


# CS5182 Computer Graphics Coding Essentials for Point Clouds Tutorial



2024/25 Semester A

City University of Hong Kong (DG)

# Outline

---

## □ Prerequisites.

- Env (Ubuntu/Win10)
- Nvidia GPU
- Python in Anaconda
- Version relationship

## □ PyTorch

- Custom C++ and CUDA extensions
- Example

## □ Auxiliary libraries

- Open3D
- PyTorch Geometric
- Trimesh

# Prerequisites

---

- ❑ Choose a stable and popular system
  - For Linux, it is recommended to use Ubuntu16.04/Ubuntu18.04/Ubuntu20.04. Notice that the latest is not recommended right now because there are some incompatibilities. Choosing the right version will fix most of the quirky bugs.
  - For Windows user, use windows 10/11 is ok for most of the current point cloud repositories. However, for further usage of multi-GPU environment, it is highly recommended to get familiar with linux command line as soon as possible.



# Prerequisites

---

- Getting access to right version for your toolkits (intro)
  - a parallel computing platform and application programming interface model created by Nvidia.
  - a free and open-source distribution of the programming languages Python and R, aiming to simplify package management and deployment.
  - an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab.



# Prerequisites

- Getting access to right version for your toolkits (make sure you have nvidia gpu and right driver)
  - <https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html>

\*\* CUDA 11.0 was released with an earlier driver version, but by upgrading to Tesla Recommended Drivers 450.80.02 (Linux) / 452.39 (Windows), minor version compatibility is possible across the CUDA 11.x family of toolkits.

The version of the development NVIDIA GPU Driver packaged in each CUDA Toolkit release is shown below.

*Table 3: CUDA Toolkit and Corresponding Driver Versions*

CUDA Toolkit	Toolkit Driver Version	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 12.6 Update 1	>=560.35.03	>=560.94
CUDA 12.6 GA	>=560.28.03	>=560.76
CUDA 12.5 Update 1	>=555.42.06	>=555.85
CUDA 12.5 GA	>=555.42.02	>=555.85

# Prerequisites

## □ Getting access to right version for your toolkits (install guide)

- A newer blog for installing cuda on windows/linux

- [https://blog.csdn.net/qq\\_51375047/article/details/140957904](https://blog.csdn.net/qq_51375047/article/details/140957904)

- Official documentation for installing anaconda and init on linux



- <https://docs.anaconda.com/anaconda/install/linux/>

```
conda create -n your_env_name python=3.8  
conda activate your_env_name
```

- PyTorch installation under conda env



- <https://pytorch.org/get-started/previous-versions/>

```
# CUDA 11.8  
conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 pytorch-cuda=11.8 -c pytorch -c nvidia  
# CUDA 12.1  
conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 pytorch-cuda=12.1 -c pytorch -c nvidia  
# CUDA 12.4  
conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 pytorch-cuda=12.4 -c pytorch -c nvidia  
# CPU Only  
conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 cpuonly -c pytorch
```

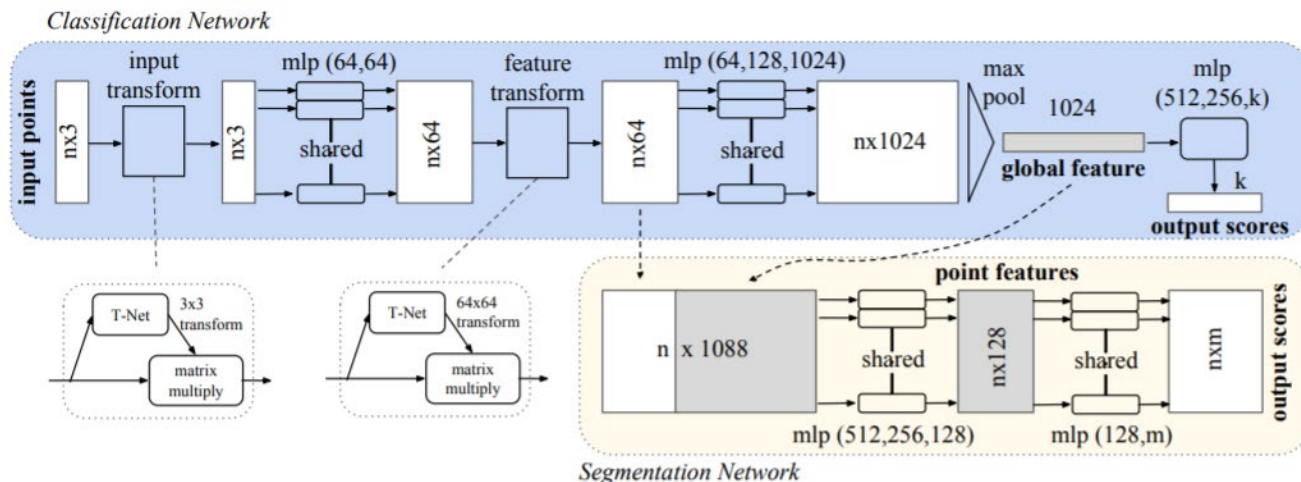
# Example: 3D Classification PointNet

## Original paper:

- [CVPR2017, Qi et al.] PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

## Architecture

- Input dim:  $N \times 3$  (N points. Each point has 3 values x, y and z)
- Output dim:
  - For classification: k scores for all the k candidate classes
  - For segmentation:  $n \times m$  scores for each of the n points and each of the m semantic subcategories.



# Example: 3D Classification PointNet

---

- ❑ Original github (tensorflow version):
  - <https://github.com/charlesq34/pointnet>
- ❑ Pytorch implementation by fxia22
  - <https://github.com/fxia22/pointnet.pytorch>
- ❑ Ask these questions before reading the github code:
  - What datasets are used, where are they downloaded, and how are the data organized?
    - ❑ ModelNet40 / ShapeNet, public website, the dataset class of pytorch in dataset.py
  - What kind of tasks, what metrics are used to evaluate?
    - ❑ Classification / segmentation, Acc / mIOU (Mean Intersection over Union)
  - How is the model organized?
    - ❑ The nn.Module class of pytorch in model.py



# Example: 3D Classification PointNet

- ❑ Original github (tensorflow version):
  - <https://github.com/charlesq34/pointnet>
- ❑ Pytorch implementation by fxia22
  - <https://github.com/fxia22/pointnet.pytorch>
- ❑ Ask these questions before reading the github code:
  - How to write a command line (cli) to run the entire training process?(i.e. find the main)
    - ❑ Here is train\_xx.py, some works use main.py

## Training

```
cd utils
python train_classification.py --dataset <dataset path> --nepoch=<number epochs> --dataset_type <modelnet40 |
python train_segmentation.py --dataset <dataset path> --nepoch=<number epochs>
```

# Example: 3D Classification PointNet

---

- ❑ Original github (tensorflow version):
  - <https://github.com/charlesq34/pointnet>
- ❑ Pytorch implementation by fxia22
  - <https://github.com/fxia22/pointnet.pytorch>
- ❑ Ask these questions before reading the github code:
  - Is there any extension code that needs to be compiled first? (usually \*.cpp/\*.cu)
    - ❑ Sometimes, not all codes are written in pytorch. In the area of 3D point cloud, Due to the increase of data dimensions, the difficulty of preprocessing and visualization is generally higher than that of two-dimensional images. PyTorch therefore allows users to write their own C ++ extensions that can be compiled and integrated into the PyTorch main code.
    - ❑ E.g. the visualization tool

```
cd script
bash build.sh #build C++ code for visualization
bash download.sh #download dataset
```

# Walk through: PyTorch Code

- The training process
- The dataset
- The model


```
.
├── LICENSE
├── misc
│   ├── modelnet_id.txt #'modelnet' dataset index
│   ├── num_seg_classes.txt #nums of seg classes
│   └── show3d.png #visualization
├── pointnet
│   ├── dataset.py #dataset process code module
│   ├── __init__.py #kind of python conventions,make this directory discoverable
│   └── model.py #model definition code module
├── README.md
├── scripts
│   ├── build.sh
│   └── download.sh
├── setup.py #Compile the basic operations
├── utils #the code of some basic operations
│   ├── render_balls_so.cpp
│   ├── show3d_balls.py
│   ├── show_cls.py
│   ├── show_seg.py
│   ├── train_classification.py
│   └── train_segmentation.py
```


# Walk through: PyTorch Code


## □ The dataset


- Take ModelNet dataset used in PointNet as an example.
- The structure of the dataset building class is shown in the figure.
- Our data source is the original coordinate information xyz information of the point cloud, which is often saved with these formats, .ply, .pcd, .obj or .xyz. And you can use the software meshlab to visualize .ply, .obj or .xyz. Or use the library named Open3D to visualize .pcd.
- .ply and .obj can also contain more information like rgb, normal and connections, which make them a 'mesh'.

```
143 class ModelNetDataset(data.Dataset):
144 >     def __init__(self, ...
166
167 >     def __getitem__(self, index): ...
189
190
191 >     def __len__(self): ...
192
```

 A9-vulcan\_aligned.xyz

 refined\_reconstruction\_0180.ply

 merged\_0002.obj

 pointcloud0.pcd

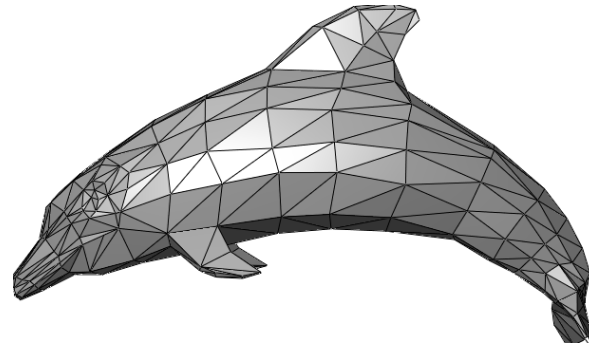
# Walk through: PyTorch Code

---

## □ Example data structure



Point cloud with color rendering for segmentation



Polygon mesh data

# Walk through: PyTorch Code

## □ The dataset

```
143 class ModelNetDataset(data.Dataset):
144     def __init__(self,
145                 root,
146                 npoints=2500,
147                 split='train',
148                 data_augmentation=True):
149         self.npoints = npoints
150         self.root = root
151         self.split = split
152         self.data_augmentation = data_augmentation
153         self.fns = []
154         with open(os.path.join(root, '{}.txt'.format(self.split)), 'r') as f:
155             for line in f:
156                 self.fns.append(line.strip())
157
158         self.cat = {}
159         with open(os.path.join(os.path.dirname(os.path.realpath(__file__)), '../misc/modelnet_id.txt'), 'r') as f:
160             for line in f:
161                 ls = line.strip().split()
162                 self.cat[ls[0]] = int(ls[1])
163
164         print(self.cat)
165         self.classes = list(self.cat.keys())
```

# Walk through: PyTorch Code

## □ The dataset

```
167 def __getitem__(self, index):
168     fn = self.fns[index]
169     cls = self.cat[fn.split('/')[0]]
170     with open(os.path.join(self.root, fn), 'rb') as f:
171         plydata = PlyData.read(f)
172         pts = np.vstack([plydata['vertex']['x'], plydata['vertex']['y'], plydata['vertex']['z']]).T
173         choice = np.random.choice(len(pts), self.npoints, replace=True)
174         point_set = pts[choice, :]
175
176         point_set = point_set - np.expand_dims(np.mean(point_set, axis=0), 0) # center
177         dist = np.max(np.sqrt(np.sum(point_set ** 2, axis=1)), 0)
178         point_set = point_set / dist # scale
179
180         if self.data_augmentation:
181             theta = np.random.uniform(0, np.pi * 2)
182             rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
183             point_set[:, [0, 2]] = point_set[:, [0, 2]].dot(rotation_matrix) # random rotation
184             point_set += np.random.normal(0, 0.02, size=point_set.shape) # random jitter
185
186         point_set = torch.from_numpy(point_set.astype(np.float32))
187         cls = torch.from_numpy(np.array([cls]).astype(np.int64))
188         return point_set, cls
189
190
191 def __len__(self):
192     return len(self.fns)
```

# Walk through: PyTorch Code

## □ The model

```
129 class PointNetCls(nn.Module):
130     def __init__(self, k=2, feature_transform=False):
131         super(PointNetCls, self).__init__()
132         self.feature_transform = feature_transform
133         self.feats = PointNetfeat(global_feat=True, feature_transform=feature_transform)
134         self.fc1 = nn.Linear(1024, 512)
135         self.fc2 = nn.Linear(512, 256)
136         self.fc3 = nn.Linear(256, k)
137         self.dropout = nn.Dropout(p=0.3)
138         self.bn1 = nn.BatchNorm1d(512)
139         self.bn2 = nn.BatchNorm1d(256)
140         self.relu = nn.ReLU()
141
142     def forward(self, x):
143         x, trans, trans_feats = self.feats(x)
144         x = F.relu(self.bn1(self.fc1(x)))
145         x = F.relu(self.bn2(self.dropout(self.fc2(x))))
146         x = self.fc3(x)
147         return F.log_softmax(x, dim=1), trans, trans_feats
```

Help links: [https://pytorch.org/tutorials/beginner/examples\\_nn/two\\_layer\\_net\\_module.html](https://pytorch.org/tutorials/beginner/examples_nn/two_layer_net_module.html)



# Walk through: PyTorch Code

## □ The training process

- Module importing, user parameters parsing and random setting

```
1 from __future__ import print_function
2 import argparse
3 import os
4 import random
5 import torch
6 import torch.nn.parallel
7 import torch.optim as optim
8 import torch.utils.data
9 from pointnet.dataset import ShapeNetDataset, ModelNetDataset
10 from pointnet.model import PointNetCls, feature_transform_regularizer
11 import torch.nn.functional as F
12 from tqdm import tqdm
```

About the version of python 2.x or 3.x, you can ignore  
Parameter parser, which accepts user input parameters  
For some system operation like making or changing directories  
Generate random numbers. Tdata set needs to be randomly shuffled  
The whole pytorch package  
A package for multi gpu training  
The optimizer package for deep learning in pytorch  
The father class of pytorch dataloader, making dataset iterable  
function libraries for neural network training  
Progress bar visualization

```
15 parser = argparse.ArgumentParser()
16 parser.add_argument(
17     '--batchSize', type=int, default=32, help='input batch size')
```

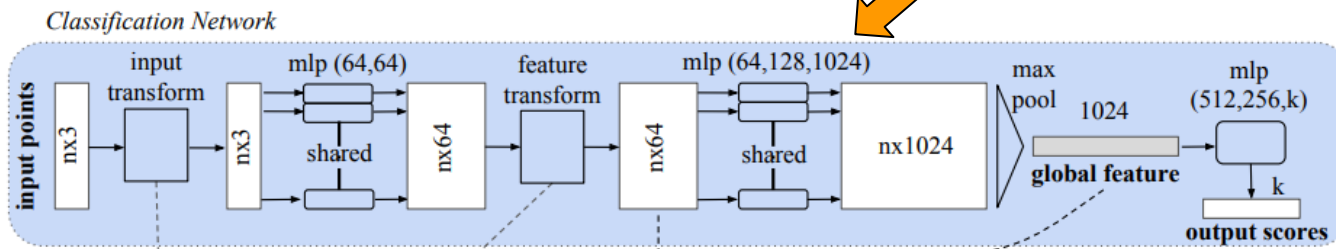
```
35 opt.manualSeed = random.randint(1, 10000)
36 print("Random Seed: ", opt.manualSeed)
37 random.seed(opt.manualSeed)
38 torch.manual_seed(opt.manualSeed)
```

```
56 > class ShapeNetDataset(data.Dataset): ...
142
143 > class ModelNetDataset(data.Dataset): ...
```

```
> class PointNetCls(nn.Module): ...
```

```
177 > def feature_transform_regularizer(trans): ...
```

Corresponding to the module in the Paper (PointNet)



# Walk through: PyTorch Code

- The training process
  - 2. Dataloader

```
40 if opt.dataset_type == 'shapenet':
41     dataset = ShapeNetDataset(
42         root=opt.dataset,
43         classification=True,
44         npoints=opt.num_points)
45
46     test_dataset = ShapeNetDataset(
47         root=opt.dataset,
48         classification=True,
49         split='test',
50         npoints=opt.num_points,
51         data_augmentation=False)
52 elif opt.dataset_type == 'modelnet40':
53     dataset = ModelNetDataset(
54         root=opt.dataset,
55         npoints=opt.num_points,
56         split='trainval')
57
58     test_dataset = ModelNetDataset(
59         root=opt.dataset,
60         split='test',
61         npoints=opt.num_points,
62         data_augmentation=False)
63 else:
64     exit('wrong dataset type')
```

```
67 dataloader = torch.utils.data.DataLoader(
68     dataset,
69     batch_size=opt.batchSize,
70     shuffle=True,
71     num_workers=int(opt.workers))
72
73 testdataloader = torch.utils.data.DataLoader(
74     test_dataset,
75     batch_size=opt.batchSize,
76     shuffle=True,
77     num_workers=int(opt.workers))
```

# Walk through: PyTorch Code

## □ The training process

- 3. The model building (in this code example the model a classifier)

```
88 classifier = PointNetCls(k=num_classes, feature_transform=opt.feature_transform)
89
90 if opt.model != '':
91     classifier.load_state_dict(torch.load(opt.model))
92
93
94 optimizer = optim.Adam(classifier.parameters(), lr=0.001, betas=(0.9, 0.999))
95 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.5)
96 classifier.cuda()
```

# Walk through: PyTorch Code

## □ The training process

### ■ 4. The training loop & saving the weights obtained by training

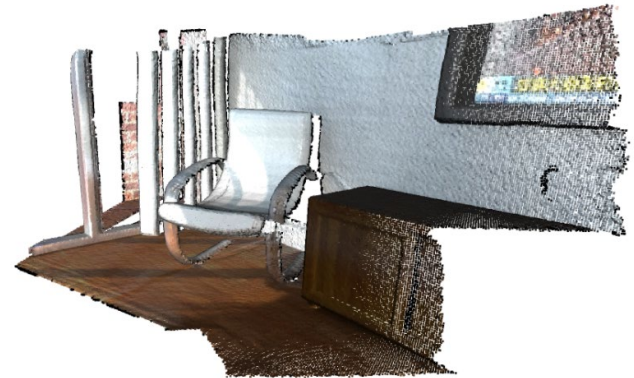
```
100 # main train loop for specific epoch
101 for epoch in range(opt.nepoch):
102     scheduler.step()
103     for i, data in enumerate(dataloader, 0):
104         points, target = data
105         target = target[:, 0]
106         points = points.transpose(2, 1)
107         points, target = points.cuda(), target.cuda()
108         optimizer.zero_grad()
109         classifier = classifier.train()
110         pred, trans, trans_feat = classifier(points)
111         loss = F.nll_loss(pred, target)
112         if opt.feature_transform:
113             loss += feature_transform_regularizer(trans_feat) * 0.001
114         loss.backward()
115         optimizer.step()
116         pred_choice = pred.data.max(1)[1]
117         correct = pred_choice.eq(target.data).cpu().sum()
118         print('%d: %d/%d train loss: %f accuracy: %f' % (epoch, i, num_batch, loss.item(), correct.item() / float(opt.batchSize)))
119
120     if i % 10 == 0:
121         j, data = next(enumerate(testdataloader, 0))
122         points, target = data
123         target = target[:, 0]
124         points = points.transpose(2, 1)
125         points, target = points.cuda(), target.cuda()
126         classifier = classifier.eval()
127         pred, _, _ = classifier(points)
128         loss = F.nll_loss(pred, target)
129         pred_choice = pred.data.max(1)[1]
130         correct = pred_choice.eq(target.data).cpu().sum()
131         print('%d: %d/%d %s loss: %f accuracy: %f' % (epoch, i, num_batch, blue('test'), loss.item(), correct.item() / float(opt.batchSize)))
132
133     torch.save(classifier.state_dict(), '%s/cls_model %d.pth' % (opt.outf, epoch))
```

# Auxiliary libraries

## □ Open3D

- an open-source library that supports rapid development of software that deals with 3D data
- based on C++ but can be used in python api

```
print("Load a ply point cloud, print it, and render it")
pcd = o3d.io.read_point_cloud("../test_data/fragment.ply")
o3d.visualization.draw_geometries([pcd],
                                   zoom=0.3412,
                                   front=[0.4257, -0.2125, -0.8795],
                                   lookat=[2.6172, 2.0475, 1.532],
                                   up=[-0.0694, -0.9768, 0.2024])
```



## □ PyTorch Geometric

- a geometric deep learning extension library for PyTorch.
- it consists of various methods for deep learning on graphs and other irregular structures, also known as geometric deep learning, from a variety of published papers.

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
```