Torch-Points3D: A modular multi-task framework for reproducible deep learning on 3D point clouds

Thomas Chaton, Nicolas Chaulet, Sofiane Horache, Loic Landrieu

Torch-Points3D: A Modular Multi-Task Framework for Reproducible Deep Learning on 3D Point Clouds

Thomas Chaton*
thomas.chaton.ai@gmail.com

Nicolas Chaulet*
Principia Labs
nicolas@principialabs.co.uk

Sofiane Horache
Centre de Robotique, CAOR
Mines ParisTech, PSL university
sofiane.horache@mines-paristech.fr

Loic Landrieu
LASTIG, Univ Gustave Eiffel, ENSG
IGN, F-94160 Saint-Mande, France
loic.landrieu@ign.fr

Abstract

We introduce Torch-Points3D, an open-source framework designed to facilitate the use of deep networks on 3D data. Its modular design, efficient implementation, and user-friendly interfaces make it a relevant tool for research and productization alike. Beyond multiple quality-of-life features, our goal is to standardize a higher level of transparency and reproducibility in 3D deep learning research, and to lower its barrier to entry.

In this paper, we present the design principles of Torch-Points3D, as well as extensive benchmarks of multiple state-of-the-art algorithms and inference schemes across several datasets and tasks. The modularity of Torch-Points3D allows us to design fair and rigorous experimental protocols in which all methods are evaluated in the same conditions.

The Torch-Points3D repository : https://github.com/nicolas-chaulet/torch-points3d.

1. Introduction

In recent years, the field of automated analysis of 3D data has been transformed by the development of new dedicated neural network architectures. This sudden spur in methodological advancements is reminiscent of the revolution undergone by image analysis in the early 2010s, initiated by AlexNet [23]. The number of methods and papers dedicated to 3D data presented at major vision conferences is now on par with images, and keeps growing each year.

The young field of deep learning for 3D has greatly pushed forward the state-of-the-art performance of automated analysis of point clouds for numerous tasks. For example, the top performance on the indoor dataset S3DIS

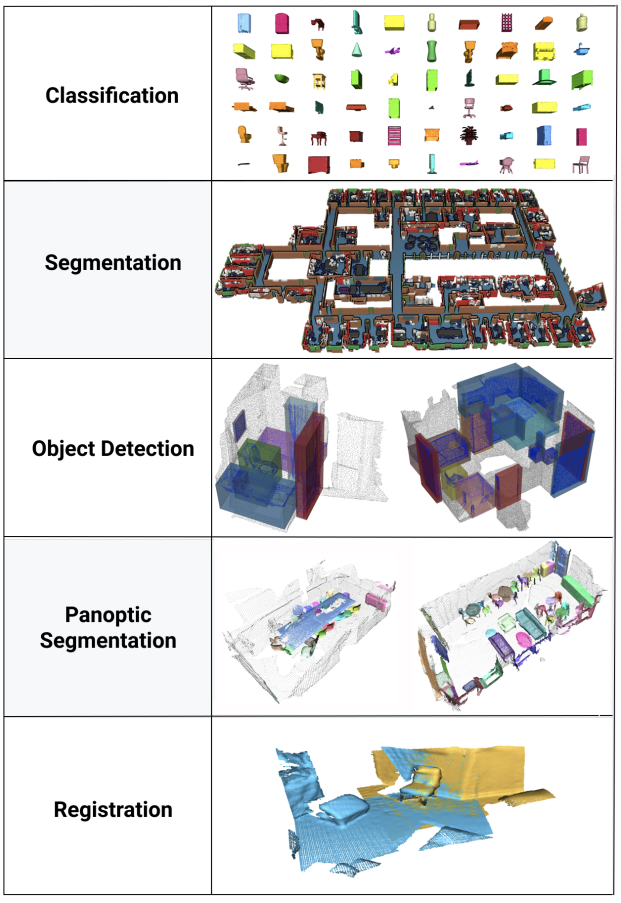


Figure 1: Torch-Points3D supports multiple tasks such as classification, segmentation, object detection, panoptic segmentation, and registration. All visuals have been produced by the framework.

* equal contribution

[1] (6-fold) have gone from 41.7% mIoU points (mean Intersection over Union) in 2017 [33], to 62.1% only a year later [25], and up to 70.6% in 2019 [38]. While this rapid methodological development is of course beneficial to the community, its fast pace comes with several shortcomings:

- Adding new datasets, tasks, or neural architectures to existing approaches requires a substantial commitment, often tantamount to a complete re-implementation. This limits the use of new networks, and prevents exhaustive comparisons.
- Handling large 3D datasets efficiently requires a significant time investment, and overcoming many implementation pitfalls. This creates soft barriers to entry, restraining the diffusion of new ideas.
- There is no standard approach for inference scheme and metrics in research papers. This makes assessing the intrinsic performance of new algorithms difficult, and their reproducibility not always straightforward.

In this paper, we introduce Torch-Points3D, a flexible and powerful development framework aiming to address these issues. In short, the purpose of our framework is to become for 3D point clouds what `torchvision` [31] or `pytorch-geometric` [13] have become for images and graphs respectively. More generally, our goal is to address the growing technical debt pervasive to machine learning research codes. This is particularly crucial for 3D data, for which many steps require special care in order to not invalidate investigations, from data loading and preprocessing to the computation of performance metrics. By proposing tried and tested implementations, which only get more robust as the user community grows, we aim to further increase the rigor of 3D deep learning.

Torch-Points3D is intended for novices as much as experts. It provides intuitive interfaces with most open-access 3D datasets, re-implementations of many of the top-performing networks, classic data augmentation schemes and validated performance metric. This allows researchers to focus on the development of core algorithms and test them on all available datasets with minimal effort. The different components of Torch-Points3D are highly customizable and can be plugged into one another with a unified system of configuration files. Users can then easily swap backbone networks for a given task, leading to the efficient selection of the best-suited algorithms, as well as facilitating comparison with new approaches. On this front, our framework makes it easy to standardize experimental protocols, ensuring both reproducibility and that models' performances are evaluated on equal footing.

Finally, we propose a multitude of quality-of-life features such as open logs with Weight and Biases [2], versatile model configuration handling with Hydra [41], and bespoke visualization functions as illustrated in Figure 1.

To illustrate the capabilities of Torch-Points3D, we propose several numerical experiments:

- We evaluated the performance of different backbone networks in a recent object detection method.
- We benchmark different methods over several datasets with a unified protocol, aiming to assess their inherent performance.
- We quantify the benefit of implemented test-time enhancers, such as voting schemes.
- We present our point clouds registration implementation within our framework, combining recent papers and reaching state-of-the-art performance.
- We share key findings about speed enhancing procedures that can be leveraged on any model supported by the framework.

2. Related Work

The first deep learning methods for 3D point clouds analysis relied on image [3] or voxel-based representations [36, 37]. PointNet [33, 34] was the first network whose architecture was specifically designed to handle unordered 3D point clouds. Since then, a multitude of approaches have been proposed, see the comprehensive review by Guo *et al*. [17]. Manipulating large 3D point clouds requires extensive implementations and to this end, several open-source frameworks have been proposed.

Kaolin Krishna Murthy J. *et al*. from Nvidia shared a Pytorch framework aiming to accelerate 3D deep learning research [19]. It implements boilerplate code for handling meshes, voxels, and point clouds.

Pytorch3D Nikhila Ravi *et al*. proposed another Pytorch-based framework, similar to Kaolin, for 3D computer vision research [35]. Its key features include bespoke data structure for storing and manipulating meshes, a differentiable mesh renderer, camera position optimization, bundle adjustment, and several mesh-based deep models [16].

Det3D Zhu Benjin. *et al*. open-sourced a 3D Object Detection Pytorch toolbox [47], providing out-of-the-box implementations of many 3D object detection algorithms [26, 42], as well as compatibility with several datasets such as KITTI [14] and nuScenes [4].

OpenPCDet and **MMDetection3D** [45] are open-source 3D object detection PyTorch toolboxes, part of the Open-

MMLab project developed by CUHK Multimedia Lab.

However, a unifying framework for multi-tasks, multi-models, multi-datasets for reproducible 3D point clouds deep learning has yet to be proposed. In this paper, we introduce Torch-Points3D, which aims to answer this need.

3. The Framework

Torch-Points3D was written from scratch according to the following design principles: it should be modular, extendible, and support multiple models, tasks, and datasets. Figure 2 illustrates the different components of our framework and how they interact together. A key design principle is that the components are independent from one another allowing users to plug and play their own contributions. This could be a dataset, a custom convolution or a new data augmentation strategy for example. In this section, we illustrate how these ideas translate into a versatile, easy-to-use interface.

3.1. Dataset Handling

The growing number of large-scale 3D public datasets has a beneficial effect on both the academic community and industrial actors interested in automated 3D point clouds analysis. While image formats have been standardized for years, this is not the case for 3D data. Hence, downloading, reading, cleaning, and processing data into a deep learning-ready format requires specific implementations, discouraging researchers to perform experiments on many datasets.

Building on `pytorch-geometric` implementations, we propose an adapted interface for handling 3D datasets, from automatic downloading to data-augmentation. To ensure maximum versatility, these operations can be set in a compact and modular configuration file system.In Section 3.6, we present the different datasets currently implemented in Torch-Points3D.

3.2. Modular Model Configuration

The majority of competitive deep learning networks for 3D analysis rely on the 3-step, U-net like approach initially proposed by PointNet++ [34]:

(i) **Encoding:** The input point cloud is iteratively subsampled, and local features are computed for each point of a subsampling level based on neighboring points in the previous subsampling level. This step is built upon a `down_conv` operation, which varies for different networks.

(ii) **InnerMost:** A global descriptor per instance is computed by pooling the last subsampling level into a single vector. This embedding can be processed further with fully connected layers.

Listing 1: Configuration file `shapeNet-fixed.yaml` for the Shapenet dataset, with a fixed number of sampled points per object.

```
data:
  class: shapenet.ShapeNetDataset
  task: segmentation # associated task
  dataroot: ./data #data path
  normal: True #use normals
  pre_transforms: # preprocessing
    - transform: NormalizeScale #to unit sphere
    - transform: GridSampling3D
      params: #size of voxel
        size: 0.02
  train_transforms: # Train data augmentation
    - transform: FixedPoints
      lparams: [2048] # random sampling
    - transform: RandomNoise
      params: #Gaussian noise
        sigma: 0.01
        clip: 0.05
  test_transforms: #Test data augmentation
    - transform: FixedPoints
      lparams: [2048] # random sampling
```

(iii) **Decoding:** The learned features are interpolated back from lower sub-samplings levels, up to the original point clouds, and processed based on their neighbors' features, forming the `up_conv` operation. Features at mirror-level of subsampling for the encoder can be concatenated for additional spatial precision, through so-called `skip` connections. The result of the innermost module can also be concatenated with point features at different subsampling level.

Based on this versatile architecture, we propose a system of configuration files able to encode most segmentation networks. For example, the official single-scale PointNet++ is implemented by the configuration file in Listing 2. The model `pointnet2` described above can now be trained on any supported dataset such as Shapenet [5] with a simple command:

```
python train.py task=segmentation \
dataset=shapenet-fixed model_type=pointnet2 \
model_name=PointNet2
```

3.3. Implemented Networks

Torch-Points3D implements several convolution methods which present an interest in terms of performance or versatility: PointNet, RandLANet, KPConv, RS-CNN, and Minkowski Engine. The PointNet-based architectures [33, 34] are the simplest point convolution methods, making them both easy to use and understand. The convolution kernels for RandLANet [18] allows for efficient point clouds processing with a random sampling strategy. KPConv [38] proposes a kernel-based generalization of 2D convolution to

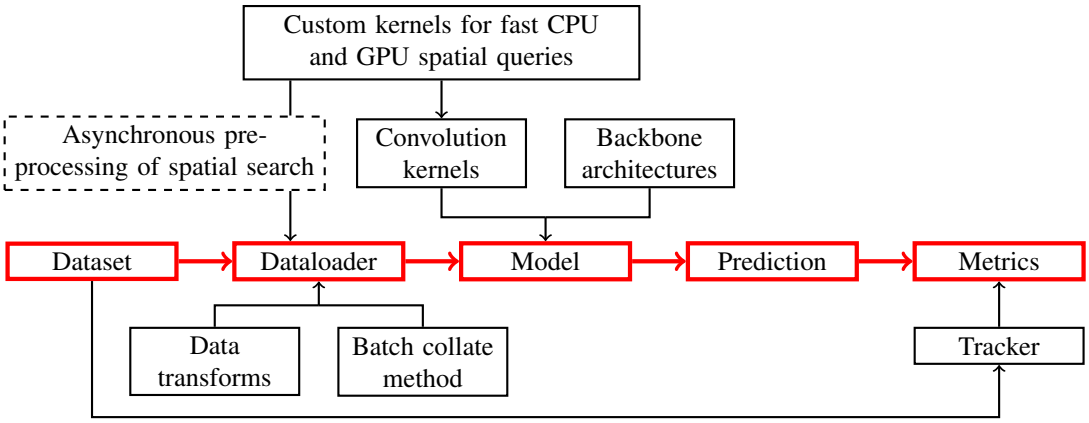


Figure 2: Overall architecture of the framework, with data flow highlighted in red. `Dataset` implements the core loading mechanism of raw data and creates objects containing the points' positions, relevant features, and labels. Those objects are then passed through data augmentation transforms and aggregated into batches in the `Dataloader`. They are finally passed to the `Model`, which outputs the prediction. A tracker evaluates the performance from predefined `Metrics` and publish results on the console and/or directly on Weight and Biases (`wandb.ai`).

Listing 2: Configuration file for PointNet++ [34].

```
pointnet2:
  class: pointnet2.PointNet2 # Model class
  conv_type: "DENSE" # Convolution type
  down_conv: #encoder
    module_name: PointNetMSGDown
    npoint: [512, 128] #subsampling levels
    radii: [[0.2], [0.4]] #neigh. radius
    nsamples: [[64], [64]] #neigh. count
    down_conv_nn: [[[FEAT+3, 64, 64, 128]],
       [[128+3, 128, 128, 256]]]
  innermost: #process learned feature
      module_name: GlobalDenseBaseModule
      nn: [256 + 3, 256, 512, 1024]
  up_conv: #decoder
      module_name: DenseFPModule
      up_conv_nn:
          [[1024 + 256, 256, 256],
            [256 + 128, 256, 128],
            [128 + FEAT, 128, 128, 128]]
      skip: True # use skip connection
  mlp_cls: #produce class scores
      nn: [128, 128]
      dropout: 0.5
```

3D point clouds, and RS-CNN [29] capture the complexity of local shapes by modeling spatial relationships between points. Minkowski Engine [7] relies on a fine-grained voxelization of point clouds, efficiently processed with sparse-convolutions.

These convolution schemes can be integrated into a backbone network architecture for semantic segmentation or object classification for example. We propose several such backbone, from a simple U-Net, to a multi-scale architecture, and more modern ResNet-like architectures as proposed in [7] or [38].

Finally, the framework also implements task-specific heads for object detection and Panoptic segmentation. In aprticular, VoteNet [32] uses Hough voting to regress bounding boxes on 3D point clouds, and PointGroup [20] uses a clustering scheme to perform instance segmentation.

While the compact configuration format described in the last section is designed to make U-Net-like networks easier, users can also easily define their own types of configuration file and model architectures in Torch-Points3D. For example, VoteNet and PointGroup use custom configurations. Our configuration system is meant to be flexible, and existing configurations serve more as guides for newcomers than rigid templates. At the time of writing, superpoint-based [25, 24] and multi-modal methods (eg. 3D points + images [11]) are not yet implemented. However, we plan to add both in the near-future.

3.4. Multi-Task Support

The ability of our framework to handle different tasks ensures its versatility and allows multi-source supervision [28]. We have currently implemented five tasks, illustrated in Figure 1, and their associated losses and metrics: classification, semantic segmentation, panoptic segmentation [22], registration, and object detection.

Adding new tasks with their associated datasets and metrics can be done in isolation from the rest of the project, allowing users to extend the framework without necessarily understanding its inner working in details.

3.5. Transparency and Reproducibility

Reproducibility of experiments is not only necessary when assessing the suitability of different networks to a

given task or dataset, but also to back scientific claims in academic papers. To this end, we have ensured the integrated compatibility of our framework with the Hydra configuration system [41] as well as the experiments tracker Weight and Biases [2] (`wandb.ai`). This online tool can store training runs along with their logs, metric visualization, configuration files, git commit hash, and our custom Pickle-based checkpoints. This total transparency allows users to compare their own experiments with our reference runs, and the models can be directly downloaded for fine-tuning tasks. In the Appendix, we reports an example of log visualization hosted on `wandb.ai`.

Another benefit of our unified approach is standardizing the learning and testing procedure. Indeed, the field of 3D analysis lacks a common ground when it comes to evaluation and augmentation strategies, both at test and training time. This makes experiments across different papers hard to compare, and could potentially obscur the intrinsic performance of new models. In Section 4.2, we propose standard protocols for different datasets and reproduce an array of experimental results.

3.6. Supported Datasets

Torch-Points3D supports multiple academic datasets with automatic data download, pre-processing, as well as automatic result submission when available.

- **ScanNet** is an indoor RGBD dataset containing 1 201 train scenes and 312 test scenes [10]. It can be used for semantic segmentation, object detection, and panoptic segmentation.
- **S3DIS** is a large-scale indoor RGB point cloud dataset covering three separate office buildings, over 6 000m^2, and containing 278 million points with instance-level object and semantic annotations. We implement three different sampling for batch-training, based on rooms [1], cubes [34], or spheres [38].
- **ModelNet10/40** is a dataset composed of over 12 000 CAD models from 10 and 40 categories [39].
- **Shapenet** is a collection of over 200 000 CAD models annotated across a hierarchy of 3 135 classes [5]. On top of classification and semantic segmentation, Torch-Points3D also implements the task of hierarchical semantic segmentation, as well as adapted metrics.
- **3DMatch** is an RGBD dataset [44] widely used for 3D reconstruction and point cloud registration.
- **KITTI Odometry** contains 21 sequences of LiDAR frames [15], with ground truth poses for the first ten. KITTI Odometry is commonly used as benchmmark for SLAM LIDAR, but can also be used to train and evaluate point cloud registration networks ([8]).

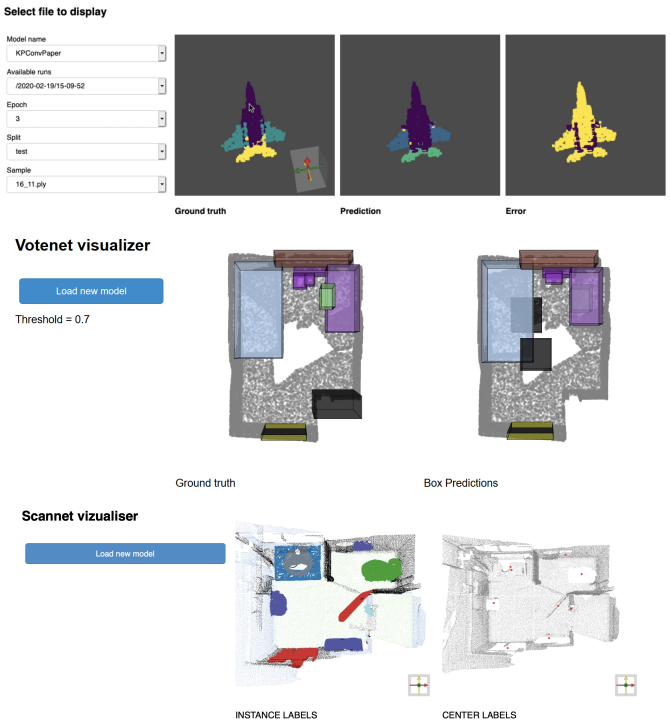


Figure 3: Several of the visualization tools available.

3.7. Built-in Visualization

Torch-Points3D provides several custom visualization tools directly available within notebooks and using the dashboarding library panel. This feature can be used to explore datasets, debug models, or illustrate predictions, as shown in Figure 3.

3.8. Ease-of-Use

As corroborated by the numerous pull-requests submitted by contributors, and testimonies from industrial partners, Torch-Points3D is accessible despite the quantity of code. Adding new datasets, new models or new base modules can be done without interfering with the rest of the code base while benefiting from existing methods for data augmentation, metric evaluation or training procedures at zero cost. A convenient way to start familiarizing one self with the framework, is by running the proposed illustrative IPython notebooks. More details as well as installation instructions are available in the Appendix.

4. Numerical Experiments

In this section, we present several case studies demonstrating some of the capabilities of Torch-Points3D , such as backbone swapping and fair benchmarking. The detailed configuration of the experiments, along with the evolution of metrics along the runs, are available as WandB projects

accessible through the framework GitHub repository.

4.1. VoteNet with Different Backbones

The VoteNet network, introduced by Qi *et al*. [32], performs end-to-end object detection in 3D point clouds. It relies on a PointNet++-like backbone network to extract point features, which are then used by an object-center voting module and a box-proposal module.

In Table 1, we assess the performance of different networks by replacing the PointNet++ backbone with more recent alternatives, such as RS-CNN [29], KPConv [38], and Minkowski Engine [7].We used the same architecture than the ones used in our semantic segmentation benchmark, see Section 4.2. Further adaptation to the task, such as truncating the decoder as recommend in VoteNet, would certainly be beneficial but are out of the scope of this paper. Otherwise, we use the hyper parameters, training and data augmentation procedures proposed in the original work. Changing the backbone is as simple as editing the model's configuration file as presented in Listing 3.

While the RS-CNN and KPConv backbones underperformed, the mean average precision 50% IoU is improved slightly by switching to a Minkowski Engine network. Overall, the PointNet++ architecture seems particularly well-suited to the task of object detection, as also observed by Xi *et al*. [40].

Note that our results differ slightly from the original paper, as our metric implementation is slightly altered: successful box recoveries only count as positive for a given class if the predicted class is correctly inferred as well.

Listing 3: Model configuration used for switching the backbone of VoteNet [32] to KPConv [38].

```
VoteNetKPConv:
    class: votenet2.VoteNet2
    conv_type: "PARTIAL_DENSE"
    define_constants:
        num_proposal: 256 # num. box proposals
        num_classes: 18 # semantic classes
    backbone:
        model_type: "KPConv" # backbone type
        extra_options:
            in_grid_size: 0.05 # input grid size
```

4.2. Benchmarking with Torch-Points3D

As one of the first and easiest to use datasets, S3DIS [1] has been used as a standard measure of the performance of state-of-the-art methods. However, there is a large discrepancy in how methods are evaluated, which makes it hard to compare their performance.We propose a common evaluation protocol, largely inspired by the one proposed by [38].

Table 1: Impact of the backbone choice on VoteNet performances. mAP@r stands for the interclass mean average precision with a detection threshold of $r\%$ IoU.

VoteNet Backbone	mAP @25	mAP @50
PointNet ++ [34]	**54.2**	30.1
RS-CNN [29]	51.6	29.5
KPConv [38]	48.9	29.2
Mink. Engine [7]	53.8	**30.2**

- **Pre-processing:** S3DIS is comprised of 6 folds, each containing a collection of point clouds corresponding to single rooms. We aggregate these clouds to obtain one cloud per fold, each corresponding to one entire level of an office building. We sample this cloud with respect to a 4cm grid.

- **Training:** In each epoch, we sample 3 000 spheres of radius 2m, centered around random points picked with a probability inversely proportional to the square root of their class frequency.

- **Optimizers:** The parameters of the optimizers are given in the configuration file presented in Listing 4. In our experiments, Stochastic gradient Descent (SGD) had slower convergence but higher generalization than momentum-based optimizers such as [21].

- **Inference:** 2m-radius spheres are sampled along a $2 \times 2 \times 2$m grid *once*. The class probability associated to points present in several spheres are averaged. To compute the final metrics on the original clouds, we perform nearest neighbor interpolation.

- **Metrics:** We report the Overall Accuracy (OA) and Mean Intersection over Union (mIoU) over classes obtained by cross-validating over the 6-folds. We add an early stopping scheme in which the model is evaluated on the epoch whose model has the highest mIoU on a validation set, obtained by withholding selected rooms from the training set.

We also devise a similar protocol for ScanNet [10]. The differing steps are as follows:

- **Pre-processing:** We subsample with a 5cm grid.

- **Training:** Batches are collections of rooms subsampled to 50 000 points.

- **Metrics:** We report the OA and mIoU on the validation set using the model of the last epoch.

Listing 4: Optimizer hyper-parameters used for training all models on S3DIS.

```
epochs: 300 # Number of epochs
batch_size: 8
optim:
    base_lr: 0.01 # Learning rate
    grad_clip: 100 #gradient clipping
    optimizer:
        class: SGD # Optimizer
        params: # SGD parameters
            momentum: 0.02
            lr: \${training.optim.base_lr}
            weight_decay: 1e-3
    lr_scheduler:
      class: ExponentialLR
      params:
        gamma: 0.9885 # /10 every 200 ep.
    bn_scheduler: # Batch Normalization Scheduler
        bn_policy: "step_decay"
        params:
            bn_momentum: 0.02
```

Table 2: Benchmarking of four different methods on the task of semantic segmentation for two different datasets: S3DIS [1] with 6-fold cross validation and ScanNet [5] evaluated on the valdiation set.

Model	S3DIS 6-folds		ScanNet	
	OA	mIoU	OA	mIoU
KPConv [38]	**86.4**	**66.3**	85.5	59.9
Mink. Engine [7]	86.0	65.9	**87.2**	**65.0**
RS-CNN [29]	83.2	62.9	79.8	47.2
PointNet++ [34]	81.06	56.7	80.7	49.3

In Table 2, we report the performance of four networks (PointNet++ [34], RS-CNN [29], Minkowski Engine [7], and KPConv [38]) on both S3DIS and ScanNet. Each of these algorithms share the exact same learning and inference procedure. This allows us to appreciate their performances *all other things being equal*.

Ze Liu *et al*. [30] interestingly demonstrate in their recent investigation that the choice of convolution type (pointnet-like, pointCNN [27], KPConv, and their own PoolPos) have little impact when evaluated with a shared architecture. We can hence attribute the performance in Table 2 to the difference in architectures, namely the depth and subsampling/upsampling operations. We also remark that Ze Liu *et al*.'s experiments are particularly easy to replicate with Torch-Points3D on any of the proposed dataset, as most convolution schemes are already implemented.

4.3. Inference Schemes

A common scheme to increase the performance of a model is to perform several inference runs—with data augmentation, and to output their average probability. While this method slows down inference, the increase in performance can be justified for non time-sensitive applications such as digital twin modeling.

Table 3: Improvement in terms of mIoU provided by inference-time voting schemes on S3DIS 6-folds.

Models	no voting	with voting
KPConv [38]	**66.3**	67.2
Mink. Engine [7]	65.9	**69.1**
RS-CNN [29]	62.9	64.6
PointNet++ [34]	56.7	59.0

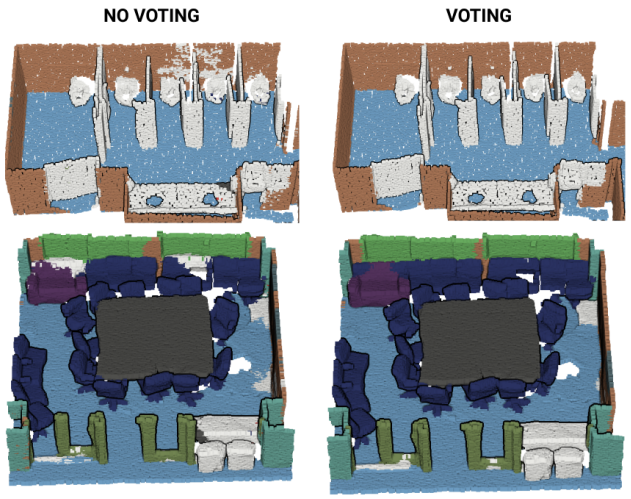


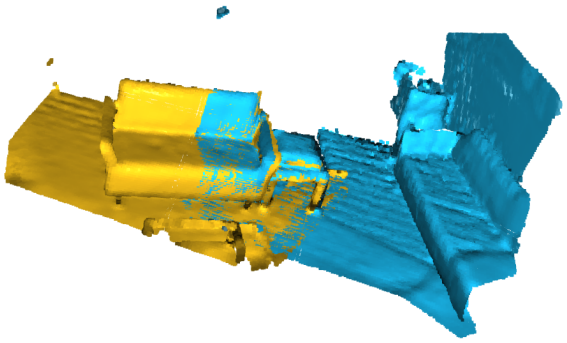
Figure 4: Segmentation predictions with and without voting. We can observe voting tends to create smoother, more accurate predictions.

In Table 3, we report the performance of different models with and without a 3-run average. The performance increase is noticeable, with an average increase from 1 to 3 points of mIoU. Interestingly, we remark that the relative order of performance of KPConv and Minkowski Engine are reversed by a non-negligible margin when using this inference scheme. In Figure 4, we illustrate the improvement provided by this inference scheme.

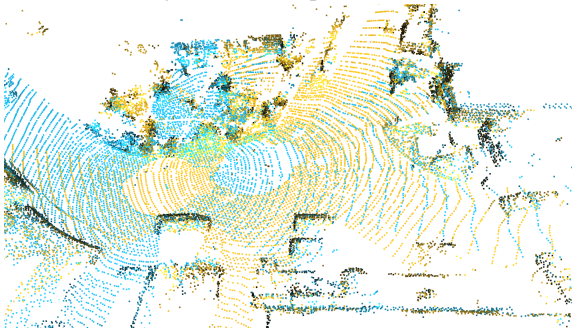
4.4. Registration

Registration is the task of finding a rigid transformation aligning several 3D point clouds. Neural networks can be trained to compute point features whose pairing determine the sought-after transformation, either end-to-end or with robust nondifferentiable estimators such as RANSAC, FGR[46], or the recent TEASER[43].

We implement a full registration pipeline within Torch-Points3D, using a Minkowski Engine backbone as sug-



(a) Registration example on 3DMatch.



(b) Registration example on KITTI odometry.

Figure 5: Qualitative registration results between two point clouds (in blue and yellow).

Table 4: Success rate (in %) on the 3DMatch and KITTI odometry datasets. Results obtained within TorchPoints3D. A *success* is defined by an error under 15 degree and 0.3 m for 3DMatch, and 2 degrees and 0.6 m for Kitti Odometry. * model taken from the paper's repository.

Methods	3DMatch	KITTI
DGR*	92.6	96.9
FCGF+ TEASER	93.6	**99.8**
FCGF+ RANSAC	**94.3**	99.6

gested by Choy *et al*. [8], and estimating transformations with TEASER[43] and RANSAC. As reported in Table 4, our implementation reaches state-of-the-art performance of point cloud registration on two datasets available in TorchPoints3D: 3DMatch [44] and KITTI Odometry [15]. In Figure 5, we present some qualitative illustrations.

4.5. CPU-Based Preprocessing

When training point-based neural networks on large point clouds, the computation of neighbors and the subsampling operations become the computational bottlenecks rather than inference or backpropagation.

In modern deep learning frameworks such as PyTorch, background processes prepare new batches of data to be run through the network while the GPU simultaneously performs tensor operations on previously prepared batches. As suggested in [38], we off-load the radius search and subsampling operations to those background processes operating on CPUs. As reported in Table 5, this allows us to achieve an 8-times overall speed-up compared to performing all computations on the GPU. This particular implementation trick—one of many—exemplifies the numerous pitfalls to overcome when implementing deep learning methods operating on 3D data.

Table 5: Training speed of the KPconv model, in thousands of points processed per second (kpts/s) during training, with radius search performed on either the GPU (Tesla T4) or the CPUs (4×2.2GHz).

	S3DIS	ScanNet
radius search on CPUs	199.9	197.4
radius search on GPU	38.6	29.2

5. Conclusion and Perspectives

We presented Torch-Points3D, a flexible and powerful framework aiming to make deep learning on 3D data both more accessible and rigorous. Our implementation allows users to evaluate, improve and combine state-of-the-art models on a growing number of tasks and datasets. The community emerging around our framework provides us with precious feedback, as well as much needed help in keeping up with a such a fast-paced domain. We welcome researchers, software engineers, and open-source enthusiasts in this endeavor.

Encouraged by the recent results of Xie *et al*. [40], we believe in the potential of transfer learning across datasets and tasks for 3D data. Our next focus will be to provide a high-level API for pre-trained, self-supervised, self-trained, and unsupervised deep learning approaches operating on 3D point clouds.

6. Acknowledgments

Thomas Chaton and Nicolas Chaulet would like to acknowledge the great support provided by Fujitsu Laboratories of Europe, both financially and for useful feedback by being one of the first users.
This research was also supported in parts by the AI4GEO project: http://www.ai4geo.eu/

References

[1] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3D semantic parsing of large-scale indoor spaces. *CVPR*, 2016. 2, 5, 6, 7

[2] L. Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com. 2, 5

[3] A. Boulch, B. L. Saux, and N. Audebert. Unstructured point cloud semantic labeling using deep segmentation networks. *Eurographics Workshop on 3D Object Retrieval*, 2017. 2

[4] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A multimodal dataset for autonomous driving. *CVPR*, 2020. 2

[5] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: an information-rich 3D model repository. Technical report, Stanford University ,Princeton University, Toyota Technological Institute at Chicago, 2015. 3, 5, 7

[6] C. Choy, W. Dong, and V. Koltun. Deep global registration. *CVPR*, 2020. 11

[7] C. Choy, J. Gwak, and S. Savarese. 4D spatio-temporal convnets: Minkowski convolutional neural networks. *CVPR*, 2019. 4, 6, 7

[8] C. Choy, J. Park, and V. Koltun. Fully convolutional geometric features. *ICCV*, 2019. 5, 8, 11

[9] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In J. Fujii, editor, *SIGGRAPH*, pages 303–312. ACM, 1996. 11

[10] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. *CVPR*, 2017. 5, 6

[11] A. Dai and M. Nießner. 3DMV: Joint 3D-multi-view prediction for 3D semantic scene segmentation. *ECCV*, 2018. 4

[12] J. Deschaud. IMLS-SLAM: scan-to-model matching based on 3D data. *CoRR*, abs/1802.08633, 2018. 11

[13] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 2

[14] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 2

[15] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. *CVPR*, 2012. 5, 8

[16] G. Gkioxari, J. Malik, and J. Johnson. Mesh R-CNN. *ICCV*, 2019. 2

[17] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3D point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 2

[18] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham. RandLA-Net: Efficient semantic segmentation of large-scale point clouds. *CVPR*, 2020. 3

[19] K. J., E. Smith, J.-F. Lafleche, C. Fuji Tsang, A. Rozantsev, W. Chen, T. Xiang, R. Lebaredian, and S. Fidler. Kaolin: a Pytorch library for accelerating 3D deep learning research. *arXiv:1911.05063*, 2019. 2

[20] L. Jiang, H. Zhao, S. Shi, S. Liu, C.-W. Fu, and J. Jia. PointGroup: Dual-set point grouping for 3D instance segmentation. *CVPR*, 2020. 4

[21] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2014. 6

[22] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár. Panoptic segmentation. *CVPR*, 2019. 4

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *NeurIPS*, 2012. 1

[24] L. Landrieu and M. Boussaha. Point cloud oversegmentation with graph-structured deep metric learning. *CVPR*, 2019. 4

[25] L. Landrieu and M. Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. *CVPR*, 2018. 2, 4

[26] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *CVPR*, 2019. 2

[27] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. PointCNN: Convolution on $\mathcal{X}$-transformed points. *NeurIPS*, 2018. 7

[28] D. Lin, S. Fidler, and R. Urtasun. Holistic scene understanding for 3D object detection with RGBD cameras. *ICCV*, 2013. 4

[29] Y. Liu, B. Fan, S. Xiang, and C. Pan. Relation-shape convolutional neural network for point cloud analysis. *CVPR*, 2019. 4, 6, 7

[30] Z. Liu, H. Hu, Y. Cao, Z. Zhang, and X. Tong. A closer look at local aggregation operators in point cloud analysis. *ECCV*, 2020. 7

[31] S. Marcel and Y. Rodriguez. Torchvision the machine-vision package of Torch. *ACM international conference on Multimedia*, 2010. 2

[32] C. R. Qi, O. Litany, K. He, and L. J. Guibas. Deep Hough voting for 3D object detection in point clouds. *ICCV*, 2019. 4, 6

[33] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *CVPR*, 2017. 2, 3

[34] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *NeurIPS*, 2017. 2, 3, 4, 5, 6, 7

[35] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Pytorch3D, 2020. 2

[36] G. Riegler, A. O. Ulusoy, and A. Geiger. OctNet: Learning deep 3D representations at high resolutions. *CVPR*, 2017. 2

[37] L. P. Tchapmi, C. B. Choy, I. Armeni, J. Gwak, and S. Savarese. SEGCloud: Semantic segmentation of 3D point clouds. *ICCV*, 2017. 2

[38] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. KPconv: Flexible and deformable convolution for point clouds. *CVPR*, 2019. 2, 3, 4, 5, 6, 7, 8

[39] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. *CVPR*, 2015. 5

[40] S. Xie, J. Gu, D. Guo, C. R. Qi, L. J. Guibas, and O. Litany. PointContrast: Unsupervised pre-training for 3D point cloud understanding. *ECCV*, 2020. 6, 8

[41] O. Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. 2, 5

[42] B. Yang, W. Luo, and R. Urtasun. PIXOR: Real-time 3D object detection from point clouds. *CVPR*, 2018. 2

[43] H. Yang, J. Shi, and L. Carlone. TEASER: Fast and certifiable point cloud registration, 2020. 7, 8

[44] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3DMatch: Learning local geometric descriptors from RGB-D reconstructions. *CVPR*, 2017. 5, 8

[45] W. Zhang, Y. Wu, T. Wang, Y. Li, K.-Y. Lin, Z. Wang, J. Shi, C. Qian, K. Chen, D. Lin, and C. C. Loy. MMDetection3D. https://github.com/open-mmlab/mmdetection3d, 2020. 2

[46] Q.-Y. Zhou, J. Park, and V. Koltun. Fast Global Registration. *ECCV*, 2016. 7

[47] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu. Class-balanced grouping and sampling for point cloud 3D object detection. *arXiv preprint arXiv:1908.09492*, 2019. 2

APPENDIX

A. Illustrative Notebooks

We propose two IPython notebooks to illustrate some of the capacities of Torch-Points3D:

- A notebook illustrating the training of the Relation-Shape CNN model for object classification with Torch-Points3D: link.
- A notebook showing the inner working of encoding and decoding in a KPConv model for part segmentation: link.

These notebook can be run on Google Colab from a browser without any installations. They are self-contained and will install automatically all required packages, as well as download the relevant datasets. Be warned that the installation of the necessary libraries and download of the datasets can take up to 30minutes. You can otherwise download the notebooks and run them locally, after installing the necessary libraries, namely torch torch-points3d and pyvista.

```
pip install torch torch-points3d pyvista
```

B. Details on the Registration Experiment

B.1. Implementation

As described in the main paper, we implemented FCGF model for point cloud registration. The encoder is composed of 4 residual blocks with output sizes of $[32, 64, 128, 256]$. Each block has a stride of 2 except for the first block. The decoder is composed of residual blocks of output sizes $[64, 64, 64, 64]$. We train with SGD with momentum of 0.8, a learning rate of 0.1, and a weight decay of 10^{-4}. The same parameters are used for both Kitti Odometry and 3DMatch.

B.2. 3Dmatch

Since 3D match is a dataset of RGBD images, we need to fuse depth images to obtain 3D point clouds. We use a TSDF voxel grid as in [9] and obtain fragments. For the training set and the validation set, 50 depth images are fused to obtain each fragment. We use a voxel subsampling of size 0.02m in the FCGF network.

B.3. KITTI Odometry

Kitti Odometry contains LiDAR scans with their associated poses, however these do not have the precision necessary to properly evaluate registration predictions. Choy *et al.* [8] use these poses as initialization, and use ICP to refine the transformation between each pairs. The pairs are defined as LiDAR frames whose center is at least distant of 10m. We use IMLS SLAM [12] to compute the poses of all sequences. As in [8, 6], the sequences $0, 1, 2, 3, 4, 5$ are used for training, $6, 7$ for validation and $8, 9, 10$ for testing. For this dataset, we use a voxel subsampling of size 0.3m.

B.4. Evaluation

To evaluate registration, we measure the rotation error and the translation error as :

$$\delta_{trans} = ||t_{est} - t^\star||_2 \tag{1}$$

$$\delta_{rot} = \arccos \frac{trace(R_{est} R^{\star\, T}) - 1}{2} \tag{2}$$

where (t_{est}, R_{est}) is the estimated translation and rotation, and $(t^\star, R^\star)$ the associated ground truth. For 3DMatch, we count a *success* when the rotation error is under 15 degrees, and the translation error is less than 0.3 m, as suggested by Choy *et al.* [6] For Kitti Odometry, the rotation error must be less than 2 degrees and the translation error must be under 0.6 m for a prediction to be considered successful.

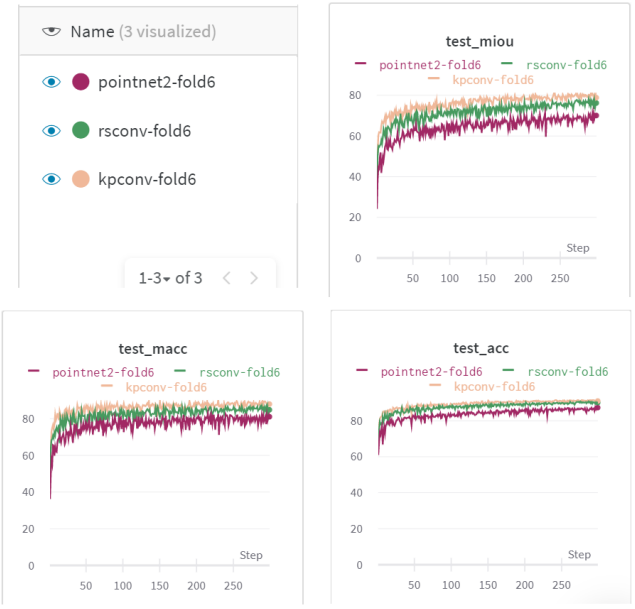


Figure 6: Training logs hosted on wandb.ai for different models on a S3DIS fold. These logs, along their trained models and full configuration, are publicly available on the contributor's wandb.ai's account (links provided on the repository).