City University of Hong Kong

Department of Computer Science

CS5182 Computer Graphics

Final Report

Student: Li Xinyan (55670594)

1. Introduction

This is my first contract with computer graphics. I studied both PointNet [1] and PointNet++ [2] papers, reproduced the PointNet's code, and compared it with PointNet++.I have tried my best to get fully understanding of deep-learning based point clouds processing. For me, the learning and implementation process is very meaningful.

2. Theoretical Support: PointNet

In is part, I will explain my understanding of PointNet from the inside principle to the network design. The understanding is based on papers and my previous knowledge.

2.1 Background Knowledge: 3D Point Clouds

The machine representation of a point in 3D space is a vector, which basically contains its coordinate (x, y, z) and sometimes other information (RGB, normal, intensity, etc.). An object in 3D space is made up of a group of points, the set of these points' vector is known as a point cloud.

3D point clouds make it possible for us to handle real object in the form of data points. Noted that point cloud is a set, so it's unordered (compare to points in 2D space), which means the permutation of points will not change the point cloud structure. 3D Point cloud is also robust to transformation, rotation and scaling (done by simple matrix multiplication).

3D point clouds can be directly observed from scanner, but in most cases they cannot be directly used in 3D application before surface reconstruction. 3D point cloud of an object is more like geometry primitive, rather than structure directly contains information between points (mesh, volumetric, multi-view RGB).

Compared with other structures, 3D point cloud is simpler (set of vectors) and easier to obtain (primitive from scanner).

2.2 Point Cloud based Tasks

Nowadays, there are more deep-learning tasks based on 3D point cloud: shape classification, part segmentation, semantic segmentation, shape retrieval, key point detection, object detection. But PointNet and PointNet++ papers are the pioneer of directly using 3D point cloud as networks' input without being transformed to other structures.

In PointNet paper, they successfully built a general network PointNet for three tasks: classification, part segmentation and semantic segmentation.

2.3 Background Knowledge: Neural Networks

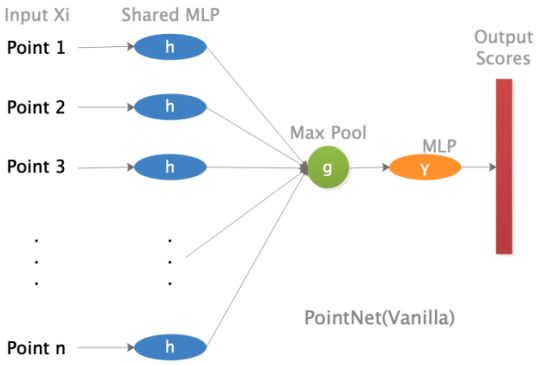
From 2.1, we know there are two essential properties of point cloud: permutation invariance and transformation invariance. So, the problem is, how to preserve these properties in a deep-learning network, or in other words, how to design a network that can satisfied these properties.

Firstly, how to achieve permutation invariance?

Generally speaking, neural network works as an approximation function, it gives different output depends on input.

Symmetric function has transformation invariance property, it can produce the same output form input with different permutation. Typical symmetric function like: Max(), Min(), Sum() and Avg().So, we can use deep neural network to approximate a symmetric function (this is what the max pool layer ***g*** is doing).

However, directly apply a symmetric operation on raw input data is meaningless, since some features will be ignored. So before max pooling, we need to increase data dimension to preserve information (use MLP ***h*** to achieve this goal).



Img1: PointNet initial structure (Vanilla)

Then, how to achieve transformation invariance? The author introduced a light network called T-Net to align input points and features. In my opinion, T-Net is something like a pre-trained Spatial Transformer Network, it can correct the deviation (or reduce the effects) bringing from transformation, rotation and scaling. By applying this automatic transformation network, the input is aligned and the change of perspective will not affect final result.

There is no need to worry too much about T-Net, it can only improve network's performance about 1~2% and it is removed in PointNet++. By the way, if you remove these two T-Nets from PointNet, the structure still make sense. An easier way to understand this operation is to treat it as a pre-trained alignment matrix.

2.4 PointNet structure

Here I use the Img2 from original paper to illustrate my idea.

i. Input: suppose an input point cloud contains n points, and each point has three dimensions in the form of 3D coordinate (x, y, z), so the input points matrix size is $n\times3$.

ii. Input transformation: align input points by multiply a 3×3matrix (T-Net), the output size is $n\times3\times3\times3 = n\times3$.

iii. The first MLP (64 nodes in hidden layer and 64 nodes in output layer): feed data to an MLP, this MLP is shared by all points (all points will use the same weights and structure). This step is based on a general idea: increase the feature dimension of data before doing feature extraction. Because the MLP's output layer is fully connected and contains 64 nodes, data now have 64 dimensions and output size of this MLP is $n\times64$.

iv. Feature transformation: align features by multiply a 64×64matrix (T-Net), the output size is $n\times64\times64\times64 = n\times64$.

v. The second MLP (64 nodes in 1st layer, 128 nodes in 2nd layer and 1024 nodes in output layer): this MLP is shared, the output will be 1024 dimensions with size $n\times1024$.

vi. Max pool layer: this layer extracts the maximum value (a.k.a. global feature) in each dimension (1024 dim in total), so the output size is 1×1024.

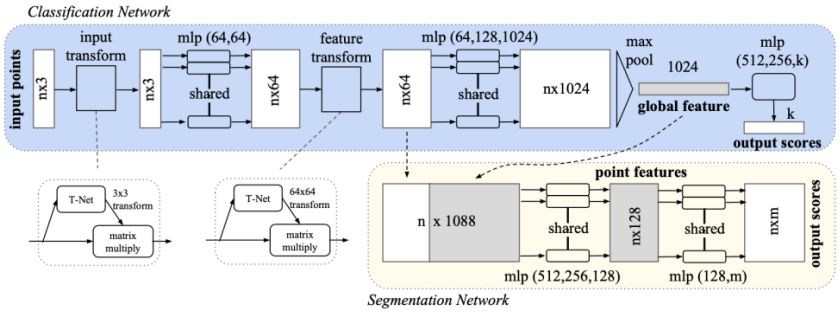
vii. The third MLP (512 nodes in 1st layer, 256 nodes in 2nd layer and k nodes in output layer): k is the number of categories. This MLP is for doing dimension reduction before produce output score for each categories, the output size of this MLP is $1\times k$, this is the last layer for classification model.

The above processes are the classification network in PointNet, it corresponds to the blue region in Img2, and the yellow region shows the segmentation network.

i. Stitching: the segmentation network stitches the output of feature transformation (low dimensional feature/local feature with size: $n\times64$) and output of max pool layer (high dimensional feature/global feature with size: $n\times1024$), together forming a $n\times1088$feature matrix. My explanation for using two kind of feature: object segmentation focuses on points, local feature is useful, the proportion of local/global feature here is 64/1024.

ii. The first MLP (512 nodes in 1st layer, 256 nodes in 2nd layer and 168 nodes in output layer): this MLP is shared and used for dimension reduction, its output size: $n\times128$.

iii. The second MLP (128 nodes in 1st layer and m nodes in output layer): m is the number of segments, for each input point, it will generate the segment scores. So, the final output size of this network is $n\times m$.



Img2: PointNet final structure (img from paper)

3. Code Reproduction

This part describes the every details of the experiment: environment, adopted codes, dataset, task and results. In summary, I referenced to codes from 3 Github repositories, used two dataset (ShapeNet subset and a transferred ModelNet40), the final results are very similar to the paper.

3.1 Experiment Environment

Python version: 3.7	Anaconda version: 1.9.12
Nvidia CUDA version: 9020	Operating system: Windows 10
PyCharm version: 2019.3 (Used as editor)	Visual Studio version: 2010 (Used as dll complier)

3.2 Resources

- Codes:

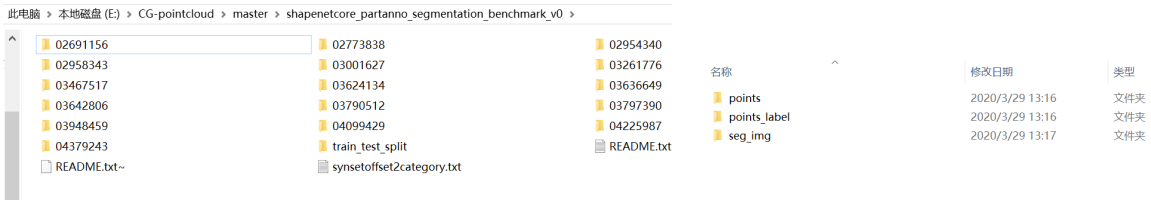
  The reproduction code mainly from Fixa22 [3], this is a pytorch version of PointNet. I also referenced codes in other repositories: charlesq34 [4][5] (he is the author of PointNet paper) and eriche2016 [6] (this is a pytorch version of PointNet++).

  The reason why I did not use the author's code: his code is purely done on Linux, I tried to execute it on Windows and MacOS but failed (it seems that the author used some modules only being supported on Linux). But Fixa22 and eriche2016's code can be reproduced on Windows.

- Dataset:

  The dataset [7] is given by the author, it's a subset of ShapeNet database. Here I used the dataset from the author: shapenetcore_partanno_segmentation_benchmark (about 1.53G). This dataset can be used for both classification and object segmentation tasks.

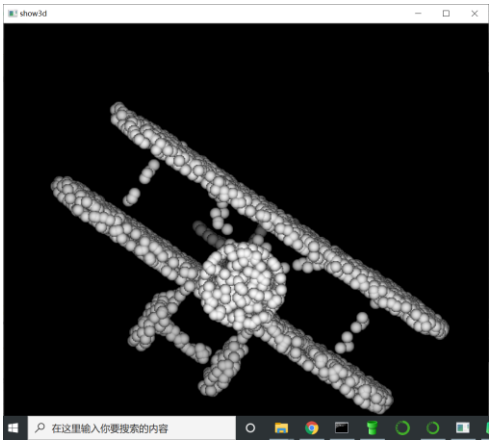
Contents in this dataset:



Img3: Dataset contents

There are 16 classes in this dataset, they are stored in the folders named by numbers. In all these class folders, there will be 3 folders: 'points', 'points_label' and 'seg_img'.

- In 'points' folder there are many .pts file, each .pts file is a point cloud suggest a real object (I wrote some code to visualize it in 3D space, shown in the img4).



Img4: visualization for 02691156\points\1a04e3eab45ca15dd86060f189eb133.pts

- In 'points_label' folder there are .seg files recording the segmentation information of corresponding object in .pts file.
- The 'seg_img' folder was not used in this experiment.

In the 'train_test_split' folder are three .json files that record the split of train, test and validation data. For 'synsetoffset2category.txt', it maps folder names to class names.

The second dataset used is from Princeton University: ModelNet40 [10], it was pre-processed in the form of ShapeNet but point clouds are stored together in .h5 format. This dataset also adopted by the author to show their networks' performance.

3.3 Tasks

I reproduce the code on two tasks:

- Classification: for a given cloud point of an object as input, classify it to a specific class

- Object Segmentation: for a given cloud point of an object as input, segment it to specific number of parts.

Task 1: Classification

The command used to train the classification model:

>> python train_classification.py --dataset=shapenetcore_partanno_segmentation_benchmark_v0 --nepoch=10 -- dataset_type=shapenet --feature_transform=True

--dataset: the path of training dataset, --nepoch: the number of epoch, --dataset_type: can be shapenet or modelnet40, it depends on the dataset used, --feature_transform: set it to True to enable feature transform network (i.e. T-Net mentioned above)

In classification task, the number of sample (the number of point cloud) in training set and test set are 12137 and 2874. Actually, use only no more than 5 epochs, this model can get a really good performance. Here are some logs I saved:

```
[4: 751/758] train loss: 0.171325 accuracy: 1.000000
[4: 752/758] train loss: 0.002450 accuracy: 1.000000
[4: 753/758] train loss: 0.147034 accuracy: 0.937500
[4: 754/758] train loss: 0.035154 accuracy: 1.000000
[4: 755/758] train loss: 0.132367 accuracy: 0.937500
[4: 756/758] train loss: 0.125144 accuracy: 0.937500
[4: 757/758] train loss: 0.020990 accuracy: 1.000000
[4: 758/758] train loss: 0.042953 accuracy: 0.562500
180it [00:33,  5.34it/s]
final accuracy 0.954070981210856
```

```
[249: 751/758] train loss: 0.000688 accuracy: 1.000000
[249: 752/758] train loss: 0.001179 accuracy: 1.000000
[249: 753/758] train loss: 0.000070 accuracy: 1.000000
[249: 754/758] train loss: 0.012517 accuracy: 1.000000
[249: 755/758] train loss: 0.000198 accuracy: 1.000000
[249: 756/758] train loss: 0.000110 accuracy: 1.000000
[249: 757/758] train loss: 0.005017 accuracy: 1.000000
[249: 758/758] train loss: 0.000338 accuracy: 0.562500
180it [00:31,  5.73it/s]
final accuracy 0.9794711203897007
```

Img5: accuracy after trained for 4/250 epoch (left/right)

- The command used to test the classification mode:

>> python show_cls.py --model=E:\CG-pointcloud\Pointnet.pytorch-master\cls\cls_model_4.pth

--model: the path of the saved model, use this model to do classification on test set.

Actually I was supervised by the some of the model's high accuracy on test set, it reached 100% correctness on much of the test data, which made me very anxious, I worried maybe I done something wrong, I double check the code, but everything worked right.

```
Namespace(model='E:\\CG-pointcloud\\Pointnet.pytorch-master\
{'Airplane': 0, 'Bag': 1, 'Cap': 2, 'Car': 3, 'Chair': 4, 'E
'Mug': 11, 'Pistol': 12, 'Rocket': 13, 'Skateboard': 14, 'Ta
{'Airplane': 4, 'Bag': 2, 'Cap': 2, 'Car': 4, 'Chair': 4, 'E
Mug': 2, 'Pistol': 3, 'Rocket': 3, 'Skateboard': 3, 'Table':
i:0  loss: 0.039242 accuracy: 1.000000
i:1  loss: 0.420761 accuracy: 0.937500
i:2  loss: 0.031309 accuracy: 1.000000
i:3  loss: 0.147253 accuracy: 0.875000
i:4  loss: 0.224696 accuracy: 0.937500
i:5  loss: 0.015196 accuracy: 1.000000
i:6  loss: 0.305850 accuracy: 0.937500
i:7  loss: 0.180132 accuracy: 0.937500
i:8  loss: 0.046713 accuracy: 1.000000
i:9  loss: 0.046006 accuracy: 1.000000
i:10  loss: 0.272572 accuracy: 0.937500
i:11  loss: 0.421177 accuracy: 0.750000
i:12  loss: 0.401127 accuracy: 0.875000
i:13  loss: 0.049356 accuracy: 1.000000
i:14  loss: 0.193866 accuracy: 0.937500
i:15  loss: 0.151959 accuracy: 0.937500
i:16  loss: 0.028744 accuracy: 1.000000
```

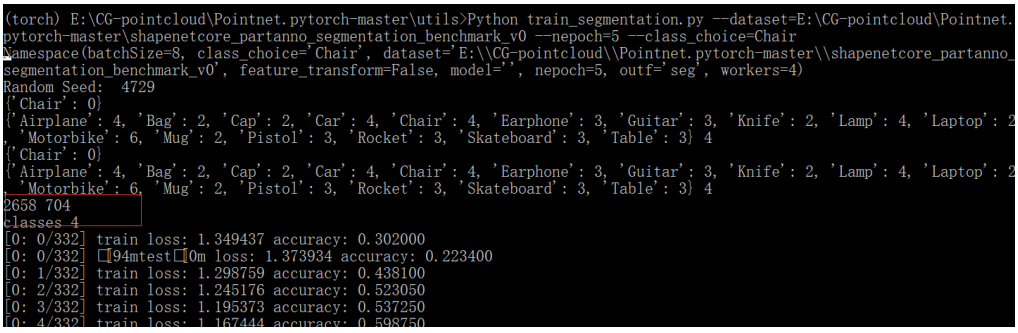
Img6: accuracy on test set

Task 2: Object Segmentation

This code trained each class separately, which means for different classes, I trained different models to do segmentation task. This is because different classes have different segment number, and the shape and segments in different classes are different. After this training process, I obtained 16 segmentation models. How to segment a particular class is learnt from the training set. The command used to train a segmentation mode:

>> Python train_segmentation.py --dataset=shapenetcore_partanno_segmentation_benchmark_v0 --nepoch=5 --class_choice=Chair

--dataset: the path of training dataset, --nepoch: the number of epoch, --class_choice: current segmentation class.

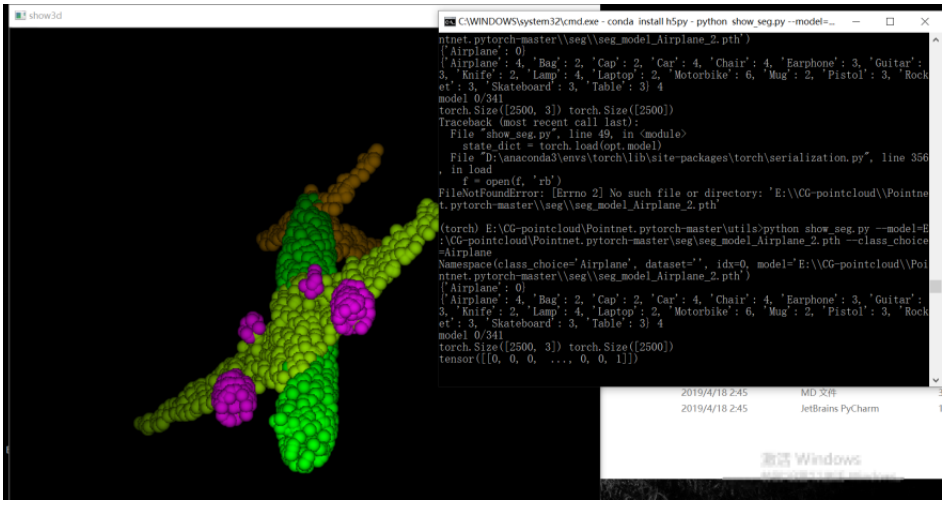


Img7: segmentation model training process

The command used to test the segmentation mode:

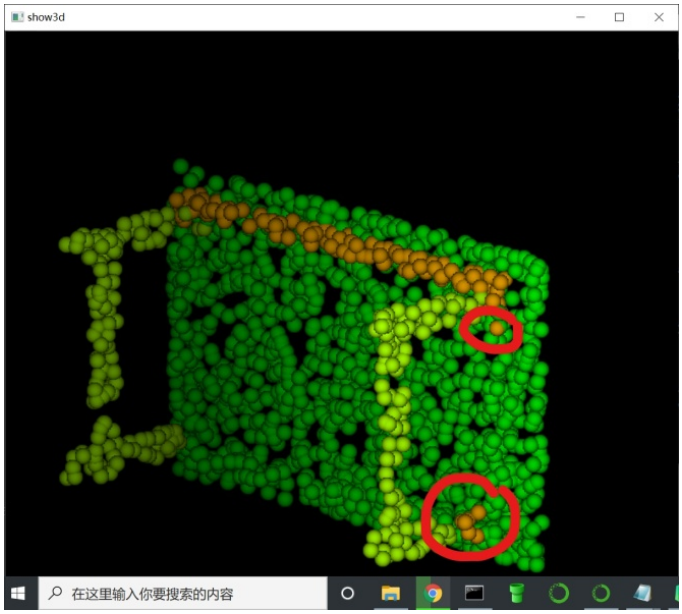
>> python show_seg.py --model= seg\seg_model_Airplane_4.pth --class_choice=Airplane --idx=0

--model/--class_choice: the path of saved model and its class name, for different classes, different model should be selected; --idx: the index of object in the test data.



Img8: visualization tool for showing segmentation result

I was pleased with the segmentation results on test set. By using the visualization tools to check the segmentation result, most of the objects are correctly divided into parts, except some misclassified points. For example, img9 shows one of the object's segmentation in class 'table', some points are wrongly segmented. But the result is acceptable, since more than 95% part of the object are correctly segmented.



Img9: some mis-segmented points

3.4 Result

1. Classification result on ShapeNet dataset and ModelNet40 dataset

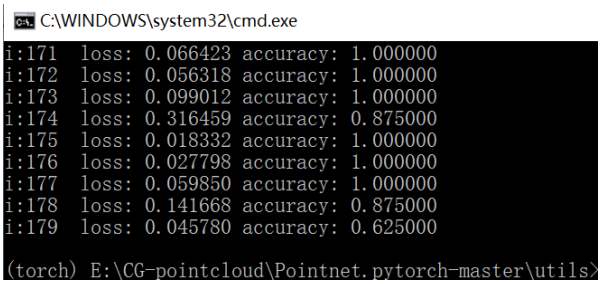
In my experiment, the accuracy is the average of all the samples; accuracy in test set.

	Result (ShapeNet) Author	Result (ShapeNet) Fxia22	Result (ModelNet40) Fxia22	Result (ShapeNet) My experiment	Result (ModelNet40) My experiment
Accuracy with T-Net	89.2	87.0	98.1	87.0	98.0
Accuracy w/o T-Net	87.1	86.4	97.7	85.6	97.9

Tab1: Result of classification task

My classification result shows high similarity with Fxia22's, this model performs better on ModelNet40 dataset. This is easy to explain, even though the ModelNet40 dataset have more classes (40), but most classes have very simple geometry structure: (like bathtub, bookshelf, cone, door, vase).

When testing the model, the dataloader returns test sample in the size of a batch in every iteration, and calculate the average accuracy on this batch. For ShapeNet, there are 2874 test samples, so the number of iteration is 2874/16=179.625(180 iterations are needed to test all samples).



Img10: meaning of iteration number

By setting the argv '—feature_transform' to 'True' or 'False', T--Net can enabled or disabled. On the ShapeNet dataset, in author's provided result, T-Net contributes to 2.1 point to final accuracy, but only 0.4 point in Fxia22's implementation, 1.4 point in my reproduction. On the ModelNet40 dataset, the contributions even less.

To sum up, for the use of T-Net, I think T-Net does improved the performance but not very much (if consider the time cost it's not worthwhile to use it).

2. Segmentation result on ShapeNet dataset

In my experiment, the accuracy of a class is the average accuracy of all the samples of this class on test set. Some classes like Motorbike and Rocket have a lower accuracy (0.3~0.5). After checking its visualization, I found that these classes have less training sample and more segments (5 to 6 segments) at the same time (which means they are complex objects with less data). Other classes like Cap, Bag, Earphone, they got less data, but then only have 2 to 3 segments at the same time, so their accuracy did not drop rapidly on test set.

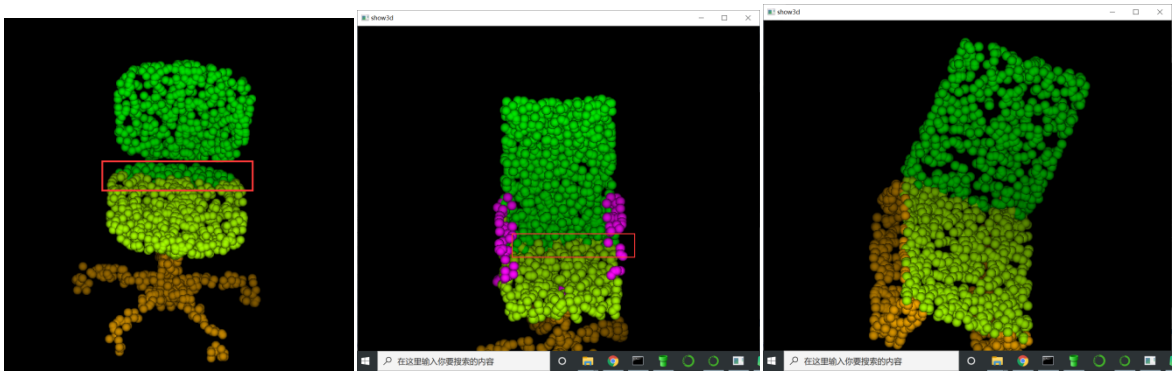
From the result table, you can see that simple classes (with less segments) will yield higher accuracy. So I think the accuracy can be strongly affected by the number of sample in training set (or similarity between samples in training set) and the number of segments.

Class	Result (ShapeNet) Author	Result (ShapeNet) Fxia22	Result (ShapeNet) My experiment	Number of sample (train/test)	Number of Segments
Airplane	83.4	73.5	78.6	1958/341	3/4
Bag	78.7	71.3	72.3	54/14	2

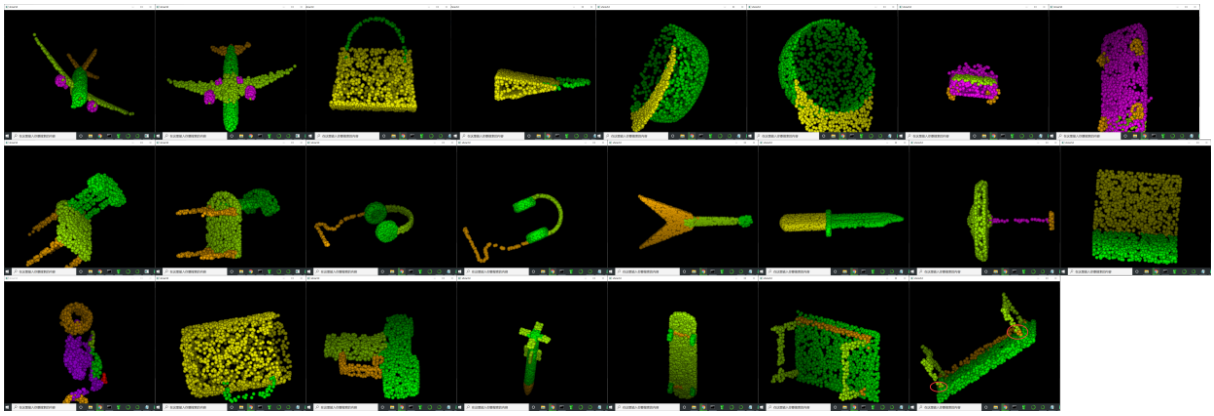
Cap	82.5	87.2	86.1	39/11	2
Car	74.9	69.5	70.5	659/158	3/4
Chair	89.6	86.1	85.5	2658/704	3
Earphone	73.0	69.5	72.0	49/14	2-3
Guitar	91.5	86.1	93.2	550/195	3
Knife	85.9	81.6	80.0	277/80	2
Lamp	80.8	77.4	78.3	1118/286	3
Laptop	95.3	92.7	96.2	324/83	2
Motorbike	65.2	41.3	33.3	125/51	4-6
Mug	93.0	86.5	90.4	130/38	2
Pistol	81.2	78.2	80.1	209/44	3
Rocket	57.9	41.2	43.8	46/12	3-5 (trick shape)
Skateboard	72.8	61.0	70.9	106/31	2-3
Table	80.6	81.1	81.1	3835/848	2-3
Average/ Overall	83.7	74.0	75.7	12137/2910	12137/2910

Tab2: Result of segmentation task

Compare to Fxia22's result, my segmentation result is slightly better. For example, Fxia22 gave an image on one of the segmentation result (Img11-left), I noticed that the junction of backrest and cushion of this armchair is wrongly segmented. But this situation did not happen in my experiment (as shown inImg11-middle/right, the backrest and cushion are in different color).



Img11: segmentation result analysis



Img12: part of the segmentation result

The above Img12 are the screenshots I saved for testing the segmentation model, almost all the objects are perfectly segmented (even with incomplete point clouds). The only bug is sometime there are some single points appeared to belong to other segments even when they were surrounded by points from the same segments. I mentioned this bug at Img9, it's very hard to understand why (maybe it was caused by the some randomness factors/noise in training set).

4. Extended exploration: PointNet++

Because I've already spent too much time on PointNet, I only studied the basic concepts of PointNet++. This part is based on my shallow knowledge, it might have some error.

4.1 The Purpose of Designing PointNet++

Some articles point out that PointNet performs poorly on semantic segmentation task due to the lack of local features. PointNet do Max pooling after 2 T-Net and 2 MLP, in segmentation task, the output of the second MLP (**n×64**, local feature) and the output of Max pooling layer (**n×1024**, global feature) are stitched together as the input for segmentation network. So the proportion of local and global feature is 64/1024, then the local feature has little proportion of the input, the connection between points (in local feature) was not learnt by the network. In object segmentation, this problem is reduced by centralized the coordinate system using objects' local coordinate, but in semantic segmentation, the problem is more tricky (there are multiple objects in global coordinate system).

Second problem is about sampling and grouping. The input of PointNet was not sampled, so all the point clouds information will be feed to the network. So when there are more points, there will be more work, even some points are redundant (this conclusion comes from the

slightly drop of performance when randomly removes some points from the input). So when handling large scene segmentation, a good sampling and grouping algorithm can reduce a lot of work.

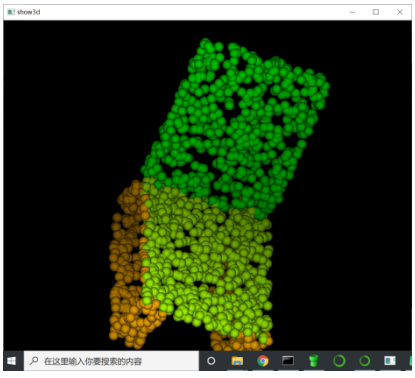
4.2 PointNet++: Focus More on Local Features

Compared to MLP, CNN (convolution neural network) can do better feature extraction by using less parameters, and the convolution operation can enlarge the receptive field of features. PointNet adopts the idea of CNN, use set abstraction to do feature extraction, including sampling, grouping and PointNet.

First operation is to do sampling on input point clouds using FPS (fast point sampling). For each sampled point, use Ball Query (Ball Query Grouping) to make a sphere, points that inside this spherical region will be seen as a group and be transformed into a local coordinate, this local coordinate can provide transformation invariance. Then this partition of the point cloud (group) is input to the PointNet to do the feature extraction, which allows us to extraction on a local region. The extracted features will be passed to the next layer, so the points of one layer are the set of the most important features (call it important because it contains more features) of the previous layer, with the number of layers increase, the extracted features will contain more and more information

4.3 PointNet++: Grouping Method

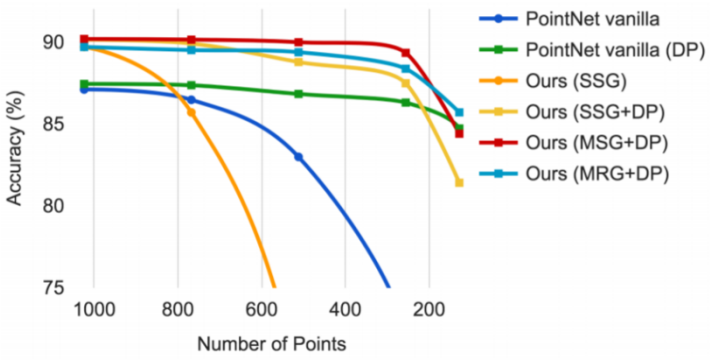
From Img13 we can see that the density of a point cloud is uneven. Some slots appear when the density is lower, and some region with more points looks densely.



Img13: different density in different region

The given experiment result in Img14 from the PointNet++ paper shows the performance of two networks when randomly dropping out some of the points. First, PointNet++ performance is better than PointNet when there are more points (because PointNet++ can

acquire more information of local features). Second, when the number of point decrease, PointNet performs better since it knows more global features than PointNet++. But both 's networks drop rapidly when the number of points decrease. So the traditional grouping (single scale grouping) is not good for the network, a more stable grouping method must be adopted.



Img14: networks with different grouping method

PointNet++ brings up 2 grouping method: Multi-Scale Grouping (MSG) and Multi-Resolution Grouping (MRG). Generally speaking, MSG is better (the red line in Img13) but it takes more time to calculate the network parameters (since the weights are not shred).

5. Appendix: Problem Fixed

You can skip this part, it's something like proof of work.

Due to different coding environment (operating system, system or software version, platform used), even re-deploy others codes are difficult sometime. In this part, I simply write down some problems I have met and solutions of them, which I think it might contribute to others.

1. Install pyTorch in Anaconda:

   It will be better to use Anaconda to deploy your codes, create environment inside Anaconda and use command 'conda install pkg' to install modules. There many toturails online using a lot of steps to install pyTorch, but the best way is to go to pyTorch's official websites and follow the official tutorial:

   1. conda install pytorch torchvision cudatoolkit=10.1 -c pytorch.
   2. conda install pytorch torchvision cudatoolkit=9.2 -c pytorch -c defaults -c numba/label/dev

   Here I made a mistake, my CUDA's version is 9020, but I used the 1st command. So, I got this error:

- The NVIDIA driver on your system is too old (found version 9020).
  - Reason: this is not because you've got an old version NVIDIA but you've got the latest pytorch version.
  - Solution: use the 2nd command. Then everything will be fine.

2. Problems in the origin train_classification.py:
   - PyTorch: The "freeze_support()" line can be omitted if the program is not going to be frozen (occurred in'utils\train_classification.py' and 'utils\train_segmentation.py').
     - Reason: This is because the missing-order of parent process and child processes.
     - Solution: go to 'utils\train_classification.py', put everything within main funtion (except the import part), like this:

       Change:

```
train_classification.py 〉No Selection
from __future__ import print_function
import argparse
import os
import random
import torch
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
from pointnet.dataset import ShapeNetDataset, ModelNetDataset
from pointnet.model import PointNetCls,
    feature_transform_regularizer
import torch.nn.functional as F
from tqdm import tqdm


parser = argparse.ArgumentParser()
parser.add_argument(
    '--batchSize', type=int, default=32, help='input batch size')
parser.add_argument(
    '--num_points', type=int, default=2500, help='input batch
        size')
parser.add_argument(
    '--workers', type=int, help='number of data loading workers',
        default=4)
parser.add_argument(
    '--nepoch', type=int, default=250, help='number of epochs to
        train for')
```

       To:

```
train_classification.py 〉No Selection
from __future__ import print_function
import argparse
import os
import random
import torch
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
from pointnet.dataset import ShapeNetDataset, ModelNetDataset
from pointnet.model import PointNetCls,
    feature_transform_regularizer
import torch.nn.functional as F
from tqdm import tqdm

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--batchSize', type=int, default=32, help='input batch
            size')
    parser.add_argument(
        '--num_points', type=int, default=2500, help='input batch
            size')
    parser.add_argument(
        '--workers', type=int, help='number of data loading
            workers', default=4)
    parser.add_argument(
        '--nepoch', type=int, default=250, help='number of epochs
            to train for')
```

   - RuntimeError: CUDA error: out of memory (in torch\nn\modules\conv.py line 176)
     - Reason: the requested memory size exceeds my system's ability (or due to some limitations)

- Solution: go to ‘utils\train_classification.py’, I cut down the batchsize to a half.

  Change:

```
parser.add_argument(
    '--batchSize', type=int, default=32, help='input batch size')
parser.add_argument(
    '--num_points', type=int, default=2500, help='input batch size')
```

  To:

```
parser.add_argument(
    '--batchSize', type=int, default=16, help='input batch size')
parser.add_argument(
    '--num_points', type=int, default=1250, help='input batch size')
```

  In addition, if you change the batch_size and num_points parameters here, remember to change the ‘show_cls.py’, you need to modify the batch_size too. There will be a problem in ‘show_cls.py’ if you change the batch size, you should also change the equation used to calculate accuracy, otherwise, you will only get half-accuracy because it will be divided by batch_size=32.

  Change:

```
47      pred_choice = pred.data.max(1)[1]
48      correct = pred_choice.eq(target.data).cpu().sum()
49      print('i:%d loss: %f accuracy: %f' % (i, loss.data.item(), correct / float(32)))
50
```

  To:

```
47      pred_choice = pred.data.max(1)[1]
48      correct = pred_choice.eq(target.data).cpu().sum()
49      print('i:%d loss: %f accuracy: %f' % (i, loss.data.item(), correct / float(16)))
50
```

■ No module named ‘pointnet.model’ (in utils\train_classification.py line 7)

- Reason: python program migration issue, because ‘pointnet.model’ is in the parent folder of our current scripts, when you want import something, python will search system path, if the module you want to import is not in the path, ‘no module named’ happens.
- Solution: add the folder that contains ‘pointnet’ to path, in ‘utils\train_classification.py’, add two line before you import ‘pointnet’ (this problem also occurred in ‘train_segment.py’):

```
from __future__ import print_function
import argparse
import os

import sys
import os
sys.path.append(os.getcwd())  #or sys.path.append("path_of_parent_folder")

import random
import torch
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
```

3. Visualization tools:

Pointnet paper's author provided a visualization tool in their open-source codes (in utils folder: render_balls_so.sh), but it was developed for Linux user. But I did some works so that it can be used on Windows. By running 'render_balls_so.sh', I got:

- g++: command not found.
  - Reason: in 'render_balls_so.sh', they used g++ command to build.'render_balls_so.so' library, which is not supported by Windows default cmd.
  - Solution:you need to download a software called 'MingGW' and add it to system path. After that you can compile cpp file directly in cmd.
- OSError: [WinError 193] %1 is not a valid Win32 application. (This error actually wasted me a whole day to solve.)
  - Reason: I don't know why this error happened, but somebody on Github gave me the suggestion to install openCV and use its extension.
  - Solution: remember to run visualization scripts using 'py -2 XXX.py' if you have python3 installed.
  - Comments: I tried to build this .so lib in a Linux virtual machine, it was built successfully on Linux, but cannot be copied to Windows and called directly, the only way is to compile it in Windows (use complier like VS2010 to do so). Noted that, only dll in x64 can be recognized by python3 (according to official, win32 dll is not supported).

However, if you want to run it using Python3 (since other visualization tools need pytorch, and pytorch only support python3 now), you need to rebuild 'render_balls_so.cpp' as a dll lib using VS2010, remember to use the x64 complier, or you will get the error anyway.

4. Handle the left-alone data:

During the training process, if the number of samples mod the batch size is equal to 1, you will get this error. This is also a designing carelessness of pytorch.

- ValueError: Expected more than 1 value per channel when training, got input size [1, 512] (this happens in training the segmentation).
  - Reason: input data later will be feed to neutral network in the size of a batch, some torch operation in this process cannot be finished using only one data.

- Solution: in Dataloader part, add a constraint parameter 'drop_last' and set it to 'True'. In train_classification/train_segmentation.py:

  Change:

```
dataloader = torch.utils.data.DataLoader(
    dataset,
    batch_size=opt.batchSize,
    shuffle=True,
    num_workers=int(opt.workers))
```

  To:

```
dataloader = torch.utils.data.DataLoader(
    dataset,
    batch_size=opt.batchSize,
    shuffle=True,
    drop_last=True,
    num_workers=int(opt.workers))
```

6. Reference

[1] Qi C R, Su H, Mo K, et al. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation[J]. 2016.

[2] Qi C R, Yi L, Su H, et al. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space[J]. 2017.

[3] Fxia22's Pointnet pytorch version: https://github.com/fxia22/pointnet.pytorch

[4] PointNet code from Qi C R's team: https://github.com/charlesq34/pointnet

[5] PointNet++ code from Qi C R's team: https://github.com/charlesq34/pointnet2

[6] eriche's Pointnet++ pytorch version: https://github.com/eriche2016/pointnet2.pytorch

[7] Stanford dataset: http://stanford.edu/~rqi/pointnet/

[8] An overview of 3D point processing processing: https://www.bilibili.com/video/BV1f7411K7mq?t=4037

[9] Inspiring presentation on Pointnrt from Qi C R: https://www.bilibili.com/video/BV1As411377S?t=1573

[10] ModelNe40t dataset: http://modelnet.cs.princeton.edu/

[11] Python calling dll on Windows: https://www.cnblogs.com/zhbzz2007/p/4515118.html

[12] Original code from Qi C R's team: https://github.com/charlesq34/pointnet

[13] Online visualization tool: http://lidarview.com/

P.S. Different teams have their own dataset in different formats (Wavefront: .obj, Stanford: .ply/.h5, Object File Format .off; STL files: .stl), which is not good for building a uniform ecosystem. Even though those data can be transformed using program, but I think is very meaningful to build up a uniform point cloud format (a standard).