Point Cloud City on Open3D ML

Installation | **Get started** | **Structure** | **Tasks & Algorithms** | **Model Zoo** | **Datasets** | **How-tos** | **Contribute**

Open3D-ML is an extension of Open3D for 3D machine learning tasks. It builds on top of the Open3D core library and extends it with machine learning tools for 3D data processing. This repo focuses on applications such as semantic point cloud segmentation and provides pretrained models that can be applied to common tasks as well as pipelines for training.

Point Cloud City was developed during the 2018 NIST Public Safety Innovation Accelerator Program - Point Cloud City NOFO awardees generated an extensive catalog of annotated 3D indoor point clouds that can be used by industry, academia, and government to advance research and development in the areas of indoor mapping, localization and navigation for public safety, as well as to demonstrate the potential value of ubiquitous indoor positioning and location-based information. These pioneering U.S. state and local governments will create a model 'Point Cloud City' and also participate in the NIST Global Cities Team Challenge initiative as the lead for an Action Cluster. This repository extends Open3D-ML to be imlemented using the Point Cloud City datasets and features the processing code, dataset integration, and machine learning model configuration files.

Open3D-ML works with **TensorFlow** and **PyTorch** to integrate easily into existing projects and also provides general functionality independent of ML frameworks such as data visualization.

Installation

Users

Open3D-ML is integrated in the Open3D v0.11+ python distribution and is compatible with the following versions of ML frameworks.

- PyTorch 1.8.2
- TensorFlow 2.5.2
- CUDA 10.1, 11.* (On `GNU/Linux x86_64`, optional)

You can install Open3D with

```
# make sure you have the latest pip version
pip install --upgrade pip
```

```
# install open3d
pip install open3d
```

To install a compatible version of PyTorch or TensorFlow you can use the respective requirements files:

```
# To install a compatible version of TensorFlow
pip install -r requirements-tensorflow.txt
# To install a compatible version of PyTorch
pip install -r requirements-torch.txt
# To install a compatible version of PyTorch with CUDA on Linux
pip install -r requirements-torch-cuda.txt
```

To test the installation use

```
# with PyTorch
$ python -c "import open3d.ml.torch as ml3d"
# or with TensorFlow
$ python -c "import open3d.ml.tf as ml3d"
```

If you need to use different versions of the ML frameworks or CUDA we recommend to build Open3D from source.

Getting started with Point Cloud City and OPEN3D-ML

Set Source Repository

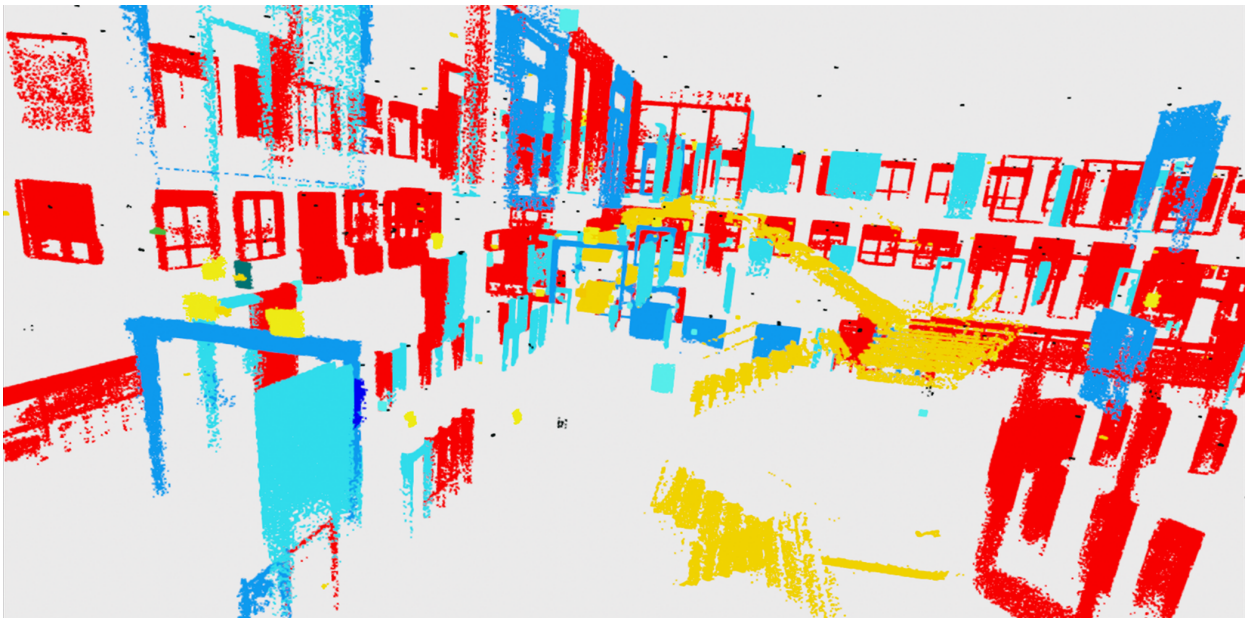
```
# in /Open3D-ML_Point_Cloud_City/
$ source set_open3d_ml_root.sh
```

Reading a dataset

The dataset namespace contains classes for reading common datasets. Here we read the SemanticKITTI dataset and visualize it.

```
import open3d.ml.torch as ml3d  # or open3d.ml.tf as ml3d

# construct a dataset by specifying dataset_path
dataset = ml3d.datasets.SemanticKITTI_PCC(dataset_path='/path/to/PCC_SKITTI/')

# get the 'all' split that combines training, validation and test set
all_split = dataset.get_split('all')

# print the attributes of the first datum
print(all_split.get_attr(0))
```

```
# print the shape of the first point cloud
print(all_split.get_data(0)['point'].shape)

# show the first 100 frames using the visualizer
vis = ml3d.vis.Visualizer()
vis.visualize_dataset(dataset, 'all', indices=range(100))
```



Loading a config file

Configs of models, datasets, and pipelines are stored in `ml3d/configs`. Users can also construct their own yaml files to keep record of their customized configurations. Here is an example of reading a config file and constructing modules from it.

```
import open3d.ml as _ml3d
import open3d.ml.torch as ml3d # or open3d.ml.tf as ml3d

framework = "torch" # or tf
cfg_file = "ml3d/configs/randlanet_semantickitti_pcc.yml"
cfg = _ml3d.utils.Config.load_from_file(cfg_file)

# fetch the classes by the name
Pipeline = _ml3d.utils.get_module("pipeline", cfg.pipeline.name, framework)
Model = _ml3d.utils.get_module("model", cfg.model.name, framework)
Dataset = _ml3d.utils.get_module("dataset", cfg.dataset.name)

# use the arguments in the config file to construct the instances
cfg.dataset['dataset_path'] = "/path/to/your/dataset"
dataset = Dataset(cfg.dataset.pop('dataset_path', None), **cfg.dataset)
model = Model(**cfg.model)
pipeline = Pipeline(model, dataset, **cfg.pipeline)
```

Semantic Segmentation

Running a pretrained model for semantic segmentation

Building on the previous example we can instantiate a pipeline with a pretrained model for semantic segmentation and run it on a point cloud of our dataset. See the model zoo for obtaining the weights of the pretrained model.

```
import os
import open3d.ml as _ml3d
import open3d.ml.torch as ml3d

cfg_file = "ml3d/configs/randlanet_semantickitti_pcc.yml"
cfg = _ml3d.utils.Config.load_from_file(cfg_file)

model = ml3d.models.RandLANet(**cfg.model)
cfg.dataset['dataset_path'] = "/path/to/your/dataset"
dataset = ml3d.datasets.SemanticKITTI(cfg.dataset.pop('dataset_path', None),
**cfg.dataset)
pipeline = ml3d.pipelines.SemanticSegmentation(model, dataset=dataset,
device="gpu", **cfg.pipeline)

# download the weights.
ckpt_folder = "./logs/"
os.makedirs(ckpt_folder, exist_ok=True)
ckpt_path = ckpt_folder + "randlanet_semantickitti_pcc_202301011330utc.pth"

# load the parameters.
pipeline.load_ckpt(ckpt_path=ckpt_path)

test_split = dataset.get_split("test")
data = test_split.get_data(0)

# run inference on a single example.
# returns dict with 'predict_labels' and 'predict_scores'.
result = pipeline.run_inference(data)

# evaluate performance on the test set; this will write logs to './logs'.
pipeline.run_test()
```

Users can also use predefined scripts to load pretrained weights and run testing.

Training a model for semantic segmentation

Similar as for inference, pipelines provide an interface for training a model on a dataset.

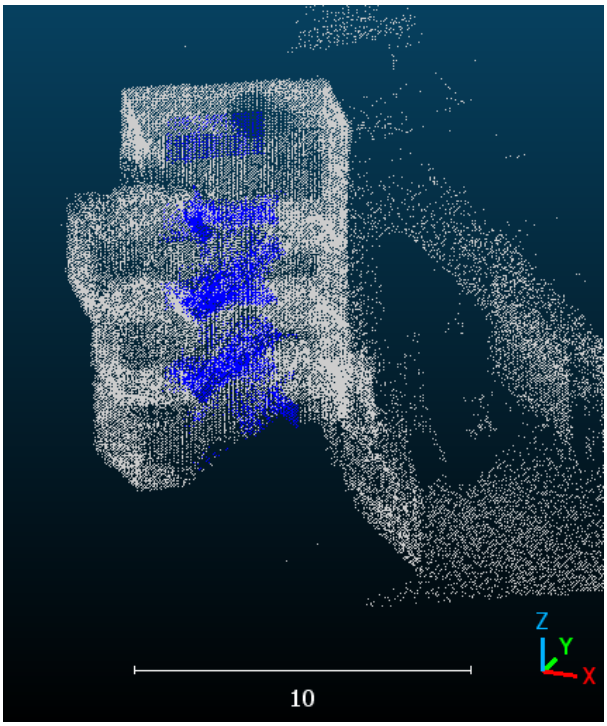
```
# use a cache for storing the results of the preprocessing (default path is
'./logs/cache')
dataset =
ml3d.datasets.SemanticKITTI_PCC(dataset_path='/path/to/SemanticKITTI_PCC/',
use_cache=True)
```

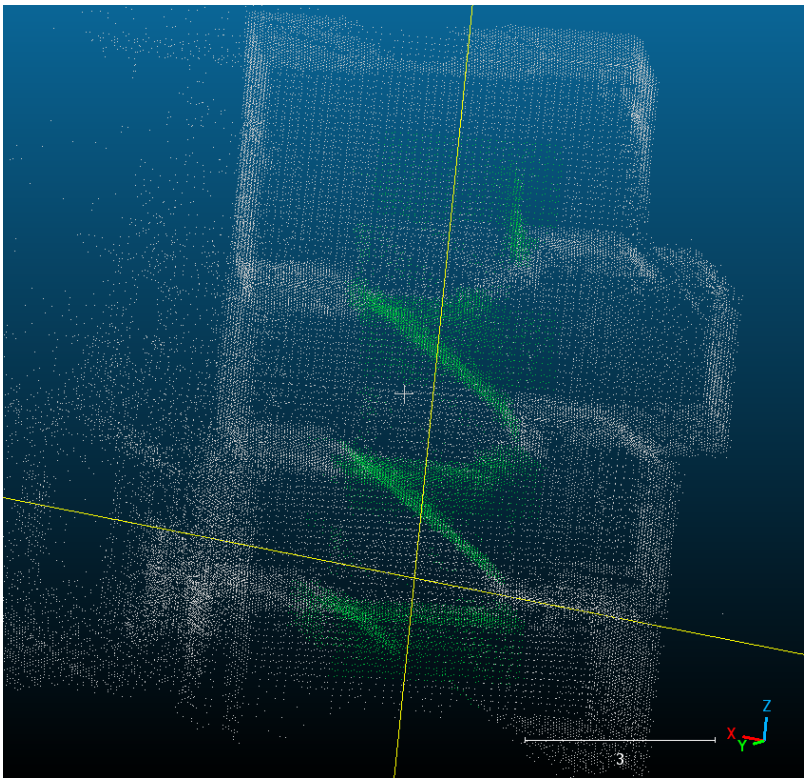
```
# create the model with random initialization.
model = RandLANet()

pipeline = SemanticSegmentation(model=model, dataset=dataset, max_epoch=100)

# prints training progress in the console.
pipeline.run_train()
```

For more examples see `examples/` and the `scripts/` directories. You can also enable saving training summaries in the config file and visualize ground truth and results with tensorboard. See this tutorial for details.

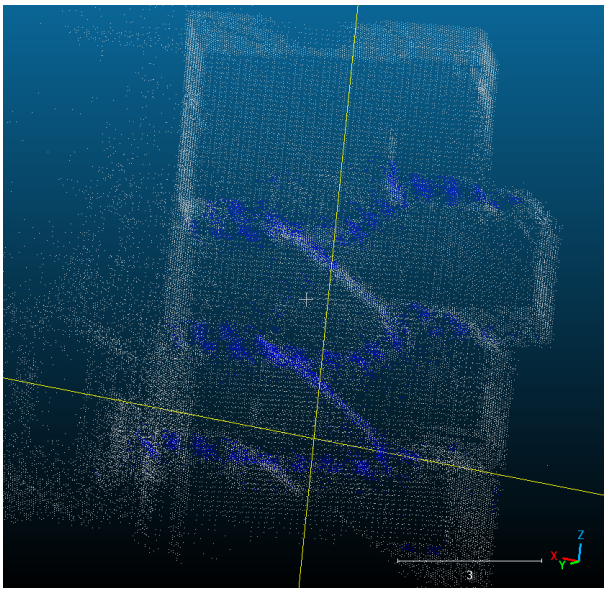
Results

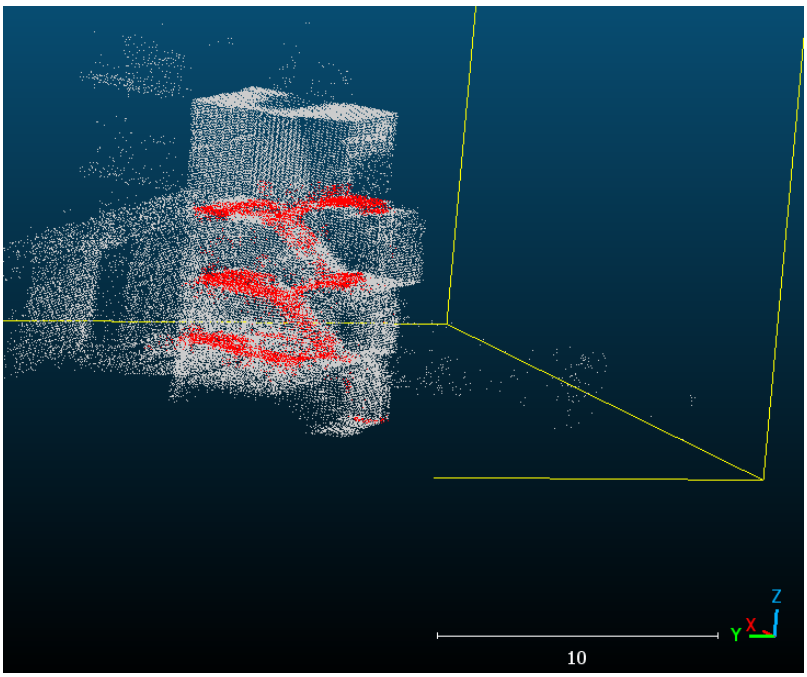
Ground Truth - Point Cloud City





KPCONV Semantic Segmentation Results





Using predefined scripts

`scripts/run_pipeline.py` provides an easy interface for training and evaluating a model on a dataset. It saves the trouble of defining specific model and passing exact configuration.

```
python scripts/run_pipeline.py {tf/torch} -c <path-to-config> --pipeline
{SemanticSegmentation/ObjectDetection} --<extra args>
```

You can use script for both semantic segmentation and object detection. You must specify either SemanticSegmentation or ObjectDetection in the `pipeline` parameter. Note that `extra args` will be prioritized over the same parameter present in the configuration file. So instead of changing param in config file, you may pass the same as a command line argument while launching the script.

For eg.

```
# Launch training for RandLANet on SemanticKITTI_PCC with torch.
python scripts/run_pipeline.py torch -c
ml3d/configs/randlanet_semantickitti_pcc.yml --dataset.dataset_path <path-to-
```

```
dataset> --pipeline SemanticSegmentation --dataset.use_cache True

# Launch testing for KPConv on SemanticKITTI_PCC with torch.
python scripts/run_pipeline.py torch -c ml3d/configs/kpconv_semantickitti_pcc.yml
--split test --dataset.dataset_path <path-to-dataset> --pipeline
SemanticSegmentation --dataset.use_cache True
```

For further help, run `python scripts/run_pipeline.py --help`.

Repository structure

The core part of Open3D-ML lives in the `ml3d` subfolder, which is integrated into Open3D in the `ml` namespace. In addition to the core part, the directories `examples` and `scripts` provide supporting scripts for getting started with setting up a training pipeline or running a network on a dataset.

```
├─ pscr_point_cloud_city  # Documentation and processing code for PCC O3D-ML
     ├─ docs              # Directory for dataset details
     ├─ scripts           # Dataset processing code
├─ docs                   # Markdown and rst files for documentation
├─ examples               # Place for example scripts and notebooks
├─ ml3d                   # Package root dir that is integrated in open3d
     ├─ configs           # Model configuration files
     ├─ datasets          # Generic dataset code; will be integratede as
open3d.ml.{tf,torch}.datasets
     ├─ metrics           # Metrics available for evaluating ML models
     ├─ utils             # Framework independent utilities; available as
open3d.ml.{tf,torch}.utils
     ├─ vis               # ML specific visualization functions
     ├─ tf                # Directory for TensorFlow specific code. same structure
as ml3d/torch.
     │                    # This will be available as open3d.ml.tf
     ├─ torch             # Directory for PyTorch specific code; available as
open3d.ml.torch
          ├─ dataloaders  # Framework specific dataset code, e.g. wrappers that
can make use of the
          │               # generic dataset code.
          ├─ models       # Code for models
          ├─ modules      # Smaller modules, e.g., metrics and losses
          ├─ pipelines    # Pipelines for tasks like semantic segmentation
          ├─ utils        # Utilities for <>
├─ scripts                # Demo scripts for training and dataset download scripts
```

Tasks and Algorithms

Semantic Segmentation

For the task of semantic segmentation, we measure the performance of different methods using the mean intersection-over-union (mIoU) over all classes. The table shows the available models and datasets for the

segmentation task and the respective scores. Each score links to the respective weight file.

This table displays results using the model KPCONV and was run using PyTorch.

Dataset	Metric	Stairway	Windows	Roof Access	Fire Sprinkler	Gas Shutoff
Enfield	mIoU	37.8	50.1	23.7	20.1	68.6
	F1	54.9	65.1	66.8	34.1	81.4
Memphis	mIoU	6.7	97.6	0.00	0.05	0.00
	F1	13.1	98.87	0.00	0.10	0.00
PCC_SKITTI	mIoU	28.9	92.7	42.8	77.8	76.7
	F1	28.9	92.7	42.8	77.8	76.7

(*) Using weights from Point Cloud City dataset. PCC_SKITTI is the combination of the Enfield and Memphis datasets with unified labels and formatted to replicate Semantic KITTI's structure.

Model Zoo

For a full list of all weight files see model_weights.txt and the MD5 checksum file model_weights.md5.

Datasets

The following is a list of datasets for which we provide dataset reader classes.

- Point Cloud City (project page)
- SemanticKITTI (project page)
- Enfield (project page)
- Memphis (project page)
- Hancock (project page)

For downloading these datasets visit the respective webpages and have a look at the scripts in `scripts/download_datasets`.

How-tos

- Process Point Cloud City - SemanticKITTI Format
- Visualize network predictions
- Visualize custom data
- Adding a new model
- Adding a new dataset
- Distributed training
- Visualize and compare input data, ground truth and results in TensorBoard
- Inference with Intel OpenVINO

Contribute

This repository is no longer maintained by NIST. The code under /pscr_point_cloud_city/scripts can be used to process the Point Cloud City dataset for machine learning tasks.

There are many ways to contribute to the original project through:

- https://github.com/isl-org/Open3D-ML/

Communication channels

- Forum: discussion on the usage of Open3D.
- Discord Chat: online chats, discussions, and collaboration with other users and developers.

Citation

Please cite our work (pdf) if you use Open3D.

```
@article{Zhou2018,
    author    = {Qian-Yi Zhou and Jaesik Park and Vladlen Koltun},
    title     = {{Open3D}: {A} Modern Library for {3D} Data Processing},
    journal   = {arXiv:1801.09847},
    year      = {2018},
}
```