most prominent methods that used point convolutions. Following the line of thoughts developed during Chapter III, we also want to define our learnable representations as point convolutions. Therefore, we are particularly meticulous about these methods. We detail how they define and use convolution kernels and how they design their architectures. Before enumerating them, we start by explaining how convolutions on point clouds are formulated.

a. Definition of a general point convolution

The formulation of a point convolution is inspired from images convolutions. There are some major differences, because a point cloud is sparse and continuous, while an image is dense and discrete, but the intuition of what happens is preserved. The convolution of an image I by a kernel g at a pixel x is defined as:

$$(I * g)(x) = \sum_{x_i \in \mathcal{N}_x} g(x_i - x) I(x_i) \tag{IV-6}$$

Where $\mathcal{N}_x$ are the neighbor pixels of x.

The first specificity of this definition is that it is a discrete version of a general convolution: x and x_i live in discrete domains included in $\mathbb{N}^2$. As opposed to the pixel positions in an image, the point positions in 3D space are continuous, but Equation (IV-6) can still be used to define the convolution of a point cloud by a kernel g at a point x. In that case, x and x_i represent point positions and live in continuous domains included in $\mathbb{R}^3$. This formulation exploits the spatial localization property of a point cloud stated in Section II.1.2: the point positions are used as the structural element in the convolution, like the pixel positions.

The other specificity is how the locality is defined. The definition domain of the kernel function g is limited to a neighborhood. More than a convolution, this is the definition of a local space filter. In the case of 3D points, this locality can be enforced using either radius neighborhoods or KNN. With the KNN definition, the neighborhoods always contain the same number of points, but their extents vary across the point cloud. With the radius definition, the number of points in the neighborhoods vary, while the extent always stays the same. We will see that both types of neighborhoods have been used in the literature.

From this definition of image convolution, a point convolution that respects these two specificities can be defined. For the sake of clarity, we use the notation introduced in Chapter II, and call x_i and f_i the points from $\mathcal{P} \in \mathbb{R}^{N \times 3}$ and their corresponding features from $\mathcal{F} \in \mathbb{R}^{N \times D}$. The general point convolution of $\mathcal{F}$ by a kernel g at a point $x \in \mathbb{R}^3$ is defined as:

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g(x_i - x) f_i \tag{IV-7}$$

In this definition, f_i is a feature vector of $\mathbb{R}^D$, and g returns a weight matrix in $\mathbb{R}^O$, where O is the output feature vector dimension. Apart from the neighborhood type, the crucial part in Equation (IV-7) is the definition of the kernel function g. This function takes the neighbors positions centered on x as input. We call them $y_i = x_i - x$ in the following. The domain of definition of g is continuous. It can be $\mathbb{R}^3$, or in the case of radius neighborhoods, a ball in $\mathbb{R}^3$. In the next paragraph, we will see that the definition of the convolution kernel is the major differentiating factor between the state-of-the-art methods.

b. The different point convolutions proposed in the literature

Many types of point convolution kernel have been proposed, and we sort them in two groups. The MLP kernels and the geometric kernels. As g is a function continuously defined in $\mathbb{R}^3$, it is straightforward to define it with the help of MLPs, instead of defining geometric weights like in image. However, using such a representation generally makes the convolution operator more complex and the convergence of the network harder. The methods using MLP convolution do not perform as well as geometric kernels. As all these methods are very close to our own point convolution definition, we enumerate them with details of their design.

Parametric Continuous Convolutions (S. Wang et al., 2018) uses Equation (IV-7), and formulates g as a MLP. In particular, they define $g = MLP(16,32, D \times O)$. It means that the $D \times O$ weight matrix is composed of spatial encodings like the ones we show in Figure 42. This figure gives a good idea of the representations that this type of kernel is able to learn. It is interesting to notice that, like any point convolution, the points where the results are computed can be different from the input points, like shown in Figure 46 We can add that the authors chose KNN and only built a single scale architecture, with 8 successive convolutional layers, which is not ideal for the learning of descriptive representations.

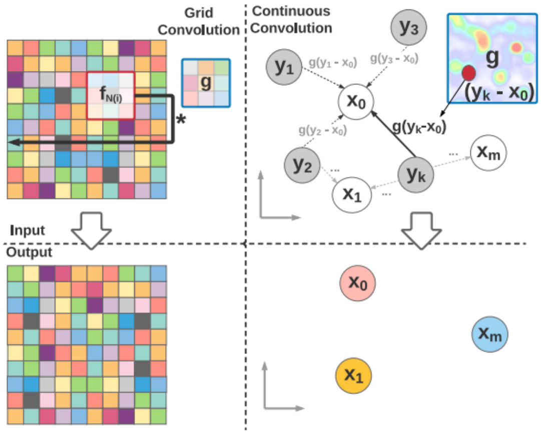


Figure 46. Illustration of deep parametric convolution compared to an image convolution (S. Wang et al., 2018).

PointCNN (Y. Li et al., 2018) heavily relies on the use of MLPs, but not following Equation (IV-7). Instead they formulate their own convolution equation, which is more complicated. First they choose to use KNN and fix their convolution size with the parameter K, number of nearest neighbors. Then they introduce two weight matrices, one learnable, and the other result of a MLP combining the information of all neighbors together. The learnable weight matrix is called $\boldsymbol{K}$ in their paper, but we will call it W to avoid confusion with the number of neighbors K. The weight matrix resulting from a MLP is called χ and defined as:

$$\chi(x) = Net(\mathcal{N}_x) \tag{IV-8}$$

In this case, we call the MLP "Net", because, in theory it should be a MLP, but in practice it is implemented with other neural network layers to optimize the computational cost. The resulting matrix χ defines $K \times K$ weights. It is very important to notice that χ depends on the neighbors of x, and in particular of their order. It means that it is not geometrically consistent. Two nearly identical neighborhoods can have different order of points, and thus a different resulting χ.

They then define their convolution as:

$$\chi\text{conv}\ (x) = \sum_{k<K} W_k \left(\sum_{i<K} \chi_{ik}(x)[MLP(y_i), f_i] \right) \tag{IV-9}$$

Given the behavior of concatenation $[\ldots, \ldots]$ with matrix multiplication. We can break down this equation into a sum of two elements:

$$\chi\text{conv}\ (x) = \sum_{k<K} W_k^1 \left(\sum_{i<K} \chi_{ik}(x) MLP(y_i) \right) + \sum_{k<K} W_k^2 \left(\sum_{i<K} \chi_{ik}(x) f_i \right) \tag{IV-10}$$

The first term combines the information of spatial encodings obtained from the neighbor coordinates and the second term combines the corresponding neighbor features (as a normal convolution would do). The first term is interesting, because it adds geometrical information at each scale of convolution, while a normal convolution only combines the features of the previous scales.

The weights χ_{ik}, which define what the authors call χ-transformation, are the cornerstone of this method. They are the only weights that depend on the neighborhood shape, and thus effectively encode a convolutional filter. In fact we can see that they are shape encodings (not exactly like spatial encodings, because they do not only depend on the point coordinates, they also depend on their order).

However, they result of a blind combination of all neighborhood information, with "Net" a non-symmetrical function. We already explained why this non-symmetry with the order of points is a weakness, but another crucial flaw is that this single neural network has to encode any shape defined by K points. "Net" thus has a huge number of parameters and needs to learn very complex representations. A single MLP network is not likely to handle this very well and the authors had to reduce the complexity of "Net" with depth-wise convolutions. Even with these neural network tricks, they admit that the learned χ-transformations are far from ideal.

Monte Carlo Convolution (Hermosilla et al., 2018) is also a MLP convolution, but introduces the notion of sample density in the definition. It means that they use Equation (IV-7) with a slight modification. They add a *Probability Density Function* (see Figure 47) that weights each neighbor feature according to the local density around its support point and normalize with the number of neighbors:

$$(\mathcal{F} * g)(x) = \frac{1}{|\mathcal{N}_x|} \sum_{x_i \in \mathcal{N}_x} \frac{1}{p(x_i|x)} g(x_i - x) f_i \qquad \text{(IV-11)}$$

The idea is quite simple to understand, a point in a denser area will have less influence, because the points around it also contribute to the result. This very smart formulation uses radius neighborhoods, which is in accordance with its care for robustness to non-uniform densities. However, they still define g with MLPs. Each $D \times O$ weight of g is computed as a $MLP(8,8,1)$. It is a very heavy definition, so they propose to use multiple $MLP(8,8,8)$ instead of multiple $MLP(8,8,1)$, to define the weights for layers with high number of features (in the end of the network). For these layers, they also use the trick of separable convolutions, which helps to get faster computations. In that case g returns a vector of size D instead of a matrix of size $D \times O$. The features are thus just reweighted independently but not combined together. They add intermediate 1×1 convolutions to mix the features.

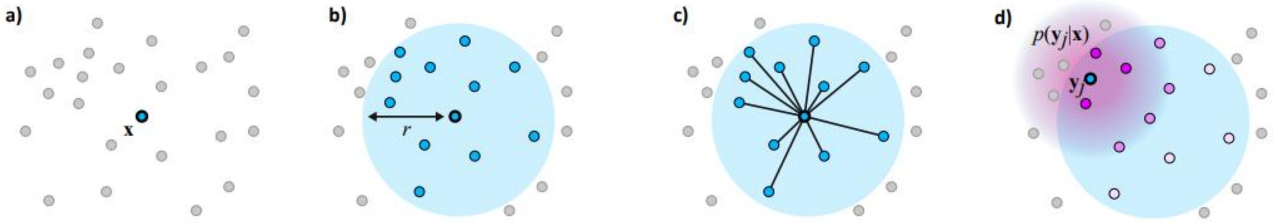


Figure 47. Illustration of the *Probability Density Function* in Monte Carlo Convolution (Hermosilla et al., 2018). The purple disk shows the bandwidth used for density estimation and the pink neighbors are the one in a denser area.

Pointwise CNN (Hua et al., 2018) is a geometric point convolution, it uses voxels bins to define the weights of function g, as shown in Figure 48. In particular, they define their convolution as in Equation (IV-7), with:

$$g(y_i) = \frac{w_{V(i)}}{n_{V(i)}} \qquad \text{(IV-12)}$$

where $V(i)$ is the index of the voxel where the point y_i is located, w are the weight matrices of each voxel, and n is the number of points in each voxel.

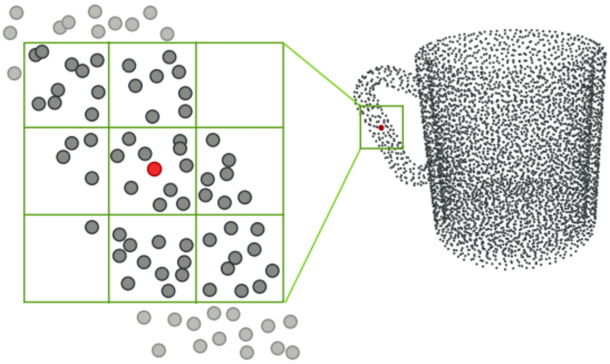


Figure 48. Illustration on Pointwise convolution (Hua et al., 2018). The nearest neighbors are binned into kernel voxels, and the voxels are convolved with kernel weights.

Locating the kernel weights with voxel bins is simple and elegant, but lacks flexibility like grid networks. The number of voxels is fixed and quite big in 3D (at least $3^3 = 27$). Furthermore, the normalization strategy used in this formulation burdens their network with unnecessary computations. In our own convolution, the subsampling strategy alleviates both varying densities and computational cost.

SpiderCNN (Xu et al., 2018), also a geometric convolution, defines its kernel as a family of polynomial functions applied with a different weight for each neighbor. In the paper, they claim that their function g is defined as:

$$g(y_i) = g^{step}(y_i) \cdot g^{Taylor}(y_i) \tag{IV-13}$$

Where g^{step} is a radius stepwise function, and g^{Taylor} is a third order polynomial function. However, in practice, they define g as:

$$g(y_i) = w_i \cdot g^{Taylor}(y_i) \tag{IV-14}$$

Where w_i is a weight that is applied at the neighbor i (in distance-wise order). This implementation detail is a significant weakness, because it leads to spatially inconsistent filters. The function g is not defined on $\mathbb{R}^3$ anymore, as its results also depends on the order of the neighbors. By contrast, our own kernel weights are located in space and the result of our convolution is invariant to point order.

Flex-convolution (Groh et al., 2018) uses linear functions to model its geometric kernel. This is an extremely simple formulation of g, where each $D \times O$ weight is a linear function of the $\mathbb{R}^3$:

$$\forall d < D, \qquad \forall o < O, \qquad g_{d,o}(y_i) = W_{d,o} y_i + b_{d,o} \tag{IV-15}$$

The number of parameters $D \times O \times 4$ is very small, and the representation learned very simple. Even if it helps to reduce the complexity of the underlying model, this convolution has a limited representative power. This could restraint the power of this network on complex tasks. Furthermore, this formulation uses KNN, which is not robust to varying densities as discussed above.

Point Convolution by Extension (Atzmon et al., 2018) design (Figure 49) is the closest to our own convolution. Its definition also uses points to carry kernel weights, and a correlation function. Like the weight of the voxels in Pointwise CNN (Hua et al., 2018), each kernel points defines a location where a weight matrix is applied. Therefore, the kernel function g is defined as:

$$g(y_i) = \sum_{k<K} \Phi(|y_i - \tilde{x}_k|) W_k \tag{IV-16}$$

Where Φ is a Gaussian function correlating the neighbor locations y_i with the kernel weight locations $\tilde{x}_k$. Each $D \times O$ weight matrix W_k is applied to the neighbors which are close to its location $\tilde{x}_k$.

However, this design is not scalable because it does not use any form of neighborhood, the sum in Equation (IV-7) is computed on the whole input cloud. As a result, the convolution computations are quadratic on the number of points. In addition, it uses a Gaussian correlation where our own formulation uses a simpler linear correlation, which helps gradient backpropagation during the learning phase.

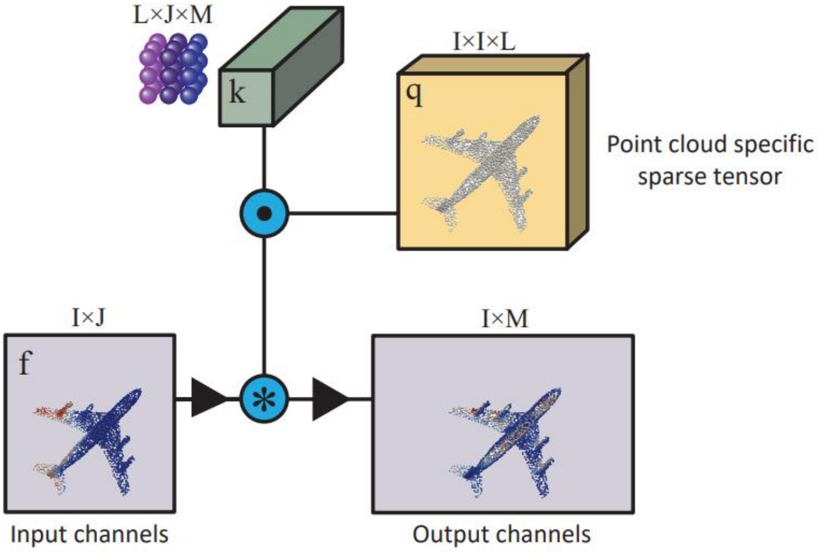


Figure 49. Computational flow of the point convolution by extension (Atzmon et al., 2018).

Conclusion

Although many point convolution definitions have been proposed, none of them are entirely satisfying in our opinion. MLPs are complex representations that do not help simplifying the underlying model. The networks using MLP convolutions involve more parameters and computations than the networks using explicit geometric kernels. They are harder to train, and need tricks to help convergence. The geometric convolutions that we listed all lack at least one crucial element, whether it is flexibility, geometric consistency, or neighborhood definition.

In our case, we define an explicit geometric convolution whose weights are directly learned, without the intermediate representation of a MLP. We will detail our design in the next section. It is inspired by image convolutions, and by our study of handcrafted point cloud representations. Section IV.3 demonstrates the efficiency of our approach, outperforming all comparable methods and ranking first in several benchmarks. Furthermore, our design also offers a straightforward deformable version. To the best of our knowledge, none of the previous works experimented a spatially deformable point convolution.

We recap several deep learning approaches for point cloud processing in Table 11.

Table 11. Comparison of state-of-the-art deep learning methods on point clouds.

Methods	Category	Brief Description
SnapNet (Boulch et al., 2017)	Projective (2D)	Viewpoint projection, image CNN, back projection to point clouds
TangentConv (Tatarchenko et al., 2018)	Projective (2D)	Pointwise projection, tangent plane projection, 2D image-like kernels
SubSparseCNN (Graham et al., 2018)	Projective (3D)	Single projection, voxel grid, sparse convolution locations
SPLATNet (Su et al., 2018)	Projective (3D)	Layer-wise projection, permutohedral lattice, bilateral convolution
SegCloud (Tchapmi et al., 2017)	Projective (3D)	Single projection, voxel grid, CRF regularization
MSDVN (Roynard et al., 2018b)	Projective (3D)	Single projection, voxel grid, multiple scales
SynSpecCNN (Yi et al., 2017)	Graph	Spectral convolution, spectral alignment, functional maps
DGCNN (Y. Wang et al., 2018)	Graph	Spatial convolution, dynamic graph update, feature space distance
SPGraph (Landrieu and Simonovsky, 2017)	Graph	Superpoint graph, MLP encodings, graph convolution for regularization
Pointnet (Qi et al., 2017a)	MLP	Pointwise encodings, dynamic feature alignment, maxpooling aggregation
SO-Net (J. Li et al., 2018)	MLP	Pointwise encodings, self-organization maps
PointNet++ (Qi et al., 2017b)	Point Convolution	MLP kernels, radius neighbors, multiscale grouping
PointCNN (Y. Li et al., 2018)	Point Convolution	MLP kernels, KNN, χ-transformation
MCConv (Hermosilla et al., 2018)	Point Convolution	MLP kernels, radius neighbors, normalization by sampling density, shared MLP weights
ParamConv (S. Wang et al., 2018)	Point Convolution	MLP kernels, KNN, single scale architecture
FlexConv (Groh et al., 2018)	Point Convolution	Geometric kernels, KNN, linear functions
SpiderCNN (Xu et al., 2018)	Point Convolution	Geometric kernels, KNN, stepwise polynomial functions, order-dependent
PCNN by Ext (Atzmon et al., 2018)	Point Convolution	Geometric kernels, no neighborhoods, kernel points, Gaussian correlation

IV.2. KPConv: a new point convolution

This section introduces our new point convolution operator named **Kernel Point Convolution** (KPConv). KPConv is a geometric convolution; it consists of a set of local 3D filters, but overcomes previous point convolution limitations as shown in previous section. KPConv is inspired by image-based convolution, but instead of kernel pixels, we use a set of kernel points to define the area where each kernel weight is applied, like shown in Figure 50. The kernel weights are thus carried by points, like the input features, and their area of influence is defined by a correlation function. The number of kernel points is not constrained by a regular grid, making our design very flexible.

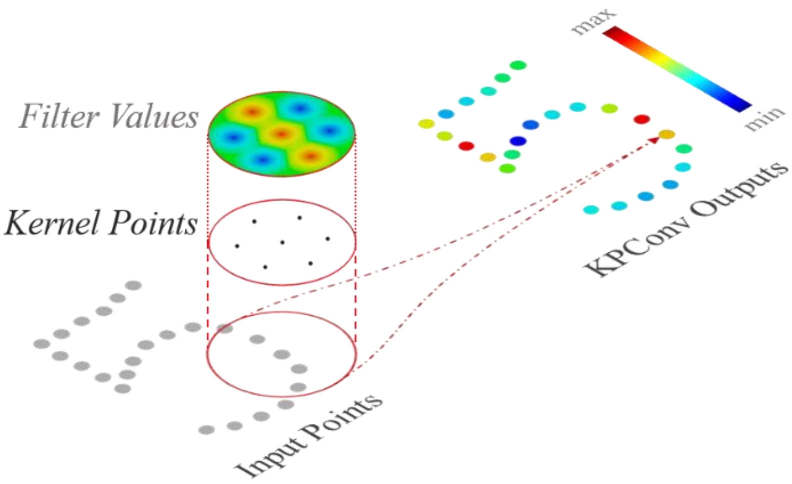


Figure 50. KPConv illustrated on 2D points. Input points with a constant scalar feature (in grey) are convolved through a KPConv that is defined by a set of kernel points (in black) with filter weights on each point.

Furthermore, we propose a deformable version of our convolution like (J. Dai et al., 2017) did for image convolution. It consists of learning local shifts applied to the kernel points. Our network generates different shifts at each convolution location, meaning that it can adapt the shape of its kernels for different regions of the input cloud. Our deformable convolution is not designed the same way as its image counterpart. Due to the different nature of the data, it needs a regularization to help the deformed kernels fit the point cloud geometry and avoid empty space.

As opposed to previous works opting for KNN, we favor radius neighborhoods. This choice is not only based on our study on handcrafted point cloud representations in Chapter III. We also stand with (Hermosilla et al., 2018), who showed that KNN is not robust in non-uniform sampling settings. The robustness of our convolution to varying densities is ensured by the combination of radius neighborhoods and regular subsampling of the input cloud, as demonstrated in our work on multiscale spherical neighborhoods (Section III.3). Compared to normalization strategies (Hermosilla et al., 2018; Hua et al., 2018), our approach also alleviates the computational cost of the convolution operation.

In addition to its deformable version, our design offers a great flexibility. It can be customized in several ways. In our experiments, we made some design choices, but, they could easily be modified in future works.

IV.2.1. A convolution kernel defined by points

KPConv is a point convolution, and can thus be defined with Equation (IV-7). As explained in the related work section, the crucial part in this equation is the definition of the kernel function g, which is where KPConv singularity lies.

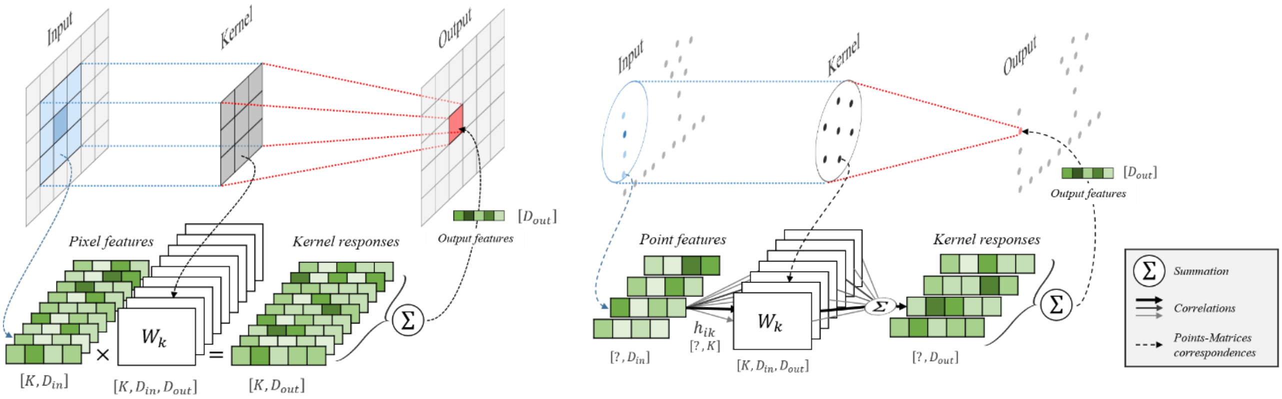

Input
Kernel
Output
Pixel features
Kernel responses
Output features
$[D_{out}]$
W_k
×
=
Σ
$[K, D_{in}]$
$[K, D_{in}, D_{out}]$
$[K, D_{out}]$
Input
Kernel
Output
$[D_{out}]$
Output features
Point features
Kernel responses
h_{ik}
$[?, K]$
W_k
Σ
$[?, D_{in}]$
$[K, D_{in}, D_{out}]$
$[?, D_{out}]$
Summation
Correlations
Points-Matrices correspondences

Figure 51. Comparison between an image convolution (left) and a KPConv (right) on 2D points for a simpler illustration. In the image, each pixel feature vector is multiplied by a weight matrix $(W_k)_{k<K}$ assigned by the alignment of the kernel with the image. In KPConv, input points are not aligned with kernel points, and their number can vary. Therefore, each point feature f_i is multiplied by all the kernel weight matrices, with a correlation coefficient h_{ik} depending on its relative position to kernel points.

In our case, this function takes the neighbor positions centered on x as input and is defined in $\mathbb{R}^3$. More precisely, as the neighbors are in the radius neighborhood $\mathcal{N}_x = \{x_i \in \mathcal{P} \mid \|x_i - x\| \leq r\}$, the centered neighbors, $y_i = x_i - x$, always are in the ball $\mathcal{B}_r^3 = \{y \in \mathbb{R}^3 \mid \|y\| \leq r\}$. $\mathcal{B}_r^3$ thus is the definition domain of g. Like image convolution kernels (see Figure 51 for a detailed comparison between image convolution and KPConv), we want g to apply different weights to different areas inside this domain. There are many ways to define areas in 3D space, and points are the most intuitive as features are also localized by them. Let $\{\tilde{x}_k\}_{k<K}$ be the K kernel points and $\{W_k\}_{k<K}$ be the associated weight matrices that map features from dimension D_{in} to D_{out}. Therefore, $\forall k < K$, $\tilde{x}_k \in \mathcal{B}_r^3$ and $W_k \in \mathbb{R}^{D_{in} \times D_{out}}$. We define the kernel function g of KPConv for any point $y_i \in \mathcal{B}_r^3$ as:

$$g(y_i) = \sum_{k<K} h(y_i, \tilde{x}_k) W_k \tag{IV-17}$$

where h is the correlation between $\tilde{x}_k$ and y_i , that should be higher when $\tilde{x}_k$ is closer to y_i. Inspired by the bilinear interpolation in (J. Dai et al., 2017), we use the linear correlation:

$$h(y_i, \tilde{x}_k) = \max\left(0, 1 - \frac{\|y_i - \tilde{x}_k\|}{\sigma}\right) \tag{IV-18}$$

where σ is the influence distance of the kernel points, and will be chosen according to the input density. Compared to a Gaussian correlation, which is used by (Atzmon et al., 2018), linear correlation is a simpler representation. We advocate this simpler correlation to ease gradient backpropagation when learning kernel deformations. A parallel can be drawn with rectified linear unit, which is the most popular activation function for deep neural networks, thanks to its efficiency for gradient backpropagation.

IV.2.2. Layout of the kernel points

Kernel point positions are critical to the convolution operator. Our kernels need to be arranged regularly to be efficient. As we claimed that one of the KPConv strengths is its flexibility, we need to find a regular disposition for any number of kernel points K.

There is no obvious regular layout of points in a sphere, so we chose to solve this issue by translating it into an optimization problem. The problem is simple; we want the K points $\tilde{x}_k$ to be as far from each other as possible inside a given sphere. We thus assign a repulsive potential to each point, defined as:

$$\forall x \in \mathbb{R}^3, \qquad E_k^{rep}(x) = \frac{1}{\|x - \tilde{x}_k\|} \tag{IV-19}$$

And add an attractive potential to the sphere center to avoid them diverging indefinitely

$$\forall x \in \mathbb{R}^3, \qquad E^{att}(x) = \|x\|^2 \tag{IV-20}$$

The problem then consists of minimizing the global energy:

$$E^{tot} = \sum_{k<K} \left(E^{att}(\tilde{x}_k) + \sum_{l \neq k} \left(E_k^{rep}(\tilde{x}_l) \right) \right) \tag{IV-21}$$

The solution is found by gradient descent with the points initialized randomly and some optional constraints. In our case, we fix one of the points at the center of the sphere. For some values of K (listed in Table 12), the points converge to a unique stable disposition. These layouts are in fact regular polyhedrons. Each polyhedron can be described by grouping points sharing a plane perpendicular to the polyhedron symmetrical axis. For a better understanding, some of the dispositions are shown in Figure 52.

The optimization is only done once. These regular layouts can be saved for future use. The surrounding points, orbiting around the center point, are not exactly on a sphere. Their distances to the center point are not the same, but the differences are marginal, so we can call the average distance to the center point r_{orbit}. The radius of the convolution neighborhood r and the influence distance σ should be adjusted to this distance. Like for many parameters of KPConv, the effect of this adjustment is intuitive. If σ is too large, the influence areas of the different kernel points will be overlapping, and if it is too small, the convolution sphere will not be entirely covered. In the following, when defining a convolution kernel, we rescale the kernel points to ensure that $r_{orbit} = 1.5 \times \sigma$, as a trade-off between overlapping and coverage. The radius of the convolution is thus directly set as the limit of the kernel point influence area, $r = 2.5 \times \sigma$. From now on, we only have to set the parameter σ and K to define our convolution kernels.

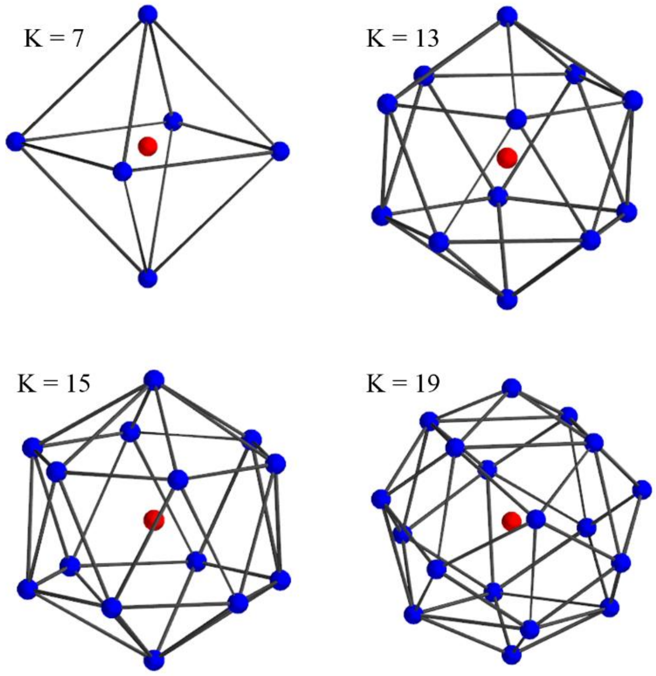


Figure 52. . Illustration of the kernel points in stable dispositions.

Table 12. Stable dispositions of the kernel point positions when the center point is fixed. If a disposition has an axis of symmetry, we describe it by the successive groups of points sharing a plane perpendicular to this axis.

K	polyhedron	groups along symmetrical axis
5	Tetrahedron	1-3
7	Octahedron	1-4-1
13	Icosahedron	1-5-5-1
15	-	1-6-6-1
18	-	1-5-5-5-1
19	-	1-4-4-4-4-1
21	-	1-6-6-6-1
25	-	4-4-4-4-4-4

IV.2.3. A flexible operator

Our design has many hyperparameters and offers a wide range of customization. In this section, we describe all the possible customizations, so that future works can explore different versions of KPConv. Such a level of customization could be seen as a weakness, as all the hyperparameters need to be tuned, but it is not the case. Thanks to the geometrical nature of our definition, most parameters can be set intuitively. We explain the choices we made for the design of the standard KPConv operator, used in our experiments.

Number of kernel points. The number of kernel points is the first customization factor. Grid kernels have a constrained number of parameters $(2k + 1)^{dim}$, where k is the size of the kernel in pixels/voxels, and dim is the dimension of the grid. For 3D grids, a kernel thus have at least 27 voxels. On the contrary, KPConv has a flexible number of kernel points, which can be set as a trade-off between computational cost and descriptive power. We choose this parameter with an empirical study presented in Section IV.3.5. Different numbers of kernel points are illustrated in Figure 52.

Kernel point layout. The kernel point layouts described above are the most regular structures to cover a spherical volume, but KPConv can use any kernel point layout. For example, the kernel point could be placed randomly, or on a grid, as shown in Figure 54.

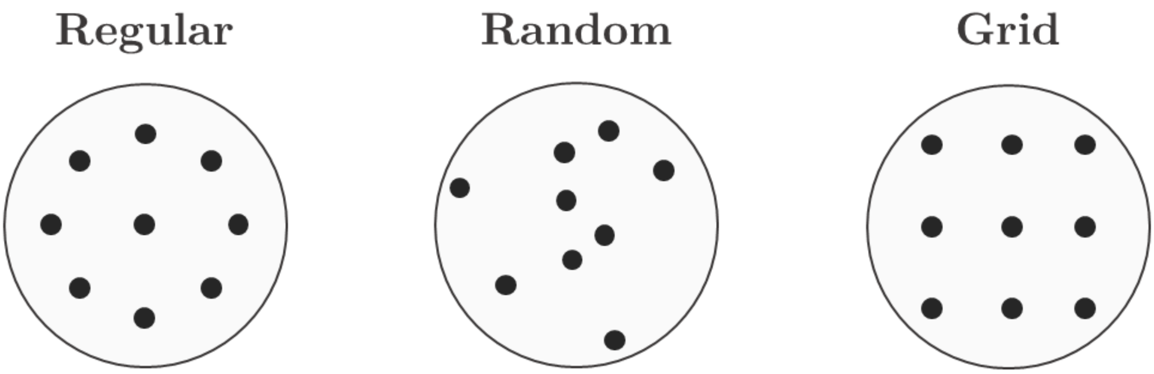


Figure 54. Example of customizations of the kernel points layout in 2D.

Influence area overlapping. In any configuration of kernel points, the ratio between the layout scale (in our case r_{orbit}) and the influence range σ controls the trade-off between overlapping and coverage of the influence areas. As a rule of thumb, we advocate the use of regular layouts, with small overlapping of the influence areas. It is also possible to define a specific influence distance for each kernel point, and adapt these values to the kernel point layout. Figure 53 shows the influence areas of the kernel points with different overlapping setups.

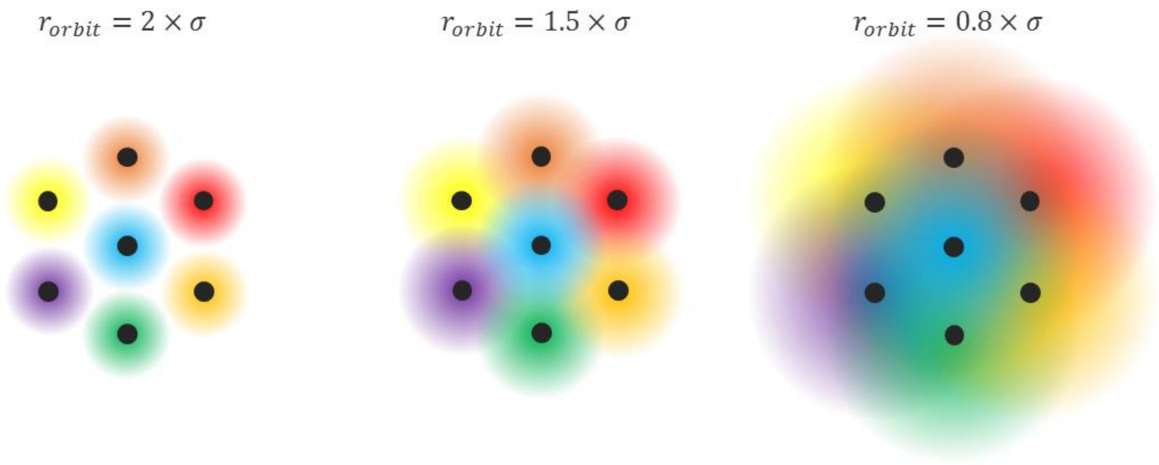

$r_{orbit} = 2 \times \sigma$
$r_{orbit} = 1.5 \times \sigma$
$r_{orbit} = 0.8 \times \sigma$

Figure 53. Example of customizations of the influence area overlapping in 2D. The area of each kernel point is shown with a different color.

Correlation function. The correlation function h can also be changed. We advocate the use of linear interpolation:

$$h_{linear}(y_i, \tilde{x}_k) = \max\left(0, 1 - \frac{\|y_i - \tilde{x}_k\|}{\sigma}\right)$$

But one could use a Gaussian correlation:

$$h_{gaussian}(y_i, \tilde{x}_k) = \exp\left(\frac{-\|y_i - \tilde{x}_k\|^2}{2\sigma^2}\right)$$

Or a rectangular correlation:

$$h_{rect}(y_i, \tilde{x}_k) = \begin{cases} 1, & \text{if } \|y_i - \tilde{x}_k\| < \sigma \\ 0, & \text{else} \end{cases}$$

In Figure 55, we can see that linear and Gaussian correlation are relatively similar, but the rectangular correlation presents artefacts in the overlapping areas.

Aggregation. Similarly to the correlation function, we can change the kernel behavior with the aggregation mode of g. We choose the summation mode in Equation (IV-17):

$$g(y_i) = \sum_{k<K} h(y_i, \tilde{x}_k) W_k$$

But it is possible to change it, for example by a closest aggregation:

$$g(y_i) = h(y_i, \tilde{x}_{k^*}) W_k \qquad \text{with} \qquad k^* = \underset{k<K}{\operatorname{argmin}} \|y_i - \tilde{x}_k\|$$

Figure 55 illustrates the two aggregation modes. We see that the sum and the closest aggregation are very similar with linear and Gaussian correlation. With the closest correlation, the notion of overlapping area does not matter anymore, as only the closest kernel point applies its weight. It is particularly interesting for the rectangular correlation, which has problems in these overlapping areas. We choose the sum aggregation because it is computationally more efficient and creates a continuous kernel function. With our linear correlation, the difference with closest aggregation is negligible.

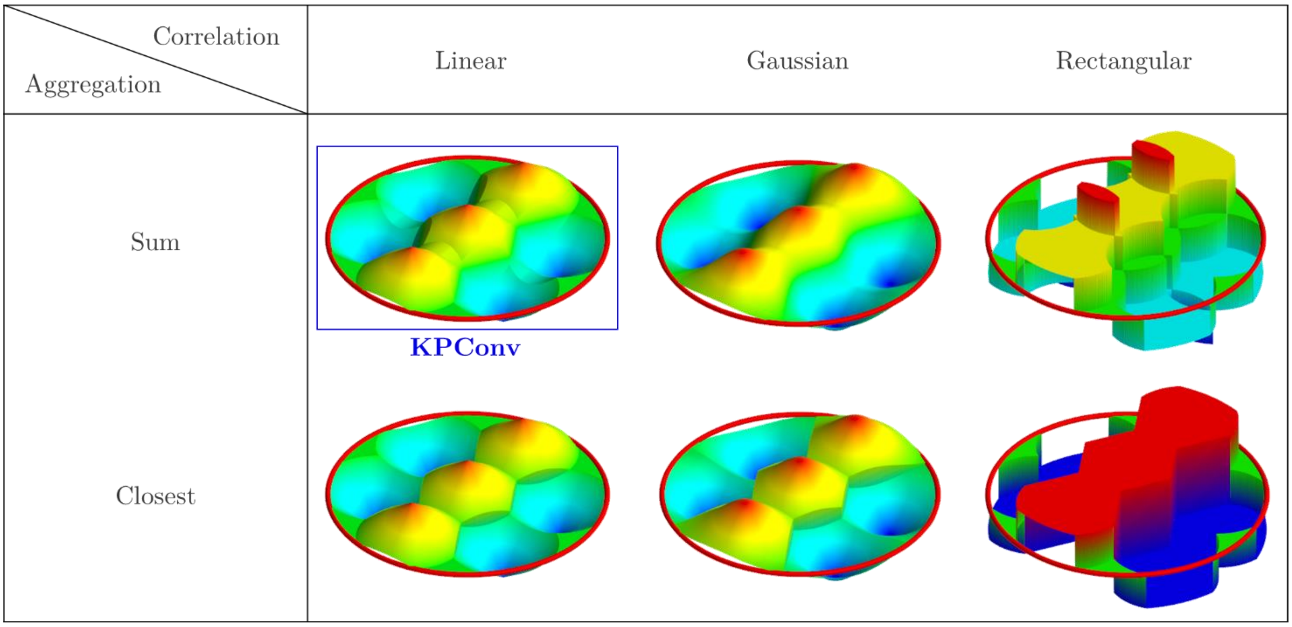


Figure 55. Example of customizations of the correlation function and aggregation mode.

Input density. The last customization factor is the ratio between the input point density and the kernel size. As explained in the introduction, the input point cloud of KPConv is subsampled by a grid, the same that was used for our multiscale neighborhoods in Section III.3.2. If we call the size of grid cells dl, then we can set the influence range of our kernel points, and thus the kernel size, with

$$\sigma = \Sigma \times dl \tag{IV-22}$$

The parameter Σ controls the density of points that our convolution can see, as shown in Figure 56. A higher value means more information, but also a higher computational cost. It is interesting to notice that in image convolution, the kernel size (3×3 for example), also controls the input density. In KPConv, the two notions are distinct and can be customized independently, with K and Σ.

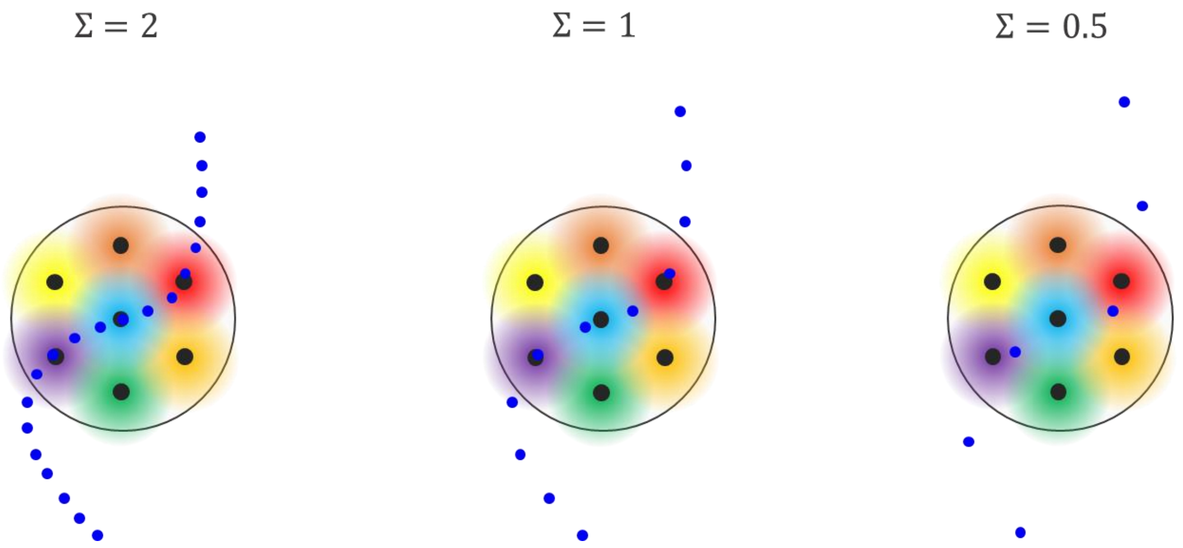


Figure 56. Customization of the input density illustrated in 2D for clarity.

We know from image convolutions that smaller kernels are better to create deep networks. This is why we limit the number of kernel points, and set $\Sigma = 1.0$, so that the density of the kernel points matches the density of input neighbors. A detailed study of the input density influence is conducted in Section IV.3.5.

Example of customization: Voxel convolution. With this level of customization, we can reproduce the behavior of previous convolutions, or imagine new ones. For example, we can define a voxel convolution, that applies its weights to the points located inside voxel bins, like (Hua et al., 2018). For this, we simply have to place $K = 27$ kernel points in a $3 \times 3 \times 3$ grid layout. To apply a constant weight inside the voxel, we just choose a rectangular correlation and the closest aggregation mode.

IV.2.4. Rigid or deformable kernels

The use of kernel points instead of fixed voxel positions in KPConv allows a great versatility, and the kernel point positions are obviously critical to the convolution operator. As explained in Section IV.2.2, our kernels need to be arranged regularly to be efficient. Indeed, the neighbors can be located anywhere inside the convolution sphere. In practice, our point clouds represent surfaces and the neighbors rarely fill the whole sphere. As it can be seen in Figure 56, only some of the kernel points have neighbors in their range. We call them active kernel point for this particular neighborhood. The rest of the kernel points are inactive. We need to cover the whole sphere with kernel points because there are many different neighborhoods and the kernel point positions are fixed.

However, our influence function h and thus our kernel function g are differentiable[1] with respect to $\tilde{x}_k$. Therefore, the kernel point positions are learnable parameters. We could consider learning one global set of $\{\tilde{x}_k\}$ for each convolution layer, but it would not bring more descriptive power than a fixed regular disposition. Instead, we designed a deformable KPConv inspired by the image deformable convolutions (J. Dai et al., 2017). The network generates a set of K shifts $\Delta(x) \in \mathbb{R}^{3\times K}$ for every convolution location and we define deformable KPConv as:

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g_{deform}\big(x_i - x, \Delta(x)\big) f_i \tag{IV-23}$$

With:

$$g_{deform}\big(y_i, \Delta(x)\big) = \sum_{k<K} h\big(y_i, \tilde{x}_k + \Delta_k(x)\big) W_k \tag{IV-24}$$

The offsets $\Delta_k(x)$ are obtained by a rigid KPConv mapping D_{in} input features to $3 \times K$ values, as shown in Figure 57. During training, the network learns the rigid kernel generating the shifts and the deformable kernel generating the output features simultaneously, but the learning rate of the first one is set to 0.1 times the learning rate of the global network, as in (J. Dai et al., 2017).

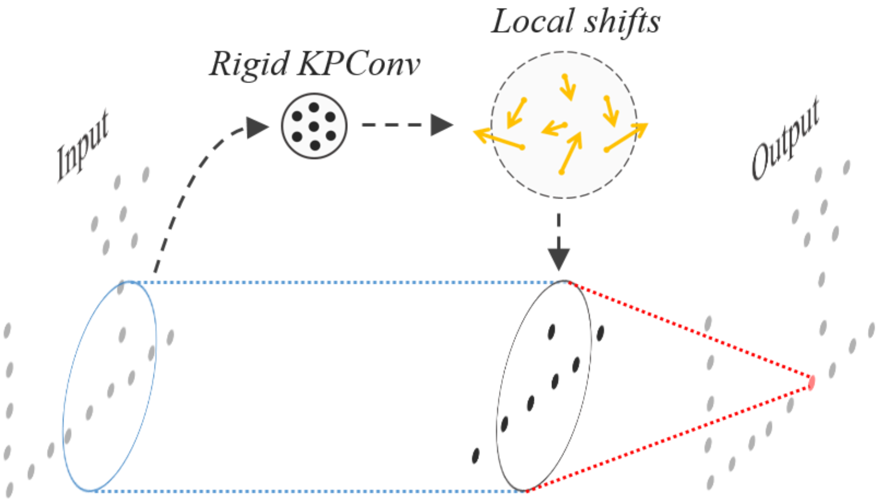


Figure 57. Deformable KPConv illustrated on 2D points.

---

[1] The function h_{linear} is actually differentiable almost everywhere. Empirically, it is the same as differentiable everywhere, and the stochastic gradient decent still works. It is the same for ReLU activations.

Unfortunately, this straightforward adaptation of image deformable convolutions is not adapted to point clouds. If we do not add anything to this design, the performances of deformable KPConv are worse than rigid KPConv. This bad behavior can be explained by the different nature of images and point clouds. Compared to an image, an input point cloud is sparse, and a kernel point can be inactive, as explained above. For an inactive kernel point at a certain location, the gradient of its shift $\Delta_k(x)$ is null. Therefore, this kernel point is not going to learn a new shift. The worse part of this behavior is that the kernel can “lose” their points during the convergence. If an active kernel point learns a shift that pulls it away from the neighbors, it will become inactive and the shift will not be learned anymore. Figure 58 illustrates the result of “lost kernel point” effect on a real network. More details on the networks we use are given later, but we can at least see that the kernel point positions, in red, are pulled away from the ground plane. This happens at any ground location because the shifts depend on the features of the previous layer, which will be very similar anywhere the point cloud represent a horizontal plane.

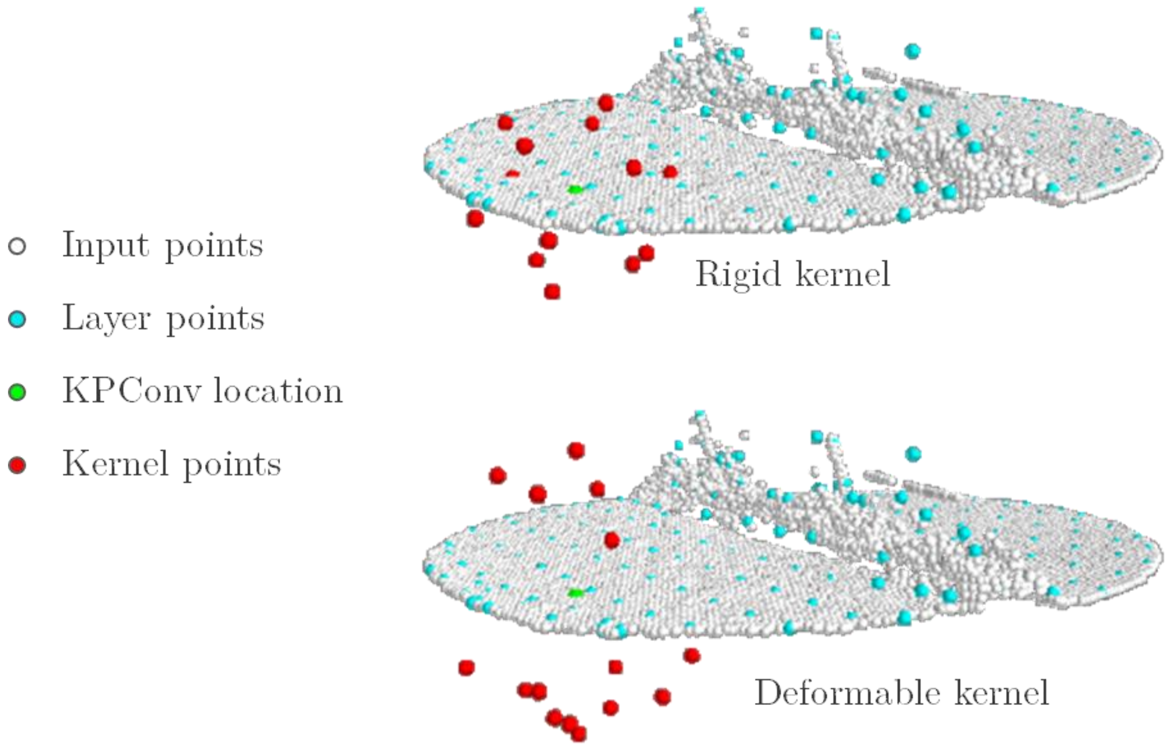


Figure 58. Illustration of the lost kernel points effect for deformable KPConv without regularization. All the points of the deformable kernel are above or under the ground. Their influence area does not reach any input point anymore, which means the gradients of their deformations are zero. They are stuck there.

A Solution to this problem is to use a regularization loss that helps the kernel points to fit the point cloud. We propose a "fitting" regularization loss $\mathcal{L}_{fit}$ that penalizes the distance between a kernel point and its closest neighbor among the input points. To avoid the kernel points to collapse together, we also add a "repulsive" regularization loss $\mathcal{L}_{rep}$ between each pair of kernel points when their influence area overlap. As a whole our regularization loss for all convolution locations x$\in \mathbb{R}^3$ is:

$$\mathcal{L}_{reg} = \sum_{x} \mathcal{L}_{fit}(x) + \mathcal{L}_{rep}(x) \tag{IV-25}$$

With:

$$\mathcal{L}_{fit}(x) = \sum_{k<K} \min_{y_i} \left(\frac{\left\| y_i - \left(\tilde{x}_k + \Delta_k(x)\right) \right\|}{\sigma} \right)^2 \tag{IV-26}$$

And:

$$\mathcal{L}_{rep}(x) = \sum_{k<K} \sum_{l \neq k} h\left(\tilde{x}_k + \Delta_k(x), \tilde{x}_l + \Delta_l(x)\right)^2 \tag{IV-27}$$

If the influence area of a kernel point does not reach any input point, the regularization loss still gives a gradient for its deformation, and helps it to get in range of input points. Therefore, our deformable KPConv is able to learn shifts that increase the number of active kernel points at each location. This is even more interesting than image deformable convolution. In addition to be able to reach further than normal convolutions and adapt the receptive fields of network features (like in image), deformable KPConv has more descriptive power than rigid KPConv because it uses all its kernel points at every location. The fitting effect of our regularization loss is shown on real point clouds in Figure 59.

IV.2.5. KPConv network layers

This section elucidates how we effectively put the KPConv theory into practice. For further details, we have released our code using Tensorflow library here: *https://github.com/HuguesTHOMAS/KPConv*.

Subsampling to deal with varying densities. As explained in the introduction, we use a subsampling strategy to control the density of input points at each layer. To ensure a spatial consistency of the point sampling locations, we favor grid subsampling. Thus, the support points of each layer, carrying the features, are chosen as the barycenters of the original input points contained in all non-empty grid cells. This is the same strategy used in Section III.3.2, with the same advantages. It is a very efficient way to deal with varying densities. Compared to a normalization strategy, our subsampling strategy alleviates the computational cost.

Pooling layer. To create architectures with multiple layer scales, we need to reduce the number of points progressively. As we already have a grid subsampling, we double the cell size at every pooling layer, along with the other related parameters, incrementally increasing the receptive field of KPConv. The features pooled at each new location can be obtained by either a max-pooling or a KPConv. We use the latter in our architectures and call it "strided KPConv", by analogy to the image strided convolution.

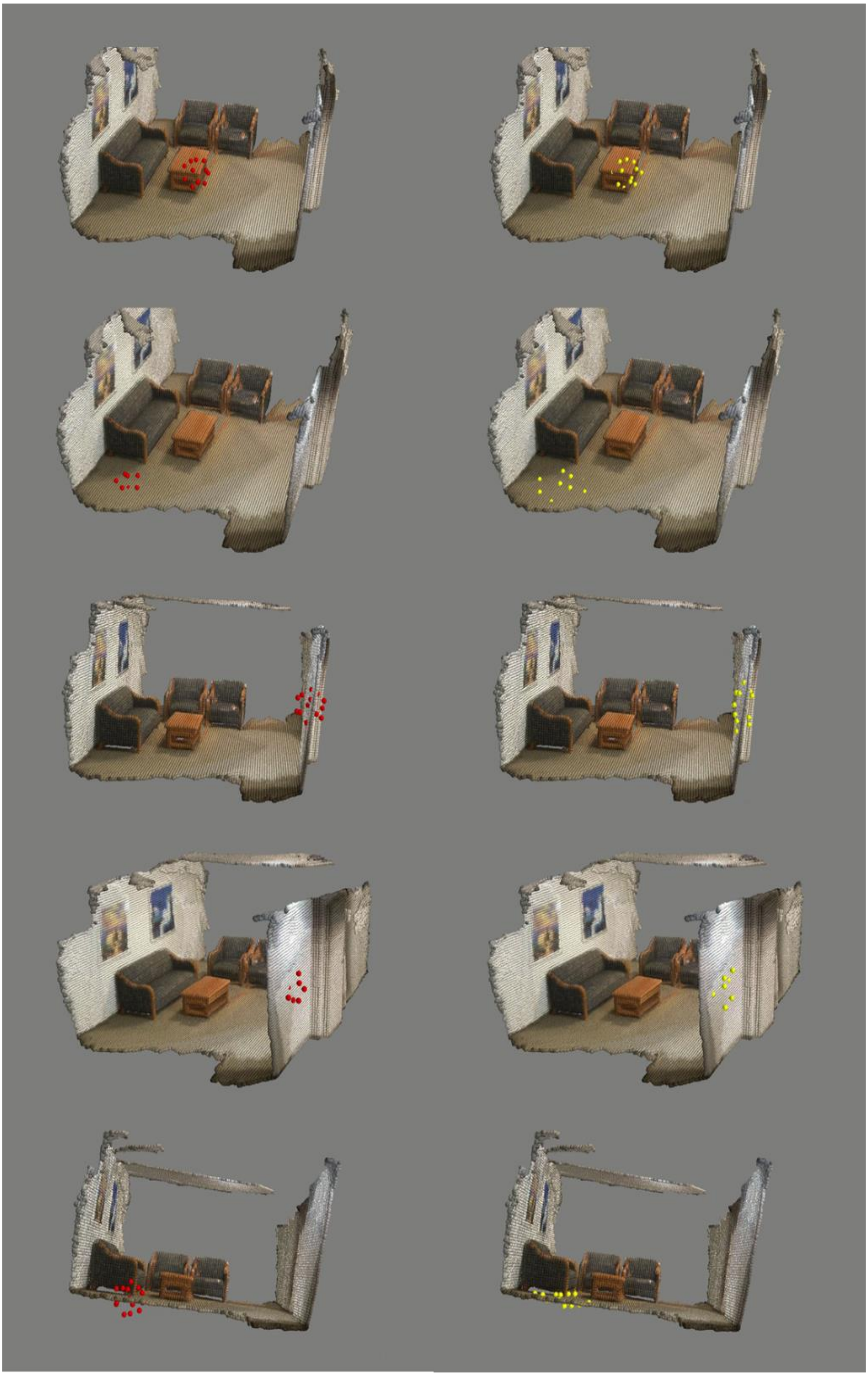


Figure 59. Illustration of deformed kernels (yellow) compared to the original rigid kernels of KPConv (red). With the regularization loss, the kernel points fit the point cloud geometry.

KPConv layer. Our convolution layer takes as input the points $\mathcal{P} \in \mathbb{R}^{N\times 3}$, their corresponding features $\mathcal{F} \in \mathbb{R}^{N\times D_{in}}$, and the matrix of neighborhood indices $\mathfrak{N} \in [\![1, N]\!]^{N'\times n_{max}}$. N' is the number of locations where the neighborhoods are computed, which can be different from N (in the case of "strided" KPConv). The neighborhood matrix is forced to have the size of the biggest neighborhood n_{max}. Because most of the neighborhoods comprise less than n_{max} neighbors, the matrix $\mathfrak{N}$ thus contains unused elements. We call them "shadow" neighbors, and they are ignored during the convolution computations (more details in Section IV.2.7.

Network parameters. Each layer j has a cell size dl_j from which we infer other parameters. As stated above, the kernel points influence distance is set as $\sigma_j = \Sigma \times dl_j$, and from σ, we infer the kernel point layout distance d_{center} and the convolution radius r. In the case of deformable kernels, the kernel points can be shifted further than d_{center}, which means we need a larger radius. We use a specific parameter ρ to control it: $r_j = \rho \times dl_j$, and allow larger receptive field.

This parameter has a very limited influence on the results, as it only defines a higher bound for the deformed kernel point distances to the center. We have to keep in mind that a higher value will cost computations but we observe in practice that a value of $\rho = 5.0$ is large enough and does not affect the computations much. Σ and ρ are proportional coefficients set for the whole network. Unless stated otherwise, we will use the following set of parameters, chosen thanks to their intuitive behavior and a small cross validation: $K = 15$, $\Sigma = 1.0$ and $\rho = 5.0$. The first subsampling cell size dl_0 is another crucial parameter but is tied to the scale of the scenes we process. It will thus depend on the dataset and, as stated above, $dl_{j+1} = 2 \times dl_j$.

IV.2.6. Kernel point network architectures

Combining analogy with successful image networks and empirical studies, we designed two network architectures for the classification and the segmentation tasks. Figure 60 shows diagrams detailing both architectures.

KP-CNN is a 5-layer classification convolutional network. Each layer contains two convolutional blocks, the first one being strided except for the first layer. Our convolutional blocks are designed like bottleneck ResNet blocks (He et al., 2016) with a KPConv replacing the image convolution, batch normalization and leaky ReLU activation (see Figure 61). In Figure 60, we show an example of point cloud from ModelNet40 dataset, subsampled at every layer. It illustrates how the convolution radius (red sphere) grows proportionally to the subsampling grid size. The green number above the layers are the features dimensions used in our blocks (D in Figure 61). After the last layer, the features are aggregated by a global average pooling and processed by the fully connected and softmax layers like in an image CNN. For the results with deformable KPConv, we only use deformable kernels in the last 5 KPConv blocks (2nd block from layer 3 and both blocks from layers 4 and 5).

KP-FCNN is a fully convolutional network for segmentation. The encoder part is the same as in KP-CNN, and the decoder part uses nearest upsampling to get the final pointwise features. Skip links are used to pass the features between intermediate layers of the encoder and the decoder. Those features

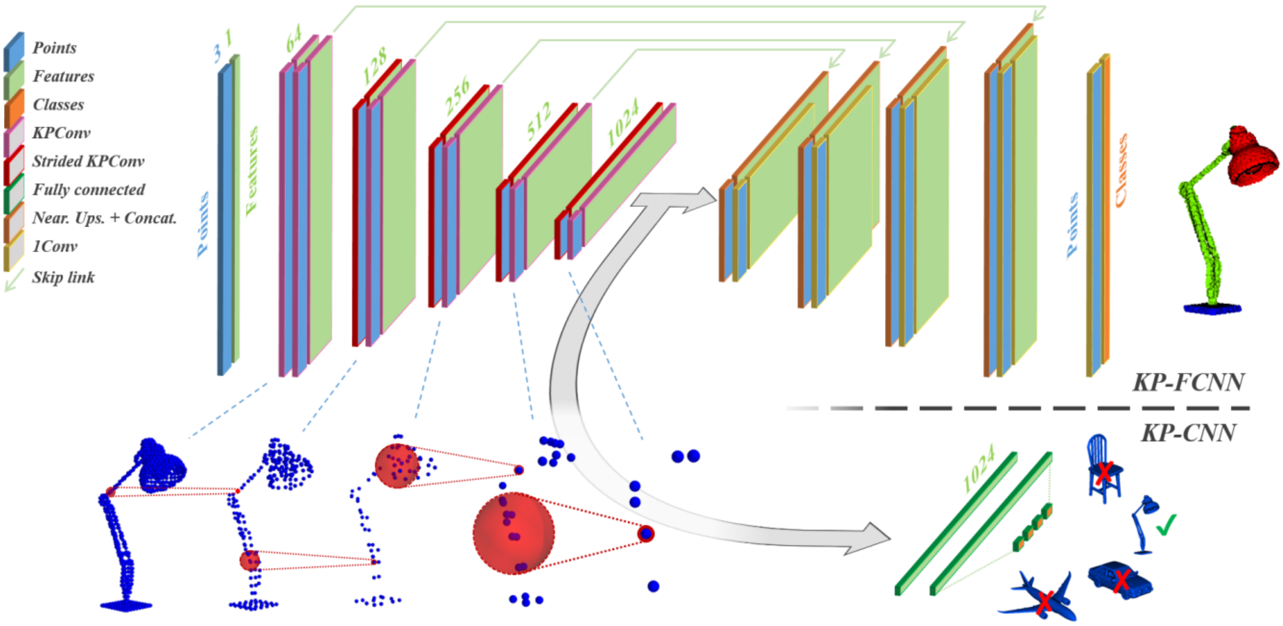

Points
Features
Classes
KPConv
Strided KPConv
Fully connected
Near. Ups. + Concat.
1Conv
Skip link
3
1
Points
Features
64
128
256
512
1024
Points
Classes
KP-FCNN
KP-CNN
1024

Figure 60. Illustration of our 2 network architectures for segmentation (top) and classification (bottom) of 3D point clouds. During a forward pass, features are transformed by consecutive operations (represented by edge colors) while points are fed to each layer as a support structure guiding the operations.

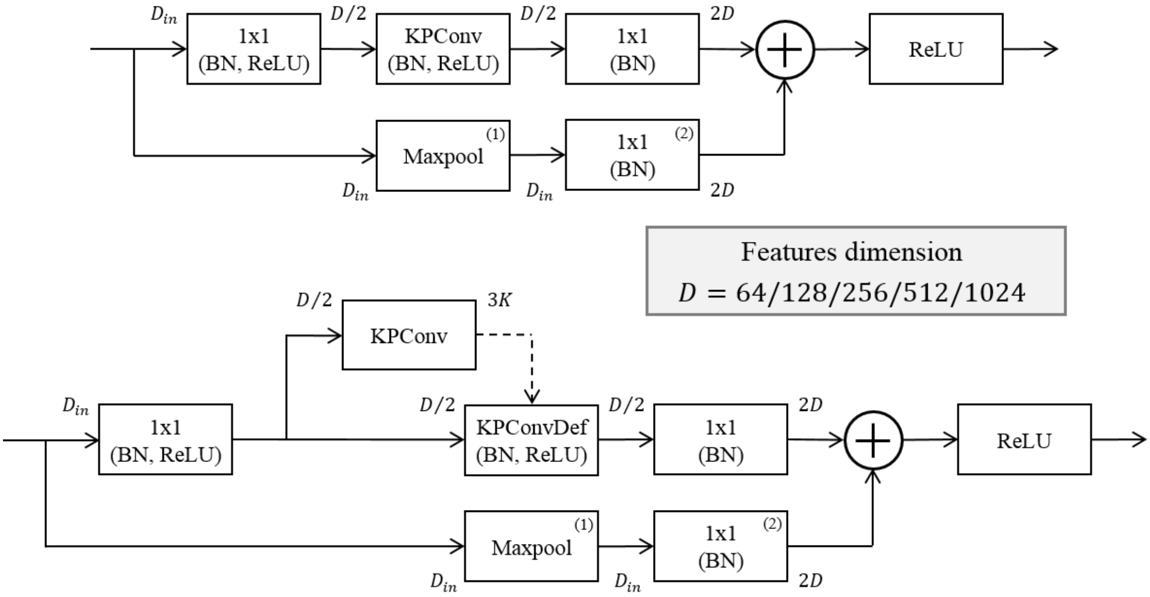

D_{in}
1x1 (BN, ReLU)
$D/2$
KPConv (BN, ReLU)
$D/2$
1x1 (BN)
$2D$
ReLU
Maxpool (1)
1x1 (BN) (2)
D_{in}
D_{in}
$2D$
Features dimension
$D = 64/128/256/512/1024$
$D/2$
KPConv
$3K$
D_{in}
1x1 (BN, ReLU)
$D/2$
KPConvDef (BN, ReLU)
$D/2$
1x1 (BN)
$2D$
ReLU
Maxpool (1)
1x1 (BN) (2)
D_{in}
D_{in}
$2D$

Figure 61. Convolutional blocks used in our architectures. Both rigid (top) and deformable (bottom) KPConv use ResNet connections, batch normalization and leaky ReLU. Optional blocks: shortcut max pooling$^{(1)}$ is only needed for strided KPConv, and shortcut 1×1 convolution$^{(2)}$ is only needed when $D_{in} \neq 2D$.

are concatenated to the upsampled ones and processed by a unary convolution, which is the equivalent of a 1×1 convolution in image or a shared MLP in PointNet. It is possible to replace the nearest upsampling operation by a KPConv, in the same way as the strided KPConv, but it does not lead to a significant improvement of the performances.

IV.2.7. Implementation details

In this section, we elucidate problems arising in the implementation of KPConv. Behind the theoretical definition of the operator and the network architectures, many issues have to be solved in practice. They are not relevant to the design of the method but to its effective application.

a. Variable batch size

The input of our network is a point cloud representing an object or a scene. We insisted a lot on the consistent spatial density as a strength of our network, but it is problematic for the implementation. Indeed, different objects do not have the same dimensionality, and thus do not need the same amount of points to be described for a given regular sampling. For example, a linear object like a pole needs fewer points than a plane like the ground. Therefore, our network needs to be able to process input point clouds of varying sizes.

In practice, a network processes batches of a few point clouds, but their sizes are variable, so we cannot stack them along a new “batch” dimension. We thus stack our point and feature tensors along their first dimension (number of points). As the neighbor and pooling indices do not point from one input cloud to another, each batch element is processed independently without any implementation trick. We only need to keep track of the batch index of each element in order to define the global pooling of KP-CNN. Since the number of points can vary a lot, we use a variable batch size by selecting as many elements as possible until a certain number of total batch points is reached. This limit is chosen so that the average batch size corresponds to the target batch size. A very similar batch strategy has already been described by (Hermosilla et al., 2018).

b. Parallel input queue

To optimize the speed of our KPConv networks, it is important to understand how the operations work, and how they should be split between GPU and CPU. In KPConv, all the operations use the points $\mathcal{P}$, the features $\mathcal{F}$ and the neighborhood indices $\mathfrak{N}$. They are implemented as Tensorflow standard operations like matrix additions, products, or indices gathering, which are automatically parallelized on a GPU. However, we first need to subsample the points $\mathcal{P}$ and compute the neighborhood indices $\mathfrak{N}$ for each layer. These variables are not modified by the network; they are just structural elements guiding the feature transformations. Therefore, we do not have to wait for the result of the layer $l-1$ to compute their value for a layer l. They can be computed before the network actually starts operating on the features. Furthermore, computing neighborhoods in a point cloud is not easily parallelizable on a GPU. As explained in Section II.3.2, accelerating structure can be used to compute neighborhoods very efficiently on a CPU.

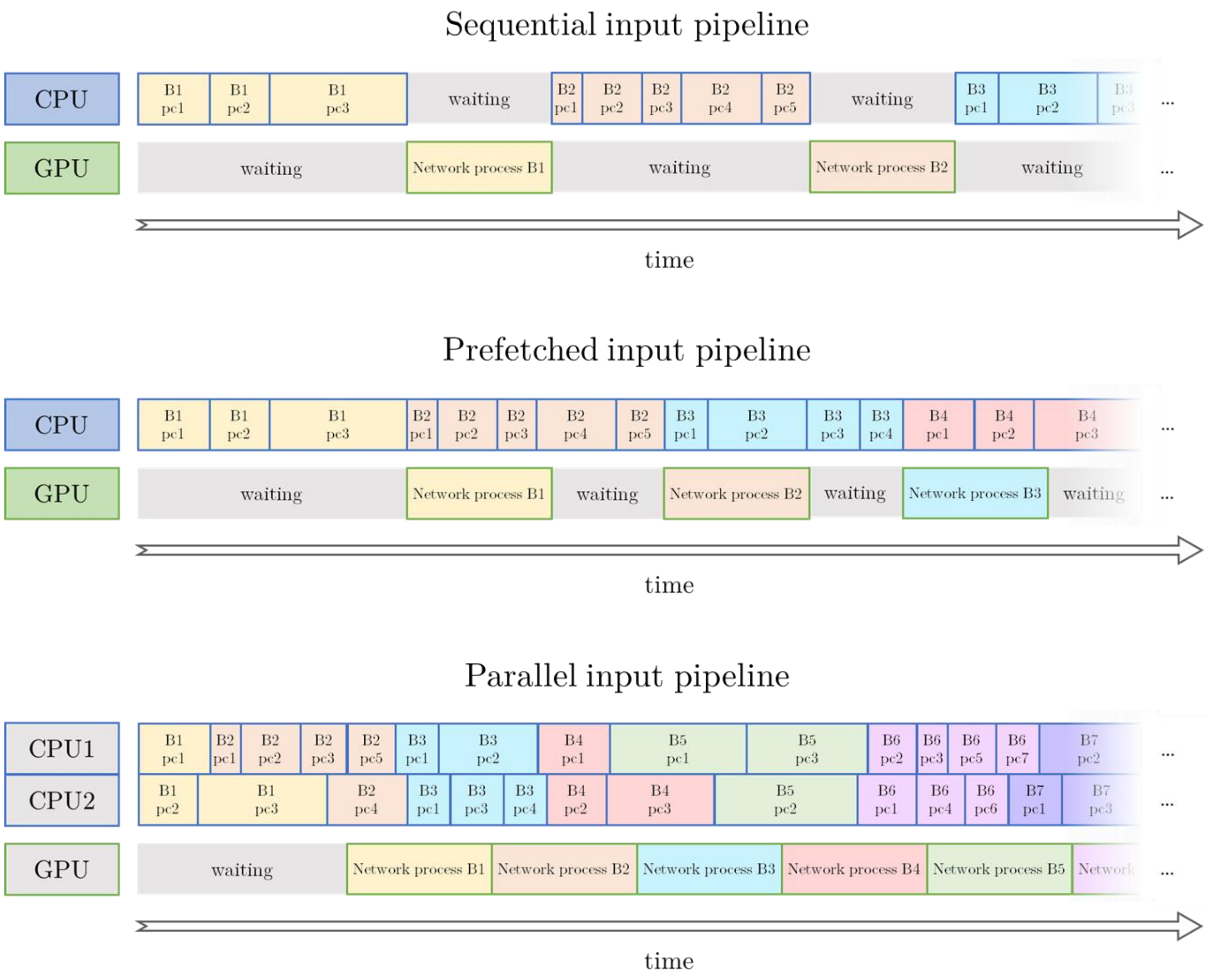

Sequential input pipeline
CPU
B1 pc1
B1 pc2
B1 pc3
waiting
B2 pc1
B2 pc2
B2 pc3
B2 pc4
B2 pc5
waiting
B3 pc1
B3 pc2
B3 pc3
...
GPU
waiting
Network process B1
waiting
Network process B2
waiting
...
time
Prefetched input pipeline
CPU
B1 pc1
B1 pc2
B1 pc3
B2 pc1
B2 pc2
B2 pc3
B2 pc4
B2 pc5
B3 pc1
B3 pc2
B3 pc3
B3 pc4
B4 pc1
B4 pc2
B4 pc3
...
GPU
waiting
Network process B1
waiting
Network process B2
waiting
Network process B3
waiting
...
time
Parallel input pipeline
CPU1
B1 pc1
B2 pc1
B2 pc2
B2 pc3
B2 pc5
B3 pc1
B3 pc2
B4 pc1
B5 pc1
B5 pc3
B6 pc2
B6 pc3
B6 pc5
B6 pc7
B7 pc2
...
CPU2
B1 pc2
B1 pc3
B2 pc4
B3 pc1
B3 pc3
B3 pc4
B4 pc2
B4 pc3
B5 pc2
B6 pc1
B6 pc4
B6 pc6
B7 pc1
B7 pc3
...
GPU
waiting
Network process B1
Network process B2
Network process B3
Network process B4
Network process B5
Network
...
time

Figure 62. Illustration of the parallelization of the input pipeline of KPConv Networks. We denote the batches as BX and the point clouds they comprise as pcX.

Considering these facts, we decided to implement a parallel input queue to preprocess these variables before feeding them to the network. On CPU, we compute the points $\mathcal{P}$ and the neighborhood indices $\mathfrak{N}$ for every layer of the network, in addition to other useful variables. On GPU, we feed the network with these inputs and compute the rest of the operations processing the features. As illustrated in Figure 62, our input pipeline uses prefetching and CPU thread parallelization. With these tricks, the speed of our networks is multiplied by 3 or 4 depending on the dataset.

c. Ignoring shadow neighbors

We explained the concept of “shadow neighbors” previously but did not detail how they were ignored in the convolution operations. Our convolution operations are parallelized on a GPU, and we cannot have a varying neighborhood size. The neighborhood indices are stored in a matrix $\mathfrak{N} \in [\![1, N]\!]^{N' \times n_{max}}$. There are N' locations and each neighborhood is a list that contains n_{max} indices. However, for most locations, there are some shadow neighbors among these n_{max} indices. During the convolution operations, the features of the neighbors are gathered with these indices. Each index points to one of

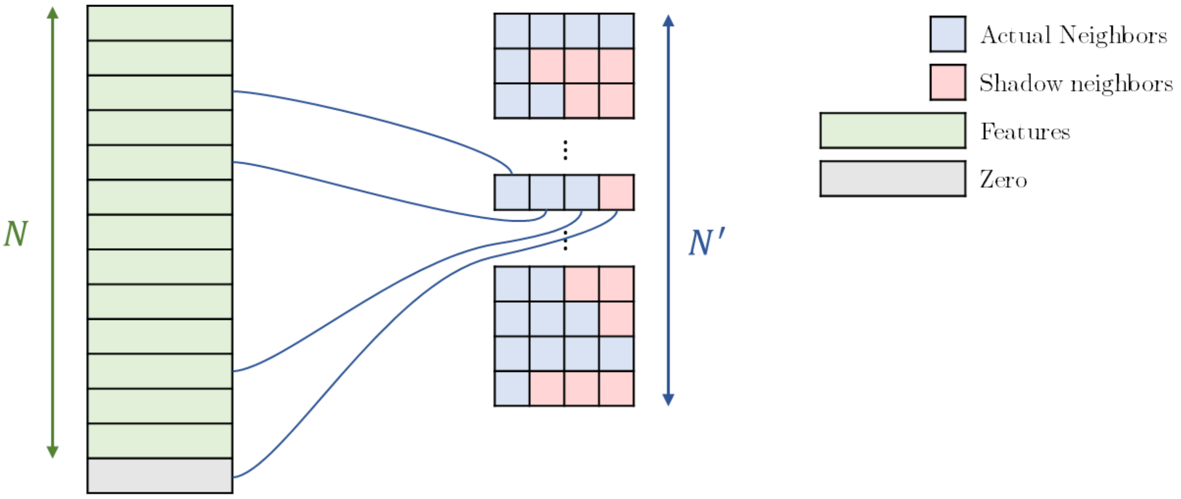


Figure 63. Illustration of the neighbor features gathering in KPConv, with the shadow neighbor trick. Each blue line indicates where a neighbor index points.

the N rows of matrix $\mathcal{F}$, and the index of the shadow neighbor, either points to one of these features, or create a pointing error in the code. The solution we found is very simple, we stack a row of zeros at the end of the feature matrix $\mathcal{F}$, and make the shadow neighbors point to it. Each shadow neighbor is encoded with the index $N + 1$. As we explained below, a zero feature is equivalent to empty space for our convolution; therefore, the result of KPConv is not influenced by these shadow neighbors. This trick is illustrated in Figure 63.

d. Hybrid neighborhood

The shadow neighbor trick allows computing KPConv without errors, but these shadow neighbors are still part of the computations the GPU has to make. This a big loss of efficiency for our network. It is even worse when we consider our batch stacking strategy. The neighborhood matrix is shared for each point cloud of a batch, which means n_{max} is the largest neighborhood size among several point clouds. In practice, we observe that this maximum value is very high, more than the double of most neighborhood sizes.

In order to alleviate the computational cost, we thus decided to limit the value n_{max}. For the largest neighborhoods, we ignore some of the neighbors, simply by cropping the neighborhood matrix to a new size $N' \times n_{lim}$, which considerably helps the network efficiency. The new n_{lim} value is chosen as the 80th percentile of the neighborhood sizes distribution. This choice ensures that 80% of the neighborhoods are untouched, and in practice, given the distribution of neighborhood sizes on real datasets, n_{lim} value is half of n_{max}. Despite 20% of the neighborhoods being cropped, only a few of them are really affected and the overall network performances are the same as without cropping.

In our implementation, the neighborhood indices are ordered by distance. Therefore, this cropping is equivalent to a hybrid neighborhood defined in Section II.3.1. We observed no difference between the performances of our networks with randomly ordered neighborhoods or hybrid neighborhood.

e. Input features and empty space

It is important to understand that our convolution is not affected by the points that carry features equal to zero. These points are equivalent to empty space. Therefore, point clouds with no features like colors are always fed to the networks with a constant feature equal to one. The value of this feature is not important, but it has to be non-zero. The network thus can "see" the points as opposed to the empty space. This constant feature encodes the geometry of the point cloud for the network. For some of the datasets, we will use additional features like colors of relative point positions. In this case, we still keep the constant feature and append it to the other features. Without it, the black/dark points would be ignored and the network would have a biased view of the geometry of the scene.

IV.2.8. Learning strategy for different tasks

Our networks can process point clouds for the following tasks:

- object classification
- object part segmentation
- scene semantic segmentation

As explained in the introduction, we focus on the semantic segmentation of real scenes, but evaluate our method on the other tasks for benchmarking purposes. For each task, we have a specific training strategy.

a. Object classification

For the object classification datasets, we use KP-CNN. We only use one dataset for this task, ModelNet40, which consists of several point clouds of different objects. We rescale the objects to fit them into a unit sphere, and consider units to be meters. Then we just have to subsample these point clouds with a grid of size dl_0 and to feed the network with them. Our network is able to process very large point clouds with $dl_0 = 0.02m$. If we compare it to a grid CNN, it is equivalent to a 100^3 voxel grid.

We use a Momentum Gradient Descent optimizer to minimize a cross-entropy loss, with a batch size of 16, a momentum of 0.98 and an initial learning rate of 10^{-3}. Our learning rate is scheduled to decrease exponentially, and we choose the exponential decay to ensure it is divided by 10 every 100 epochs. A 0.5 probability dropout is used in the final fully connected layers. The network converges in 200 epochs. In the case of deformable kernels, the regularization loss is added to the output loss with a multiplicative factor of 0.1.

b. Object part segmentation

For this task, we use KP-FCNN architecture with the same parameters as in the classification task, a batch size of 16, a momentum of 0.98 and an initial learning rate of 10^{-3}. The same learning rate decay is used but we do not add dropout on the last segmentation layers.

c. Scene segmentation

The scene segmentation datasets are more interesting in our opinion, as they offer more diversity, and come from real scenes. We conduct experiments on four different datasets for this task, to evaluate the performances of KPConv in many scenarios. With this task, we face the same issues as in Section III.3.4. The dataset is too big to be processed as a whole by our network, but we can segment smaller parts of the scene and we need to elaborate a learning strategy.

We choose to use KP-FCNN to segment spherical subclouds of the scene. The spheres containing the input subclouds are smaller than the whole dataset but large enough to cover several objects. We call their radius R and ensure its value is equal to $50 \times dl_0$ (in accordance to ModelNet40 experiment). R and dl_0 are chosen for each dataset independently, considering the size of the objects and the level of details needed. Lower values mean more details, but less spatial context for big objects. Because our network processes a whole sphere and not a single point, our active learning strategy, used for had-crafted features in Chapter III, is not viable with KPConv networks. Furthermore, the training of a neural network is very long and it would not be feasible to use the same kind of trial and error process. We elaborate two strategies for the picking of input spheres across the scenes.

The first one, called **random picking**, consists of picking an equal number of sphere centers randomly in each class of the dataset. It is the same as the random picking of training points for hand-crafted features. The main weakness of this strategy is that it is sensible to varying densities. The areas with a higher density have a higher probability to be picked and will thus be seen more times than the lower density areas. This creates a bias in the network training.

The second strategy, named **regular picking**, is designed to be robust to varying densities and to pick the same amount of spheres in every locations of the scene. We ensure the spatial regularity of the picking by keeping track of a potential value at each point. When a sphere is picked, we update the potential of the surrounding points. To keep a smooth potential function, the surrounding potentials are updated with Tukey weights defined as a function of the distance to the sphere center:

$$Tukey(d) = \begin{cases} \left(1 - \left(\frac{d}{d_{Tuk}}\right)^2\right)^2 & if\ d \leq d_{Tuk} \\ 0 & if\ d > d_{Tuk} \end{cases} \qquad \text{(IV-28)}$$

The points closer than the distance d_{Tuk} are updated. This range is chosen smaller than the actual input sphere radius: $d_{Tuk} = R/3$. It creates overlapping between input spheres and helps the training. Each new sphere is picked as the minimum of the potentials across the scenes. This picking strategy is illustrated in Figure 64.

We use the regular picking strategy with a range $d_{Tuk} = R/3$ for both training and test in our experiments. It helps the network to be robust to varying densities. The range, lower than the input sphere radius, ensures that every point is tested multiple times by different sphere locations. At training, it helps the network with more variety in the data. During the test phase, it gives more robustness to the predictions, which are computed with a voting scheme (the predicted probabilities for each point are averaged).

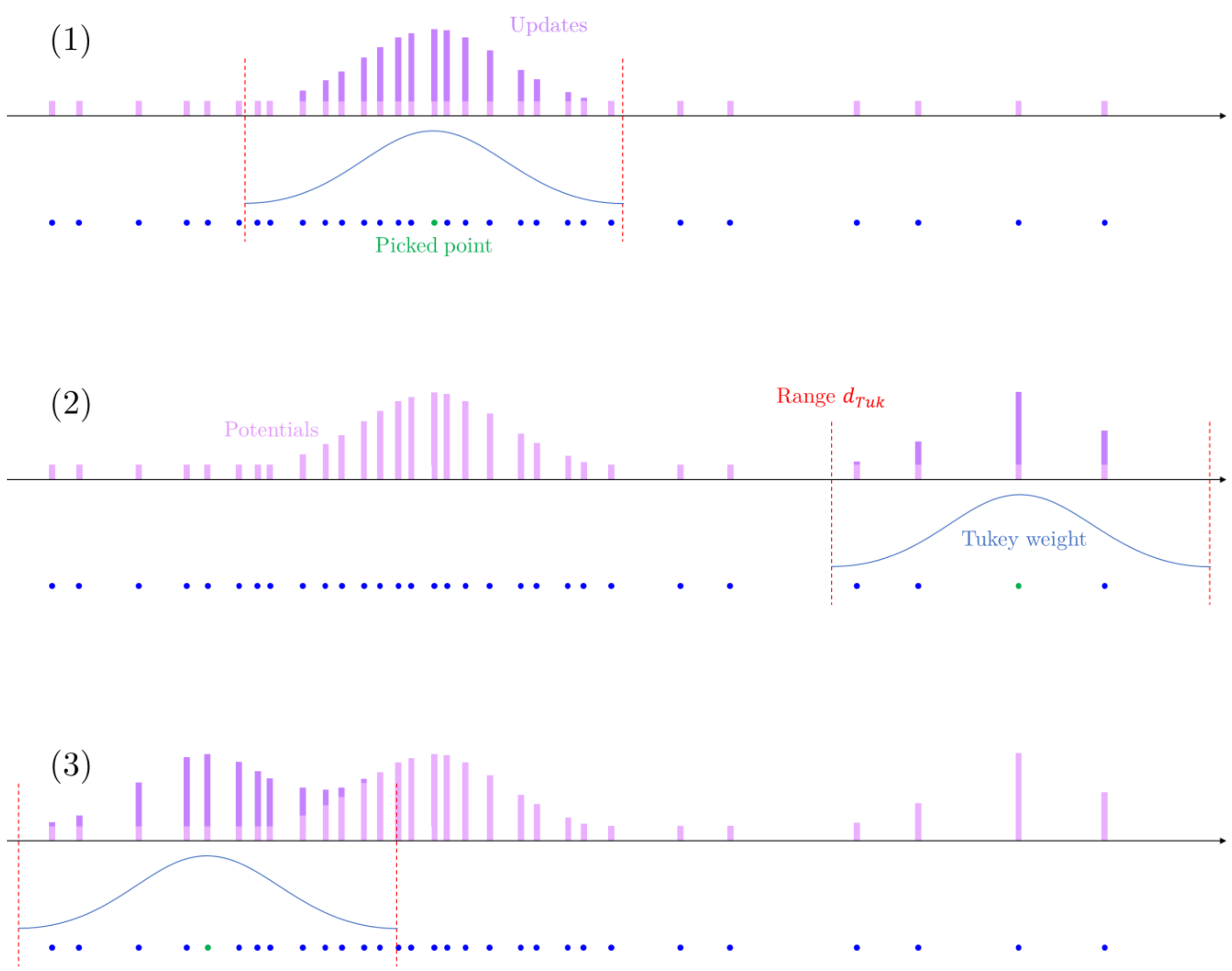


Figure 64. Illustration of the spatially regular picking strategy with potential updates in 1D for clarity. This strategy helps picking input spheres regularly in the space, without being biased by the varying densities.

The training parameters are changed with a learning rate of 10^{-2}, and a batch size of 10. We use the same learning rate schedule and no dropout. Among all experiments, the network needs 400 epochs at most to converge. We can generate any number of input spheres, so we define an epoch as 500 optimizer steps, which is equivalent to 5000 spheres seen by the network.

d. Data augmentation

Eventually, as usually done in Deep Learning, we use data augmentation strategies to increase the variety of input data and help the robustness of our network. In this paragraph, we list all the strategies used for data augmentation in our experiments. These operations are part of the input pipeline, running in parallel on CPU, as they operate on the input point clouds (or subclouds), before they are fed to the network.

Anisotropic Scaling. The input point clouds are scaled independently in each dimension. The scaling factor is picked uniformly in $[0.8, 1.2]$. Our code also allows choosing an isotropic scaling instead. Nevertheless, we found that the anisotropic deformations are beneficial for the robustness of the network.

Vertical rotation. In some experiments, we rotate the point clouds around the vertical axis with a random angle in $[0, 2\pi]$. The vertical direction has a geometrical meaning as most objects are oriented by gravity.

Flipping. We can choose to flip the input point cloud along the three dimensions. This strategy can replace rotation if the dataset is aligned. Alternatively, it can complete vertical rotation augmentation. In that case, we only have to flip along one of the first two dimensions, as the other is redundant.

Noise. A Gaussian noise is added to the point coordinates to perturb their positions. It helps the network to be robust to the noise in real data. The standard deviation of the noise is set with a smaller value than dl_0, to avoid losing too much details in the shapes.

Color annealing. For colorized datasets, we found that adding color features helps some classes but harms others. We decided to help the network to build representations that use both geometric features and colors feature, by erasing the color features of some input clouds. We introduce a color annealing probability of 0.2, which is sufficient to force the network to learn geometric features.

IV.3. Experiments

In this experiment section, we evaluate the performances of our two neural networks for classification and segmentation of 3D point clouds. We first present the experiments conducted for the three tasks of object classification, object part segmentation and scene semantic segmentation. They were conducted on six datasets, all presented in Section II.2.2:

- ModelNet40 (Wu et al., 2015).
- ShapeNetPart (Yi et al., 2016)
- Scannet (A. Dai et al., 2017)
- S3DIS (Armeni et al., 2016)
- Semantic3D (Hackel et al., 2017)
- Paris-Lille-3D (Roynard et al., 2018a)

We give detailed quantitative results, qualitative results, and computational costs in terms of memory and speed. Overall, rigid and deformable KPConv both perform very well, outperforming state-of-the-art methods in almost any scenario, which shows the strength of our convolution operator and the networks we build with it. We find that rigid KPConv achieves better performances on simpler tasks, like object classification, or small segmentation datasets. Deformable KPConv thrives on more difficult tasks, like large segmentation datasets, which offers many object instances and a greater diversity.

In addition to the confirmation that KPConv is able to learn strong representations that lead to top notch performances, we want more insights on these representations. We first conduct an ablation study to understand the role of the crucial parameter K. We learn that deformable KPConv is more robust to a lower number of kernel points, which implies a greater descriptive power. Then we visualize geometric features that our network was able to learn in different layers. These visualizations confirm our intuitions about the representations that could be learned by neural networks on point clouds. They offer a particularly useful insight for the understanding of the mechanisms involved in the learning of deep neural networks on point clouds. Eventually, we explain the stronger performances of deformable KPConv in difficult scenarios. A qualitative study of KPConv Effective Receptive Field (Luo et al., 2016) shows that deformable kernels improve the network ability to adapt to the geometry of the scene objects.

IV.3.1. 3D shape classification on ModelNet40

Our first experiment was conducted on the most commonly used model dataset, ModelNet40. Although it is not a semantic segmentation dataset, it is a very common experiment to assess the performances of 3D deep learning methods. For benchmarking purposes we use the point clouds provided by (Qi et al., 2017b) and follow standard training/test splits. We choose to ignore the normals as they come from the artificial nature of the data. In real point clouds, the normals have to be computed.

As we explained earlier, we set the first subsampling grid size to $dl_0 = 2cm$. We do not add any feature as input and only use the constant feature equal to 1. Among the available augmentation procedures, we choose to scale, flip and perturb the point coordinates. On this dataset, our network handles batches comprising around **109,000** points with an average batch size of 16 points clouds. Each point cloud thus counts **6,800** points on average. More details on training speeds are given in Section IV.3.4.

As shown in Table 13, our networks outperform other state-of-the-art methods using only points. We do not take into account methods using normals as additional input. In terms of overall accuracy, KPConv outperforms competing point convolutions by at least 0.6 points (Atzmon et al., 2018) and up to 2.7 points (Groh et al., 2018).

We also notice that rigid KPConv performances are slightly better than deformable KPConv. We suspect that it can be explained by the task simplicity. If deformable kernels add more descriptive power, they also increase the overall network complexity, which can disturb the convergence or lead to overfitting on simpler tasks like this shape classification

IV.3.2. 3D shape segmentation on ShapeNetPart

Our second experiment evaluates KPConv on ShapeNetPart, which is an object part segmentation task. This task is the same as scene semantic segmentation in theory, but instead of scenes, point clouds representing objects are segmented. Due to the smaller object sizes, compared to real 3D scenes, this dataset has been used more often in the literature.

We use the same augmentation procedure as for ModelNet40, and ignore normals for the same reason. The positions (x, y, z) are added as additional features to the constant 1, like in previous work. The point clouds are smaller than in ModelNet40, but we keep the same parameters as before. Our network thus handles batches comprising around **38,000** points with an average batch size of 16 points clouds. Each point cloud thus counts **2,300** points on average. In this lighter setup, our network processes the batches faster as shown in Section IV.3.4.

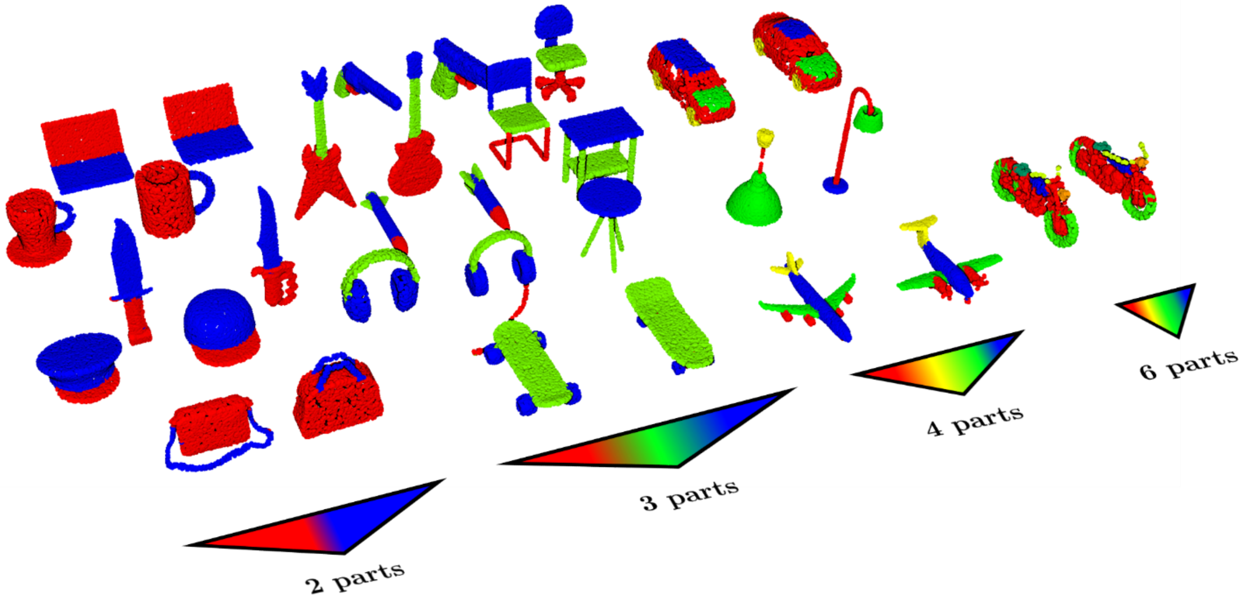


Figure 65. Example of objects from ShapeNetPart dataset, segmented with KP-FCNN. The different objects have varying number of parts.

Table 13 shows the instance average, and the class average mIoU, details each class mIoU in Appendix A.1. KP-FCNN outperforms all other algorithms, including those using additional inputs like images or normals. Shape segmentation is a more difficult task than shape classification, and we see that KPConv has better performances with deformable kernels. Some examples of segmentations results are shown in Figure 65. We provide more examples well-segmented objects of each class in Appendix A.3.

Table 13. Shape Classification and Segmentation results. For generalizability to real data, we only consider scores obtained without shape normals on ModelNet40 dataset. The metrics are overall accuracy (OA) for ModelNet40, class average IoU (mcIoU) and instance average IoU (mIoU) for ShapeNetPart.

	ModelNet40	ShapeNetPart	
Methods	OA	mcIoU	mIoU
SPLATNet (Su et al., 2018)	-	83.7	85.4
SGPN (W. Wang et al., 2018)	-	82.8	85.8
3DmFV-Net (Ben-Shabat et al., 2017)	91.6	81.0	84.3
SynSpecCNN (Yi et al., 2017)	-	82.0	84.7
RSNet (Huang et al., 2018)	-	81.4	84.9
SpecGCN (C. Wang et al., 2018)	91.5	-	85.4
PointNet++ (Qi et al., 2017b)	90.7	81.9	85.1
SO-Net (J. Li et al., 2018)	90.9	81.0	84.9
PCNN by Ext (Atzmon et al., 2018)	92.3	81.8	85.1
SpiderCNN (Xu et al., 2018)	90.5	82.4	85.3
MCConv (Hermosilla et al., 2018)	90.9	-	85.9
FlexConv (Groh et al., 2018)	90.2	84.7	85.0
PointCNN (Y. Li et al., 2018)	92.2	84.6	86.1
DGCNN (Y. Wang et al., 2018)	92.2	85.0	84.7
SubSparseCNN (Graham et al., 2018)	-	83.3	86.0
Rigid KPConv	**92.9**	85.0	86.2
Deformable KPConv	92.7	**85.1**	**86.4**

In addition to these strong performances, we show in Figure 66, some examples of mis-segmented objects. We can see that the errors often come from data related issues. The objects are sometimes misaligned, like the first plane pointing backwards. Other mistakes come from wrong object classes, like the lightbulb considered as a lamp, which is different. Some objects also have unusual shapes that the network has never seen like the first bag or the second plane. There also are annotation mistakes that we can see on the skateboards. Some examples are just very hard like the second bag where the handle on the main part is practically invisible in 3D, or the first lamp, which really looks like it stands on the floor and not like it hangs on the ceiling.

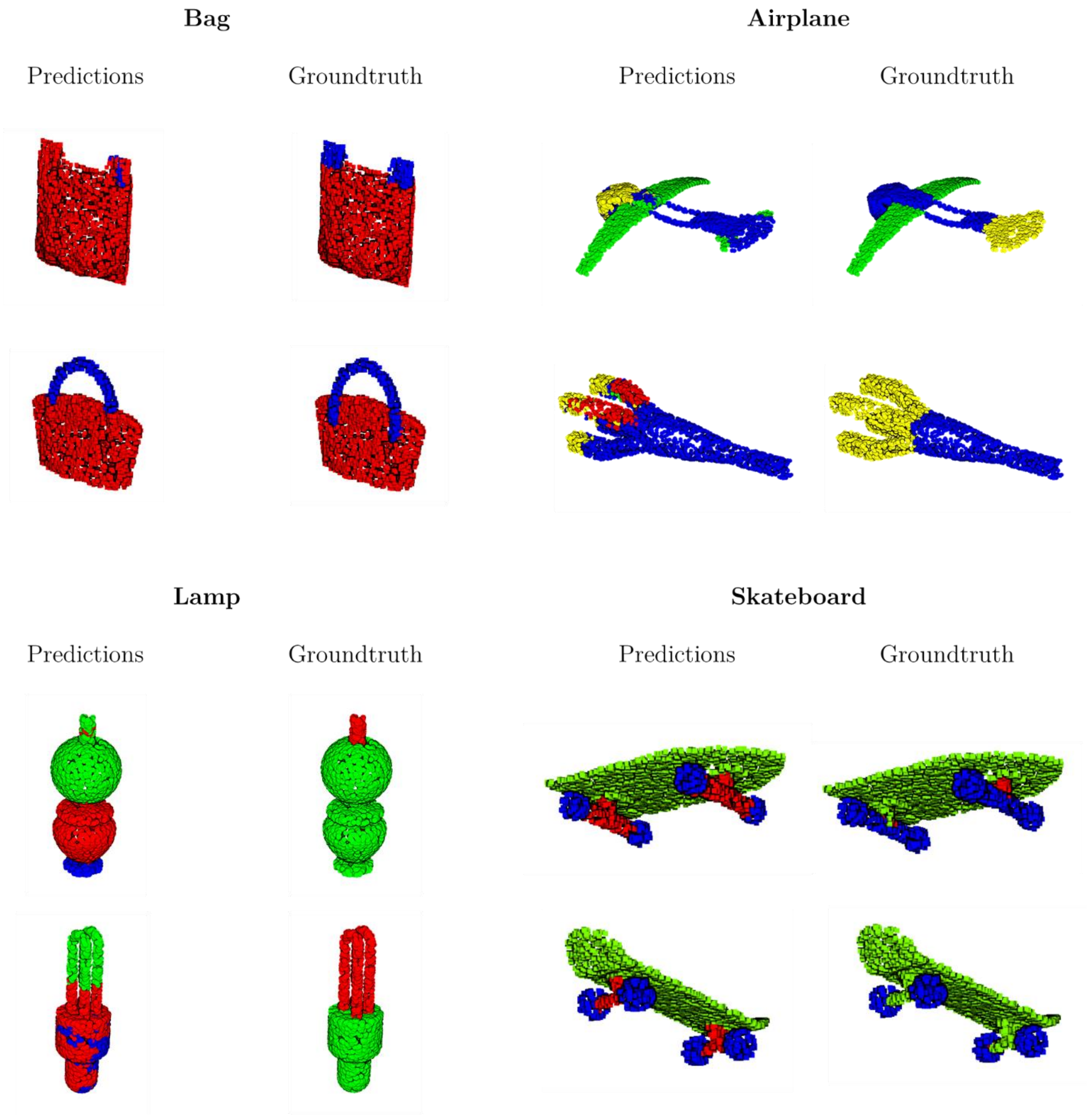

Bag
Predictions
Groundtruth
Airplane
Predictions
Groundtruth
Lamp
Predictions
Groundtruth
Skateboard
Predictions
Groundtruth

Figure 66. Example among the worst results of our method on ShapeNetPart dataset. We can see that the errors are due to groundtruth mistakes, object misalignment or unusual shapes.

IV.3.3. 3D scene segmentation on several datasets

Our third experiment shows how our segmentation architecture generalizes to real indoor and outdoor data. To this end, we choose to test our network on 4 datasets of different natures. Scannet, for indoor cluttered scenes, S3DIS, for indoor large spaces, Semantic3D, for outdoor fixed scans, and Paris-Lille-3D, for outdoor mobile scans. Because outdoor objects are larger than indoor objects, we use $dl_0 = 6cm$ on Semantic3D and Paris-Lille-3D, and $dl_0 = 4cm$ on Scannet and S3DIS. Given our rule to have $R = 50 \times dl_0$, we thus have input radiuses of $3m$ for outdoor datasets and $2m$ for indoor datasets.

As shown in Table 14, our architecture ranks second on Scannet and outperforms all other segmentation architectures on the other datasets. Compared to other point convolution architectures (Atzmon et al., 2018; Y. Li et al., 2018; Xu et al., 2018), KPConv performances exceed previous scores by 19 mIoU points on Scannet and 9 mIoU points on S3DIS. SubSparseCNN (Graham et al., 2018) score on Scannet was not reported in their original paper, so it is hard to compare without knowing their experimental setup. We can notice that, in the same experimental setup on ShapeNetPart segmentation, KPConv outperformed SubSparseCNN by nearly 2 mIoU points.

Table 14. mIoU scores on 3D scene segmentation datasets. Scannet, Semantic3D (Sem3D) and Paris-Lille-3D (PL3D) scores are taken from their respective online benchmarks (reduced-8 challenge for Semantic3D). S3DIS scores are given for Area-5 (see appendix for k-fold).

Methods	Scannet	Sem3D	S3DIS	PL3D
Pointnet (Qi et al., 2017a)	-	-	41.1	-
Pointnet++ (Qi et al., 2017b)	33.9	-	-	-
SnapNet (Boulch et al., 2017)	-	59.1	-	-
SPLATNet (Su et al., 2018)	39.3	-	-	-
SegCloud (Tchapmi et al., 2017)	-	61.3	48.9	-
RF_MSSF (ours)	-	62.7	49.8	56.3
Eff 3D Conv (Zhang et al., 2018)	-	-	51.8	-
TangentConv (Tatarchenko et al., 2018)	43.8	-	52.6	-
MSDVN (Roynard et al., 2018b)	-	65.3	54.7	66.9
RSNet (Huang et al., 2018)	-	-	56.5	-
FCPN (Rethage et al., 2018)	44.7	-	-	-
PointCNN (Y. Li et al., 2018)	45.8	-	57.3	-
PCNN (Atzmon et al., 2018)	49.8	-	-	-
SPGraph (Landrieu and Simonovsky, 2017)	-	73.2	58.0	-
ParamConv (S. Wang et al., 2018)	-	-	58.3	-
SubSparseCNN (Graham et al., 2018)	**72.5**	-	-	-
Rigid KPConv	68.6	**74.6**	65.4	72.3
Deformable KPConv	68.4	73.1	**67.1**	**75.9**

Among these 4 datasets, KPConv deformable kernels improved the results on Paris-Lille-3D and S3DIS while the rigid version was better on Scannet and Semantic3D. If we follow our assumption, we can explain the lower scores on Semantic3D by the lack of diversity in this dataset. Indeed, despite comprising 15 scenes and 4 billion points, it contains a majority of ground, building and vegetation points and a few real objects like cars or pedestrians. Although this is not the case of Scannet, which comprises more than 1,500 scenes with various objects and shapes, our validation studies are not reflected by the test scores on this benchmark. We found that the deformable KPConv outperformed its rigid counterpart on several different validation sets (see Section IV.3.5). As a conclusion, these results show that the descriptive power of deformable KPConv is useful to the network on large and diverse datasets.

Illustrations of segmented scenes on these four datasets are shown in Figure 67, Figure 68, Figure 69 and Figure 70. We show the results, the groundtruth, and in addition, we highlight the errors in red for clarity. Several comments can be made about these results.

First, we can compare with our multiscale feature predictions. We can see that the errors are more located on coherent parts of the scene, as opposed to the random error patches of the multiscale framework. Furthermore, in Paris-Lille-3D, we see that our network detects the tree trunks as vegetation in most cases and that the cars close to trees are well segmented. All these clues show that our neural network has a better context awareness. It is able to aggregate the information of adjacent objects without confusing them. This kind of aggregation scheme is typical in neural networks and is part of the reason they outperform handcrafted features.

We also see several spots where our network mistakes are understandable.

- On Paris-Lille-3D (Figure 67), the network is unable to differentiate a tree trunk from a contiguous trashcan. This can come from the fact that the tree trunks in the training scenes are always separated from other objects.

- On Paris-Lille-3D (Figure 67), our method was able to distinguish the flowerpots from the plants inside them. This distinction leads to errors because there is no flowerpot class and they were annotated as vegetation. Our network understood that it was not really vegetation, but could not find any suitable class for them. Therefore, one is classified as barrier and the other as trash can.

- On Semantic3D (Figure 68), we can see an error for the small bush in the foreground. The annotation says it is high vegetation and the network low vegetation. Given the size of the bush, we can say this is an annotation mistake.

- On Semantic3D (Figure 68), there are many points on the ground under cars and around some bollards that are annotated as hardscape, which is another annotation mistake. Our network manage to separate the bollard and the cars from the ground better than the annotation.

- On S3DIS (Figure 69), A column on the right was confused with a wall. We can see that the other column was well segmented, as it was clearly extruded from the wall, but the other is not as obviously extruded, and even a human could think this is a part of the wall.

- On S3DIS (Figure 69), we can see that the boards are not entirely segmented. When considering this class is indistinguishable in 3D and the network only relies on color to segment them, it is remarkable that it manages to find a large portion of them. It means that our network is able to learn color features in addition to geometric features.

- On Scannet (Figure 70), the annotation sometimes seems sloppy. In the background on the left, we can see a wall that was not well annotated, with part of it denoted as door. The network predictions for this part seem closer to the reality.

- On Scannet (Figure 70), some classes are confusing. On the right a furniture is distinctly segmented by the network, but with the wrong class. It is predicted as cabinet, were the annotation say otherfurniture. It looks like a cabinet, but the class otherfurniture sometimes contains elements of other classes that were not well annotated.

We provide more illustrations of segmented scenes in Appendix A.2.

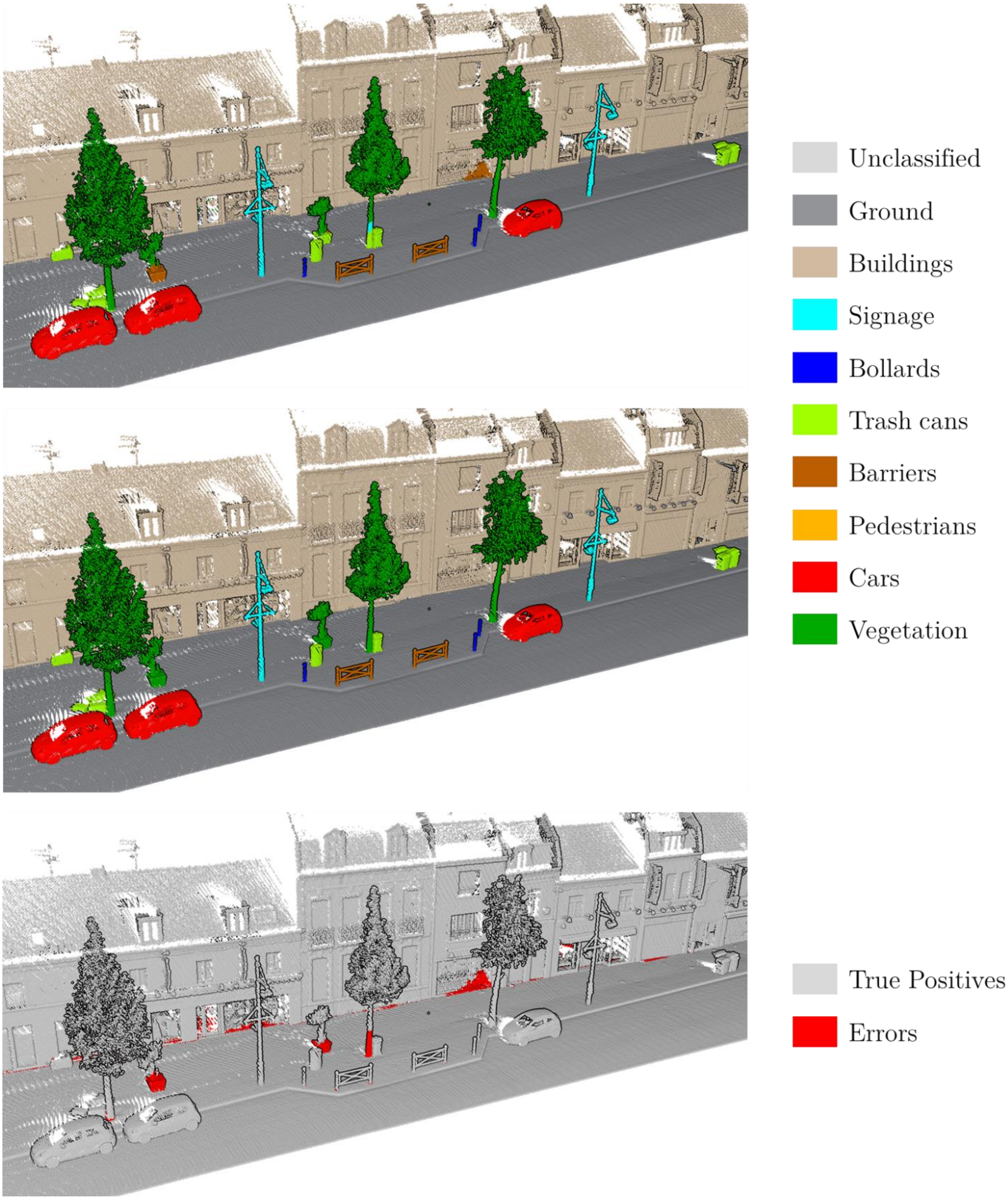

Unclassified
Ground
Buildings
Signage
Bollards
Trash cans
Barriers
Pedestrians
Cars
Vegetation
True Positives
Errors

Figure 67. Paris-Lille-3D scene segmented with KP-FCNN. Top: predictions. Middle: Groundtruth. Bottom: Errors.

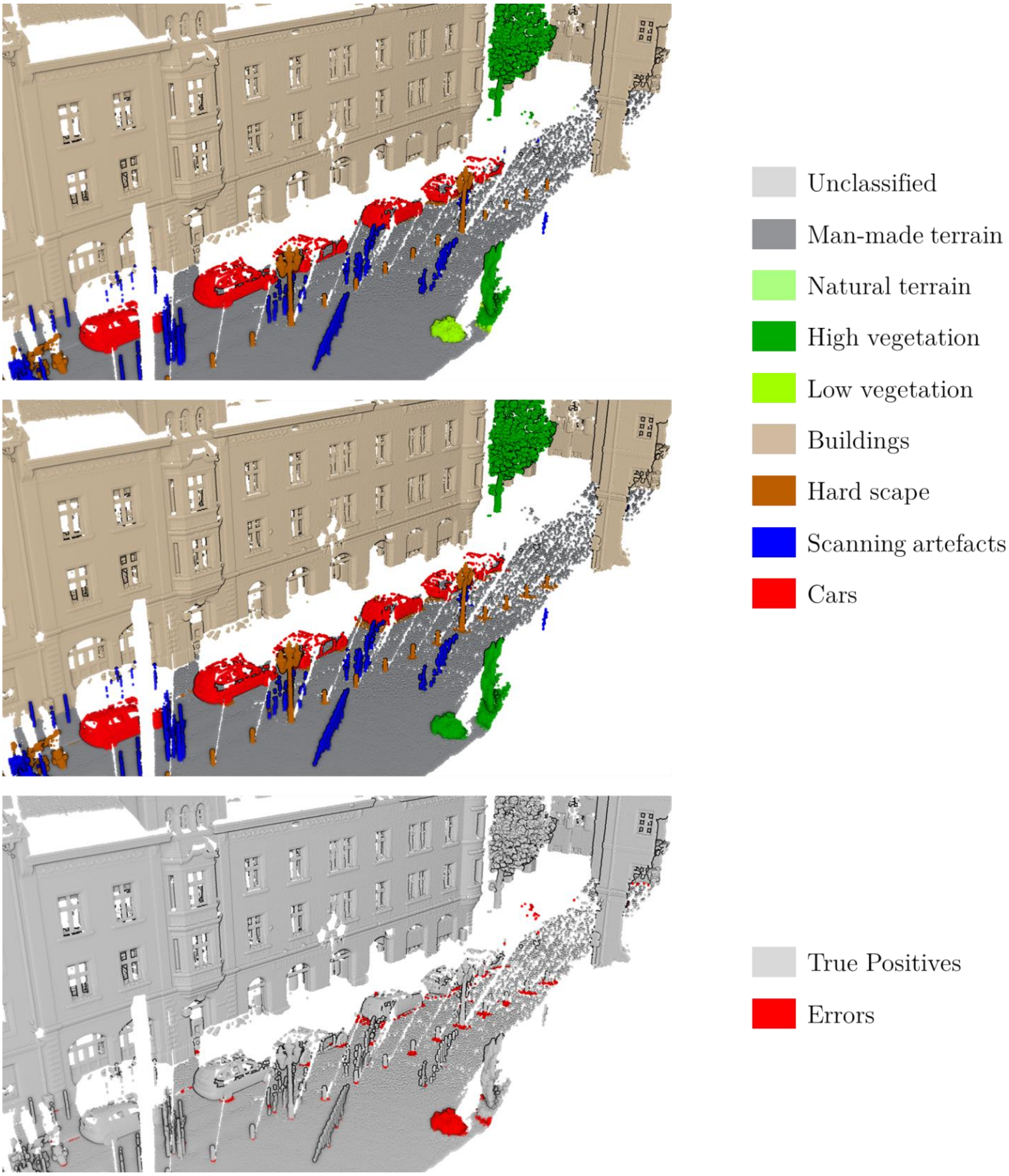

Unclassified
Man-made terrain
Natural terrain
High vegetation
Low vegetation
Buildings
Hard scape
Scanning artefacts
Cars
True Positives
Errors

Figure 68. Semantic3D scene segmented with KP-FCNN. Top: predictions. Middle: Groundtruth. Bottom: Errors.

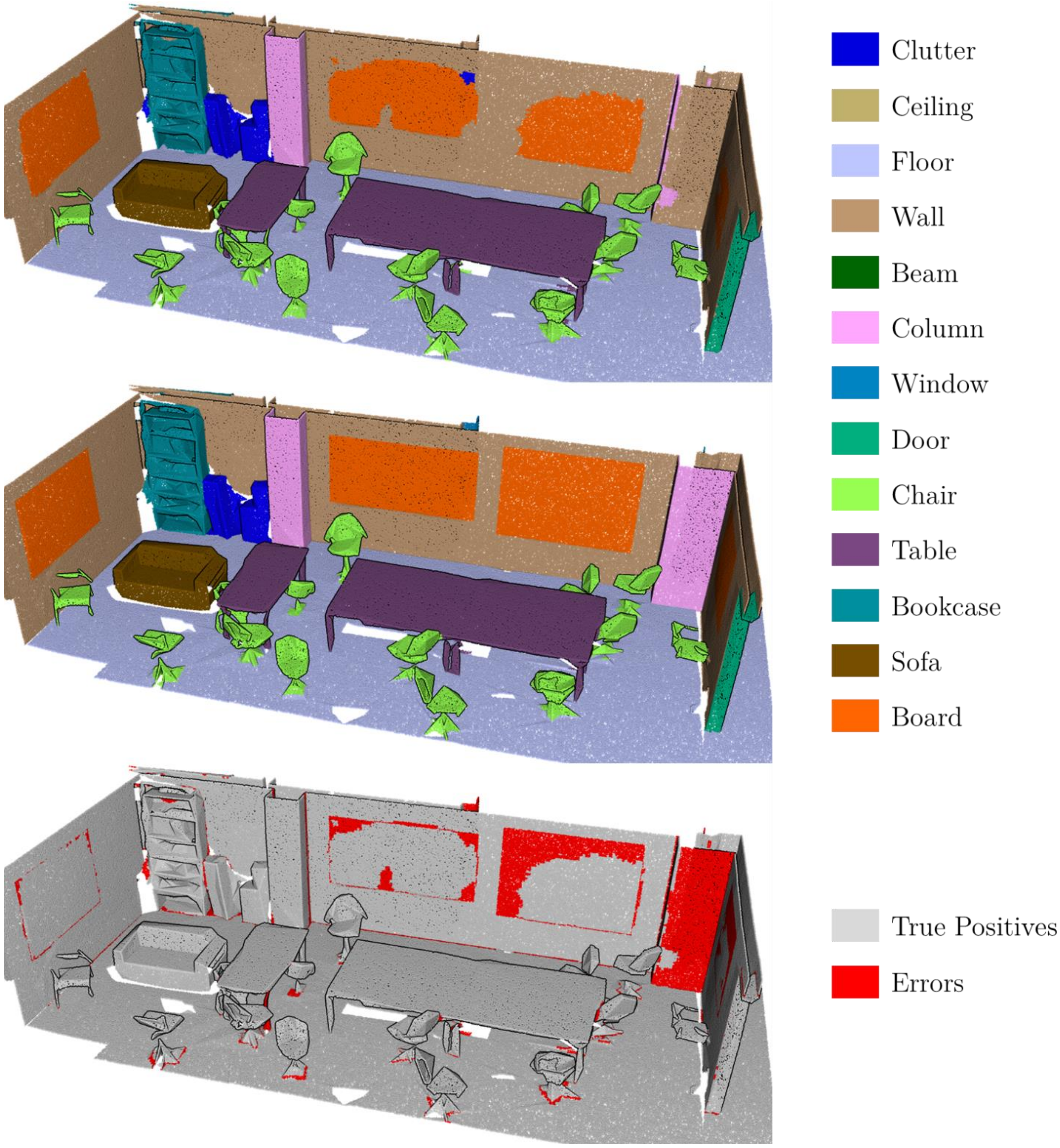

Clutter
Ceiling
Floor
Wall
Beam
Column
Window
Door
Chair
Table
Bookcase
Sofa
Board
True Positives
Errors

Figure 69. S3DIS scene segmented with KP-FCNN. Top: predictions. Middle: Groundtruth. Bottom: Errors.

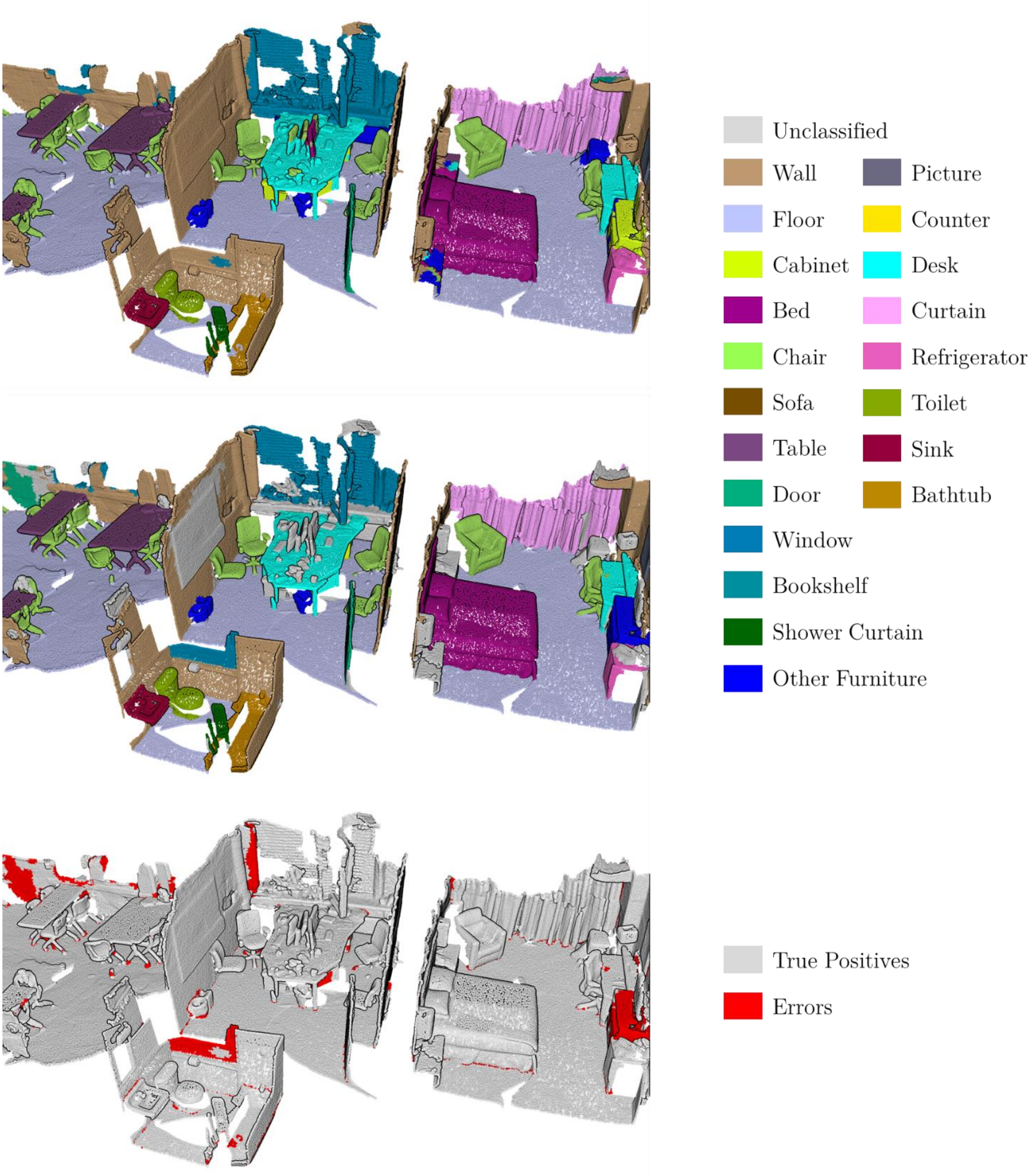

Unclassified
Wall
Picture
Floor
Counter
Cabinet
Desk
Bed
Curtain
Chair
Refrigerator
Sofa
Toilet
Table
Sink
Door
Bathtub
Window
Bookshelf
Shower Curtain
Other Furniture
True Positives
Errors

Figure 70. Multiple Scannet scenes segmented with KP-FCNN. Top: predictions. Middle: Groundtruth. Bottom: Errors. The scenes are placed together for visualization, but have been segmented independently.

IV.3.4. Model sizes and speeds

Table 15 shows some statistics of our models on different datasets. First, we notice that KP-FCNN and KP-CNN have similar number of parameters, because the decoder part of KP-FCNN only involves light 1×1 convolutions. We see that the running speeds are different from one dataset to another, which is not surprising. Indeed, the number of operations performed during a forward pass of our network depends on the number of points of the current batch, and the maximum number of neighbors of these points. Our models have been prototyped with a RTX 2080Ti in this experiment.

Table 15. Model running statistics on 4 datasets: ModelNet40, ShapeNetPart, Scannet, Semantic3D. We give the model sizes and the training and inference speed for both rigid and deformable version.

		MN40	SNP	Scannet	Sem3D
Avg pts/elem		6800	2370	8950	3800
Avg pts/batch		109K	38K	90K	38K
Params	*rigid*	14.3M	14.2M	14.1M	14.1M
	deform	15.2M	15.0M	14.9M	14.9M
Training (batch/s)	*rigid*	3.5	5.5	4.3	8.8
	deform	3.1	4.3	3.9	7.1
Inference (batch/s)	*rigid*	8.7	16.7	9.3	17.5
	deform	8.0	12.2	8.1	15.0

The speed of our networks is comparable, maybe even faster than competing point convolutional networks. However, a valid comparison cannot be shown here as every method is evaluated on a different hardware, and using different libraries. A more extensive study should be conducted, where we compare our networks with competing approaches on the same hardware and in the same experimental conditions.

IV.3.5. Impact of the number of kernel points

In this experiment, we conduct an ablation study to support our claim that deformable KPConv has a stronger descriptive power than rigid KPConv. The idea is to impede the capabilities of the network, in order to reveal the real potential of deformable kernels. We use Scannet dataset (same parameters as before) and use the official validation set, because the test set cannot be used for such evaluations. As depicted in Figure 71, the deformable KPConv only loses 1.5 mIoU points when restricted to 4 kernel points. In the same configuration, the rigid KPConv loses 3.5 mIoU points.

As stated in Section IV.3.3, we can also see that deformable KPConv performs better than rigid KPConv with 15 kernel points. Although it is not the case on the test set, we tried different validation sets that confirmed the superior performances of deformable KPConv. This is not surprising as we obtained the same results on S3DIS. Deformable KPConv seems to thrive on indoor datasets, which offer more diversity than outdoor datasets. To understand why, we need to go beyond numbers and see what is effectively learned by the two versions of KPConv.

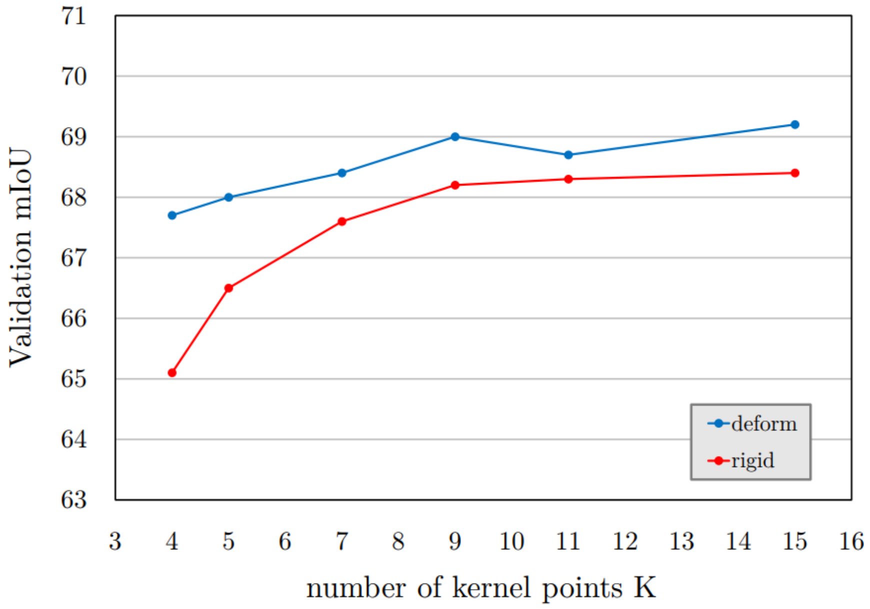


Figure 71. Ablation study on Scannet validation set. Evolution of the mIoU when reducing the number of kernel points.

IV.3.6. Visualization of learned features

To achieve a deeper understanding of KPConv, we offer a first insight of the learning mechanisms with the visualization of learned features. In this experiment, we trained KP-CNN on ModelNet40 with rigid KPConv. We added random rotation augmentations around vertical axis to increase the input shape diversity. Then we visualize each learned feature by coloring the points according to their level of activation for this feature. In Figure 72, we choose input point clouds maximizing the activation for different features at the first and third layer. For a cleaner display, we projected the activations from the subsampled points of the layer to the original input points. We observe that, in its first layer, the network is able to learn low-level features like vertical/horizontal planes (a/b), linear structures (c), or corners (d). In the later layers, the network detects more complex shapes like small buttresses (e), balls (f), cones (g), or stairs (h). However, it is difficult to see a difference between rigid and deformable KPConv. This tool is very useful to understand what KPConv can learn in general, but we need another one to compare the two versions.

IV.3.7. Study of the Effective Receptive Field

To apprehend the differences between the representations learned by rigid and deformable KPConv, we can compute its Effective Receptive Field (ERF) (Luo et al., 2016) at different locations. The ERF is a measure of the influence that each input point has on the result of a KPConv layer at a particular location. It is computed as the gradient of KPConv responses at this particular location with respect to the input point features. As we can see in Figure 73, the ERF varies depending on the object it is centered on. We see that rigid KPConv ERF has a relatively consistent range on every type of object, whereas deformable KPConv ERF seems to adapt to the object size. Indeed, it covers the whole bed, and concentrates more on the chair that on the surrounding ground. When centered on a flat surface, it also seems to ignore most of it and to reach for further details in the scene. This adaptive behavior shows that deformable KPConv improves the network ability to adapt to the geometry of the scene objects, and explains the better performances on indoor datasets.

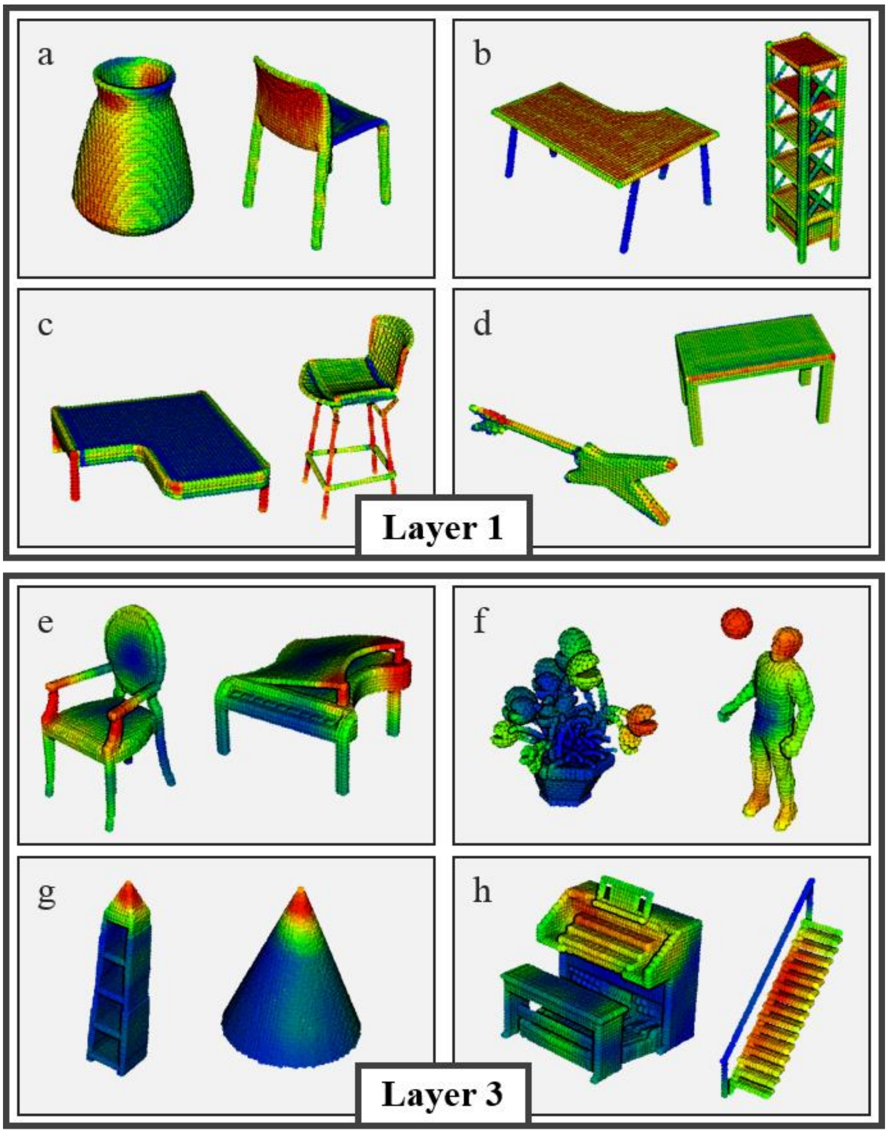


Figure 72. Low and high level features learned in KP-CNN. Each feature is displayed on 2 input point clouds taken from ModelNet40. High activations are in red and low activations in blue.

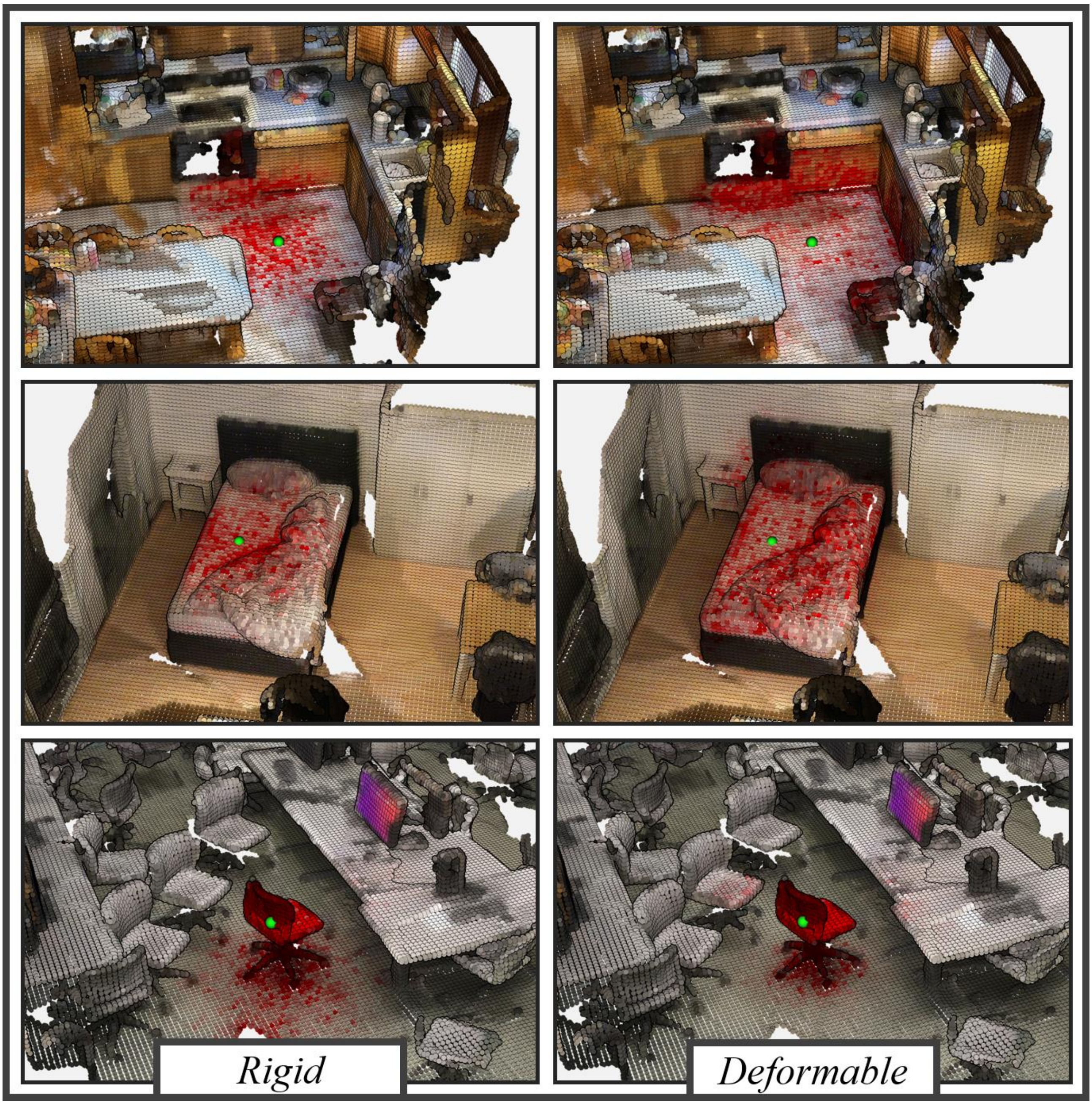


Figure 73. KPConv ERF at layer 4 of KP-FCNN, trained on Scannet. The green dots represent the ERF centers. ERF values are merged with scene colors as red intensity. The more red a point is the more influence it has on the green point features.

Conclusion

Overview

In this thesis, our goal was to learn new representations for 3D point cloud semantic segmentation. We first established the important notions that were needed for the design of such representation-learning algorithms.

Our line of thoughts, revolving around the three tiers of representations described in the introduction, conducted us to study handcrafted representations before designing a deep learning method capable of learning its own representations. This study resulted in several contributions for point clouds representation learning. First, the definition of a new multiscale spherical neighborhood, and the corresponding multiscale features proved that KNN were not adapted for the representation of point clouds at various scales. Thanks to their strong descriptive power, they were adopted by Terra3D with minor modifications for industrial applications. Our first semantic segmentation algorithm, using our multi-scale handcrafted features, and a new active learning strategy, outperforms the other handcrafted approaches in the literature. It thus offers a strong baseline for semantic segmentation deep learning architectures. Eventually, our subsampling scheme, which controls density in spherical neighborhoods, is also crucial for the design of learnable point cloud representations.

Thanks to this thorough study of handcrafted representations for point clouds, we elaborated a novel convolution operator (KPConv) for representation learning on point clouds. KPConv operates like an image convolution but directly on 3D points. Its weights are located by kernel points in a spherical domain. This new operator offers a great flexibility in its design and usage. We also defined a deformable version of this convolution operator that learns local shifts effectively deforming the convolution kernels to make them fit the point cloud geometry. We proposed a new regularization loss to ensure a good behavior of deformable convolutions on point clouds. The convolutional network architectures using KPConv operator process point clouds directly and outperform state-of-the art semantic segmentation methods in almost any situation. Eventually, we provided valuable visualizations of the features learned by our convolution kernels and their Effective Receptive Fields. These insights help us understand the mechanisms of learnable representations in our architectures.

Our networks reproduce the representation-learning scheme of image CNN, but on 3D point clouds. They reach the third tier of representation described in the introduction and are able to learn their own representations. We released our source code, hoping to help further research on point cloud convolutional architectures and representation learning.

Future works

Applications to instance segmentation. The power of such learned representations is not limited to semantic segmentation. Our architectures can easily be adapted to new applications, like image convolutional networks. For example, we could use KPConv in instance detection and instance segmentation networks. Such instance segmentation networks could be particularly useful for modelization in the context of large-scale 3D scenes and for object detection in the context of autonomous vehicles.

Image instance segmentation networks, like Mask R-CNN (He et al., 2017), use the same backbone architecture as image semantic segmentation. However, many questions are raised by the different nature of point clouds. Some methods use bounding boxes to detect object instances. Does boxes make sense in 3D point clouds? Should such boxes be oriented? Other shapes like spheres or cylinders could be used to locate objects.

Deformable convolutions are very interesting in the context of instance detection. Their ability to focus their receptive field on particular objects is well adapted to this task. Furthermore, recent methods like RepPoints (Yang et al., 2019), use deformable convolution kernels directly as object detectors instead of defining explicit bounding boxes. This kind of approaches could be particularly interesting for point clouds. This application would raise many questions to explore. Should the formulation be the same for point clouds? Would our regularization loss be important on such detector kernels? How could this object detection network be extended to instance segmentation?

Learning better representations. We visualized very interesting representations learned by our networks, but are they entirely satisfying? In all our experiments, our networks were trained on only one dataset. The number and the diversity of shapes offered by each dataset independently is quite limited, compared to image datasets. Each dataset only covers one type of environment, with the same kind of objects. In images, the datasets are larger, and offers a greater diversity. Images have different illuminations, distortions, and the same object can have very different shapes or scales. For example, image networks are able to recognize a cat, whether it is sleeping on a couch or facing close to the camera. As point clouds capture the 3D geometry of objects, their shapes and scales are not varying so much in this type of data, the only variations come from the differences between instances of the same class (two different car models for example) and from the occlusions.

Due to this difference, point cloud deep learning approaches might not be learning representations as strong as image networks. We believe that KPConv networks could thrive on large and more diverse point cloud datasets. First, KPConv efficiency allows building very deep networks with more layers, which should be essential for larger datasets. Moreover, our kernel combines a strong descriptive power, compared to other simpler representations, like the linear kernels of (Groh et al., 2018); and great learnability, the weights of MLP convolutions like (Y. Li et al., 2018; S. Wang et al., 2018) are more complex to learn.

A first step in this direction could be to train our architectures on multiple datasets together, using multi-task learning. This is the same idea that was used for ShapeNetPart dataset. Multiple network heads, one for each dataset, are trained together and sharing the features learned by one KPConv network. In that way, the network would have to learn representations that work for every environment and objects. However, many issues should be explored. Would the network really learn representations that are more general, or just learn different sets of representations, specific to each dataset? Would a network have stronger performances on one dataset if it were trained on several datasets?

The last question is very interesting considering the nature of point clouds. In image applications, networks need to be robust to all the variations of the objects, as they are inherent to the nature of the data. However, point clouds often are acquired industrially, which means their quality is much more controlled than images taken with smartphones. It is not sure that the lessons learned from image deep learning are applicable to point cloud deep learning. Given the low diversity of the test scenes, more specialized architectures, adapted to specific datasets and environments could be better than networks that are more general.

Extend to the fourth dimension. After showing such strong performances on images, deep learning approaches have been adapted with more or less success to videos. It is a lot more difficult to use deep learning approaches on videos as images are turned into dense 3D matrices when time dimension is included. However, point clouds in time and space remain sparse and the application of deep learning approaches on these four-dimensional point clouds is more straightforward.

Few works have explored the subject of space-time point clouds. Application like 3D flow estimation and simultaneous localization and mapping generally use two consecutive 3D point clouds instead of one 4D point cloud. Tranquil clouds networks also learn temporally coherent features with consecutive 3D point clouds.

Many applications include the time dimension: 3D flow estimation or simultaneous localization and mapping for example. However, few works have explored the subject of space-time point clouds. To solve these problems, algorithms generally use two consecutive 3D point clouds instead of one 4D point cloud. We can also give the example of a network named Tranquil clouds (Prantl and Chentanez, 2019), which learns temporally coherent features with consecutive 3D point clouds.

To go further than these methods, we could imagine a four-dimensional KPConv kernel that is defined in space and time. The fact that point clouds remain very sparse even if we add the fourth dimension means that our methods could extend very well to this new data. Deformable kernels could help our kernel to use fewer points and still be descriptive, despite the curse of dimensionality.

We believe this thesis opens the path for many new methods and applications in the field of 3D deep learning. We hope to see future works built on top of KPConv and our contributions for representation learning.

Publications

This thesis has led the following publications.

National conferences

Thomas, H., Deschaud, J. E., Marcotegui, B., & Goulette, F. (2018) Descripteurs sphériques multi-échelles pour la classification sémantique de nuages de points 3D. In *RFIAP et CFPT 2018*

International conferences

Thomas, H., Goulette, F., Deschaud, J. E., & Marcotegui, B. (2018, September). Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods. *In 2018 International Conference on 3D Vision (3DV)* (pp. 390-398). IEEE.

Thomas, H., Qi, C. R., Deschaud, J. E., Marcotegui, B., Goulette, F., Guibas, L. J. (2019). KPConv: Flexible and Deformable Convolution for Point Clouds. *In The IEEE International Conference on Computer Vision (ICCV), 2019.*

References

Aijazi, A., Checchin, P., Trassoudaine, L., 2013. Segmentation based classification of 3D urban point clouds: A super-voxel based approach with evaluation. Remote Sens. 5, 1624–1650.

Aldoma, A., Vincze, M., Blodow, N., Gossow, D., Gedikli, S., Rusu, R.B., Bradski, G., 2011. CAD-model recognition and 6DOF pose estimation using 3D cues, in: Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference On. IEEE, pp. 585–592.

Armeni, I., Sener, O., Zamir, A.R., Jiang, H., Brilakis, I., Fischer, M., Savarese, S., 2016. 3d semantic parsing of large-scale indoor spaces, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1534–1543.

Atzmon, M., Maron, H., Lipman, Y., 2018. Point convolutional neural networks by extension operators. ArXiv Prepr. ArXiv180310091.

Ben-Shabat, Y., Lindenbaum, M., Fischer, A., 2018. 3DmFV: Three-Dimensional Point Cloud Classification in Real-Time Using Convolutional Neural Networks. IEEE Robot. Autom. Lett. 3, 3145–3152.

Ben-Shabat, Y., Lindenbaum, M., Fischer, A., 2017. 3d point cloud classification and segmentation using 3d modified fisher vector representation for convolutional neural networks. ArXiv Prepr. ArXiv171108241.

Blanco, J.L., Rai, P.K., 2014. nanoflann: a C++11 header-only library for Nearest Neighbor (NN) search with KD-trees.

Blomley, R., Jutzi, B., Weinmann, M., 2016. CLASSIFICATION OF AIRBORNE LASER SCANNING DATA USING GEOMETRIC MULTI-SCALE FEATURES AND DIFFERENT NEIGHBOURHOOD TYPES. ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci. 3.

Boscaini, D., Masci, J., Rodolà, E., Bronstein, M., 2016. Learning shape correspondence with anisotropic convolutional neural networks, in: Advances in Neural Information Processing Systems. pp. 3189–3197.

Boulaassal, H., Landes, T., Grussenmeyer, P., Tarsha-Kurdi, F., 2007. Automatic segmentation of building facades using terrestrial laser data, in: ISPRS Workshop on Laser Scanning 2007 and SilviLaser 2007. pp. 65–70.

Boulch, A., Marlet, R., 2016. Deep learning for robust normal estimation in unstructured point clouds, in: Computer Graphics Forum. Wiley Online Library, pp. 281–290.

Boulch, A., Saux, B.L., Audebert, N., 2017. Unstructured point cloud semantic labeling using deep segmentation networks, in: Eurographics Workshop on 3D Object Retrieval. p. 1.

Brodu, N., Lague, D., 2012. 3D terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology. ISPRS J. Photogramm. Remote Sens. 68, 121–134.

Bruna, J., Zaremba, W., Szlam, A., Lecun, Y., 2014. Spectral networks and locally connected networks on graphs, in: International Conference on Learning Representations (ICLR2014), CBLS, April 2014.

Castillo, E., Liang, J., Zhao, H., 2013. Point cloud segmentation and denoising via constrained nonlinear least squares normal estimates, in: Innovations for Shape Analysis. Springer, pp. 283–299.

Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., 2015. Shapenet: An information-rich 3d model repository. ArXiv Prepr. ArXiv151203012.

Chehata, N., Guo, L., Mallet, C., 2009. Airborne lidar feature selection for urban classification using random forests. Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. 38, W8.

Christoph Stein, S., Schoeler, M., Papon, J., Worgotter, F., 2014. Object partitioning using local convexity, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 304–311.

Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nie\s sner, M., 2017. Scannet: Richly-annotated 3d reconstructions of indoor scenes, in: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y., 2017. Deformable convolutional networks, in: Proceedings of the IEEE International Conference on Computer Vision. pp. 764–773.

Defferrard, M., Bresson, X., Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering, in: Advances in Neural Information Processing Systems. pp. 3844–3852.

Demantke, J., Mallet, C., David, N., Vallet, B., 2011. Dimensionality based scale selection in 3D lidar point clouds. Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. 38, W12.

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

Deschaud, J.-E., Goulette, F., 2010. A fast and accurate plane detection algorithm for large noisy point clouds using filtered normals and voxel growing, in: 3DPVT.

Douillard, B., Underwood, J., Kuntz, N., Vlaskine, V., Quadros, A., Morton, P., Frenkel, A., 2011. On the segmentation of 3D LIDAR point clouds, in: Robotics and Automation (ICRA), 2011 IEEE International Conference On. IEEE, pp. 2798–2805.

Duan, C., Chen, S., Kovacevic, J., 2019. 3D Point Cloud Denoising via Deep Neural Network Based Local Surface Estimation, in: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 8553–8557.

Elbaz, G., Avraham, T., Fischer, A., 2017. 3D point cloud registration for localization using a deep neural network auto-encoder, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4631–4640.

Feng, C., Taguchi, Y., Kamat, V.R., 2014. Fast plane extraction in organized point clouds using agglomerative hierarchical clustering, in: 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 6218–6225.

Fischler, M.A., Bolles, R.C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24, 381–395.

Fleishman, S., Cohen-Or, D., Silva, C.T., 2005. Robust moving least-squares fitting with sharp features, in: ACM Transactions on Graphics (TOG). ACM, pp. 544–552.

Golovinskiy, A., Funkhouser, T., 2009. Min-cut based segmentation of point clouds, in: 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops. IEEE, pp. 39–46.

Golovinskiy, A., Kim, V.G., Funkhouser, T., 2009. Shape-based recognition of 3D point clouds in urban environments, in: 2009 IEEE 12th International Conference on Computer Vision. Presented at the 2009 IEEE 12th International Conference on Computer Vision, pp. 2154–2161. https://doi.org/10.1109/ICCV.2009.5459471

Graham, B., Engelcke, M., van der Maaten, L., 2018. 3d semantic segmentation with submanifold sparse convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9224–9232.

Groh, F., Wieschollek, P., Lensch, H., 2018. Flex-Convolution (Million-Scale Point-Cloud Learning Beyond Grid-Worlds). ArXiv Prepr. ArXiv180307289.

Gu, X., Wang, Y., Wu, C., Lee, Y.J., Wang, P., 2019. HPLFlowNet: Hierarchical Permutohedral Lattice FlowNet for Scene Flow Estimation on Large-scale Point Clouds, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3254–3263.

Guerrero, P., Kleiman, Y., Ovsjanikov, M., Mitra, N.J., 2018. PCPNet learning local shape properties from raw point clouds, in: Computer Graphics Forum. Wiley Online Library, pp. 75–85.

Hackel, T., Savinov, N., Ladicky, L., Wegner, J.D., Schindler, K., Pollefeys, M., 2017. Semantic3D. net: A new Large-scale Point Cloud Classification Benchmark. ArXiv Prepr. ArXiv170403847.

Hackel, T., Wegner, J.D., Schindler, K., 2016a. Contour detection in unstructured 3d point clouds, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1610–1618.

Hackel, T., Wegner, J.D., Schindler, K., 2016b. Fast semantic segmentation of 3D point clouds with strongly varying density. ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci. Prague Czech Repub. 3, 177–184.

He, K., Gkioxari, G., Dollár, P., Girshick, R., 2017. Mask r-cnn, in: Proceedings of the IEEE International Conference on Computer Vision. pp. 2961–2969.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778.

Henaff, M., Bruna, J., LeCun, Y., 2015. Deep convolutional networks on graph-structured data. ArXiv Prepr. ArXiv150605163.

Hermosilla, P., Ritschel, T., Vázquez, P.-P., Vinacua, À., Ropinski, T., 2018. Monte Carlo convolution for learning on non-uniformly sampled point clouds, in: SIGGRAPH Asia 2018 Technical Papers. ACM, p. 235.

Hernández, J., Marcotegui, B., 2009. Point cloud segmentation towards urban ground modeling, in: 2009 Joint Urban Remote Sensing Event. IEEE, pp. 1–5.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W., 1992. Surface reconstruction from unorganized points. ACM.

Hua, B.-S., Tran, M.-K., Yeung, S.-K., 2018. Pointwise convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 984–993.

Huang, Q., Wang, W., Neumann, U., 2018. Recurrent slice networks for 3d segmentation of point clouds, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2626–2635.

Johnson, A.E., Hebert, M., 1999. Using spin images for efficient object recognition in cluttered 3D scenes. IEEE Trans. Pattern Anal. Mach. Intell. 21, 433–449.

Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. ArXiv Prepr. ArXiv160902907.

Klokov, R., Lempitsky, V., 2017. Escape from Cells: Deep Kd-Networks for The Recognition of 3D Point Cloud Models. ArXiv Prepr. ArXiv170401222.

Lalonde, J.-F., Vandapel, N., Hebert, M., 2007. Data structures for efficient dynamic processing in 3-d. Int. J. Robot. Res. 26, 777–796.

Lalonde, J.-F., Vandapel, N., Huber, D.F., Hebert, M., 2006. Natural terrain classification using three-dimensional ladar data for ground robot mobility. J. Field Robot. 23, 839–861.

Landrieu, L., Raguet, H., Vallet, B., Mallet, C., Weinmann, M., 2017. A structured regularization framework for spatially smoothing semantic labelings of 3D point clouds. ISPRS J. Photogramm. Remote Sens. 132, 102–118.

Landrieu, L., Simonovsky, M., 2017. Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. ArXiv Prepr. ArXiv171109869.

Lawin, F.J., Danelljan, M., Tosteberg, P., Bhat, G., Khan, F.S., Felsberg, M., 2017. Deep projective 3D semantic segmentation, in: International Conference on Computer Analysis of Images and Patterns. Springer, pp. 95–107.

Lefebvre, S., Hoppe, H., 2006. Perfect spatial hashing, in: ACM Transactions on Graphics (TOG). ACM, pp. 579–588.

Li, J., Chen, B.M., Hee Lee, G., 2018. So-net: Self-organizing network for point cloud analysis, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9397–9406.

Li, L., Sung, M., Dubrovina, A., Yi, L., Guibas, L., 2018. Supervised Fitting of Geometric Primitives to 3D Point Clouds. ArXiv Prepr. ArXiv181108988.

Li, R., Wang, S., Zhu, F., Huang, J., 2018. Adaptive graph convolutional neural networks, in: Thirty-Second AAAI Conference on Artificial Intelligence.

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B., 2018. PointCNN: Convolution on X-transformed points, in: Advances in Neural Information Processing Systems. pp. 820–830.

Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R., 2015. Gated graph sequence neural networks. ArXiv Prepr. ArXiv151105493.

Liu, X., Qi, C.R., Guibas, L.J., 2018. Learning Scene Flow in 3D Point Clouds. ArXiv Prepr. ArXiv180601411.

Luo, W., Li, Y., Urtasun, R., Zemel, R., 2016. Understanding the effective receptive field in deep convolutional neural networks, in: Advances in Neural Information Processing Systems. pp. 4898–4906.

Marton, Z.-C., Pangercic, D., Blodow, N., Kleinehellefort, J., Beetz, M., 2010a. General 3D modelling of novel objects from a single view, in: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference On. IEEE, pp. 3700–3705.

Marton, Z.-C., Pangercic, D., Rusu, R.B., Holzbach, A., Beetz, M., 2010b. Hierarchical object geometric categorization and appearance classification for mobile manipulation, in: Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference On. IEEE, pp. 365–370.

Masci, J., Boscaini, D., Bronstein, M., Vandergheynst, P., 2015. Geodesic convolutional neural networks on riemannian manifolds, in: Proceedings of the IEEE International Conference on Computer Vision Workshops. pp. 37–45.

Maturana, D., Scherer, S., 2015. Voxnet: A 3d convolutional neural network for real-time object recognition, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 922–928.

Meagher, D., 1982. Geometric modeling using octree encoding. Comput. Graph. Image Process. 19, 129–147.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M., 2017. Geometric deep learning on graphs and manifolds using mixture model cnns, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5115–5124.

Muja, M., Lowe, D.G., 2009. Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP 1 2, 2.

Munoz, D., Bagnell, J.A., Vandapel, N., Hebert, M., 2009. Contextual classification with functional max-margin markov networks, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, pp. 975–982.

Museth, K., 2013. VDB: High-resolution sparse volumes with dynamic topology. ACM Trans. Graph. TOG 32, 27.

Najafi, M., Namin, S.T., Salzmann, M., Petersson, L., 2014. Non-associative higher-order markov networks for point cloud classification, in: European Conference on Computer Vision. Springer, pp. 500–515.

Niemeyer, J., Rottensteiner, F., Soergel, U., 2014. Contextual classification of lidar data and building object detection in urban areas. ISPRS J. Photogramm. Remote Sens. 87, 152–165.

Niemeyer, J., Wegner, J.D., Mallet, C., Rottensteiner, F., Soergel, U., 2011. Conditional random fields for urban scene classification with full waveform LiDAR data, in: Photogrammetric Image Analysis. Springer, pp. 233–244.

Ovsjanikov, M., Ben-Chen, M., Solomon, J., Butscher, A., Guibas, L., 2012. Functional maps: a flexible representation of maps between shapes. ACM Trans. Graph. TOG 31, 30.

Öztireli, A.C., Guennebaud, G., Gross, M., 2009. Feature preserving point set surfaces based on non-linear kernel regression, in: Computer Graphics Forum. Wiley Online Library, pp. 493–501.

Papon, J., Abramov, A., Schoeler, M., Worgotter, F., 2013. Voxel cloud connectivity segmentation-supervoxels for point clouds, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2027–2034.

Pauly, M., Keiser, R., Gross, M., 2003. Multi-scale feature extraction on point-sampled surfaces, in: Computer Graphics Forum. Wiley Online Library, pp. 281–289.

Poppinga, J., Vaskevicius, N., Birk, A., Pathak, K., 2008. Fast plane detection and polygonalization in noisy 3D range images, in: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, pp. 3378–3383.

Prantl, L., Chentanez, N., 2019. Tranquil Clouds: Neural Networks for Learning Temporally Coherent Features in Point Clouds. ArXiv Prepr. ArXiv190705279.

Qi, C.R., Su, H., Mo, K., Guibas, L.J., 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation. Proc Comput. Vis. Pattern Recognit. CVPR IEEE 1, 4.

Qi, C.R., Su, H., Niessner, M., Dai, A., Yan, M., Guibas, L.J., 2016. Volumetric and Multi-View CNNs for Object Classification on 3D Data. ArXiv Prepr. ArXiv160403265.

Qi, C.R., Yi, L., Su, H., Guibas, L.J., 2017b. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, in: Advances in Neural Information Processing Systems. pp. 5099–5108.

Rabbani, T., Van Den Heuvel, F., Vosselmann, G., 2006. Segmentation of point clouds using smoothness constraint. Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. 36, 248–253.

Rakotosaona, M.-J., La Barbera, V., Guerrero, P., Mitra, N.J., Ovsjanikov, M., 2019. POINTCLEANNET: Learning to Denoise and Remove Outliers from Dense Point Clouds. ArXiv Prepr. ArXiv190101060.

Ravanbakhsh, S., Schneider, J., Poczos, B., 2016. Deep Learning with Sets and Point Clouds. ArXiv Prepr. ArXiv161104500.

Rethage, D., Wald, J., Sturm, J., Navab, N., Tombari, F., 2018. Fully-convolutional point networks for large-scale point clouds, in: Proceedings of the European Conference on Computer Vision (ECCV). pp. 596–611.

Riegler, G., Osman Ulusoy, A., Geiger, A., 2017. Octnet: Learning deep 3d representations at high resolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3577–3586.

Roveri, R., Öztireli, A.C., Pandele, I., Gross, M., 2018. Pointpronets: Consolidation of point clouds with convolutional neural networks, in: Computer Graphics Forum. Wiley Online Library, pp. 87–99.

Roynard, X., Deschaud, J.-E., Goulette, F., 2018a. Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. Int. J. Robot. Res. 37, 545–557.

Roynard, X., Deschaud, J.-E., Goulette, F., 2018b. Classification of Point Cloud Scenes with Multiscale Voxel Deep Network. ArXiv Prepr. ArXiv180403583.

Roynard, X., Deschaud, J.E., Goulette, F., 2016. Fast and Robust Segmentation and Classification for Change Detection in Urban Point Clouds, in: ISPRS 2016-XXIII ISPRS Congress.

Rusu, R.B., Blodow, N., Beetz, M., 2009. Fast point feature histograms (FPFH) for 3D registration, in: 2009 IEEE International Conference on Robotics and Automation. IEEE, pp. 3212–3217.

Rusu, R.B., Bradski, G., Thibaux, R., Hsu, J., 2010. Fast 3d recognition and pose using the viewpoint feature histogram, in: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference On. IEEE, pp. 2155–2162.

Rusu, R.B., Marton, Z.C., Blodow, N., Dolha, M., Beetz, M., 2008. Towards 3D point cloud based object maps for household environments. Robot. Auton. Syst. 56, 927–941.

Sappa, A.D., Devy, M., 2001. Fast range image segmentation by an edge detection strategy, in: Proceedings Third International Conference on 3-D Digital Imaging and Modeling. IEEE, pp. 292–299.

Schnabel, R., Wahl, R., Klein, R., 2007. Efficient RANSAC for point-cloud shape detection, in: Computer Graphics Forum. Wiley Online Library, pp. 214–226.

Schoeler, M., Papon, J., Wörgötter, F., 2015. Constrained planar cuts - Object partitioning for point clouds, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5207–5215. https://doi.org/10.1109/CVPR.2015.7299157

Serna, A., Marcotegui, B., 2014. Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning. ISPRS J. Photogramm. Remote Sens. 93, 243–255.

Serna, A., Marcotegui, B., Goulette, F., Deschaud, J.-E., 2014. Paris-rue-Madame database: a 3D mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods, in: 4th International Conference on Pattern Recognition, Applications and Methods ICPRAM 2014.

Settles, B., 2009. Active learning literature survey. University of Wisconsin-Madison Department of Computer Sciences.

Shapovalov, R., Velizhev, E., Barinova, O., 2010. Nonassociative markov networks for 3d point cloud classification. The, in: International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XXXVIII, Part 3A. Citeseer.

Simonovsky, M., 2018. Deep Learning on Attributed Graphs (PHD diss.). Université Paris-Est.

Simonovsky, M., Komodakis, N., 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3693–3702.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. ArXiv Prepr. ArXiv14091556.

Song, H., Feng, H.-Y., 2009. A progressive point cloud simplification algorithm with preserved sharp edge data. Int. J. Adv. Manuf. Technol. 45, 583–592.

Song, H., Feng, H.-Y., OuYang, D., 2008. Automatic detection of tangential discontinuities in point cloud data. J. Comput. Inf. Sci. Eng. 8, 021001.

Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.-H., Kautz, J., 2018. Splatnet: Sparse lattice networks for point cloud processing, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2530–2539.

Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E., 2015. Multi-view convolutional neural networks for 3d shape recognition, in: Proceedings of the IEEE International Conference on Computer Vision. pp. 945–953.

Tarsha-Kurdi, F., Landes, T., Grussenmeyer, P., 2007. Extended RANSAC algorithm for automatic detection of building roof planes from LiDAR data. Photogramm. J. Finl. 21, 97–109.

Tatarchenko, M., Park, J., Koltun, V., Zhou, Q.-Y., 2018. Tangent convolutions for dense prediction in 3d, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3887–3896.

Tchapmi, L., Choy, C., Armeni, I., Gwak, J., Savarese, S., 2017. Segcloud: Semantic segmentation of 3d point clouds, in: 2017 International Conference on 3D Vision (3DV). IEEE, pp. 537–547.

Thomas, H., Goulette, F., Deschaud, J.-E., Marcotegui, B., 2018. Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods, in: 2018 International Conference on 3D Vision (3DV). IEEE, pp. 390–398.

Vallet, B., Brédif, M., Serna, A., Marcotegui, B., Paparoditis, N., 2015. TerraMobilita/iQmulus urban point cloud analysis benchmark. Comput. Graph. 49, 126–133.

Verma, N., Boyer, E., Verbeek, J., 2018. Feastnet: Feature-steered graph convolutions for 3d shape analysis, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2598–2606.

Vosselman, G., Gorte, B.G., Sithole, G., Rabbani, T., 2004. Recognising structure in laser scanner point clouds. Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. 46, 33–38.

Wang, C., Samari, B., Siddiqi, K., 2018. Local spectral graph convolution for point set feature learning, in: Proceedings of the European Conference on Computer Vision (ECCV). pp. 52–66.

Wang, H., Wang, C., Luo, H., Li, P., Chen, Y., Li, J., 2015. 3-D point cloud object detection based on supervoxel neighborhood with Hough forest framework. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 8, 1570–1581.

Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., Tong, X., 2017. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. ACM Trans. Graph. TOG 36, 72.

Wang, S., Suo, S., Ma, W.-C., Pokrovsky, A., Urtasun, R., 2018. Deep parametric continuous convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2589–2597.

Wang, W., Yu, R., Huang, Q., Neumann, U., 2018. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2569–2578.

Wang, Y., Solomon, J.M., 2019. Deep Closest Point: Learning Representations for Point Cloud Registration. ArXiv Prepr. ArXiv190503304.

Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M., 2018. Dynamic graph cnn for learning on point clouds. ArXiv Prepr. ArXiv180107829.

Wani, M.A., Arabnia, H.R., 2003. Parallel edge-region-based segmentation algorithm targeted at reconfigurable multiring network. J. Supercomput. 25, 43–62.

Weinmann, M., Jutzi, B., Hinz, S., Mallet, C., 2015a. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. ISPRS J. Photogramm. Remote Sens. 105, 286–304.

Weinmann, M., Jutzi, B., Mallet, C., 2013. Feature relevance assessment for the semantic interpretation of 3D point cloud data. ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci. 5, W2.

Weinmann, M., Schmidt, A., Mallet, C., Hinz, S., Rottensteiner, F., Jutzi, B., 2015b. Contextual classification of point cloud data by exploiting individual 3D neigbourhoods. ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci. II-3 2015 Nr W4 2, 271–278.

Wohlkinger, W., Vincze, M., 2011. Ensemble of shape functions for 3d object classification, in: 2011 IEEE International Conference on Robotics and Biomimetics. IEEE, pp. 2987–2992.

Wolf, D., Prankl, J., Vincze, M., 2015. Fast semantic segmentation of 3D point clouds using a dense CRF with learned parameters, in: Robotics and Automation (ICRA), 2015 IEEE International Conference On. IEEE, pp. 4867–4873.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J., 2015. 3d shapenets: A deep representation for volumetric shapes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1912–1920.

Wurm, K.M., Hornung, A., Bennewitz, M., Stachniss, C., Burgard, W., 2010. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems, in: Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation.

Xu, Y., Fan, T., Xu, M., Zeng, L., Qiao, Y., 2018. Spidercnn: Deep learning on point sets with parameterized convolutional filters, in: Proceedings of the European Conference on Computer Vision (ECCV). pp. 87–102.

Yang, Z., Liu, S., Hu, H., Wang, L., Lin, S., 2019. RepPoints: Point Set Representation for Object Detection. ArXiv Prepr. ArXiv190411490.

Ye, X., Li, J., Huang, H., Du, L., Zhang, X., 2018. 3d recurrent neural networks with context fusion for point cloud semantic segmentation, in: Proceedings of the European Conference on Computer Vision (ECCV). pp. 403–417.

Yi, L., Kim, V.G., Ceylan, D., Shen, I., Yan, M., Su, H., Lu, C., Huang, Q., Sheffer, A., Guibas, L., 2016. A scalable active framework for region annotation in 3d shape collections. ACM Trans. Graph. TOG 35, 210.

Yi, L., Su, H., Guo, X., Guibas, L.J., 2017. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2282–2290.

Yu, L., Li, X., Fu, C.-W., Cohen-Or, D., Heng, P.-A., 2018. EC-Net: an Edge-aware Point set Consolidation Network, in: Proceedings of the European Conference on Computer Vision (ECCV). pp. 386–402.

Zhang, C., Luo, W., Urtasun, R., 2018. Efficient Convolutions for Real-Time Semantic Segmentation of 3D Point Clouds, in: 2018 International Conference on 3D Vision (3DV). IEEE, pp. 399–408.

Zhou, Y., Tuzel, O., 2018. Voxelnet: End-to-end learning for point cloud based 3d object detection, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4490–4499.

A. More KPConv semantic segmentation results

A.1. Class scores on semantic segmentation datasets

In this appendix, we provide more details on our segmentation experiments, for benchmarking purposes with future works. We give class scores for our experiments on ShapeNetPart (Table 5) and S3DIS (Tables 6 and 7) dataset. Scannet, Semantic3D and Paris-Lille-3D are online benchmarks, the class scores can be found on their respective website.

Table 16. Segmentation mIoUs for all classes of ShapeNetPart.

Method	class avg.	inst. avg.	aero	bag	cap	car	chair	ear	guit	knif	lamp	lapt	moto	mug	pist	rock	skate	table
Kd-Net (Klokov and Lempitsky, 2017)	77.4	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
SO-Net (J. Li et al., 2018)	81.0	84.9	82.8	77.8	88.0	77.3	90.6	73.5	90.7	83.9	82.8	94.8	69.1	94.2	80.9	53.1	72.9	83.0
PCNN (Atzmon et al., 2018)	81.8	85.1	82.4	80.1	85.5	79.5	90.8	73.2	91.3	86.0	85.0	95.7	73.2	94.8	83.3	51.0	75.0	81.8
PointNet++ (Qi et al., 2017b)	81.9	85.1	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
SynSpecCNN (Yi et al., 2017)	82.0	84.7	81.6	81.7	81.9	75.2	90.2	74.9	**93.0**	86.1	84.7	95.6	66.7	92.7	81.6	60.6	**82.9**	82.1
DGCNN (Y. Wang et al., 2018)	82.3	85.1	84.2	83.7	84.4	77.1	90.9	78.5	91.5	87.3	82.9	96.0	67.8	93.3	82.6	59.7	75.5	82.0
SpiderCNN (Xu et al., 2018)	82.4	85.3	83.5	81.0	87.2	77.5	90.7	76.8	91.1	87.3	83.3	95.8	70.2	93.5	82.7	59.7	75.8	82.8
SubSparseCNN (Graham et al., 2018)	83.3	86.0	84.1	83.0	84.0	80.8	**91.4**	78.2	91.6	**89.1**	85.0	95.8	73.7	95.2	84.0	58.5	76.0	82.7
SPLATNet (Su et al., 2018)	83.7	85.4	83.2	84.3	89.1	80.3	90.7	75.5	92.1	87.1	83.9	96.3	75.6	**95.8**	83.8	64.0	75.5	81.8
PointCNN (Y. Li et al., 2018)	84.6	86.1	84.1	86.5	86.0	80.8	90.6	79.7	92.3	88.4	**85.3**	96.1	77.2	95.3	84.2	64.2	80.0	83.0
FlexConv (Groh et al., 2018)	85.0	84.7	83.6	**91.2**	**96.7**	79.5	84.7	71.7	92.0	86.5	83.2	**96.6**	71.7	95.7	86.1	**74.8**	81.4	**84.5**
Rigid KPConv	85.0	86.2	83.8	86.1	88.2	**81.6**	91.0	**80.1**	92.1	87.8	82.2	96.2	77.9	95.7	**86.8**	65.3	81.7	83.6
Deformable KPConv	**85.1**	**86.4**	**84.6**	86.3	87.2	81.1	91.1	77.8	92.6	88.4	82.7	96.2	**78.1**	**95.8**	85.4	69.0	82.0	83.6

Table 17. Semantic segmentation IoU scores on S3DIS Area-5. Additionally, we give the mean class recall, a measure that some previous works call mean class accuracy.

Method	mIoU	mRec	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book.	sofa	board	clut.
Pointnet (Qi et al., 2017a)	41.1	49.0	88.8	97.3	69.8	0.1	3.9	46.3	10.8	52.6	58.9	40.3	5.9	26.4	33.2
SegCloud (Tchapmi et al., 2017)	48.9	57.4	90.1	96.1	69.9	0.0	18.4	38.4	23.1	75.9	70.4	58.4	40.9	13.0	41.6
Eff 3D Conv (Zhang et al., 2018)	51.8	68.3	79.8	93.9	69.0	0.2	28.3	38.5	48.3	71.1	73.6	48.7	59.2	29.3	33.1
TangentConv (Tatarchenko et al., 2018)	52.6	62.2	90.5	97.7	74.0	0.0	20.7	39.0	31.3	69.4	77.5	38.5	57.3	48.8	39.8
RNN Fusion (Ye et al., 2018)	57.3	63.9	92.3	**98.2**	79.4	0.0	17.6	22.8	62.1	74.4	80.6	31.7	66.7	62.1	56.7
SPGraph (Landrieu and Simonovsky, 2017)	58.0	66.5	89.4	96.9	78.1	0.0	**42.8**	48.9	61.6	84.7	75.4	69.8	52.6	2.1	52.2
ParamConv (S. Wang et al., 2018)	58.3	67.1	92.3	96.2	75.9	**0.3**	6.0	**69.5**	63.5	66.9	65.6	47.3	68.9	59.1	46.2
Rigid KPConv	65.4	70.9	92.6	97.3	81.4	0.0	16.5	54.5	**69.5**	90.1	80.2	74.6	66.4	63.7	58.1
Deformable KPConv	**67.1**	**72.8**	**92.8**	97.3	**82.4**	0.0	23.9	58.0	69.0	**91.0**	**81.5**	**75.3**	**75.4**	**66.7**	**58.9**

Table 18. Semantic segmentation IoU scores on S3DIS k-fold. Additionally, we give the mean class recall, a measure that some previous works call mean class accuracy.

Method	mIoU	mRec	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book.	sofa	board	clut.
Pointnet (Qi et al., 2017a)	47.6	66.2	88.0	88.7	69.3	42.4	23.1	47.5	51.6	42.0	54.1	38.2	9.6	29.4	35.2
RSNet (Huang et al., 2018)	56.5	66.5	92.5	92.8	78.6	32.8	34.4	51.6	68.1	60.1	59.7	50.2	16.4	44.9	52.0
SPGraph (Landrieu and Simonovsky, 2017)	62.1	73.0	89.9	95.1	76.4	62.8	47.1	55.3	68.4	**73.5**	**69.2**	63.2	45.9	8.7	52.9
PointCNN (Y. Li et al., 2018)	65.4	75.6	**94.8**	**97.3**	75.8	63.3	51.7	58.4	57.2	71.6	69.1	39.1	61.2	52.2	58.6
Rigid KPConv	69.6	78.1	93.7	92.0	82.5	62.5	49.5	65.7	**77.3**	57.8	64.0	68.8	71.7	60.1	59.6
Deformable KPConv	**70.6**	**79.1**	93.6	92.4	**83.1**	**63.9**	**54.3**	**66.1**	76.6	57.8	64.0	**69.3**	**74.9**	**61.3**	**60.3**

A.2. Illustration of 3D scenes semantic segmentation

In this appendix, we provide more illustrations of 3D scenes segmented with our KP-FCNN network. These results are computed on the validation set of each dataset in order to see the errors that our algorithm made.

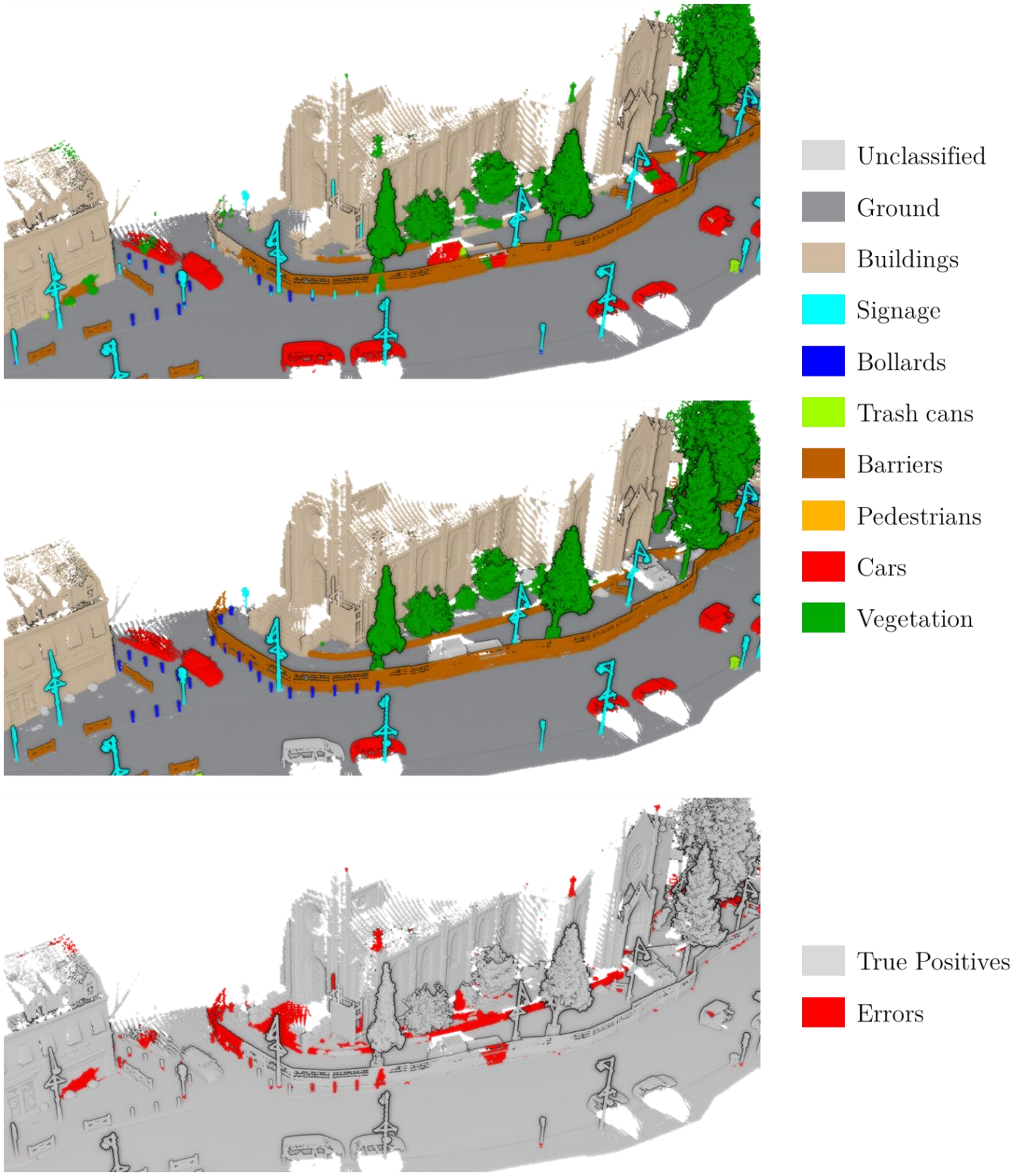

Unclassified
Ground
Buildings
Signage
Bollards
Trash cans
Barriers
Pedestrians
Cars
Vegetation
True Positives
Errors

Figure 74. Illustration of a 3D scene from Paris-Lille-3D dataset segmented with KPConv. Top: predictions. Middle: Groundtruth. Bottom: Errors.

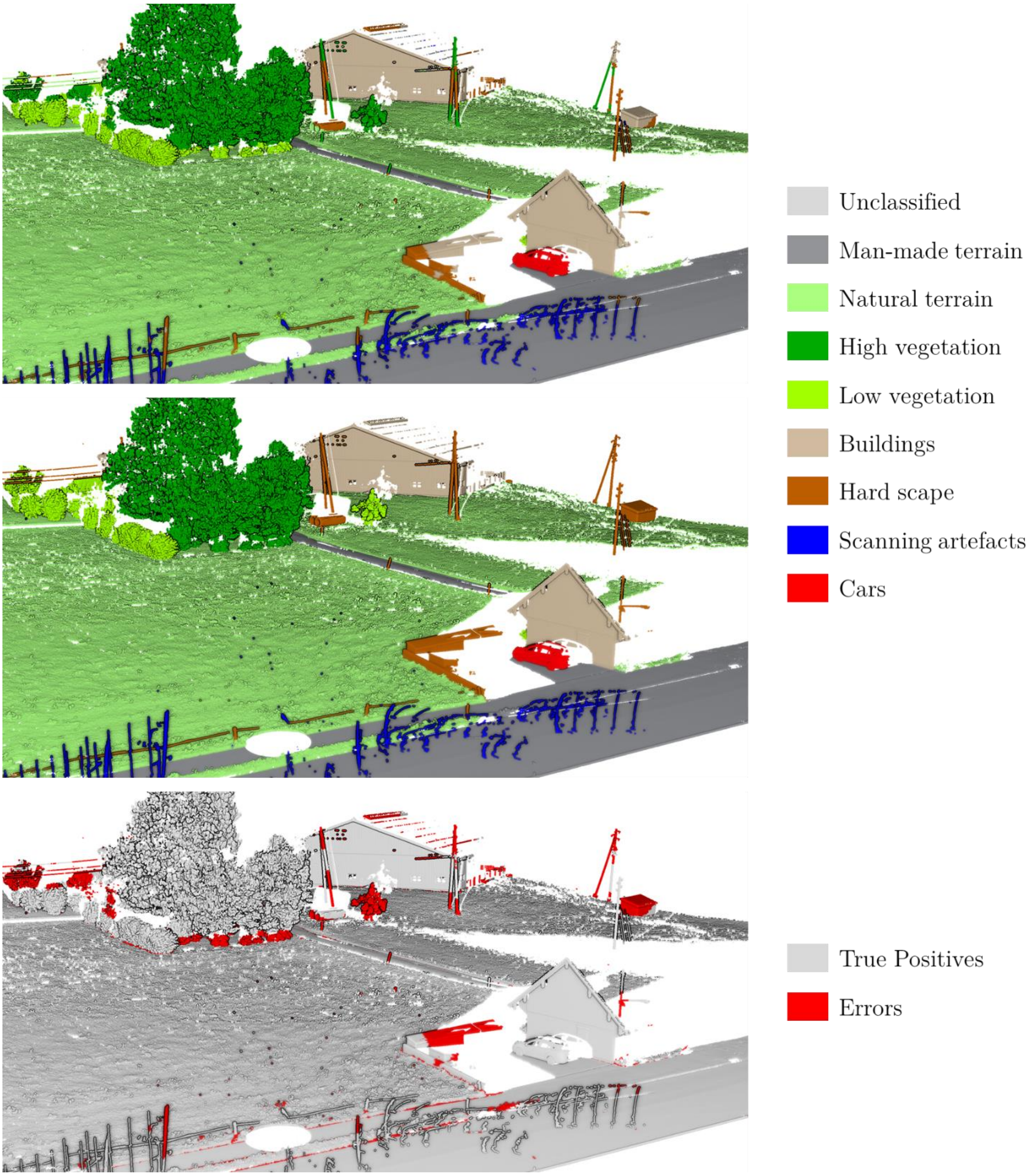

Unclassified
Man-made terrain
Natural terrain
High vegetation
Low vegetation
Buildings
Hard scape
Scanning artefacts
Cars
True Positives
Errors

Figure 75. Illustration of a 3D scene from Semantic3D dataset segmented with KPConv. Top: predictions. Middle: Groundtruth. Bottom: Errors.

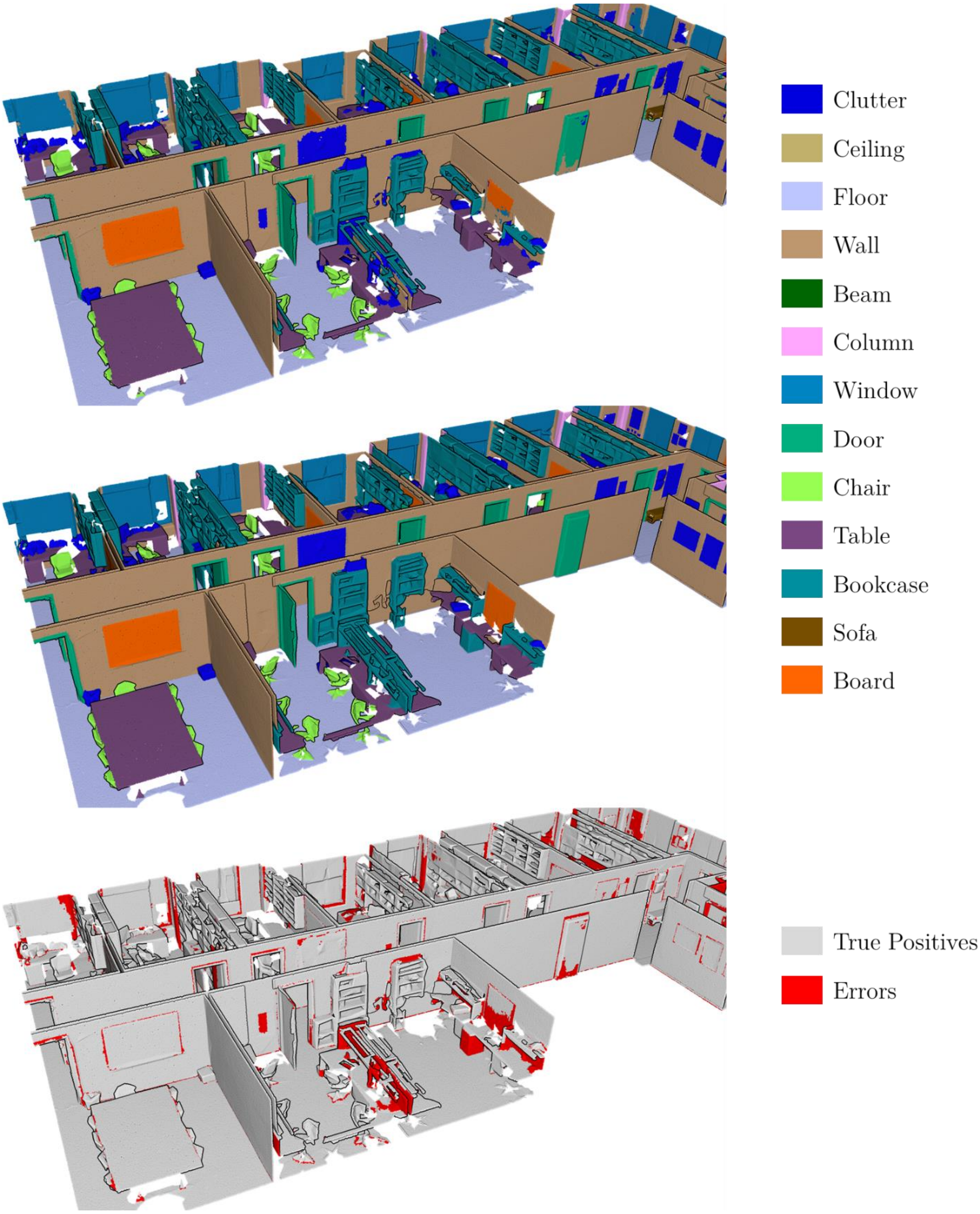

Clutter
Ceiling
Floor
Wall
Beam
Column
Window
Door
Chair
Table
Bookcase
Sofa
Board
True Positives
Errors

Figure 76. Illustration of a 3D scene from S3DIS dataset segmented with KPConv. Top: predictions. Middle: Groundtruth. Bottom: Errors.

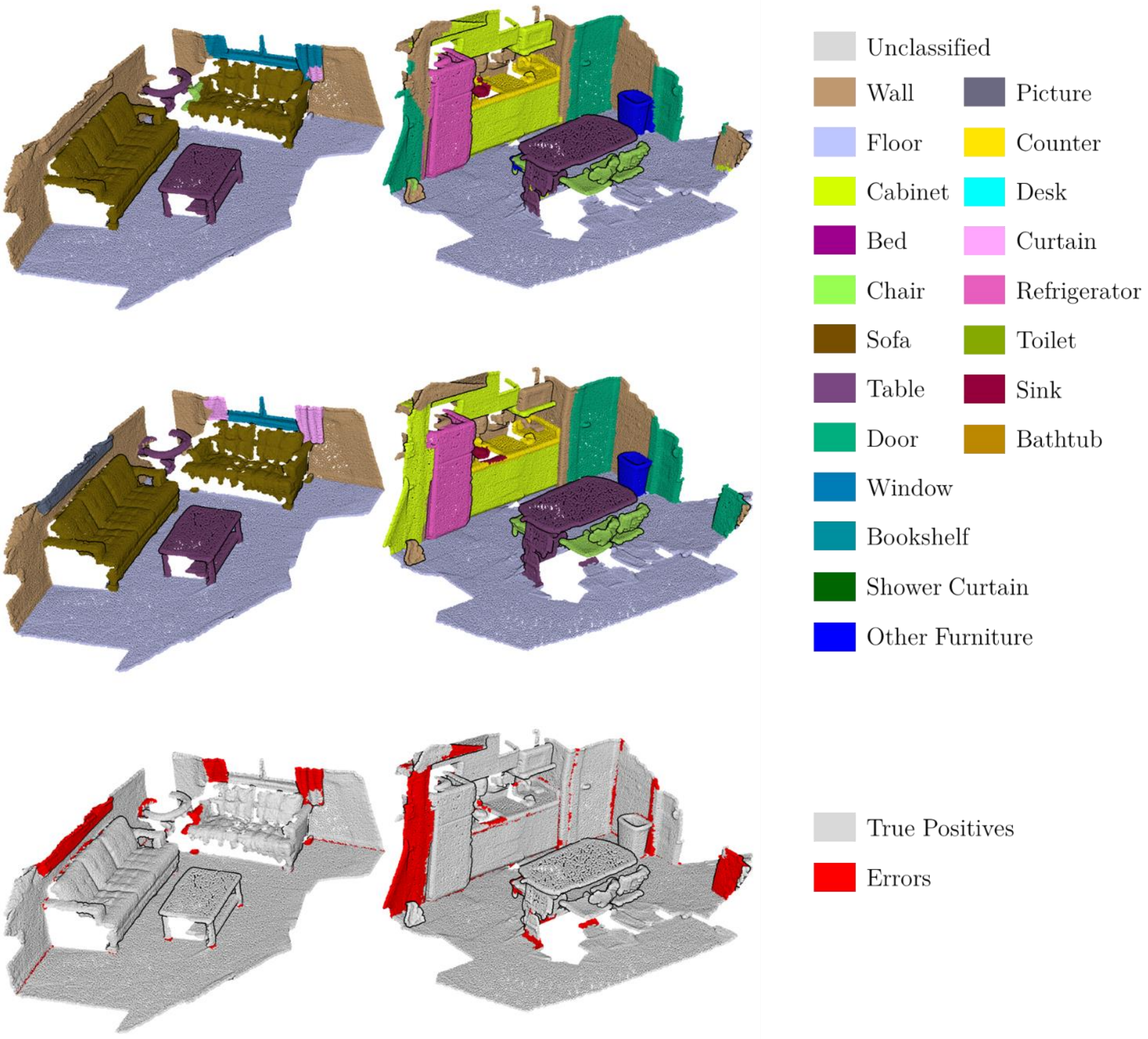

Unclassified
Wall
Picture
Floor
Counter
Cabinet
Desk
Bed
Curtain
Chair
Refrigerator
Sofa
Toilet
Table
Sink
Door
Bathtub
Window
Bookshelf
Shower Curtain
Other Furniture
True Positives
Errors

Figure 77. Illustration of two 3D scenes from Scannet dataset segmented with KPConv. Top: predictions. Middle: Groundtruth. Bottom: Errors.

A.3. Illustration of 3D objects part segmentation

In this appendix, we provide illustrations of 3D objects segmented with our KP-FCNN network on ShapeNetPart dataset.

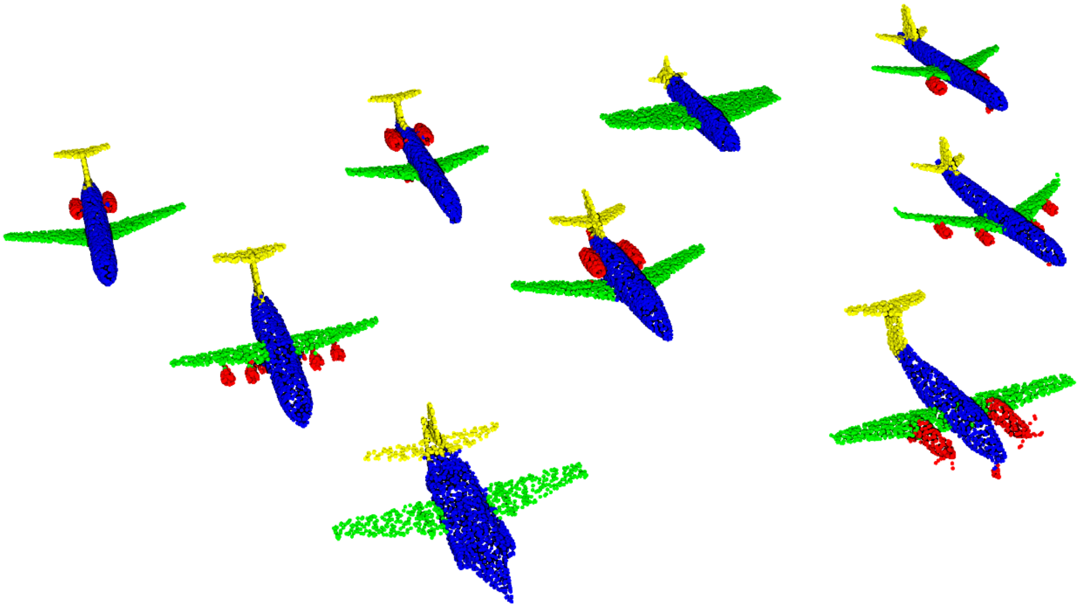


Figure 78. Examples of airplanes from ShapeNetPart dataset, segmented with KP-FCNN.

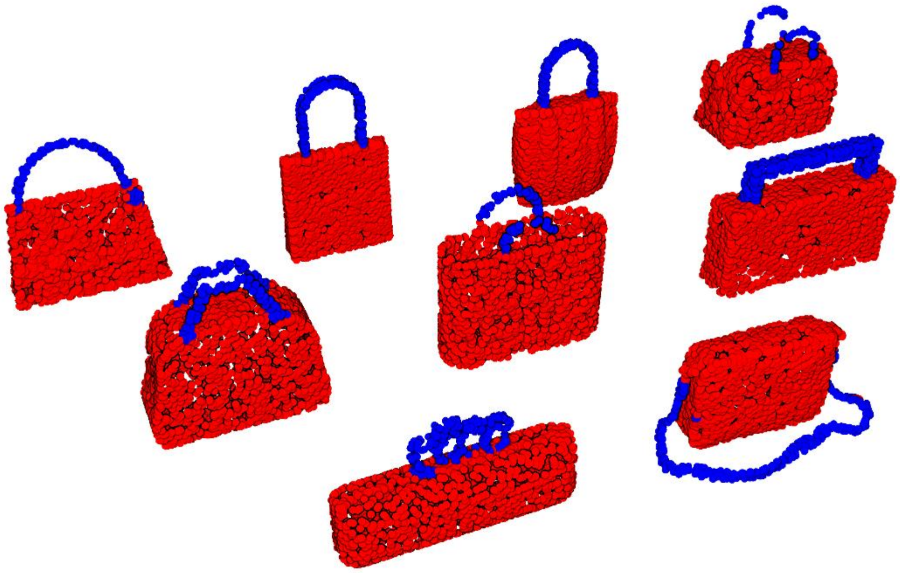


Figure 79. Examples of bags from ShapeNetPart dataset, segmented with KP-FCNN.

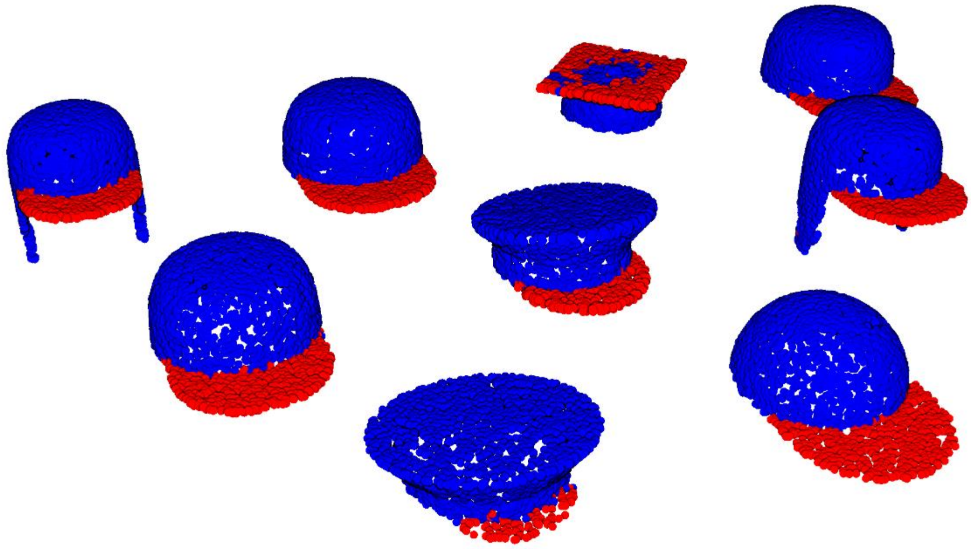


Figure 80. Examples of caps from ShapeNetPart dataset, segmented with KP-FCNN.

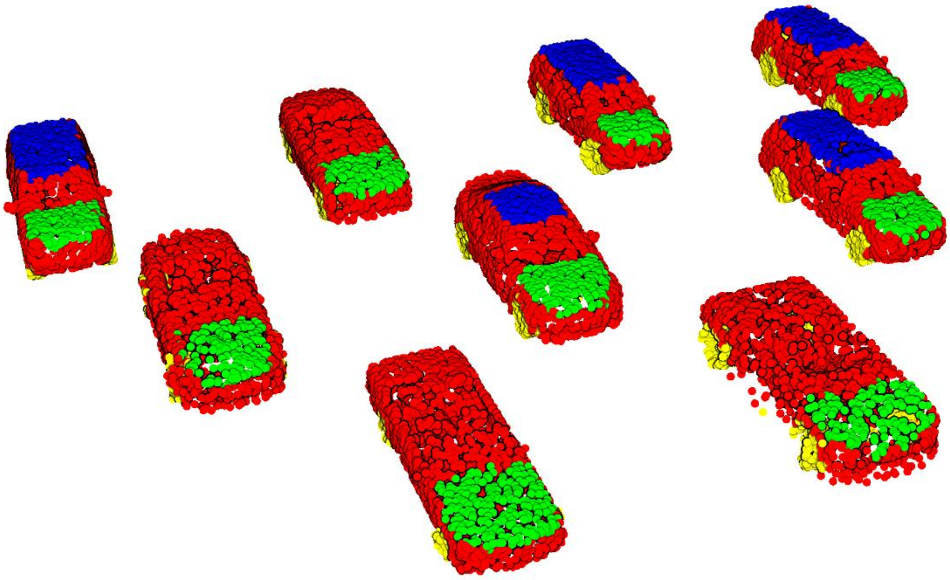


Figure 81. Examples of cars from ShapeNetPart dataset, segmented with KP-FCNN.

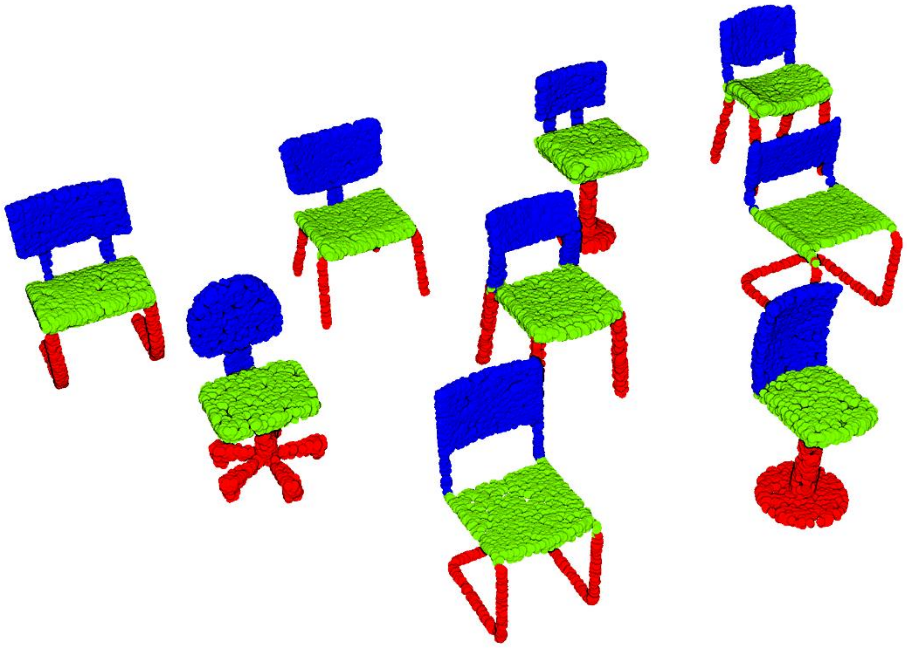


Figure 82. Examples of chairs from ShapeNetPart dataset, segmented with KP-FCNN.

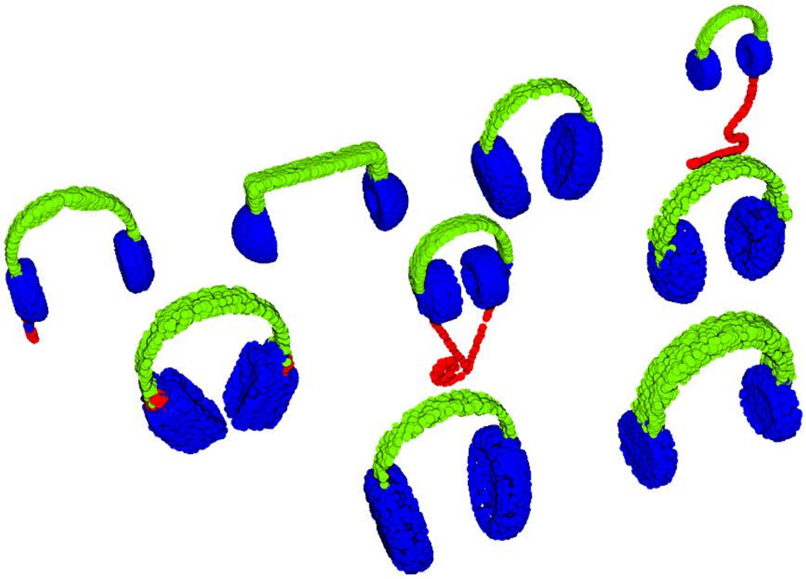


Figure 83. Examples of earphones from ShapeNetPart dataset, segmented with KP-FCNN.

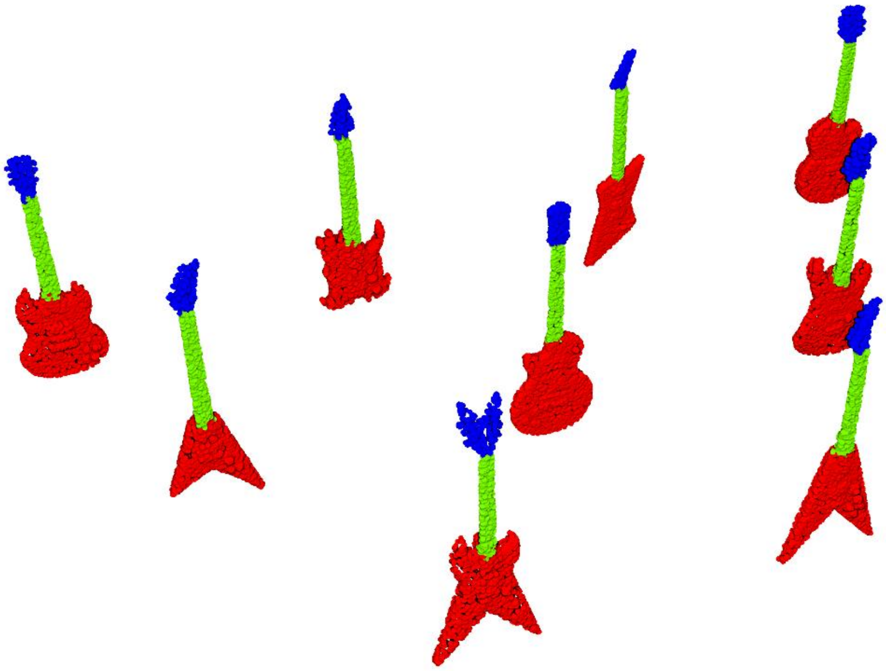


Figure 84. Examples of guitars from ShapeNetPart dataset, segmented with KP-FCNN.

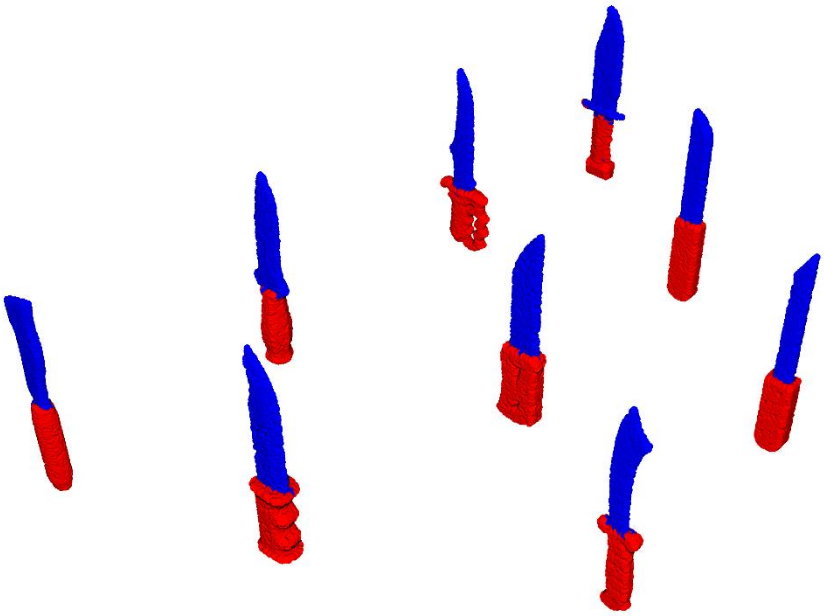


Figure 85. Examples of knives from ShapeNetPart dataset, segmented with KP-FCNN.

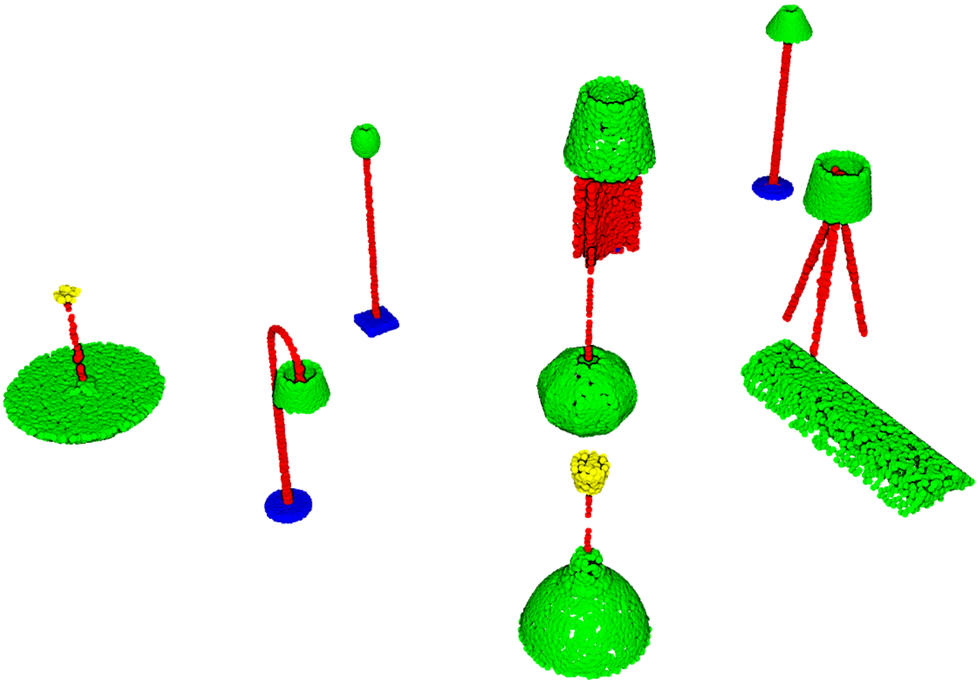


Figure 86. Examples of lamps from ShapeNetPart dataset, segmented with KP-FCNN.

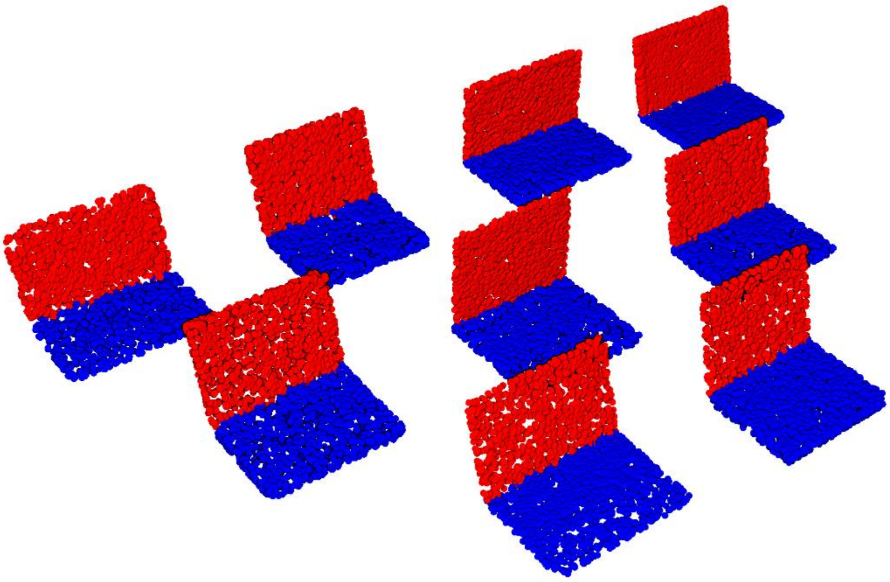


Figure 87. Examples of laptops from ShapeNetPart dataset, segmented with KP-FCNN.

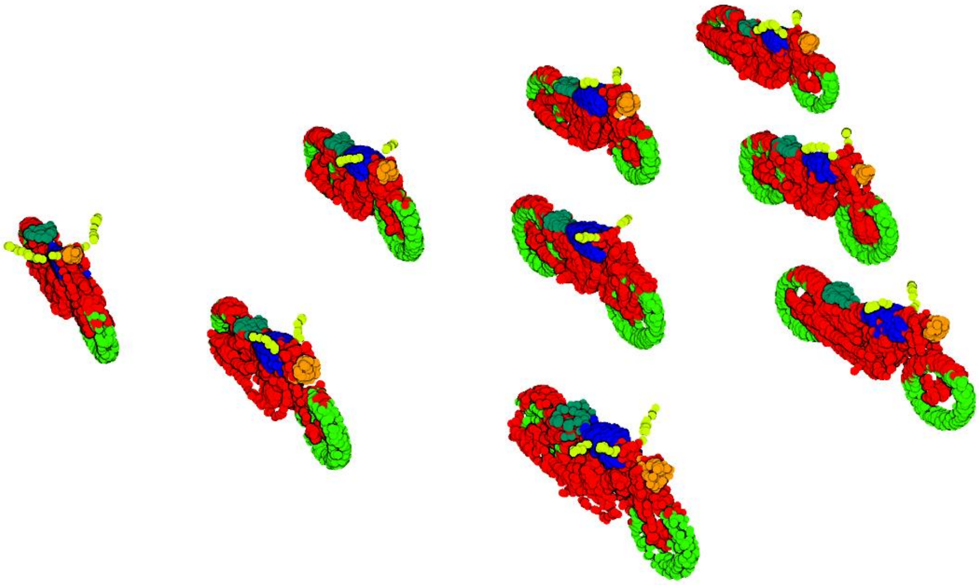


Figure 88. Examples of motorbikes from ShapeNetPart dataset, segmented with KP-FCNN.

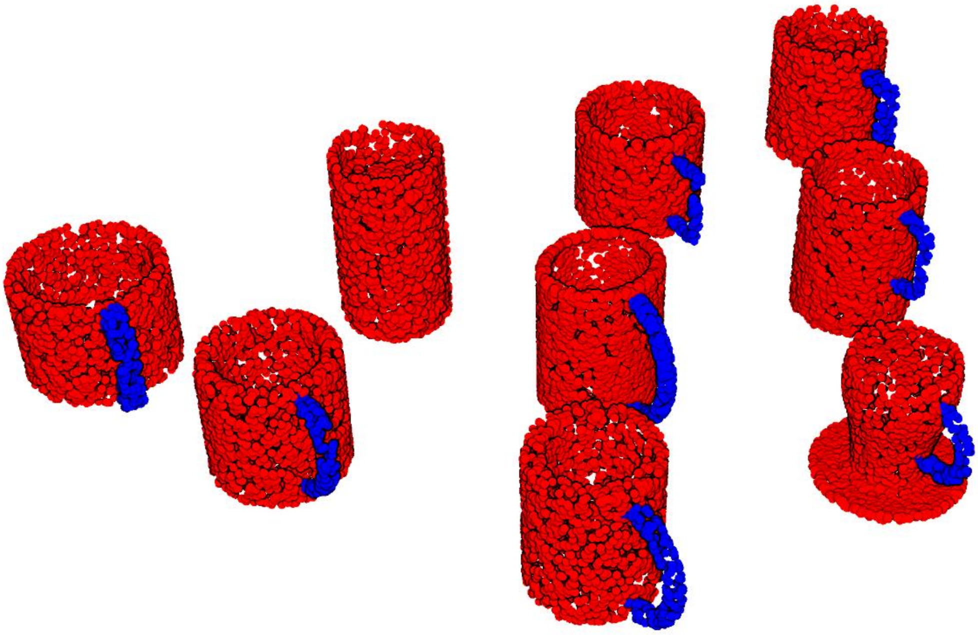


Figure 89. Examples of mugs from ShapeNetPart dataset, segmented with KP-FCNN.

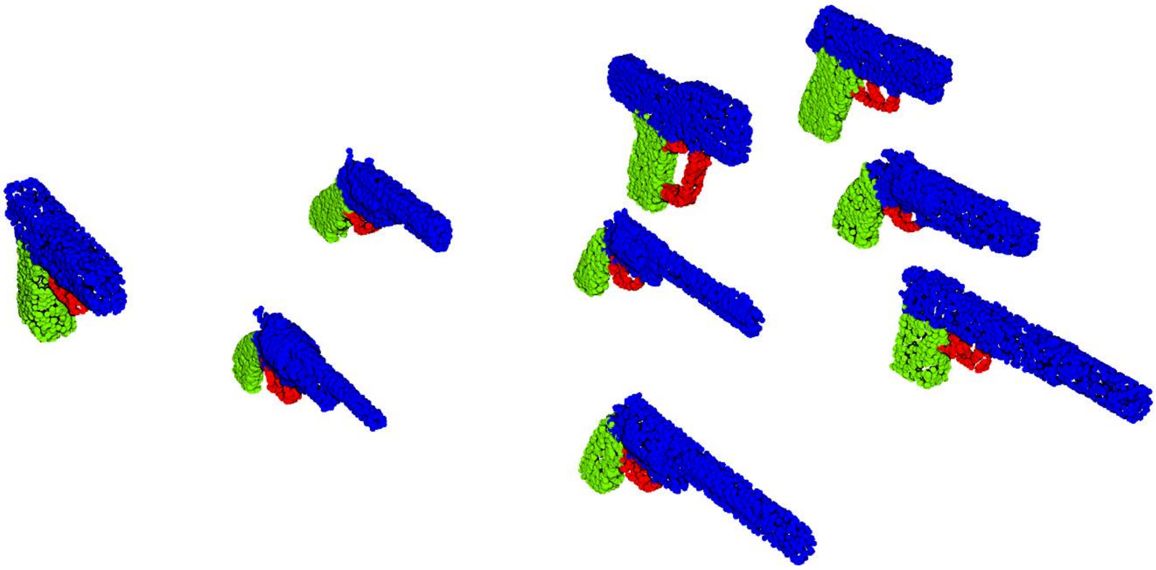


Figure 90. Examples of pistols from ShapeNetPart dataset, segmented with KP-FCNN.

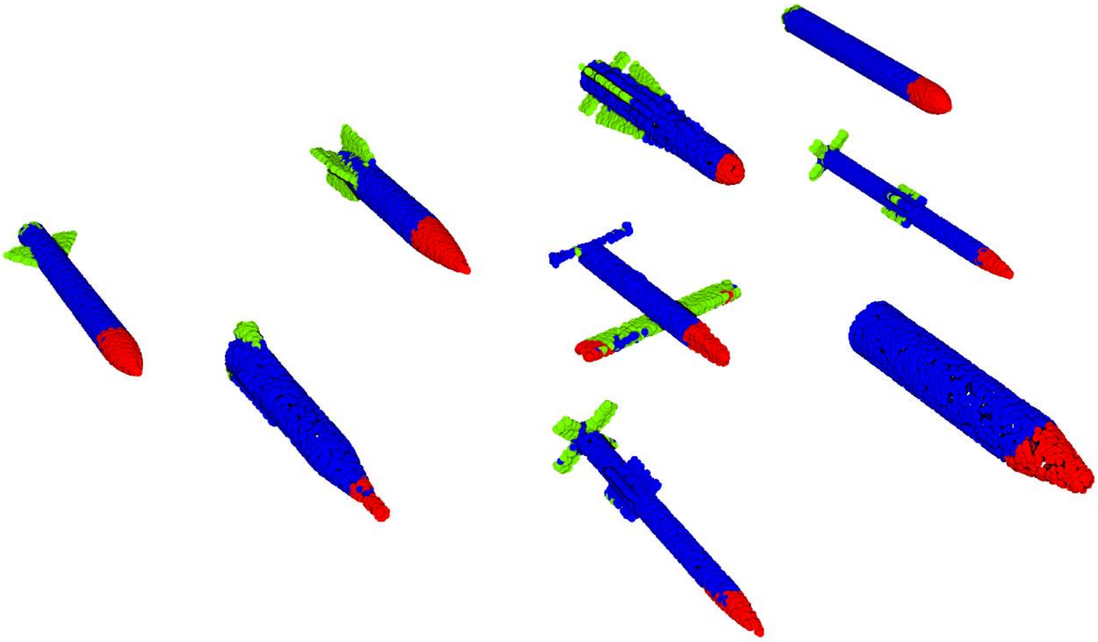


Figure 91. Examples of rockets from ShapeNetPart dataset, segmented with KP-FCNN.

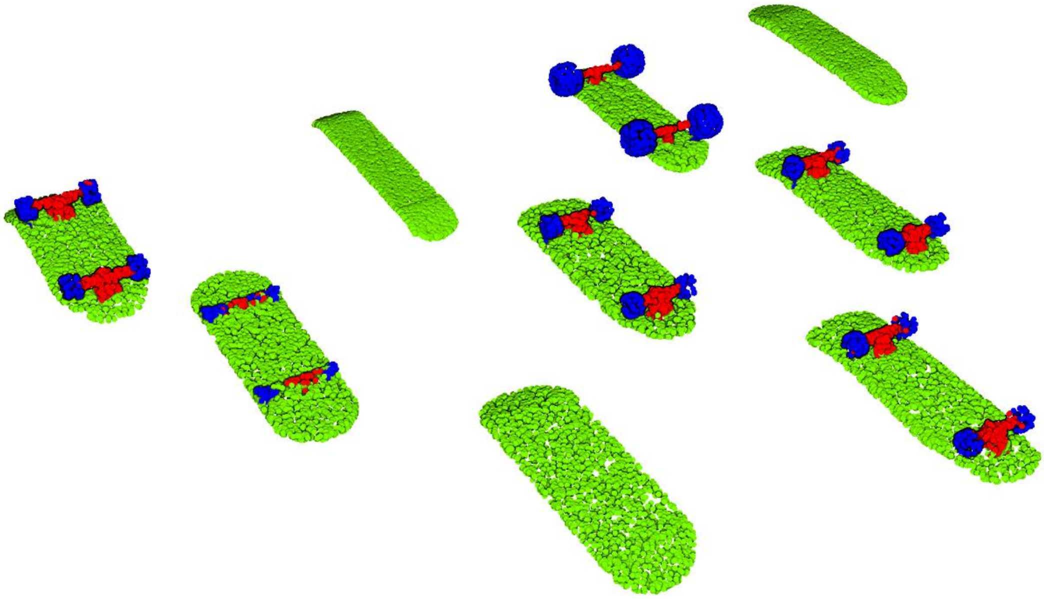


Figure 92. Examples of skateboards from ShapeNetPart dataset, segmented with KP-FCNN.

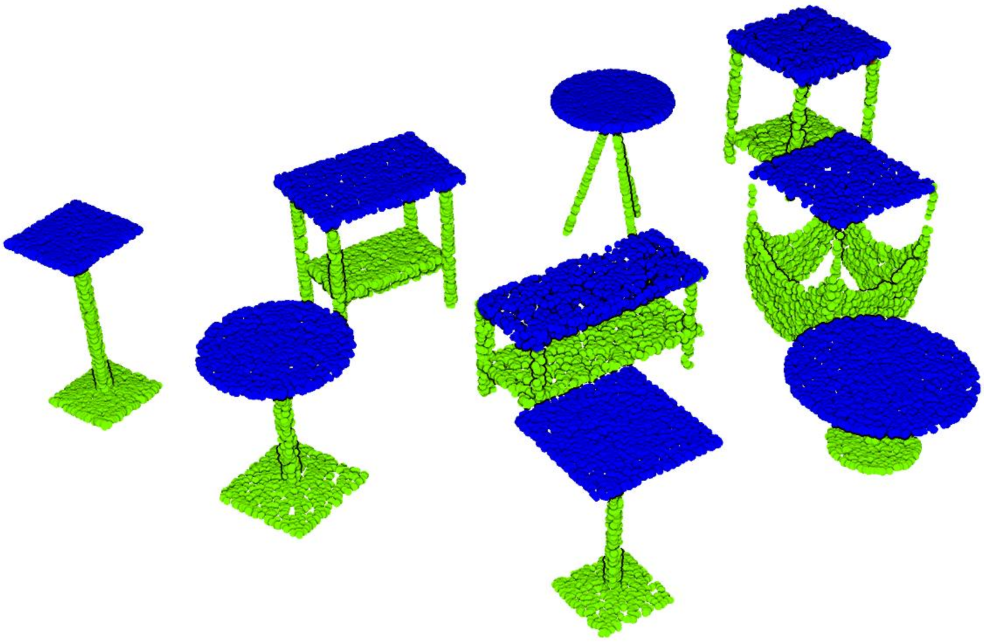


Figure 93. Examples of tables from ShapeNetPart dataset, segmented with KP-FCNN.

RÉSUMÉ

Aujourd’hui, de nouvelles technologies permettent l’acquisition de scènes 3D volumineuses et précises sous la forme de nuages de points. Les nouvelles applications ouvertes par ces technologies, comme les véhicules autonomes ou la maintenance d'infrastructure, reposent sur un traitement efficace des nuages de points à grande échelle. Les méthodes d'apprentissage profond par convolution ne peuvent pas être utilisées directement avec des nuages de points. Dans le cas des images, les filtres convolutifs ont permis l’apprentissage de nouvelles représentations, jusqu’alors construites « à la main » dans les méthodes de vision par ordinateur plus anciennes. En suivant le même raisonnement, nous présentons dans cette thèse une étude des représentations construites « à la main » utilisées pour le traitement des nuages de points. Nous proposons ainsi plusieurs contributions, qui serviront de base à la conception d’une nouvelle représentation convolutive pour le traitement des nuages de points. Parmi elles, une nouvelle définition de voisinages sphériques multi-échelles, une comparaison avec les k plus proches voisins multi-échelles, une nouvelle stratégie d'apprentissage actif, la segmentation sémantique des nuages de points à grande échelle, et une étude de l'influence de la densité dans les représentations multi-échelles. En se basant sur ces contributions, nous introduisons la « Kernel Point Convolution » (KPConv), qui utilise des voisinages sphériques et un noyau défini par des points. Ces points jouent le même rôle que les pixels du noyau des convolutions en image. Nos réseaux convolutionnels surpassent les approches de segmentation sémantique de l’état de l’art dans presque toutes les situations. En plus de ces résultats probants, nous avons conçu KPConv avec une grande flexibilité et une version déformable. Pour conclure notre réflexion, nous proposons plusieurs éclairages sur les représentations que notre méthode est capable d'apprendre.

MOTS CLÉS

3D, apprentissage profond, nuage de points, convolution, sémantisation

ABSTRACT

In the recent years, new technologies have allowed the acquisition of large and precise 3D scenes as point clouds. They have opened up new applications like self-driving vehicles or infrastructure monitoring that rely on efficient large scale point cloud processing. Convolutional deep learning methods cannot be directly used with point clouds. In the case of images, convolutional filters brought the ability to learn new representations, which were previously hand-crafted in older computer vision methods. Following the same line of thought, we present in this thesis a study of hand-crafted representations previously used for point cloud processing. We propose several contributions, to serve as basis for the design of a new convolutional representation for point cloud processing. They include a new definition of multiscale radius neighborhood, a comparison with multiscale k-nearest neighbors, a new active learning strategy, the semantic segmentation of large scale point clouds, and a study of the influence of density in multiscale representations. Following these contributions, we introduce the Kernel Point Convolution (KPConv), which uses radius neighborhoods and a set of kernel points to play the role of the kernel pixels in image convolution. Our convolutional networks outperform state-of-the-art semantic segmentation approaches in almost any situation. In addition to these strong results, we designed KPConv with a great flexibility and a deformable version. To conclude our argumentation, we propose several insights on the representations that our method is able to learn.

KEYWORDS

3D, deep learning, point clouds, convolution, semantic segmentation