Author	Student ID	Email
Junzhi Chen	3038732738	junzhi@berkeley.edu
Songlin Zhao	3038737171	zhao_songlin@berkeley.edu
Zhen Tong	3038732400	tonytong@berkeley.edu
Jianwen Chen	3038737119	jimchenxm@berkeley.edu

1 Abstract

PointNet is a neural network architecture that can process raw point cloud data without pre-processing or feature extraction steps. The main challenge with processing point cloud data is that it is unordered, making it difficult for traditional to do training. PointNet addresses this challenge by extracting global features and ensuring permutation invariance through shared MLP and max-pooling operations. Additionally, PointNet incorporates the Joint Alignment Network to eliminate the influence of translation, rotation, and other factors on point cloud classification and segmentation. The network also concatenates data and global features during segmentation to aggregate local and global information.

In this assignment, students will implement the PointNet Classification and PointNet Segmentation modules. Students can learn about point cloud data and pre-processing, implement shared MLP and Joint Alignment Network, tune the alpha value to observe the accuracy change, and write the code for the loss function to understand better what the network tries to learn.

2 Point-to-point Response to the Review Comments

1. Reviewer 1

- **Content and Correctness**

  grade: Small improvement needed

  Comment: This assignment introduces students to PointNet for point cloud processing. The assignment begins with an simple analytic question that asks students to think about what makes the network permutation-invariant. The bulk of this assignment lives in the Colab notebook, and it covers everything from transforming and sampling data, building the Tnet transform class and eventually building a full PointNet Classifier. It is especially helpful to include the 3D scatterplots, which are really helpful in understanding the distinct distribution that PointNet models work with. There is one small bug in the class definition, and it looks like it is simply a typo and can be fixed really easily. At this moment there is an AttributeError in the forward call of Tnet where "matrix = self.last_last(xb).view(-1,self.k,self.k) + init", I believe the self.last_last should be self.last_fc. Other than that the assignment is excellent.

  I can't seem to find the Latex source as required by the guidelines, but the PDF is very detailed and contains both analytical questions and their solutions. This should be doable in about 2 hours by an average 182 student.

  Response: 1. Indeed, there is a small typo of a function name. We have already fixed that. 2. We will include the latex source in our final submission by providing a github link on the submission.

- **Scaffolding**

  grade: Excellent work, no actions needed.

  Comment: The homework contains most of the necessary information. The classes in the coding questions are clearly separated as independent modules that can be assembled together. The data is automatically downloaded from the internet and there are test cases that act as sanity checks. The more difficult coding questions like the T-net and the transform class also have very detailed instructions that students can follow. There are pictures outlining the architecture of each component of the models and also text cells showing the relevant mathematical formulas

  Response: Thanks for the comment!

- **Readability/Clarity**

  grade: Excellent work, no actions needed.

  Comment: The instructions in this problem are very clear with very few typos. The use of images and visualization tools make the instructions easier to follow. Aside from one small typo in the code, the assignment runs smoothly with problems.

  Response: We have proofread the coding notebook and fixed all the typos we can find.

- **Commentary on HW**

  grade: Excellent work, no actions needed.

Comment: I assume the first 2 pages in the PDF is the commentary. The commentary is a very succint summarization of the actual assignment and it makes easier to understand the scope of the problems.

Response: Thanks for the comment!

- **Going above and beyond**

  grade: Excellent work, no actions needed.

  Comment: The assignment demonstrates some key concepts introduced in the paper and does a good job in simplifying and condensing the contents of the paper into one assignment problem. The visualizations are solid and really help with the understanding, especially the part of PointNet segmentation. Together it covers most of the concepts introduced in the paper and delivers them in a way that is understandable for most students.

  Response: Thanks for the comment!

2. Reviewer 2

- **Content and Correctness**

  grade: Small improvement needed

  Comment: Content: Doing the full problem would probably take well over 2 hours to do for an average CS 182 student (maybe 4 hours total?). I think the classifier notebook itself takes over 2 hours. I'm not sure about the segmentation notebook due to access issues, but I'm sure it would take some time to complete as well. The implementation looks straightforward to do for a CS 182 student. However, I think some parts in the solution require students to code up transferring to GPU instead of CPU, which I feel maybe should be handled by the given code. Overall, the content is extremely well done. The key concepts are highlighted clearly in the introduction. The homework thoughtfully guides students through understanding the concepts step-by-step. I would just consider making the homework a bit shorter by maybe providing given solutions in some parts to make students have less tasks to implement.

  Correctness: The segmentation homework and solution notebook are not openable due to invalid authentication credentials. Maybe there are sharing permission issues? The solution for T-Net's forward method has a minor mistake. The last line should be "self.last_fc", not "self.last_last". Otherwise, the project looks almost fully correct with very few spelling typos here and there. For example, the subsection after T-Net should be "Transform Class", not "Transfrom Class".

  Response: 1. We have deleted some coding problems 2. We tested that the segmentation notebook can be accessed normally and we don't know why the reviewer cannot access it. 3. Indeed, there is a small typo of a function name. We have already fixed that. 4. We have proofread the coding notebook and fixed all the typos we can find. 5. There is only one part that involves transferring CPU to GPU, where student need to create a new matrix and add it with existing matrix that is already in GPU. So it's natural that we need to transfer the newly created matrix to GPU after creating it. If we handle it for students, the structure of that problem will be messy.

- **Scaffolding**

  grade: Excellent work, no actions needed.

  Comment: The assignment has good scaffolding, providing all the necessary code/datasets required to engage with the material. All of the code needed to be implemented is well-described and has many comments to explain and guide students in the right direction. There are good autograding tests which verify student implementations.

  Response: Thanks for the comment!

- **Readability/Clarity**

  grade: Small improvement needed.

  Comment: I would say there are a few spelling or grammar mistakes here and there, but this definitely does not affect the overall readability and clarity. Everything is thoroughly explained and easily understandable. One suggestion would be to include an overview of everything in the classifier notebook in the beginning. For the "Understand the Data" section, I think it should be slightly clarified that there are ten classes that we are aiming to classify. It is only apparent until a bit later into the notebook.

  Response: 1. We have proofread the coding notebook and fixed all the typos we can find. 2. We have added an overview according to the suggestion. Thanks for the suggestion! 3. We have a markdown note directly after downloading the data and before "understanding the data", indicating that the objects are labeled 0,1,...,9, reviewer might not see it.

- **Commentary on HW**

  grade: Excellent work, no actions needed.

  Comment: The commentary on the homework is very well written. It meets the page-length requirement. It also does a great job in explaining the key concepts by dividing into 3 sections (permutation invariant, T-Net, and segmentation) and showing how the homework guides students through those 3 concepts. The commentary is very helpful to understanding the concepts of the PointNet paper and achieve the learning goals of the homework.

  Response: Thanks for the comment!

- **Going above and beyond**

  grade: Excellent work, no actions needed.

  Comment: The project does a very good job going above and beyond. There are a plentiful number of visualizations to illustrate ideas for students. There is a dataset for students to visualize, train, fine-tune. There are also multiple graphs at the end for model performance and a confusion matrix. The analytical problems are very creative. There are ways to go further maybe but the project definitely exceeds expectations already in this area of the rubric in my opinion. One suggestion is maybe including 1-2 more analytical questions.

  Response: Thanks for the comment!

3. Reviewer 3

- **Content and Correctness**

  grade: Small improvement needed.

  Comment: Overall, the content of the assignment is very strong, and the Colab notebooks run smoothly in general. There is sufficient exposition provided about point clouds and the paper for someone unfamiliar with it, and it engages well with the key concepts of the paper through good problems and visualizations. The 2 Colab notebooks build up the models used for Classification and Segmentation well to deepen students' understanding of the paper. However, there is a minor bug in each Colab notebook. In the Classifier notebook, there is a bug in the Tnet class in "matrix = self.last_last(xb).view(-1,self.k,self.k) + init ". self.last_last should be self.last_fc. In the Segmentation notebook, there is a bug in downloading expert_verified.zip, which can be fixed by modifying the wget command.

  Response: 1. Indeed, there is a small typo of a function name. We have already fixed that. 2. We have modified wget to make downloading expert_verified.zip runs normally.

- **Scaffolding**

  grade: Excellent work, no actions needed.

  Comment: The assignment has good scaffolding, providing all the necessary code/datasets required to engage with the material. All of the code needed to be implemented is well-described and has many comments to explain and guide students in the right direction. There are good autograding tests which verify student implementations.

  Response: Thanks for the comment!

- **Readability/Clarity**

  grade: Small improvement needed.

  Comment: The assignment is overall very easy to understand and follow. Any notation used is explained. There are a few minor grammatical errors, but they do not impede understanding of the assignment.

  Response: 1. We have proofread the coding notebook and fixed all the typos we can find.

- **Commentary on HW**

  grade: Excellent work, no actions needed.

  Comment: The commentary provided is helpful, explaining all of the key concepts of the paper. The HW achieves all of the learning goals through insightful questions and guided coding assignments which build up understanding of the overall network architecture. Another strong point is how it connects new ideas in the paper to what we have previously learned in class, such as UNets.

  Response: Thanks for the comment!

- **Going above and beyond**

  grade: Excellent work, no actions needed.

Comment: The assignment does a great job of simplifying the paper. There are also great visualizations that help students better understand point clouds in general; how parts of the network, like the T-net, transform point clouds; and how the model performs in segmentation. The confusion matrix in the Classifier notebook is also a great tool to help show students the model's strengths and weaknesses, and the problem on the confusion matrix pushes students to think why that might be the case.

Response: Thanks for the comment!

3 Commentary on the HomeWork

3.1 Key Concept

3.1.1 Background

Point Cloud Data Point clouds are sets of 3D points that represent the shape and structure of objects or scenes as unordered point sets. Due to the unordered property of the point cloud, traditional networks cannot directly use the point cloud data to train.

PointNet PointNet was introduced by Charles R. Qi et al. in a 2017 paper titled "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." It is a neural network architecture that directly takes raw point clouds as input and processes them without pre-processing or feature extraction steps. PointNet can learn meaningful features from point clouds, such as local and global geometric information, and can be used for tasks such as 3D object classification, semantic segmentation, and point cloud generation.

3.1.2 Core Concept of Paper

Make the Network Permutation Invariant Extracting a global feature is essential for classification and segmentation tasks because the input data cannot reflect the global information. The main challenge with processing point cloud data is that it is unordered. We want the same output features regardless of the order of the input point cloud. We need a symmetric function for unordered input to have a permutation-invariant network. To handle the above problem, we apply shared MLP on the individual points in the input point cloud to expand the input dimension. And we achieve permutation-invariant by using max-pooling operations to aggregate information across the points.

Joint Alignment Network The Joint Alignment Network (T-Net) can eliminate the influence of translation, rotation, and other factors on point cloud classification and segmentation. The input point cloud passes through this network to obtain an affine transformation matrix. Then, this matrix is multiplied by the original point cloud to obtain a new nx3 point cloud. The same method is applied to the feature space to ensure the invariance of features.

Concatenating data and global feature during segmentation Different from object classification which just needs global feature information, part segmentation needs both global and local feature. To do local and global information aggregation, we use a method similar to U-Net. PointNet aggregates local and global features simply by concentrating the global feature and local feature of each point in the hidden layer of MLP. With this modification, the network can predict each point category relying on global semantics and local geometry.

3.1.3 How HW engages with the concept

Understand Point Cloud Data The data we download is ModelNet10. It consists of two parts: vertives and faces.

- We first display the contents in the data file.

- We then let students to write some functions to pre-process the data. We first sample point-cloud from faces of the object. Then we do normalization, rotation, and adding noise.

Make the Network Permutation Invariant

- Firstly, we want students to understand shared MLP in the network. The shared MLP is the same as doing 1d convolution with kernel size 1. So we first let student implement the sharedMLP layer, and compared it with torch.conv1d package.
- We then let students the forward pass of Transform network and PointNetClassifier network so that they have a clear understanding of the whole network.

Joint Alignment Network

- Let students implement the forward pass of T-net in the notebook to understand the architecture of T-Net.
- In order to constrain the matrix learned by T-Net to be a rotation matrix, a penalty term is added to the loss of the module and has a coefficient alpha. Let students tune the alpha value and observe the accuracy change due to the alpha. This can help students understand the loss design and the intuition for the proposal of T-Net in PointNet.
- Let students write the code for loss function to get a better understanding of what the network tries to learn: classification error + regularization term.

Concatenating data and global feature during segmentation This part is similar to the U-net structure we learned in class. Since the segmentation gives each point a class label, so we need both local and global information when doing segmentation.

- We let students implement the stack operation that concatenates previous hidden layer outputs and the global feature.

3.2 Homework without Solution

3.2.1 Introduction

PointNet is a deep learning model for point cloud processing, which is a type of 3D data representation commonly used in computer vision and robotics applications. Point clouds are sets of 3D points that represent the shape and structure of objects or scenes in the form of unordered point sets.

PointNet was introduced by Charles R. Qi et al. in a 2017 paper titled "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." It is a neural network architecture that directly takes raw point clouds as input and processes them without any pre-processing or feature extraction steps. PointNet is capable of learning meaningful features from point clouds, such as local and global geometric information, and can be used for tasks such as 3D object classification, semantic segmentation, and point cloud generation.

The PointNet architecture uses multi-layer perceptrons (MLPs) and max pooling operations to process individual points in the point cloud, as well as symmetric functions to aggregate the features of all points into a global feature vector that captures the overall structure of the point cloud. PointNet can be extended with additional modules, such as PointNet++ for hierarchical feature learning or PointNet-based convolutional neural networks (CNNs) for local feature extraction.

PointNet has been widely used in various applications, such as autonomous driving, robotics, virtual reality, and augmented reality, due to its ability to process raw point cloud data directly and learn meaningful features from unstructured 3D data.

The structure of the PointNet is shown in Figure 1. Please follow the instructions in this notebook. You will build and train PointNet on Point Cloud classification and segmentation dataset.

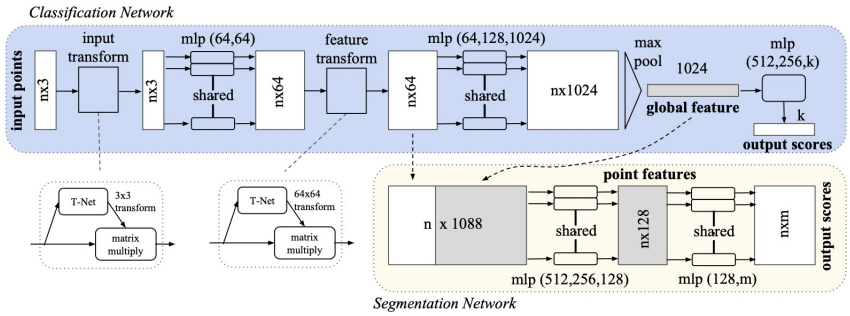


Figure 1: PointNet

3.2.2 Analysis Problem

A point cloud is a discrete *set* of data points in space. Because of this set-valued nature of point clouds, concepts from graph neural nets are often relevant in their processing.
For the first problem, we deal with 2D point cloud to get start. In Figure 1, we consider a simple network to process a 2d point cloud $X = \{x_i\}_{i=1}^{n} \in \mathbb{R}^{n\times 2}$, where n is the number of points. The original features of each point are its horizontal and vertical coordinates.

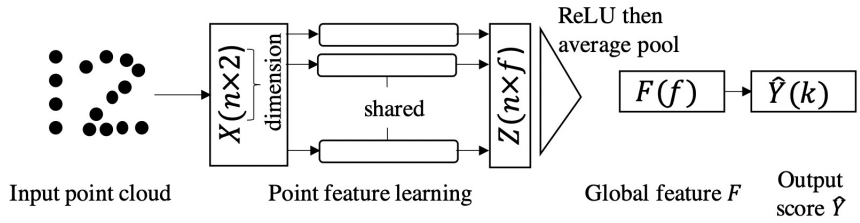


Figure 2: 2d point cloud processing network.

For example, an input point cloud with ground-truth of digit 1 could be represented as $\begin{bmatrix} 0 & 4 \\ 0 & 3 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$ where each row is a different point in the point cloud.
(a) The point feature learning module in Figure 1 learns f-dimensional features for each point separately. Specifically, it learns (shared) weights $W_1 \in \mathbb{R}^{2\times f}$ to get hidden layer outputs $Z = XW_1$. We then apply the nonlinear activation function element-wise and then use average pooling to yield the f -dimensional global feature vector $F \in \mathbb{R}^f$. Suppose we swap the first two points of the input point cloud X, i.e. $x_1, x_2, ..., x_n$ to $x_2, x_1, ..., x_n$. Show that the global feature F will not change.
Note: In reality, the network here is permutation invariant, as changing the ordering of the n input points in X will not affect the global feature F

(b) We know that we can measure the vector distance by the norm, but how can we measure distance between data out-of-order? The **Hausdorff** distanceis the longest distance you can be forced to travel by an adversary who chooses a point in one of the two sets, from where you then must travel to the other set.
The Hausdorff distance, denoted by $d_H(A, B)$, between two non-empty subsets A and B of a metric space (X, d) is defined as:

$$d_H(A, B) = \max\{\sup_{a\in A} \inf_{b\in B} d(a, b), \sup_{b\in B} \inf_{a\in A} d(a, b)\}$$

Intuitively, the Hausdorff distance measures the greatest distance from any point in A to its closest point in B, and vice versa. In other words, it quantifies how far apart the two sets are, taking into account the distances between all pairs of points in the sets.

For instance, in cloud points context, an object, say a chair, can be seen as a point set S_A in coordinate, and the other chair can be seen as another set S_B; another object, say desk, can be

interpreted into S_C. Our job is to find a appropriate function(NN), that :

$$d_H(S_A, S_B) \leq d_H(f(S_A), f(S_B))$$

$$d_H(S_A, S_C) \geq d_H(f(S_A), f(S_C))$$

Formally, a toy model can reveal the idea of PointNet.

There are 2 different Gaussian distribution $X \sim \mathcal{N}(\vec{\mu}_1, \Sigma_1)$ and $Y \sim \mathcal{N}(\vec{\mu}_2, \Sigma_2)$ The 8 points

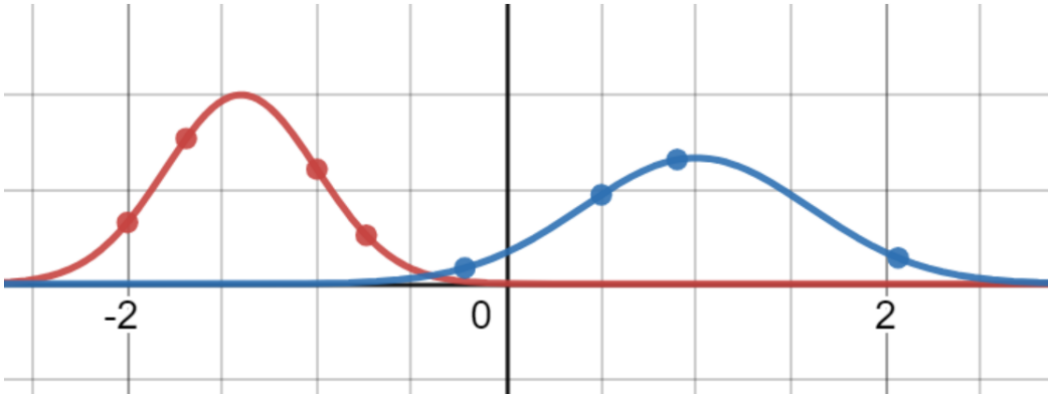


Figure 3: One dimension sample

sampled from the distribution are (from left to right)

$$X_0 = (-2, -1.69, -0.74, -1),\ Y_0 = (-0.22, 0.5, 0.9, 2.066)$$

Now compute the discrete Hausdorff distance between two sample.

(c)Now we have the sample rotated:

$$X_1 = RX_0$$

$R \in \mathbb{R}^{n \times n}, \det(RR^T) = 1$, in the example, $n = 4$. The toy PointNet will do the following things to make the output Hausdorff distance closer:

- Transform the two set to higher-dimension by input each point into a same function:

$$\mathbf{X}_0 H = \begin{bmatrix} h(X_0[1]) \\ h(X_0[2]) \\ h(X_0[3]) \\ h(X_0[4]) \end{bmatrix}, \mathbf{X}_1 H = \begin{bmatrix} h(X_1[1]) \\ h(X_1[2]) \\ h(X_1[3]) \\ h(X_1[4]) \end{bmatrix}$$

- Doing Maxpooling for each point among high-dimension.

$$g(\mathbf{X}_0 H) = \begin{bmatrix} \max(h(X_0[1])) \\ \max(h(X_0[2])) \\ \max(h(X_0[3])) \\ \max(h(X_0[4])) \end{bmatrix}, \quad g(\mathbf{X}_1 H) = \begin{bmatrix} \max(h(X_1[1])) \\ \max(h(X_1[2])) \\ \max(h(X_1[3])) \\ \max(h(X_1[4])) \end{bmatrix}$$

Intuitively explain why do we need to increase the dimension by $h(\cdot)$, and what is max layer $g(\cdot)$ doing?
(hint: think of the Hausdorff distance also have a max)

3.2.3 PointNet Classifier

(a) Finish the coding part in the colab notebook . Fine tuning the hyper-parameters to do the training

i Feel free to tune the alpha, and observe the output transform matrix of t-net and visualize the item in dataset before and after applying transform matrix, identify the similarities and differences between the input item and item applied transform matrix. What effect do you think alpha have? And when alpha is large what t-net is doing? Do you think whether alpha value can improve model performance?

ii Observe the confusion matrix, which objects have classification accuracy that is significantly higher than the others? Which objects have classification accuracy that is significantly higher than the others? Briefly explain why.

3.2.4 Understanding the structure and data dimension

The T-Net can get a learned Transform Matrix that transforms an object to a canonical position. The structure of this model can be seen as a small Point-Net. To better understand the model, we list the layers in T-Net, and we represent the shared-MLP in conv1d() way. B is the batch size of the data, and N is the number of points in an object. Furthermore, the brackets, like(B×N) define a vector $\in \mathbb{R}^M, (M = B \times N)$ dimension, rather than a matrix A$\in \mathbb{R}^{B\times N}$

	Input	Output
covn1d[64]	B× 3× N	
conv1d[128]		
conv1d[1024]		B× 1024× N
MaxPool	B× 1024× N	
Flatten		(B×1024)
Linear[B×9]	(B×1024)	
View layer[B,3, 3]		B×3×3

3.2.5 PointNet Segmentation

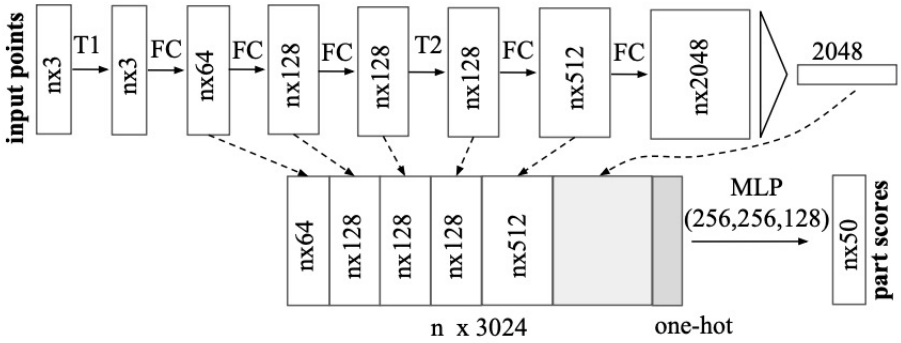


Figure 4: Segmentation PointNet

x

Finish the coding part in the colab notebook and answering the following questions.
(a) Observe the segmentation output of the module, what part of the object has the lowest segmentation accuracy? Briefly explain why.
(b) Briefly describe the difference between classification PointNet and segmentation PointNet. Briefly explain why segmentation PointNet is designed this way.

3.2.6 Classifier.ipynb Code (without solution)

PointNet

In this assignment, students will implement the PointNet Classification and PointNet Segmentation modules. Students can learn about point cloud data and pre-processing, implement shared MLP and Joint Alignment Network, tune the alpha value to observe the accuracy change, and write the code for the loss function to understand better what the network tries to learn.

This homework is based on the paper PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

This homework refers to the article Deep Learning on Point clouds

Overview

In this assignment, you will implement the PointNet Classification and PointNet Segmentation modules. In this homework, you will learn about point cloud data and pre-processing, you will implement shared MLP and Joint Alignment Network, tune the hyperparameters to observe the accuracy change, and write the code for the loss function to understand better what the network tries to learn.

Preparing the point cloud data

```
import math
import random
import os
import numpy as np
np.random.seed(7)
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import scipy.spatial.distance
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
from tqdm.notebook import trange, tqdm
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import itertools
```

```
import plotly.graph_objects as go
import plotly.express as px
```

```
!pwd
```

```
!pip install path.py;
from path import Path
```

Download the dataset

When you download a file using **wget** in Google Colab, the file is saved to the virtual machine's temporary storage associated with the current runtime session. By default, the downloaded file will be saved in the current working directory, which is /content.

The downloaded file will be lost when the runtime is disconnected.

Download the dataset

```
#@title Download the dataset
%cd /content
!wget http://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zip
!unzip -q ModelNet10.zip;
```

The dataset contains 10 classes, labeled from 0, 1, ..., 9

```
path_unsampled = Path("ModelNet10")
# read all the directories in ModelNet10
folders = [dir for dir in sorted(os.listdir(path_unsampled)) if os.path.isdir(path_unsampled/dir)]
classes = {folder: i for i, folder in enumerate(folders)};
classes
```

Understand the data

The dataset consists of **off** files that a 3D mesh has represented by **vertices** and **triangular faces**

Take the data in the file chair/train/chair_0001.off as an example:

- The first line is a symbol **OFF** representing the file is an .off file.
- The second line has three numbers. The first number is the number of vertices which is 2382 and the second number is the number of faces which is 2234.
- The following 2382 lines are coordinates of vertices in the space.
- The last 2234 lines are faces in the space. In each line, the first number is 3, the following 3 numbers are three vertices of a triangle.

display the first 6 lines in the file

```
#@title display the first 6 lines in the file
file_path = path_unsampled/"chair/train/chair_0001.off"
!sed -n '1,6p' "$file_path" | cat
```

display the first line 3000-3005 in the file

```
#@title display the first line 3000-3005 in the file
!sed -n '3000,3005p' "$file_path" | cat
```

Read the data

```
def read_off(file):
  '''
  return: verts: List[List]
          faces: List[List]
  '''
  if 'OFF' != file.readline().strip():
    raise('Not a valid OFF header')
  # read the first line
  n_verts, n_faces, __ = tuple([int(s) for s in file.readline().strip().split(' ')])
  # read vertices and facess
  verts = [[float(s) for s in file.readline().strip().split(' ')] for i_vert in range(n_verts)]
  faces = [[int(s) for s in file.readline().strip().split(' ')][1:] for i_face in range(n_faces)]
  return verts, faces

def read_sampled_off(file):
  if 'OFF' != file.readline().strip():
    raise('Not a valid OFF header')
  file.readline()
  file.readline()
  n_points = tuple([int(s) for s in file.readline().strip().split(' ')])[0]
  file.readline()
  points = [[float(s) for s in file.readline().strip().split(' ')] for i in range(n_points)]
  return points


# read vertices and faces of file chair_0001.off
with open(path_unsampled/"chair/train/chair_0001.off", 'r') as f:
  verts, faces = read_off(f)
```

Visualize the *data*

```
def visualize_rotate(data):
  x_eye, y_eye, z_eye = 1.25, 1.25, 0.8
  frames=[]

  def rotate_z(x, y, z, theta):
    w = x+1j*y
    return np.real(np.exp(1j*theta)*w), np.imag(np.exp(1j*theta)*w), z

  for t in np.arange(0, 10.26, 0.1):
    xe, ye, ze = rotate_z(x_eye, y_eye, z_eye, -t)
    frames.append(dict(layout=dict(scene=dict(camera=dict(eye=dict(x=xe, y=ye, z=ze))))))
  fig = go.Figure(data=data,
                  layout=go.Layout(updatemenus=[dict(type='buttons',
                                    showactive=False,
                                    y=1,
                                    x=0.8,
                                    xanchor='left',
                                    yanchor='bottom',
                                    pad=dict(t=45, r=10),
                                    buttons=[dict(label='Play',
```

```
                                        method='animate',
                                        args=[None, dict(frame=dict(duration=50, redraw=True),
                                                         transition=dict(duration=0),
                                                         fromcurrent=True,
                                                         mode='immediate')]
                                              )
                                        ]
                                    )
                                ]
                                ),
                    frames=frames)
  return fig


def pcshow(xs,ys,zs):
    data=[go.Scatter3d(x=xs, y=ys, z=zs,
                                  mode='markers')]
    fig = visualize_rotate(data)
    fig.update_traces(marker=dict(size=2,
                      line=dict(width=2,
                      color='DarkSlateGrey')),
                      selector=dict(mode='markers'))
    fig.show()
```

visualize vertices in 3d space

```
#@title visualize vertices in 3d space
x,y,z = np.array(verts).T
pcshow(x,y,z)
```

Transform the data into real point cloud

As you can see in the visualization, the point cloud for vertices does not really look like a chair.

This is because we haven't utilized faces information yet. Remember, we want the point cloud for an object to be evenly distributed among the faces of the object.

In the following PointCloudData class, we will do the following:

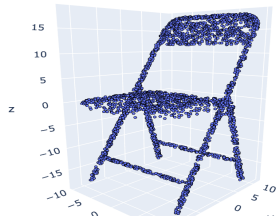
- Sample points from faces of each object
- Normalize the data
- Add rotation to the pointcloud
- Add random gaussian noise to the point cloud

```
# The class that handles reading data
class PointCloudData(Dataset):
    def __init__(self, root_dir, folder="train", sampled=False, k=1024, norm=True, rotation=True, noise=True):
      self.root_dir = root_dir
      self.sampled = sampled
      self.k = k
      folders = [dir for dir in sorted(os.listdir(root_dir)) if os.path.isdir(root_dir/dir)]
      self.classes = {folder: i for i, folder in enumerate(folders)}
      self.files = []
      self.norm = norm
      self.rotation = rotation
      self.noise = noise
      for directory in self.classes.keys():
        new_dir = root_dir/Path(directory)/folder
        for file in os.listdir(new_dir):
          if file.endswith('.off'):
            data = {}
            data['path'] = new_dir/file
            data['category'] = directory
            self.files.append(data)

    def __len__(self):
      return len(self.files)

    def sample(self, verts, faces):
      ###################################################################
      # TODO: sample k points among all faces with weights by their areas
      # i.e., we want larger triangles to have more points to be sampled in
      # the plane of triangle with larger area. The sampled points in each face
      # should be proportional to its area
      # Input: self.k: number of samples
      #        verts: List[List] contains all vertices of an object
      #        faces: List[List] contains the indices of vertices of each triangle
      # Return: numpy array of shape [k, 3]
```

```
    # hint: 1. read the following two functions: triangle_area and sample_point
    #       2. random.choices can be helpful
    ##########################################################################
    raise NotImplementedError()
    ##########################################################################

  def triangle_area(self, pt1, pt2, pt3):
    '''
    input: Coordinates of three vertices of a triangle
    output: The area of the triangle
    '''
    side_a = np.linalg.norm(pt1 - pt2)
    side_b = np.linalg.norm(pt2 - pt3)
    side_c = np.linalg.norm(pt3 - pt1)
    s = 0.5 * (side_a + side_b + side_c)
    return max(s * (s - side_a) * (s - side_b) * (s - side_c), 0)**0.5


  def sample_point(self, pt1, pt2, pt3):
    '''
    input: Coordinates of three vertices of a triangle
    output: The coordinate of a randomly sampled point in the triangle
    '''
    s, t = sorted([np.random.random(), np.random.random()])
    f = lambda i: s * pt1[i] + (t-s) * pt2[i] + (1-t) * pt3[i]
    return [f(0), f(1), f(2)]

  def __getitem__(self, idx):
    path = self.files[idx]['path']
    # print(path)
    category = self.files[idx]['category']

    if self.sampled:
      with open(path, 'r') as f:
        pointcloud = np.array(read_sampled_off(f))
    else:
      with open(path, 'r') as f:
        verts, faces = read_off(f)
        pointcloud = self.sample(verts, faces)

    if self.normalize:
      pointcloud = self.normalize(pointcloud)
    if self.rotation:
      pointcloud = self.rotate(pointcloud)
    if self.noise:
      pointcloud = self.add_noise(pointcloud)


    return {'pointcloud': torch.from_numpy(pointcloud),
            'category': self.classes[category]}

  def normalize(self, pointcloud):
    ##########################################################################
    # TODO: Given an unnormalized pointcloud, return the normalized pointclud.
    # we want the normalized_pointcloud to have the following property:
    # 1. The mean of all points in pointcloud is (0,0,0)
    # 2. All the point in the pointcloud are in the unit sphere with center (0,0,0)
    #
    # Input: pointcloud: numpy array of shape [k, 3] where k is
    #        the number of sampled points of the pointcloud
    # Return: norm_pointcloud: numpy array of shape [K, 3] which is normalized
    ##########################################################################
    raise NotImplementedError()
    ##########################################################################
    return norm_pointcloud

  def rotate(self, pointcloud):
    theta = random.random() * 2. * math.pi # rotation angle
    rot_matrix = np.array([[ math.cos(theta), -math.sin(theta),    0],
                           [ math.sin(theta),  math.cos(theta),    0],
                           [0,                              0,      1]])

    pointcloud = rot_matrix.dot(pointcloud.T).T
    return pointcloud

  def add_noise(self, pointcloud):
    noise = np.random.normal(0, 0.02, (pointcloud.shape))
    noisy_pointcloud = pointcloud + noise
    return noisy_pointcloud
```

Test your implementation

The sampling result should look like this:



```
# Testing your sample function, please visualize your sampled data
train_dataset_grade = PointCloudData(path_unsampled, folder='train', sampled=False, k=3000,
                              norm=False, rotation=False, noise=False)
with open(path_unsampled/"chair/train/chair_0001.off", 'r') as f:
  verts, faces = read_off(f)
pointcloud_grade = train_dataset_grade.sample(verts, faces)
pcshow(*pointcloud_grade.T)


# Testing normalize
torch.manual_seed(89)
correct1 = np.array([[-0.65969586, -0.61102563, -0.4375487],
                     [-0.17281342,  0.41985238,  0.22575168],
                     [ 0.57366514,  0.21148148, -0.09146313],
                     [-0.21824756, -0.01621679,  0.14690676],
                     [ 0.55209243,  0.02533382, -0.38807073],
                     [-0.07500052, -0.02942534,  0.5444241 ]])
pointcloud = torch.randn(6,3).numpy()
res1 = train_dataset_grade.normalize(pointcloud)
print('Testing normalize:')
if not np.allclose(res1, correct1, rtol=1e-03):
  print("Error")
  print("Your answer is: ", res1)
  print("The expected answer is: ", correct1)
else:
  print('passed')
```

read the data and visualize the data

```
#@title read the data and visualize the data
train_dataset_visualize = PointCloudData(path_unsampled, folder='train', sampled=False, k=3000,
                              norm=True, rotation=True, noise=False)
test_dataset_visualize = PointCloudData(path_unsampled, folder='test', sampled=False, k=3000,
                             norm=True, rotation=False, noise=False)


inv_classes = {i: cat for cat, i in train_dataset_visualize.classes.items()};
inv_classes


print('Train dataset size: ', len(train_dataset_visualize))
print('Test dataset size: ', len(test_dataset_visualize))
print('Number of classes: ', len(train_dataset_visualize.classes))
print('Sample pointcloud shape: ', train_dataset_visualize[0]['pointcloud'].size())
print('Sample pointcloud dtype: ', train_dataset_visualize[0]['pointcloud'].dtype)
print('Class: ', inv_classes[train_dataset_visualize[0]['category']])


item_visualize = train_dataset_visualize[820] # feel free to change the index to visualize different pointcloud
pointcloud = item_visualize['pointcloud']
category = item_visualize['category']
print('Class:', inv_classes[category])
pcshow(*pointcloud.T)
```

Download the pre-sampled data

During training, since sampling data from faces takes a lot of time. To save time, we have pre-sampled the data for you. We sampled 1024 points in each pointcloud as in the paper. Now we download the pre-sampled data.

```
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/c
!unzip -q pre_sample.zip;
```

```
path = Path("off_points")
trainDataset = PointCloudData(path, folder='train', sampled=True, norm=True, rotation=True, noise=True)
testDataset = PointCloudData(path, folder='test', sampled=True, norm=True, rotation=False, noise=False)


inv_classes = {i: cat for cat, i in trainDataset.classes.items()};
inv_classes


print('Train dataset size: ', len(trainDataset))
print('Test dataset size: ', len(testDataset))
print('Number of classes: ', len(trainDataset.classes))
print('Sample pointcloud shape: ', trainDataset[0]['pointcloud'].shape)
print('Sample pointcloud dtype: ', trainDataset[0]['pointcloud'].dtype)
print('Class: ', inv_classes[trainDataset[0]['category']])


item = trainDataset[423]
pointcloud = item['pointcloud']
category = item['category']
pcshow(*pointcloud.T)
print(inv_classes[category])


trainLoader = DataLoader(dataset=trainDataset, batch_size=32, shuffle=True)
testLoader = DataLoader(dataset=testDataset, batch_size=64)
```

Model Structure

The first transformation network is a mini-PointNet that takes raw point cloud as input and regresses to a 3 × 3 matrix. It's composed of a shared MLP(64, 128, 1024) network (with layer output sizes 64, 128, 1024) on each point, a max pooling across points and two fully connected layers with output sizes 512, 256.

Understanding Shared MLP

An important concept in the paper is Shared-MLP. Shared-MLP takes an input of shape [n, m] and returns a [n, k] matrix. The reason that it's called Shared-MLP is that it does the same as what nn.Linear does except that the matrix used to transform it is the same across the layer. In the following problem, you can understand that Shared-MLP is essentially the same as convolution with kernel size 1.

Implement the one-layer shared MLP and compare it with Conv1d to understand what Shared-MLP is and how Shared-MLP works.

- Implement the Matrix-based Shared-MLP
- Check the Matrix-based Shared-MLP has the same numbers of parameter as Conv1d
- Check the forward output of them are the same.

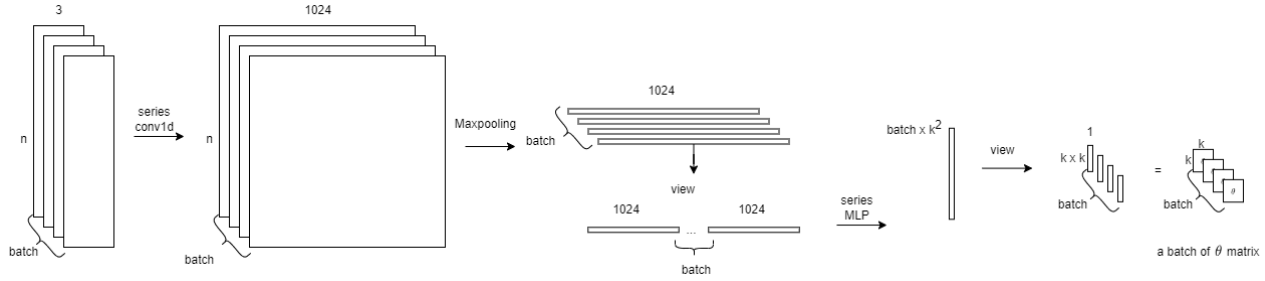
After that, you are free to use conv1d as shared-MLP in latter design

```
class SharedMLPLayer(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(SharedMLPLayer, self).__init__()
        torch.manual_seed(42)
        ################################################################
        # TODO: Implement the __init__ function
        # input: input_dim is the feature dimension of the input x
        #        output_dim is the feature dimension of the output
        #
        # hint: initialize the random params with given torch.randn.
        ################################################################
        raise NotImplementedError()
        ######################### END ####################################

    def forward(self, x):
        ################################################################
        # input: a tensor of shape [N, input_dim]
        # output a tensor of shape [N, output_dim]
        ################################################################
        raise NotImplementedError()
        ######################### END ####################################
        return out

k = 3
c = 64
sharedMLP = SharedMLPLayer(k, c)

# check the weight shape of sharedMLP
for name, param in sharedMLP.named_parameters():
    print(name, param.shape)
```

```
# An equivalent Conv1d layer implemend, and will be used to compare later
class Conv1d_layer(nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size, stride, bias=False):
        super(Conv1d_layer, self).__init__()
        self.conv1d_layer = nn.Conv1d(in_channel, out_channel, kernel_size, stride, bias=False)

        torch.manual_seed(42)
        self.conv_weights = torch.randn(in_channel, out_channel)
        conv_weights_ = self.conv_weights.t()
        conv_weights_ = torch.unsqueeze(conv_weights_, dim=-1)
        self.conv1d_layer.weight.data = conv_weights_

    def forward(self, x):
        out = self.conv1d_layer(x)
        out = torch.squeeze(out)
        out = torch.transpose(out, 0, 1)
        return out

in_channel = 3
out_channel = 64
kernel_size = 1
stride = 1

# Check the shape of conv1d_layer
conv1d_layer = Conv1d_layer(in_channel, out_channel, kernel_size, stride=1, bias=False)
for name, param in conv1d_layer.named_parameters():
    print(name, param.shape)


# Check the Matrix-based Shared-MLP has the same weights of Conv1d
print("Check the parameter number of sharedMLP and conv1d_layer is the same")
if not torch.allclose(conv1d_layer.conv_weights, sharedMLP.sharedMatrix, atol=1e-5):
    print("Error")
    print("Your sharedMLP weight = \n", sharedMLP.sharedMatrix.shape)
    print("Expected sharedMLP weight should be equal to conv1d_layer \n", conv1d_layer.conv_weights.shape)
    print("Difference = ", torch.norm(conv1d_layer.conv_weights-sharedMLP.sharedMatrix).item())
assert torch.allclose(conv1d_layer.conv_weights, sharedMLP.sharedMatrix, atol=1e-5), "conv_weights and sharedMLP.sharedMatrix
print("passed")


print("\nCheck the forward output of the sharedMLP and conv1d_layer is the same")
x0 = torch.randn(5, 3)
y0 = sharedMLP(x0)
z0 = conv1d_layer(torch.transpose(torch.unsqueeze(x0, 0), 1, 2))
if not torch.allclose(y0, z0, atol=1e-5):
    print("Error")
    print("y0 = ", y0)
    print("z0 = ", z0)
    print("Difference = ", torch.norm(y0-z0).item())
assert torch.allclose(y0, z0, atol=1e-5), "y0 and z0 are not equal"
print("passed")
```

T-Net

The T-Net module is a type of Spatial Transformer Network (STN) that learns a kxk transformation matrix for a given point cloud, which is then used to transform the point cloud to a canonical pose. It consists of two parts: a convolutional network and a fully connected network. The convolutional network maps the input point cloud to a feature space, consisting of a series of convolutional layers with batch normalization and ReLU activation. The fully connected network takes the feature space and learns the transformation matrix, consisting of fully connected layers with batch normalization and ReLU activation. Finally, the T-Net applies the transformation matrix to the input point cloud to transform it to a canonical pose.




```
class Tnet(nn.Module):
    """
    T-Net is a type of spatial transformer network (STN) that learns a kxk transformation matrix
    for a given point cloud. The matrix is then used to transform the point cloud to a canonical
    pose. It consists of two parts: a convolutional network and a fully connected network.
```

```
    The convolutional network maps the input point cloud to a feature space and the fully connected
    network learns the transformation matrix from the feature space.
    """
    def __init__(self, hidden_sizes_conv=[64, 128, 1024], hidden_sizes_fc=[512, 256], k=3):
        super().__init__()
        self.k=k
        self.hidden_sizes_conv=hidden_sizes_conv
        self.hidden_sizes_fc=hidden_sizes_fc

        self.conv = self._build_conv()
        self.fc = self._build_fc()
        self.last_fc = nn.Linear(self.hidden_sizes_fc[-1],self.k*self.k)

    def _build_conv(self):
        ######################################################################
        # TODO: Build the convolutional network that maps the input point cloud
        # to a feature space. The hidden dimension is hidden_sizes_conv
        #
        # Hint: construct a series of convolutional layers with batch
        # normalization and ReLU activation.
        #   The convolution layers follows the following structure:
        #   [conv1d]-> [Batch Norm Layer] -> [ReLU]-> [conv1d]-> ...
        #   Take the integer in hidden_size_conv for convolution1d layer
        #   size and batch_norm layer size
        ######################################################################
        layers = []
        prev_size = self.k
        for layer_id, size in enumerate(self.hidden_sizes_conv):
            raise NotImplementedError()
        ################################ END ###################################
        return nn.Sequential(*layers)

    def _build_fc(self):
        '''
        This function implements the "series mlp" part in the above graph
        The fully connected layers that this function build transforms an input
        of shape [bs, 1024] to [bs, k^2] where k is the desired size of the
        transformation matrix
        '''
        layers = []
        prev_size = self.hidden_sizes_conv[-1]
        for layer_id, size in enumerate(self.hidden_sizes_fc):
            bn = nn.BatchNorm1d(size)
            fc = nn.Linear(prev_size, size)
            layers.append(fc)
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        return nn.Sequential(*layers)


    def forward(self, input):
        ######################################################################
        # TODO: Performs the forward pass of the T-Net.
        # It first applies the convolutional network to the input point cloud
        # to obtain a feature space. Then it applies maxpool along the first dimension
        # Then, it applies the fully connected network to the feature space to
        # obtain the kxk transformation matrix. Finally, it applies the
        # transformation matrix to the input point cloud to transform it to a
        # canonical pose.
        # input: [batch, k=3, N], N is the points number in a object
        # output: [batch, k, k]
        #
        # Hint: the forward structure is as follows:
        # [ConvLayers]->[MaxPooling]->[Flatten]->[Fully Connected Layers]->[theta_Matrix + identity]
        # Remember that the identity matrix requires gradient
        ######################################################################
        # input.shape (bs,n,3)
        bs = input.size(0)
        raise NotImplementedError()
        ############################ END ###################################
        return matrix
```

Test your T-Net construction is correct.

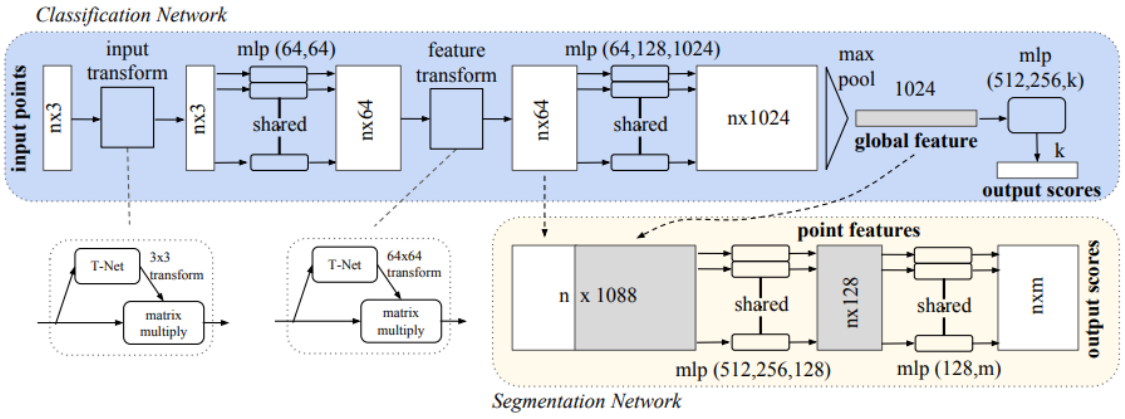
- Test the layer parameter is correct, you should follow the comment in the coding part
- Test the T-Net forward is correct.

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters())
```

```
torch.manual_seed(42)
test_t_net = Tnet()

print("Getting the T-Net parameters")
if count_parameters(test_t_net)!=803081:
    print("Error")
    print("test_t_net parameters number = ", count_parameters(test_t_net))

assert count_parameters(test_t_net)==803081
print('passed')


torch.manual_seed(42)
x1 = torch.randn(3, 3, 5)

y1 = torch.tensor([[[ 1.4712e+00,  1.1447e+00,  6.5780e-02],
         [ 2.6862e-01,  1.5355e+00, -7.9635e-01],
         [-3.1744e-01,  4.8485e-01,  1.2669e+00]],

        [[ 1.0652e+00, -2.9729e-02, -9.1289e-04],
         [-2.0753e-01,  1.6646e+00,  5.0989e-01],
         [-2.5312e-01,  7.1402e-01,  8.2575e-01]],

        [[ 1.3445e+00,  6.7090e-01, -4.4554e-01],
         [ 2.4452e-01,  1.1833e+00, -5.8614e-01],
         [-5.3094e-02, -1.3413e-01,  9.4217e-01]]])

pred_y1 = test_t_net(x1)

print("Getting T-Net out:", end = (" "))
if not torch.allclose(y1, pred_y1, rtol=1e-03, atol=1e-03):
    print("Error")
    print("Your answer is:", pred_y1)
    print("The expected answer is:", y1)

assert torch.allclose(y1, pred_y1, rtol=1e-03, atol=1e-03),  "different y_pred and y"
print("passed")
```

Transform Class

The Transform class is every thing before last MLPs in PointNet. It is a neural network architecture that uses two pairs of spatial transform net (STN) and shared MLP layers to extract global features from a point cloud data of (nx3) shape. The STN is implemented using the T-Net and computes the 3x3 transform matrix, which is then multiplied with the input point cloud to get a transformed point cloud of the same shape. The transformed point cloud is then input into the shared MLP layers along with the feature transform matrix. The output from the shared MLP layers is max pooled along the feature dimension to get a global feature vector. The output also includes the point and feature transform matrices.



Implement the Transform part

```
class Transform(nn.Module):
    def __init__(self, input_size=3, feature_size=64, sharedMLP1_layers=[64, 64], sharedMLP2_layers=[64, 128, 1024], batch_nor
        """
        Transform class is all the pipeline to get a global feature

                 ____________________                                  _______________                 ____________________
                |                    |                                |               |               |                    |
        x -->   |   input transform  | --> y (canonical point cloud) --> |  shared MLP   | --> feature -->|feature transform|
                |____________________|                                |_______________|               |____________________|
        The transform class is a neural network architecture that go throught 2 pairs of spactial transform net and shared MLP
        The STN is the T-Net that implement above, and the shared-MLP can be regarded as a one-dimensional convolutional layer

        the input x as a point cloud data of (nx3) shape first compute the 3x3 transform matrix and multiplied with the transf
```

```
        the last_activate bool is True when you want to add the last layer with activation function.
        """
        super().__init__()
        self.batch_norm = True

        self.input_transform = Tnet(k=3)
        self.feature_transform = Tnet(k=64)

        self.sharedMLP1 = self._build_sharedMLP(input_size, sharedMLP1_layers, last_activate=True)
        self.sharedMLP2 = self._build_sharedMLP(feature_size, sharedMLP2_layers, last_activate=False)

    def _build_sharedMLP(self, input_dim, sharedMLP_layers, last_activate = True):
        ######################################################################
        # TODO: Build the shared MLP layers. Take the sharedMLP_layers list as
        # sharedMLP_layers is a list of dimensions of hidden layers
        # last_activate represents whether apply activation to the last layer
        # hidden dimension in Conv1d, Batch norm
        #       The structure is [Conv1d]->[Batch Norm]->[ReLU]
        ######################################################################
        layers = []
        prev_size = input_dim
        for layer_id, size in enumerate(sharedMLP_layers):
            raise NotImplementedError()
        ######################################################################
        return nn.Sequential(*layers)

    def forward(self, input):     #input:[batch_size, 3, 1024] output:[batch_size, 1024]
        ######################################################################
        # TODO: Implement the code to multiply the transform matrix and the point
        # cloud.
        #   The transformed x should be the same shape of x [batch_size, 3, N]
        #   The transformed feature is [batchsize, 64, N]
        #   The maxpooled feature is [batch_size, 1024, 1]
        # Hint:
        # 1. Get the transform matrix [batch_size, 3, 3] by the T-Net
        # 2. Batch matrix multiply the input x and transform matrix
        # 3. Input the data into the Shared MLP
        # 4. Batch matrix multiply the feature and the feature_transform matrix
        # 5. Input the output into the Shared MLP with feature dimension
        # 6. Maxpooling along the feature dimension
        # 7. output the output data, points transform matrix, and the feature
        #       transform matrix
        ######################################################################
        raise NotImplementedError()
        ######################################################################
        return output, matrix3x3, matrix64x64


# Check the default parameter number to see the model is correct
torch.manual_seed(42)
test_transform_net = Transform()

print("Getting the parameter number of the transform net:", end = (" "))
test_tranform_net_param_num = count_parameters(test_transform_net)
if test_tranform_net_param_num!=2812105:
    print("Error")
    print("Your test_transform_net parameters number = ", count_parameters(test_transform_net))
    print("Expected answer = 2812105")
    print("Difference = ", torch.absolute(test_tranform_net_param_num!=2812105))

assert count_parameters(test_transform_net)==2812105
print("passed")

# Check the forward output is correct
torch.manual_seed(42)
x2 = torch.randn(2, 3, 5)
pred_y2, pred_mat1, pred_mat2 = test_transform_net(x2)

mat1 = torch.tensor([[[ 1.0655,  0.4169,  0.1452],
         [ 0.0240,  1.7885, -0.6011],
         [-0.5582,  0.6857,  1.0256]],

        [[ 1.5976,  0.8703, -0.4317],
         [ 0.2260,  1.2361,  0.0457],
         [ 0.0986,  0.0844,  1.0267]]])

print("Getting the correct forward output : ", end = (""))
if not torch.allclose(mat1, pred_mat1, rtol=1e-03, atol=1e-03):
    print("Error")
    print("Your pred_mat1 is \n", pred_mat1)
    print("Expected answer mat1 is \n", mat1)
    print("Difference = ", torch.norm(pred_mat1- mat1))
```

```
assert torch.allclose(mat1, pred_mat1, rtol=1e-03, atol=1e-03),  "different pred_mat1 and mat1"
print("passed")
```

PointNet Classifier

The following code defines a PyTorch module called PointNet for classifying point cloud data. The PointNet module includes a Transform class, which takes in 3D point cloud data as input and generates global features and transformation matrices. The global features are then passed through a multi-layer perceptron (MLP) to generate scores for classification. The MLP consists of linear layers, batch normalization, ReLU activation, and dropout layers. The PointNet module outputs the logsoftmax of the scores along with the 3x3 and 64x64 transformation matrices generated by the Transform class. The PointNet module can be customized with different layer configurations, batch normalization, and dropout rates.

```
class PointNet(nn.Module):
    def __init__(self, sharedMLP1_layers=[64, 64], sharedMLP2_layers=[64, 128, 1024], dropout_rate = 0.3,  classes = 10, batch
        """
        Point Net the whole neural network for the classification of point cloud data
                    _________                  ___
        input x--->|Transform|---> global feature--->|MLP|---> scores
                   |_________|                      |___|
            args:   sharedMLP1_layers is the first shared MLP in Transform class
                    sharedMLP2_layers is the second shared MLP in Transorm class
                The MLP has the structure of
                    [Linear] -> [Batch Norm] -> [ReLU] -> [Dropout] -> [Linear] -> ... -> [Linear] -> [Batch Norm] -> [ReLU] -
        """
        super().__init__()
        self.transform = Transform(input_size=3, feature_size=64, sharedMLP1_layers=sharedMLP1_layers, sharedMLP2_layers=share
        self.batch_norm = batch_norm
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, classes)
        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.dropout = nn.Dropout(dropout_rate)
        self.logsoftmax = nn.LogSoftmax(dim=1)

    def forward(self, input):
        ########################################################################
        # TODO: get the output y, 3x3 transform matrix and 64x64 transform
        # matrix from self.transform net.
        # Then, y->[fc1]->[bn1]->[relu]->[fc2]->[dropout]->[bn2]->[relu]->[fc3]->z
        # return logsoftmax(z), 3x3 transform matrix and 64x64 transform matrix
        ########################################################################
        raise NotImplementedError()
        ############################ END ###################################
        return self.logsoftmax(output), matrix3x3, matrix64x64


# Test your implementation
sharedMLP1_layers=[64, 64]
sharedMLP2_layers=[64, 128, 1024]

torch.manual_seed(42)
test_point_net = PointNet(sharedMLP1_layers, sharedMLP2_layers)

print("Getting the parameter number of the pointnet:", end = " ")
test_point_net_param_num = count_parameters(test_point_net)
if test_point_net_param_num!=3472339:
    print("Error")
    print("Your test_transform_net parameters number = ", count_parameters(test_point_net))
    print("The expected answer is 3472339")
    print("Difference = ", torch.absolute(torch.tensor(test_point_net_param_num-3472339)))

assert count_parameters(test_point_net)==3472339
print("passed")

torch.manual_seed(42)
x3 = torch.randn(3, 3, 5)
w, pred_mat3x3, pred_matfxf = test_point_net(x3)

mat3x3 = torch.tensor([[[ 1.4712e+00,  1.1447e+00,  6.5780e-02],
         [ 2.6862e-01,  1.5355e+00, -7.9635e-01],
         [-3.1744e-01,  4.8485e-01,  1.2669e+00]],

        [[ 1.0652e+00, -2.9729e-02, -9.1289e-04],
         [-2.0753e-01,  1.6646e+00,  5.0989e-01],
         [-2.5312e-01,  7.1402e-01,  8.2575e-01]],

        [[ 1.3445e+00,  6.7090e-01, -4.4554e-01],
         [ 2.4452e-01,  1.1833e+00, -5.8614e-01],
         [-5.3094e-02, -1.3413e-01,  9.4217e-01]]])
```

```
        [ 5.5091e-02,  1.5113e-01,  9.4217e-01]]])
print("Check the output correctness by mat3x3", end = (" "))
if not torch.allclose(mat3x3, pred_mat3x3, rtol=1e-03, atol=1e-03):
    print("Error")
    print("Your pred_mat3x3 is \n", pred_mat1)
    print("The answer mat3x3 is \n", mat1)
    print("Difference = ", torch.norm(pred_mat3x3- mat3x3))
assert torch.allclose(pred_mat3x3, mat3x3, rtol=1e-03, atol=1e-03),  "different pred_mat1 and mat1"
print("passed")
```

Train the model

Loss formula

One last thing before training the model is defining the loss of this network. Our loss function is formed by two part: softmax classification loss and regularization term:

1. Softmax Classification Loss: Use cross entropy loss to calculate softmax classification loss, the formula is following:

   $H(p, q) = -\sum_{x \in X} p(x) \cdot log\ q(x)$

2. Regularization Term: Constrain the feature transformation matrix learned by T-net to be close to orthogonal matrix, so the regularization term formula is following:

   $\|I - AA^T\|_F^2$

By combining the above two parts, our overall loss function is:

$L = H(p, q) + \alpha \times (\|I - A_1 A_1^T\|_F^2 + \|I - A_2 A_2^T\|_F^2)$

$Note: A_1, A_2$ is 3-dimensional and 64-dimensional feature transform matrix formed by T-net

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)


class ClsExperiment:
    def __init__(self, train_data, val_data, model: nn.Module,
                 lr: float, save=True, alpha=None):
        self.train_data = train_data
        self.val_data = val_data
        self.optimizer = torch.optim.Adam(model.parameters(), lr=lr)
        self.save=save
        if alpha==None:
          self.alpha=0.0001
        else:
          self.alpha=alpha
        self.model = model.cuda()
        self.loss=[]
        self.acc=[]

    def train(self, epochs):
        epoch_iterator = trange(epochs)
        # validation
        if self.val_data:
            self.evaluate("Initial ")
        for epoch in epoch_iterator:
            self.model.train()
            running_loss = 0.0
            data_iterator = tqdm(self.train_data)
            for i, data in enumerate(data_iterator, 0):
                inputs, labels = data['pointcloud'].to(device).float(), data['category'].to(device)
                self.optimizer.zero_grad()

                outputs, m3x3, m64x64 = self.model(inputs.transpose(1,2))

                loss = self.getloss(outputs, labels, m3x3, m64x64, self.alpha)
                loss.backward()
                self.optimizer.step()

                # print statistics
                running_loss += loss.item()
                if i % 10 == 9:    # print every 10 mini-batches
                    self.loss.append(running_loss / 10)
                    data_iterator.set_postfix(loss=running_loss / 10)
                    running_loss = 0.0
            # validation
            if self.val_data:
                self.evaluate("Epoch:{} ".format(epoch+1))
            if self.save:
```

```
            torch.save(self.model.state_dict(), "save_"+str(epoch)+".pth")

    def evaluate(self, ttype):
        self.model.eval()
        correct = total = 0
        with torch.no_grad():
            for data in self.val_data:
                inputs, labels = data['pointcloud'].to(device).float(), data['category'].to(device)
                inputs = inputs.transpose(1,2)
                outputs, __, __ = self.model(inputs)
                num_correct = self.get_num_correct(outputs.data, labels)
                total += labels.size(0)
                correct += num_correct
            val_acc = 100. * correct / total
            self.acc.append(val_acc)
            print(ttype+'Test accuracy: %d %%' % val_acc)
            #epoch_iterator.set_postfix(val_acc=val_acc)

    def get_num_correct(self, outputs, labels):
          ##########################################################################
          # TODO: given outputs of the model, and ground truth labels,
          # get the number of correct predictions among bs objects
          # Inputs: outputs: [bs, 10]
          #         labels: [bs]
          # Return: the number of correct predictions among bs objects
          # hint: torch.max can be helpful
          ##########################################################################
          raise NotImplementedError()
          ##########################################################################
          return num_correct.item()

    def getloss(self, outputs, labels, m3x3, m64x64, alpha):
        ###############################################################################
        # TODO: Get the loss of the model. The loss of the model consists of three parts:
        # 1. Cross entropy loss between outputs and labels
        # 2. The norm of two T-net transform matrices: I - m3x3^T @ m3x3 and
        #                                              I - m64x64^T @ m64x64
        # 3. The final overall loss is corssEctropy(y, y_hat) + alpha*(|I-m3x3^T@ m3x3|_2 + |I-m64x64^T@ m64x64|_2) / batch_si
        #
        # Inputs: outputs: [bs, 10]
        #         labels: [bs]
        #         m3x3: [bs, 3, 3]
        #         m64x64: m3x3: [bs, 64, 64]
        # Return: the loss
        #
        #
        # hint: 1. Remember to add an extra parameter (requires_grad=True) when you create
        # a new matrix using pytorch if the created matrix is interacted with other matrices
        # in the model
        ###############################################################################
        raise NotImplementedError()
        ###############################################################################
        return loss

    def plt_loss(self):
      x1 = range(0, len(self.loss))
      y1 = self.loss
      plt.plot(x1, y1, 'o-')
      plt.title('Train loss vs. epoches')
      plt.ylabel('Train loss')
      plt.show()

    def plt_accuracy(self):
      x1 = range(0, len(self.acc))
      y1 = self.acc
      plt.plot(x1, y1, 'o-')
      plt.title('Test accuracy vs. epoches')
      plt.ylabel('Tes accuracy')
      plt.show()

    def get_pred_label(self):
      final_preds = []
      final_labels = []
      with torch.no_grad():
          for data in self.val_data:
              inputs, labels = data['pointcloud'].to(device).float(), data['category'].to(device)
              outputs, __, __ = self.model(inputs.transpose(1,2))
              _, preds = torch.max(outputs.data, 1)
              final_preds += list(preds.cpu().numpy())
              final_labels += list(labels.cpu().numpy())
      return final_preds, final_labels
```

```
    def plot_confusion_matrix_(self, cm, classes, normalize, title='Confusion matrix', cmap=plt.cm.Blues):
      if normalize:
          cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
          print("Normalized confusion matrix")
      else:
          print('Confusion matrix, without normalization')

      plt.figure(figsize=(8,8))
      plt.imshow(cm, interpolation='nearest', cmap=cmap)
      plt.title(title)
      plt.colorbar()
      tick_marks = np.arange(len(classes))
      plt.xticks(tick_marks, classes, rotation=45)
      plt.yticks(tick_marks, classes)

      fmt = '.2f' if normalize else 'd'
      thresh = cm.max() / 2.
      for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
          plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")

      plt.tight_layout()
      plt.ylabel('True label')
      plt.xlabel('Predicted label')

    def plt_confusion_matrix(self, normalize=True):
      preds, labels = self.get_pred_label()
      cm = confusion_matrix(labels, preds)
      self.plot_confusion_matrix_(cm, list(classes.keys()), normalize)
```

Test your implementation

```
torch.manual_seed(99)
pointnet_test = PointNet()
clsexp_test = ClsExperiment(trainLoader, testLoader, pointnet_test, lr=0.001, save=False)
# Testing get_num_correct
correct1 = torch.tensor([3])
outputs = torch.randn(8, 4)
labels = torch.tensor([0,2,1,1,3,2,3,0])
res1 = torch.tensor(clsexp_test.get_num_correct(outputs, labels))
print('Testing get_num_correct:')
if not torch.allclose(res1, correct1, rtol=1e-03):
  print("Error")
  print("Your answer is: ", res1)
  print("The expected answer is: ", correct1)
else:
  print('passed')

# Testing get_loss
m3x3 = torch.randn(8, 3,3)
m64x64 = torch.randn(8, 64,64)
res2 = clsexp_test.getloss(outputs, labels, m3x3, m64x64, alpha = 0.0001)
correct2 = torch.tensor(1.4213)
print('Testing getloss:')
if not torch.allclose(res2, correct2, rtol=1e-03):
  print("Error")
  print("Your answer is: ", res2)
  print("The expected answer is: ", correct2)
else:
  print('passed')
```

Fine tuning the hyper-parameters to reach a testing accuray of 75 within 15 epochs

```
lr=1
alpha=100
sharedMLP1_layers=[64, 64]
sharedMLP2_layers=[64, 128, 1024]
dropout_rate=0.8


pointnet = PointNet(sharedMLP1_layers, sharedMLP2_layers, dropout_rate)
pointnet.to(device)
clsexp = ClsExperiment(trainLoader, testLoader, pointnet, lr, False, alpha)
clsexp.train(epochs=15)
```

Visualize the pointcloud after applying T-net transformation matrix

```
data=trainDataset[55]
inputs, labels = data['pointcloud'].to(device).float(), data['category']
```

```
__, matrix, _=pointnet(inputs.view(1, 1024, 3).transpose(1,2))
index=random.randint(0, len(data['pointcloud']))
inputs = inputs.view(1024, 3).cpu()
print("Before applying transform matrix visualization:\n")
pcshow(*inputs.T)
print("After applying transform matrix visualization:\n")
after_transform=inputs@matrix[0].cpu()
after_transform=after_transform.detach().numpy()
pcshow(*after_transform.T)
print(matrix[0].T@matrix[0])
```

```
clsexp.plt_loss()
```

```
clsexp.plt_accuracy()
```

```
clsexp.plt_confusion_matrix()
```

3.2.7 Segmentation.ipynb Code (without solution)

PointNet Segmentation part

This is a homework based on the paper [PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation]

Dataset

Download the dataset by wget

```
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookie
!unzip -q expert_verified.zip
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookie
!unzip -q points.zip
```

```
from __future__ import print_function, division
import os
import torch
import pandas as pd
from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
from torch.utils.data.dataset import random_split
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import random
from tqdm.notebook import trange, tqdm

import plotly.graph_objects as go
import plotly.express as px
```

Load the dataset

```
def read_pts(file):
    verts = np.genfromtxt(file)
    #return utils.cent_norm(verts)
    return verts

def read_seg(file):
    verts = np.genfromtxt(file, dtype= (int))
    return verts

def sample_2000(pts, pts_cat):
    res1 = np.concatenate((pts,np.reshape(pts_cat, (pts_cat.shape[0], 1))), axis= 1)
    res = np.asarray(random.choices(res1, weights=None, cum_weights=None, k=2000))
    images = res[:, 0:3]
    categories = res[:, 3]
    categories-=np.ones(categories.shape)
    return images, categories
```

```
class Data(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, root_dir, valid=False, transform=None):

        self.root_dir = root_dir
        self.files = []
        self.valid=valid

        newdir = root_dir + '/expert_verified/points_label/'

        for file in os.listdir(newdir):
            if file.find("(")!=-1:
                continue
            o = {}
            o['category'] = newdir + file
            o['img_path'] = root_dir + '/points/'+ file.replace('.seg', '.pts')
            self.files.append(o)


    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        img_path = self.files[idx]['img_path']
        category = self.files[idx]['category']
        with open(img_path, 'r') as f:
            image1 = read_pts(f)
        with open(category, 'r') as f:
            category1 = read_seg(f)
        image2, category2 = sample_2000(image1, category1)
        if not self.valid:
            theta = random.random()*360

        return {'image': np.array(image2, dtype="float32"), 'category': category2.astype(int)}



root_dir="/content"
dset = Data(root_dir , transform=None)
train_num = int(len(dset) * 0.95)
val_num = int(len(dset) *0.05)
if int(len(dset)) - train_num -  val_num >0 :
    train_num = train_num + 1
elif int(len(dset)) - train_num -  val_num < 0:
    train_num = train_num -1
train_dataset, val_dataset = random_split(dset, [train_num, val_num])
val_dataset.valid=True

print('######### Dataset class created #########')
print('Number of images: ', len(dset))
print('Sample image shape: ', dset[0]['image'].shape)

Seg_train_loader = DataLoader(dataset=train_dataset, batch_size=32)
Seg_val_loader = DataLoader(dataset=val_dataset, batch_size=32)
```

Visualize item in dataset

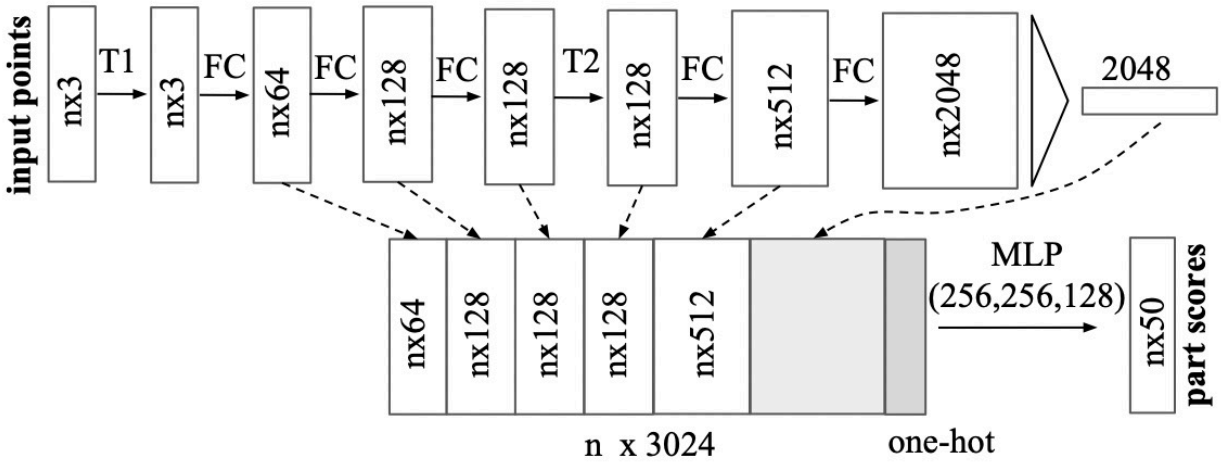
```
def visualize_rotate(data):
    x_eye, y_eye, z_eye = 1.25, 1.25, 0.8
    frames=[]

    def rotate_z(x, y, z, theta):
        w = x+1j*y
        return np.real(np.exp(1j*theta)*w), np.imag(np.exp(1j*theta)*w), z

    for t in np.arange(0, 10.26, 0.1):
        xe, ye, ze = rotate_z(x_eye, y_eye, z_eye, -t)
        frames.append(dict(layout=dict(scene=dict(camera=dict(eye=dict(x=xe, y=ye, z=ze))))))
    fig = go.Figure(data=data,
                    layout=go.Layout(
                        updatemenus=[dict(type='buttons',
                                    showactive=False,
                                    y=1,
                                    x=0.8,
                                    xanchor='left',
                                    yanchor='bottom',
                                    pad=dict(t=45, r=10),
                                    buttons=[dict(label='Play',
                                                  method='animate',
                                                  args=[None, dict(frame=dict(duration=50, redraw=True),
                                                                   transition=dict(duration=0),
                                                                   fromcurrent=True,
                                                                   mode='immediate'
                                                                   )]
                                                  )
                                             ]
                                    )
                                ]
                    ),
                    frames=frames
            )

    return fig


def segpcshow(xs,ys,zs, category):
    data=[go.Scatter3d(x=xs, y=ys, z=zs,
                                 mode='markers')]
    fig = visualize_rotate(data)
    fig.update_traces(marker=dict(size=2,
                      line=dict(width=2,
                      color=category)),
                      selector=dict(mode='markers'))
    fig.show()


item_visualize = train_dataset[1000] # feel free to change the index to visualize different pointcloud
pointcloud = item_visualize['image']
category = item_visualize['category']
segpcshow(*pointcloud.T, category)


def _set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)

def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
```

Model Sturcture

Segmentation Model

Refer the Supplementary C in [PointNet paper]

The output from the classification global transform section forms a vector [f1 , . . . , fK], which is a global signature of the input set. We can easily train a SVM or multi-layer perceptron classifier on the shape global features for classification. However, point segmentation requires a combination of local and global knowledge. We implement this by using U-Net-Like structure, concatenate the global and local feature together. The feature in every layer is stacked together to form point feature for every point. The architecture can be seen in the following figure. The architecture can be seen in the following figure.



To get started with implementing PointNet segmentation, we need to change `Transform` to `SegTransform`, `PointNet` to `PointNetSeg` in order to correspond with the paper implementation.

To accelerate training, we limit the segmentation dataset to only one category(plane). So we simplify the segmentation model by not concentrating the one-hot vector to point feature. As a result, the resulting point feature will only have 3008 dimensions.

HINT: use `Conv1d` to do shared MLP

▾ T-net part is same as T-net in PointNet classification

```
class Tnet(nn.Module):
    """
    T-Net is a type of spatial transformer network (STN) that learns a kxk transformation matrix
    for a given point cloud. The matrix is then used to transform the point cloud to a canonical
    pose. It consists of two parts: a convolutional network and a fully connected network.
    The convolutional network maps the input point cloud to a feature space and the fully connected
    network learns the transformation matrix from the feature space.
    """
    def __init__(self, hidden_sizes_conv=[64, 128, 1024], hidden_sizes_fc=[512, 256], k=3):
        super().__init__()
        self.k=k
        self.hidden_sizes_conv=hidden_sizes_conv
        self.hidden_sizes_fc=hidden_sizes_fc
        self.fc1 = nn.Linear(hidden_sizes_fc[-1],k*k)
        self.conv = self._build_conv()
        self.fc = self._build_fc()

    def _build_conv(self):
        layers = []
        prev_size = self.k
        for layer_id, size in enumerate(self.hidden_sizes_conv):
            bn = nn.BatchNorm1d(size)
            conv = nn.Conv1d(prev_size, size,1)  #share mlp can be implemented by three 1*1 convolution filter
            layers.append(conv)
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        return nn.Sequential(*layers)

    def _build_fc(self):

        layers = []
        prev_size = self.hidden_sizes_conv[-1]
        for layer_id, size in enumerate(self.hidden_sizes_fc):
            bn = nn.BatchNorm1d(size)
            fc = nn.Linear(prev_size, size)
            layers.append(fc)
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        return nn.Sequential(*layers)


    def forward(self, input):

        # input.shape (bs,n,3)
        bs = input.size(0)

        xb = self.conv(input)   #3->64->128->1024
        #torch.nn.MaxPool1d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
        #bs*n*1024->bs*1024*1
        # print(xb.shape)
        pool = nn.MaxPool1d(xb.size(-1))(xb)
        # print(pool.shape)

        flat = nn.Flatten(1)(pool)   #flatten the tensor bs*1*1024->bs*1024
        #fully connected layer
        #1024->512->256
        xb = self.fc(flat)

        #initialize with identity matrix bs*k*k
        init = torch.eye(self.k, requires_grad=True).repeat(bs,1,1)
        if xb.is_cuda:
          init=init.cuda()
        #b*256->b*(k*k)->b*k*k + init
        matrix = self.fc1(xb).view(-1,self.k,self.k) + init
        #input b*n*k
        return matrix
```

SegTransform, PointNetSeg part

```
class SegTransform(nn.Module):
    """
    SegTransform is a class transform point cloud into point feature. When the data go through
    the network, the features generated by every layer in the network stack together to form the
    point features.
    """
    def __init__(self):
        super().__init__()
        self.input_transform = Tnet(k=3)
```

```
        self.feature_transform = Tnet(k=128)
        self.fc1 = nn.Conv1d(3,64,1)
        self.fc2 = nn.Conv1d(64,128,1)
        self.fc3 = nn.Conv1d(128,128,1)
        self.fc4 = nn.Conv1d(128,512,1)
        self.fc5 = nn.Conv1d(512,2048,1)


        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(128)
        self.bn4 = nn.BatchNorm1d(512)
        self.bn5 = nn.BatchNorm1d(2048)

    def forward(self, input):
        """
        input: bs*3*n tensor
        output:
          outs: List of tensor, contain every output of the fc layer
          matrix3x3: bs*3*3 matrix output by t-net
          matrix128x128: bs*128*128 matrix output by t-net
        """
        #
        n_pts = input.size()[2]
        outs = []
        matrix3x3 = self.input_transform(input)
        xb = torch.bmm(torch.transpose(input,1,2), matrix3x3).transpose(1,2)
        out1 = F.relu(self.bn1(self.fc1(xb)))
        outs.append(out1)
        ########################################################################
        # TODO: Performs the forward pass of the SegTransform.
        # It is used to generate the point feature that is then feeded into the final MLP part.

        # Hint: Make sure adding batch normarlization layer and relu after every fc layer.
        # Hint: The final return output should be like [out1, out2, out3...].
        # Hint: In order to multiply the t-net output with the input, you need to adjust the shape of the input using torch.transpo
        # Hint: The global feature (output after maxpool) is repeated n times to concatenate to every local feature.
        #       You may find torch.repeat and torch.transpose useful here.
        ########################################################################
        #bs*64*n -> bs*128*n
        out2 = ...
        outs.append(out2)
        out3 = ...
        # 128->128
        outs.append(out3)
        #bs*128*128
        matrix128x128 = ...
        #bs*n*128 * bs*128*128 -> bs*n*128 -> bs*128*n
        out4 = ...
        outs.append(out4)
        # 128->512
        out5 = ...
        outs.append(out5)
        #512->2048
        xb = ...
        #bs*2048*n -> bs*2048*1
        xb = ...
        #bs*2048->n*bs*2048->2048*bs*n->bs*2048*n
        out6 = ...
        outs.append(out6)
        ########################################################################

        return outs, matrix3x3, matrix128x128


class PointNetSeg(nn.Module):
    """
    PointNetSeg is a class transform point feature into scores of each point
    In this dataset the category number is 4
    """
    def __init__(self, hidden_sizes_fc=[256, 256, 128, 4]):
        super().__init__()
        self.transform = SegTransform()
        self.hidden_sizes_fc=hidden_sizes_fc
        self.fc = self._build_fc()
        self.logsoftmax = nn.LogSoftmax(dim=1)

    def _build_fc(self):
        layers = []
        prev_size = 3008
        for layer_id, size in enumerate(self.hidden_sizes_fc):
            bn = nn.BatchNorm1d(size)
            fc = nn.Conv1d(prev_size, size, 1)
```

```
            layers.append(fc)
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        return nn.Sequential(*layers)

    def forward(self, input):
        """
        input: bs*3*n tensor
        output:
          out: logsoftmax score of each point, of shape bs*category_number*n
          matrix3x3: bs*3*3 matrix output by t-net
          matrix128x128: bs*128*128 matrix output by t-net
        """
        inputs, matrix3x3, matrix128x128 = self.transform(input)
        #bs*3008*n
        stack = torch.cat(inputs,1)
        output = self.fc(stack)

        return self.logsoftmax(output), matrix3x3, matrix128x128
```

Test SegTransform, PointNetSeg

```
_set_seed(2017)
model = SegTransform()

if count_parameters(model)!=6970057:
    print("Error")
    print("test_t_net parameters number = ", count_parameters(model))

assert count_parameters(model)==6970057

x1 = torch.randn(3, 3, 5)

y1=torch.tensor([0.1483, 0.0000, 1.2474, 0.6692, 0.8206, 0.0000, 0.9475, 0.0000, 0.0000, 0.4241])

pred_y1, _, _ = model(x1)
assert torch.allclose(y1, pred_y1[0].view(-1)[35:45], rtol=1e-03, atol=1e-03),  "different y_pred and y"
print("SegTransform test pass!")


_set_seed(2017)
model = PointNetSeg()

if count_parameters(model)!=7840853:
    print("Error")
    print("test_t_net parameters number = ", count_parameters(model))

assert count_parameters(model)==7840853

x1 = torch.randn(3, 3, 5)

y1=torch.tensor([-1.8602, -1.3863, -1.5023, -2.0664, -2.1121, -1.3051, -2.0492, -0.6164, -1.7983, -1.9253])

pred_y1, _, _ = model(x1)
assert torch.allclose(y1, pred_y1.view(-1)[35:45], rtol=1e-03, atol=1e-03),  "different y_pred and y"
print("PointNetSeg test pass!")
```

Training loop for segmentation

```
class ExperimentSeg:
    def __init__(self, train_data, val_data, model: nn.Module,
                 lr: float, save=True):
        self.train_data = train_data
        self.val_data = val_data
        self.model = model.cuda()
        self.optimizer = torch.optim.Adam(pointnet.parameters(), lr=lr)
        self.save=save
        self.loss=[]
        self.acc=[]

    def train(self, epochs):
        epoch_iterator = trange(epochs)
        # validation
        if self.val_data:
            self.evaluate("Initial ")
        for epoch in epoch_iterator:
            self.model.eval()
```

```
            correct = total = 0
            self.model.train()
            running_loss = 0.0
            data_iterator = tqdm(self.train_data)
            for i, data in enumerate(data_iterator, 0):
                inputs, labels = data['image'].to(device), data['category'].to(device)
                self.optimizer.zero_grad()
                outputs, m3x3, m64x64 = self.model(inputs.transpose(1,2))

                loss = self.getloss(outputs, labels, m3x3, m64x64)
                loss.backward()
                self.optimizer.step()

                # print statistics
                running_loss += loss.item()
                if i % 10 == 9:    # print every 10 mini-batches
                    self.loss.append(running_loss / 10)
                    data_iterator.set_postfix(loss=running_loss / 10)
                    running_loss = 0.0

            # validation
            if self.val_data:
                self.evaluate("Epoch:{} ".format(epoch+1))

            # save the model
            if self.save:
                torch.save(self.model.state_dict(), "save_"+str(epoch)+".pth")

    def evaluate(self, ttype):
        self.model.eval()
        correct = total = 0
        with torch.no_grad():
            for data in self.val_data:
                inputs, labels = data['image'].to(device).float(), data['category'].to(device)
                outputs, __, __ = self.model(inputs.transpose(1,2))
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0) * labels.size(1)
                correct += (predicted == labels).sum().item()
            val_acc = 100. * correct / total
            self.acc.append(val_acc)
            print(ttype+'Valid accuracy: %d %%' % val_acc)
            #epoch_iterator.set_postfix(val_acc=val_acc)

    def getloss(self, outputs, labels, m3x3, m128x128, alpha = 0.0001):
        criterion = torch.nn.NLLLoss()
        bs=outputs.size(0)
        id3x3 = torch.eye(3, requires_grad=True).repeat(bs,1,1)
        id128x128 = torch.eye(128, requires_grad=True).repeat(bs,1,1)
        if outputs.is_cuda:
            id3x3=id3x3.cuda()
            id128x128=id128x128.cuda()
        diff3x3 = id3x3-torch.bmm(m3x3,m3x3.transpose(1,2))
        diff128x128 = id128x128-torch.bmm(m128x128,m128x128.transpose(1,2))
        return criterion(outputs, labels) + alpha * (torch.norm(diff3x3)+torch.norm(diff128x128)) / float(bs)

    def plt_loss(self):
      x1 = range(0, len(self.loss))
      y1 = self.loss
      plt.plot(x1, y1, 'o-')
      plt.title('Test loss vs. epoches')
      plt.ylabel('Test loss')
      plt.show()

    def plt_accuracy(self):
      x1 = range(0, len(self.acc))
      y1 = self.acc
      plt.plot(x1, y1, 'o-')
      plt.title('Validation accuracy vs. epoches')
      plt.ylabel('Validation accuracy')
      plt.show()

    def visualize_pred_label(self):
      for data in self.val_data:
          inputs, labels = data['image'].to(device), data['category'].to(device)
          outputs, __, __ = self.model(inputs.transpose(1,2))
          _, predicted = torch.max(outputs.data, 1)
          index=random.randint(0, len(data['image']))
          inputs, labels, predicted = inputs.cpu(), labels.cpu(), predicted.cpu()
          print("True Label visualization:\n")
          segpcshow(*inputs[index].T, labels[index])
          print("Predicted Label visualization:\n")
          segpcshow(*inputs[index].T, predicted[index])
          break
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
pointnet = PointNetSeg()
pointnet.to(device)
segexp = ExperimentSeg(Seg_train_loader, Seg_val_loader, pointnet, 0.001, save=False)
segexp.train(5)
```

Visualize the result

```
segexp.plt_loss()
```

```
segexp.plt_accuracy()
```

```
segexp.visualize_pred_label()
```

3.3 Homework with Solution

3.3.1 Introduction

PointNet is a deep learning model for point cloud processing, which is a type of 3D data representation commonly used in computer vision and robotics applications. Point clouds are sets of 3D points that represent the shape and structure of objects or scenes in the form of unordered point sets.

PointNet was introduced by Charles R. Qi et al. in a 2017 paper titled "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." It is a neural network architecture that directly takes raw point clouds as input and processes them without any pre-processing or feature extraction steps. PointNet is capable of learning meaningful features from point clouds, such as local and global geometric information, and can be used for tasks such as 3D object classification, semantic segmentation, and point cloud generation.

The PointNet architecture uses multi-layer perceptrons (MLPs) and max pooling operations to process individual points in the point cloud, as well as symmetric functions to aggregate the features of all points into a global feature vector that captures the overall structure of the point cloud. PointNet can be extended with additional modules, such as PointNet++ for hierarchical feature learning or PointNet-based convolutional neural networks (CNNs) for local feature extraction.

PointNet has been widely used in various applications, such as autonomous driving, robotics, virtual reality, and augmented reality, due to its ability to process raw point cloud data directly and learn meaningful features from unstructured 3D data.

The structure of the PointNet is shown in Figure 1. Please follow the instructions in this notebook. You will build and train PointNet on Point Cloud classification and segmentation dataset.

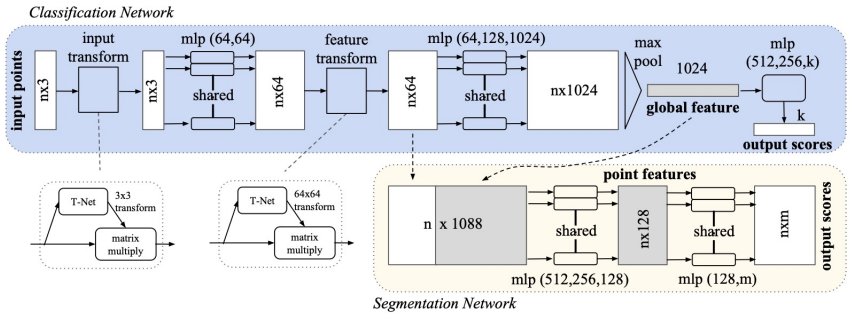


Figure 5: PointNet

3.3.2 Analysis Problem

A point cloud is a discrete *set* of data points in space. Because of this set-valued nature of point clouds, concepts from graph neural nets are often relevant in their processing.
For the first problem, we deal with 2D point cloud to get start. In Figure 1, we consider a simple network to process a 2d point cloud $X = \{x_i\}_{i=1}^{n} \in \mathbb{R}^{n\times 2}$, where n is the number of points. The original features of each point are its horizontal and vertical coordinates.

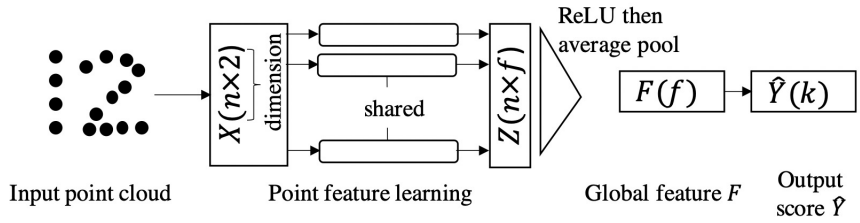


Figure 6: 2d point cloud processing network.

For example, an input point cloud with ground-truth of digit 1 could be represented as $\begin{bmatrix} 0 & 4 \\ 0 & 3 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$ where each row is a different point in the point cloud.
(a) The point feature learning module in Figure 1 learns f-dimensional features for each point separately. Specifically, it learns (shared) weights $W_1 \in \mathbb{R}^{2\times f}$ to get hidden layer outputs $Z = XW_1$. We then apply the nonlinear activation function element-wise and then use average pooling to yield the f -dimensional global feature vector $F \in \mathbb{R}^f$. Suppose we swap the first two points of the input point cloud X, i.e. $x_1, x_2, ..., x_n$ to $x_2, x_1, ..., x_n$. Show that the global feature F will not change.
Note: In reality, the network here is permutation invariant, as changing the ordering of the n input points in X will not affect the global feature F .
Solution: Expand X row-wise, derive X_1:

$$Z = XW_1 = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} W_1 = \begin{bmatrix} x_1W_1 \\ x_2W_1 \\ ... \\ x_nW_1 \end{bmatrix} \tag{1}$$

The global feature F is the average all row vectors of Z:

$$F = \frac{1}{n}\sum_{i=1}^{n} ReLU(x_iW_1) \tag{2}$$

We can see that swapping the order of x_1, x_2 will not affect F . *Note*: Actually, permuting the order of points $x_1, x_2, ..., x_n$ will not change F as it is a sum of all input point coordinates transformed by a weight matrix. Additionally, ReLU is invariant to the permutation of the order of input points.
(b) We know that we can measure the vector distance by the norm, but how can we measure distance between data out-of-order? The Hausdorff distance is the longest distance you can be

forced to travel by an adversary who chooses a point in one of the two sets, from where you then must travel to the other set.
The Hausdorff distance, denoted by $d_H(A, B)$, between two non-empty subsets A and B of a metric space (X, d) is defined as:

$$d_H(A, B) = \max\{\sup_{a\in A} \inf_{b\in B} d(a, b), \sup_{b\in B} \inf_{a\in A} d(a, b)\}$$

Intuitively, the Hausdorff distance measures the greatest distance from any point in A to its closest point in B, and vice versa. In other words, it quantifies how far apart the two sets are, taking into account the distances between all pairs of points in the sets.
For instance, in cloud points context, an object, say a chair, can be seen as a point set S_A in coordinate, and the other chair can be seen as another set S_B; another object, say desk, can be interpreted into S_C. Our job is to find a appropriate function(NN), that :

$$d_H(S_A, S_B) \leq d_H(f(S_A), f(S_B))$$

$$d_H(S_A, S_C) \geq d_H(f(S_A), f(S_C))$$

Formally, a toy model can reveal the idea of PointNet.

There are 2 different Gaussian distribution $X \sim \mathcal{N}(\vec{\mu}_1, \Sigma_1)$ and $Y \sim \mathcal{N}(\vec{\mu}_2, \Sigma_2)$ The 8 points

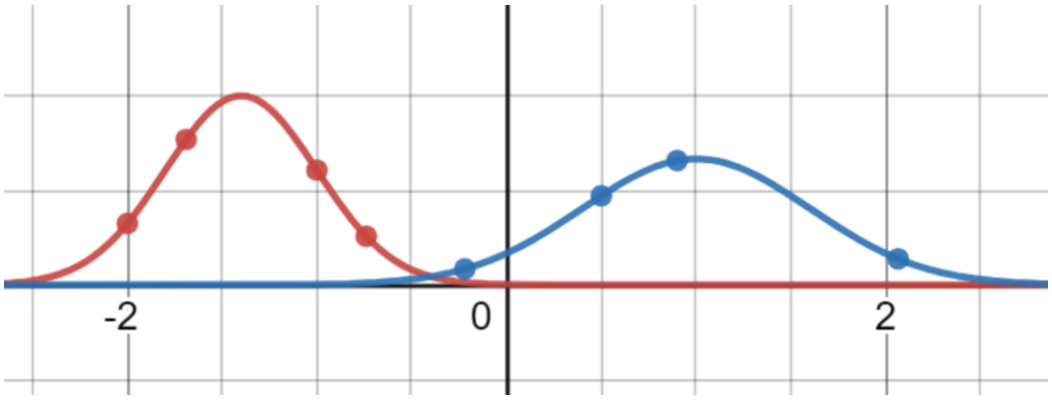


Figure 7: One dimension sample

sampled from the distribution are (from left to right)

$$X_0 = (-2, -1.69, -0.74, -1) Y_0 = (-0.22, 0.5, 0.9, 2.066)$$

Now compute the discrete Hausdorff distance between two sample.
Solution:

$$\sup_{x\in X_0} \inf_{y\in Y_0} d(x, y) = |-2-(-0.22)| = 1.78$$

$$\sup_{y\in Y_0} \inf_{x\in X_0} d(x, y) = |-1-2.066| = 3.066$$

$$d_H(X_0, Y_0) = \max(1.78, 3.066) = 3.066$$

(c)Now we have the sample rotated:

$$X_1 = RX_0$$

$R \in \mathbb{R}^{n\times n}, \det(RR^T) = 1$, in the example, $n = 4$. The toy PointNet will do the following things to make the output Hausdorff distance closer:

- Transform the two set to higher-dimension by input each point into a same function:

$$\mathbf{X}_0 H = \begin{bmatrix} h(X_0[1]) \\ h(X_0[2]) \\ h(X_0[3]) \\ h(X_0[4]) \end{bmatrix}, \mathbf{X}_1 H = \begin{bmatrix} h(X_1[1]) \\ h(X_1[2]) \\ h(X_1[3]) \\ h(X_1[4]) \end{bmatrix}$$

- Doing Maxpooling for each point among high-dimension.

$$g(\mathbf{X}_0 H) = \begin{bmatrix} \max(h(X_0[1])) \\ \max(h(X_0[2])) \\ \max(h(X_0[3])) \\ \max(h(X_0[4])) \end{bmatrix}, \quad g(\mathbf{X}_1 H) = \begin{bmatrix} \max(h(X_1[1])) \\ \max(h(X_1[2])) \\ \max(h(X_1[3])) \\ \max(h(X_1[4])) \end{bmatrix}$$

Intuitively explain why do we need to increase the dimension by $h(\cdot)$, and what is max layer $g(\cdot)$ doing?
(hint: think of the Hausdorff distance also have a max)

Solution:
By the question above, we know that the the date out-of-order should increase their dimension along the input feature dimension. After transformed into the high dimension space by input the points into the $h(\cdot)$ function, some negative points can the chance to cast to the positive side, and then the R rotation matrix affect will be eliminated. The graph below can help you illustrate:The Max function is doing selecting job among high dimension features that can lower the distance before taking Hausdorff distance.

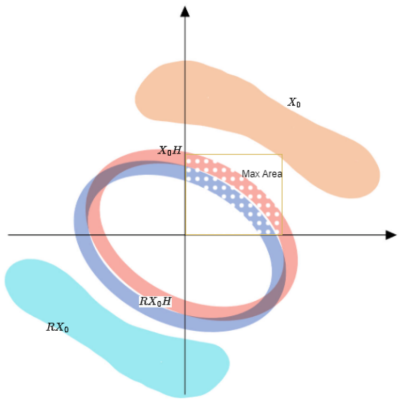


Figure 8: Toy PointNet Illistration

3.3.3 PointNet Classifier

(a) Coding solution of classifier.ipynb: colab notebook . Fine tuning the hyper-parameters to do the training

i Feel free to tune the alpha, and observe the output transform matrix of t-net and visualize the item in dataset before and after applying transform matrix, identify the similarities and differences between the input item and item applied transform matrix. What effect do you think alpha have? And when alpha is large what t-net is doing? Do you think whether alpha value can improve model performance?

Solution: Alpha is the regularizer in the loss function, larger alpha means we inject inductive bias into the model, makeing the transform matrix outputed by T-net be close to orthogonal matrix. With large alpha, T-net output an orthogonal transform matrix. Applying transform matrix, rotating the item in 3d-coordinate system. By tuning different alpha from 0.0001 to 10, we can find that alpha doesn't help model performance much.

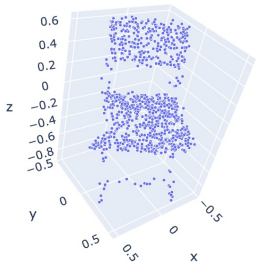


Figure 9: Chair before transform

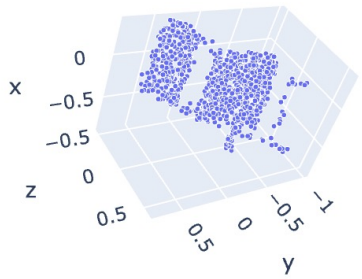


Figure 10: Chair after transform

ii Observe the confusion matrix, which objects have classification accuracy that is significantly higher than the others? Which objects have classification accuracy that is significantly higher than the others? Briefly explain why.

Solution: Chair, monitor and table have classification accuracy that is significantly higher than the others. Dresser and night stand have classification accuracy that is significantly lower than the others. This is because chair, monitor and table are significantly different from other objects in the dataset, thus they are easier to classify. While dresser and night stand are similar to each other, thus they are harder to classify.

3.3.4 Understanding the structure and data dimension

The T-Net can get a learned Transform Matrix that transforms an object to a canonical position. The structure of this model can be seen as a small Point-Net. To better understand the model, we list the layers in T-Net, and we represent the shared-MLP in conv1d() way. B is the batch size of the data, and N is the number of points in an object. Furthermore, the brackets, like(B×N) define a vector $\in \mathbb{R}^M, (M = B \times N)$ dimension, rather than a matrix A$\in \mathbb{R}^{B \times N}$

	Input	Output
covn1d[64]	B× 3× N	B× 64× N
conv1d[128]	B× 64× N	B× 128× N
conv1d[1024]	B× 128× N	B× 1024× N
MaxPool	B× 1024× N	B× 1024 × 1
Flatten	B× 1024 × 1	(B×1024)
Linear[B×9]	(B×1024)	(B×9)
View layer[B,3, 3]	(B×9)	B×3×3

3.3.5 PointNet Segmentation

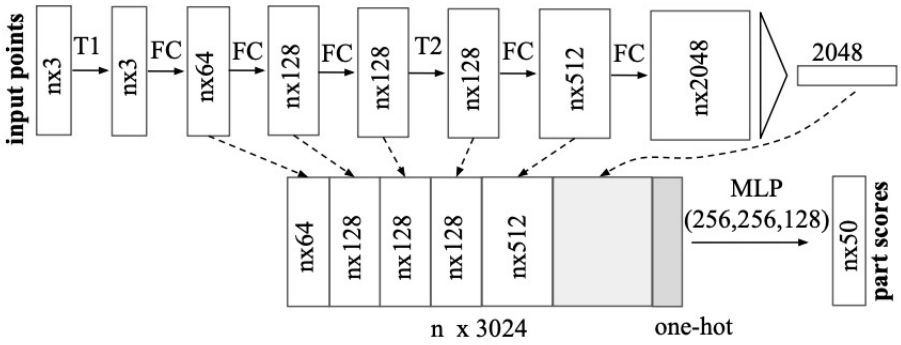


Figure 11: Segmentation PointNet

Coding solution of segmentation.ipynb: colab notebook and answering the following questions.
(a) Observe the segmentation output of the module, what part of the object has the lowest segmentation accuracy? Briefly explain why.
Solution: The tail part of the plane is inseparable from the plane body in most case. This is because they don't have clear borders, which is hard for the module to segment.
(b) Briefly describe the difference between classification PointNet and segmentation PointNet. Briefly explain why segmentation PointNet is designed this way.
Solution: Feature dimension before max pool is 2048 in Segmentation PointNet while in Classification PointNet is 1024. What's more, Segmentation PointNet concatenate every output of the mlp layer with the global feature to form the feature feeded to the final segmentation mlp. This is because segmentation task needs more information than classification task. The concatenate operation is using the U-Net way of thinking to preserve local feature while generating global feature. Higher feature dimension is to keep more information after max pool.

3.3.6 classifier_sol.ipynb Code pdf

PointNet

In this assignment, students will implement the PointNet Classification and PointNet Segmentation modules. Students can learn about point cloud data and pre-processing, implement shared MLP and Joint Alignment Network, tune the alpha value to observe the accuracy change, and write the code for the loss function to understand better what the network tries to learn.

This homework is based on the paper PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

This homework refers to the article Deep Learning on Point clouds

Overview

In this assignment, you will implement the PointNet Classification and PointNet Segmentation modules. In this homework, you will learn about point cloud data and pre-processing, you will implement shared MLP and Joint Alignment Network, tune the hyperparameters to observe the accuracy change, and write the code for the loss function to understand better what the network tries to learn.

Preparing the point cloud data

```
import math
import random
import os
import numpy as np
np.random.seed(7)
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import scipy.spatial.distance
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
from tqdm.notebook import trange, tqdm
import itertools


import plotly.graph_objects as go
import plotly.express as px


!pwd
```

```
/content
```

```
!pip install path.py;
from path import Path
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting path.py
  Downloading path.py-12.5.0-py3-none-any.whl (2.3 kB)
Collecting path
  Downloading path-16.6.0-py3-none-any.whl (26 kB)
Installing collected packages: path, path.py
Successfully installed path-16.6.0 path.py-12.5.0
```

Download the dataset

When you download a file using **wget** in Google Colab, the file is saved to the virtual machine's temporary storage associated with the current runtime session. By default, the downloaded file will be saved in the current working directory, which is /content.

The downloaded file will be lost when the runtime is disconnected.

Download the dataset

```
#@title Download the dataset
%cd /content
!wget http://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zip
!unzip -q ModelNet10.zip;
```

```
/content
--2023-05-05 01:29:15--  http://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zip
Resolving 3dvision.princeton.edu (3dvision.princeton.edu)... 128.112.136.74
Connecting to 3dvision.princeton.edu (3dvision.princeton.edu)|128.112.136.74|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zip [following]
--2023-05-05 01:29:16--  https://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zip
Connecting to 3dvision.princeton.edu (3dvision.princeton.edu)|128.112.136.74|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 473402300 (451M) [application/zip]
Saving to: 'ModelNet10.zip'

ModelNet10.zip      100%[===================>] 451.47M  13.4MB/s    in 36s

2023-05-05 01:29:53 (12.4 MB/s) - 'ModelNet10.zip' saved [473402300/473402300]
```

The dataset contains 10 classes, labeled from 0, 1, ..., 9

```
path_unsampled = Path("ModelNet10")
# read all the directories in ModelNet10
folders = [dir for dir in sorted(os.listdir(path_unsampled)) if os.path.isdir(path_unsampled/dir)]
classes = {folder: i for i, folder in enumerate(folders)};
classes
```

```
{'bathtub': 0,
 'bed': 1,
 'chair': 2,
 'desk': 3,
 'dresser': 4,
 'monitor': 5,
 'night_stand': 6,
 'sofa': 7,
 'table': 8,
 'toilet': 9}
```

Understand the data

This dataset consists of **.off** files that contain meshes represented by **vertices** and **triangular faces**

Take the data in the file 'chair/train/chair_0001.off' as an example:

- The first line is a symbol **OFF** representing the file is an .off file.
- The second line has three numbers. The first number is the number of vertices which is 2382 and the second number is the number of faces which is 2234.
- The following 2382 lines are coordinates of vertices in the space.
- The last 2234 lines are faces in the space. In each line, the first number is 3, the following 3 numbers are three vertices of a triangle.

display the first 6 lines in the file

```
#@title display the first 6 lines in the file
file_path = path_unsampled/"chair/train/chair_0001.off"
!sed -n '1,6p' "$file_path" | cat
```

```
OFF
2382 2234 0
-9.699500 0.066250 -1.575088
-9.696500 -8.837050 -18.008024
-9.699500 -9.052550 -17.887559
-9.694500 -8.621550 -18.128488
```

display the first line 3000-3005 in the file

```
#@title display the first line 3000-3005 in the file
!sed -n '3000,3005p' "$file_path" | cat
```

```
3 571 572 573
3 572 574 573
3 573 574 575
3 574 576 575
3 575 576 577
3 576 578 577
```

Read the data

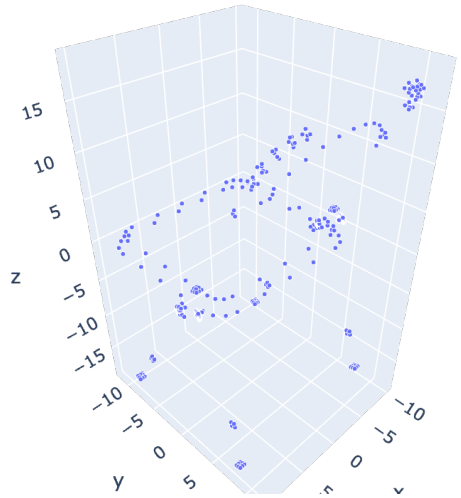
```
def read_off(file):
  '''
  return: verts: List[List]
          faces: List[List]
  '''
  if 'OFF' != file.readline().strip():
    raise('Not a valid OFF header')
  # read the first line
  n_verts, n_faces, __ = tuple([int(s) for s in file.readline().strip().split(' ')])
  # read vertices and facess
  verts = [[float(s) for s in file.readline().strip().split(' ')] for i_vert in range(n_verts)]
  faces = [[int(s) for s in file.readline().strip().split(' ')][1:] for i_face in range(n_faces)]
  return verts, faces

def read_sampled_off(file):
  if 'OFF' != file.readline().strip():
    raise('Not a valid OFF header')
  file.readline()
  file.readline()
  n_points = tuple([int(s) for s in file.readline().strip().split(' ')])[0]
  file.readline()
  points = [[float(s) for s in file.readline().strip().split(' ')] for i in range(n_points)]
  return points


# read vertices and faces of file chair_0001.off
with open(path_unsampled/"chair/train/chair_0001.off", 'r') as f:
  verts, faces = read_off(f)
```

Visualize the *data*

```
def visualize_rotate(data):
  x_eye, y_eye, z_eye = 1.25, 1.25, 0.8
  frames=[]

  def rotate_z(x, y, z, theta):
    w = x+1j*y
    return np.real(np.exp(1j*theta)*w), np.imag(np.exp(1j*theta)*w), z

  for t in np.arange(0, 10.26, 0.1):
    xe, ye, ze = rotate_z(x_eye, y_eye, z_eye, -t)
    frames.append(dict(layout=dict(scene=dict(camera=dict(eye=dict(x=xe, y=ye, z=ze))))))
  fig = go.Figure(data=data,
                  layout=go.Layout(updatemenus=[dict(type='buttons',
                                    showactive=False,
                                    y=1,
                                    x=0.8,
                                    xanchor='left',
                                    yanchor='bottom',
                                    pad=dict(t=45, r=10),
                                    buttons=[dict(label='Play',
                                            method='animate',
                                            args=[None, dict(frame=dict(duration=50, redraw=True),
                                                          transition=dict(duration=0),
                                                          fromcurrent=True,
                                                          mode='immediate')]
                                                )
                                            ]
                                       )
                                   ]
                                   ),
                  frames=frames)
  return fig


def pcshow(xs,ys,zs):
    data=[go.Scatter3d(x=xs, y=ys, z=zs,
                                  mode='markers')]
    fig = visualize_rotate(data)
    fig.update_traces(marker=dict(size=2,
                      line=dict(width=2,
                      color='DarkSlateGrey')),
                      selector=dict(mode='markers'))
    fig.show()
```

visualize vertices in 3d space

```
#@title visualize vertices in 3d space
x,y,z = np.array(verts).T
pcshow(x,y,z)
```



Transform the data into real point cloud

As you can see in the visualization, the point cloud for vertices does not really look like a chair.

This is because we haven't utilized faces information yet. Remember, we want the point cloud for an object to be evenly distributed among the faces of the object.

In the following PointCloudData class, we will do the following:

- Sample points from faces of each object
- Normalize the data
- Add rotation to the pointcloud
- Add random gaussian noise to the point cloud

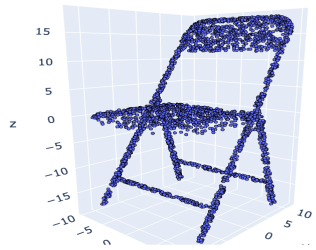
```
# The class that handles reading data
class PointCloudData(Dataset):
    def __init__(self, root_dir, folder="train", sampled=False, k=1024, norm=True, rotation=True, noise=True):
      self.root_dir = root_dir
      self.sampled = sampled
      self.k = k
      folders = [dir for dir in sorted(os.listdir(root_dir)) if os.path.isdir(root_dir/dir)]
      self.classes = {folder: i for i, folder in enumerate(folders)}
      self.files = []
      self.norm = norm
      self.rotation = rotation
      self.noise = noise
      for directory in self.classes.keys():
        new_dir = root_dir/Path(directory)/folder
        for file in os.listdir(new_dir):
          if file.endswith('.off'):
            data = {}
            data['path'] = new_dir/file
```

```
            data['category'] = directory
            self.files.append(data)

    def __len__(self):
      return len(self.files)

    def sample(self, verts, faces):
      ###########################################################################
      # TODO: sample k points among all faces with weights by their areas
      # i.e., we want larger triangles to have more points to be sampled in
      # the plane of triangle with larger area. The sampled points in each face
      # should be proportional to its area
      # Input: self.k: number of samples
      #        verts: List[List] contains all vertices of an object
      #        faces: List[List] contains the indices of vertices of each triangle
      # Return: numpy array of shape [k, 3]
      # hint: 1. read the following two functions: triangle_area and sample_point
      #       2. random.choices can be helpful
      ###########################################################################
      areas = np.zeros((len(faces)))
      verts = np.array(verts)
      for i in range(len(areas)):
        areas[i] = self.triangle_area(verts[faces[i][0]], verts[faces[i][1]], verts[faces[i][2]])
      # sample k points among all faces with weights by their areas
      sampled_faces = random.choices(faces, weights=areas, k=self.k)
      pointcloud = np.zeros((self.k, 3))
      for i in range(len(sampled_faces)):
        pointcloud[i] = self.sample_point(verts[sampled_faces[i][0]],
                                          verts[sampled_faces[i][1]],
                                          verts[sampled_faces[i][2]])
      return pointcloud
      ###########################################################################

    def triangle_area(self, pt1, pt2, pt3):
      '''
      input: Coordinates of three vertices of a triangle
      output: The area of the triangle
      '''
      side_a = np.linalg.norm(pt1 - pt2)
      side_b = np.linalg.norm(pt2 - pt3)
      side_c = np.linalg.norm(pt3 - pt1)
      s = 0.5 * (side_a + side_b + side_c)
      return max(s * (s - side_a) * (s - side_b) * (s - side_c), 0)**0.5


    def sample_point(self, pt1, pt2, pt3):
      '''
      input: Coordinates of three vertices of a triangle
      output: The coordinate of a randomly sampled point in the triangle
      '''
      s, t = sorted([np.random.random(), np.random.random()])
      f = lambda i: s * pt1[i] + (t-s) * pt2[i] + (1-t) * pt3[i]
      return [f(0), f(1), f(2)]

    def __getitem__(self, idx):
      path = self.files[idx]['path']
      # print(path)
      category = self.files[idx]['category']

      if self.sampled:
        with open(path, 'r') as f:
          pointcloud = np.array(read_sampled_off(f))
      else:
        with open(path, 'r') as f:
          verts, faces = read_off(f)
          pointcloud = self.sample(verts, faces)

      if self.normalize:
        pointcloud = self.normalize(pointcloud)
      if self.rotation:
        pointcloud = self.rotate(pointcloud)
```

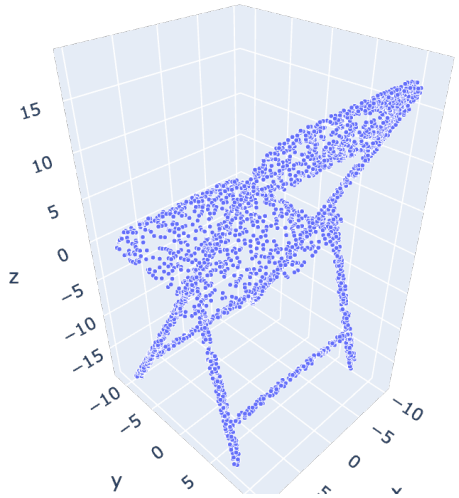
```
    if self.noise:
      pointcloud = self.add_noise(pointcloud)


    return {'pointcloud': torch.from_numpy(pointcloud),
            'category': self.classes[category]}

  def normalize(self, pointcloud):
    ############################################################################
    # TODO: Given an unnormalized pointcloud, return the normalized pointclud.
    # we want the normalized_pointcloud to have the following property:
    # 1. The mean of all points in pointcloud is (0,0,0)
    # 2. All the point in the pointcloud are in the unit sphere with center (0,0,0)
    #
    # Input: pointcloud: numpy array of shape [k, 3] where k is
    #        the number of sampled points of the pointcloud
    # Return: norm_pointcloud: numpy array of shape [K, 3] which is normalized
    ############################################################################
    pointcloud = pointcloud - np.mean(pointcloud, axis=0)
    norm_pointcloud = pointcloud / np.max(np.linalg.norm(pointcloud, axis=1))
    ############################################################################
    return norm_pointcloud

  def rotate(self, pointcloud):
    theta = random.random() * 2. * math.pi # rotation angle
    rot_matrix = np.array([[ math.cos(theta), -math.sin(theta),    0],
                           [ math.sin(theta),  math.cos(theta),    0],
                           [0,                             0,      1]])

    pointcloud = rot_matrix.dot(pointcloud.T).T
    return pointcloud

  def add_noise(self, pointcloud):
    noise = np.random.normal(0, 0.02, (pointcloud.shape))
    noisy_pointcloud = pointcloud + noise
    return noisy_pointcloud
```

Test your implementation

The sampling result should look like this:



```
# Testing your sample function, please visualize your sampled data
train_dataset_grade = PointCloudData(path_unsampled, folder='train', sampled=False, k=3000,
                             norm=False, rotation=False, noise=False)
with open(path_unsampled/"chair/train/chair_0001.off", 'r') as f:
  verts, faces = read_off(f)
pointcloud_grade = train_dataset_grade.sample(verts, faces)
pcshow(*pointcloud_grade.T)
```




```
# Testing normalize
torch.manual_seed(89)
correct1 = np.array([[-0.65969586, -0.61102563, -0.4375487],
                     [-0.17281342,  0.41985238,  0.22575168],
                     [ 0.57366514,  0.21148148, -0.09146313],
                     [-0.21824756, -0.01621679,  0.14690676],
                     [ 0.55209243,  0.02533382, -0.38807073],
                     [-0.07500052, -0.02942534,  0.5444241 ]])
pointcloud = torch.randn(6,3).numpy()
res1 = train_dataset_grade.normalize(pointcloud)
print('Testing normalize:')
if not np.allclose(res1, correct1, rtol=1e-03):
  print("Error")
  print("Your answer is: ", res1)
  print("The expected answer is: ", correct1)
else:
  print('passed')
```

```
Testing normalize:
passed
```

read the data and visualize the data

```
#@title read the data and visualize the data
train_dataset_visualize = PointCloudData(path_unsampled, folder='train', sampled=False, k=3000,
                              norm=True, rotation=True, noise=False)
test_dataset_visualize = PointCloudData(path_unsampled, folder='test', sampled=False, k=3000,
                             norm=True, rotation=False, noise=False)
```

```
inv_classes = {i: cat for cat, i in train_dataset_visualize.classes.items()};
inv_classes
```

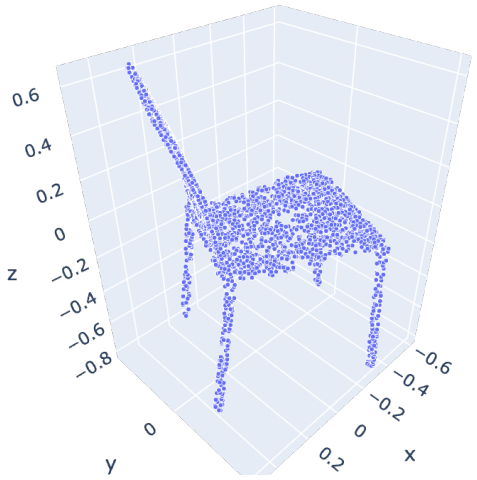
```
{0: 'bathtub',
 1: 'bed',
 2: 'chair',
 3: 'desk',
 4: 'dresser',
 5: 'monitor',
 6: 'night_stand',
 7: 'sofa',
 8: 'table',
 9: 'toilet'}
```

```
print('Train dataset size: ', len(train_dataset_visualize))
print('Test dataset size: ', len(test_dataset_visualize))
print('Number of classes: ', len(train_dataset_visualize.classes))
print('Sample pointcloud shape: ', train_dataset_visualize[0]['pointcloud'].size())
print('Sample pointcloud dtype: ', train_dataset_visualize[0]['pointcloud'].dtype)
print('Class: ', inv_classes[train_dataset_visualize[0]['category']])
```

```
Train dataset size:  3991
Test dataset size:  908
Number of classes:  10
Sample pointcloud shape:  torch.Size([3000, 3])
Sample pointcloud dtype:  torch.float64
Class:  bathtub
```

```
item_visualize = train_dataset_visualize[820] # feel free to change the index to visualize different pointcloud
pointcloud = item_visualize['pointcloud']
category = item_visualize['category']
print('Class:', inv_classes[category])
pcshow(*pointcloud.T)
```

```
Class: chair
```



Download the pre-sampled data

During training, since sampling data from faces takes a lot of time. To save time, we have pre-sampled the data for you. We sampled 1024 points in each pointcloud as in the paper. Now we download the pre-sampled data.

```
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cooki
!unzip -q pre_sample.zip;
```

```
--2023-05-05 01:30:15--  https://docs.google.com/uc?export=download&confirm=t&id=1nt-KtkxvpfJVwBZL51uEnwG_khK67M
Resolving docs.google.com (docs.google.com)... 172.253.118.100, 172.253.118.139, 172.253.118.138, ...
Connecting to docs.google.com (docs.google.com)|172.253.118.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-10-4s-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/oeib96vjbr
Warning: wildcards not supported in HTTP.
--2023-05-05 01:30:15--  https://doc-10-4s-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7
Resolving doc-10-4s-docs.googleusercontent.com (doc-10-4s-docs.googleusercontent.com)... 74.125.200.132, 2404:68
Connecting to doc-10-4s-docs.googleusercontent.com (doc-10-4s-docs.googleusercontent.com)|74.125.200.132|:443...
HTTP request sent, awaiting response... 200 OK
Length: 27331205 (26M) [application/zip]
Saving to: 'pre_sample.zip'

pre_sample.zip      100%[===================>]  26.06M  45.4MB/s    in 0.6s

2023-05-05 01:30:16 (45.4 MB/s) - 'pre_sample.zip' saved [27331205/27331205]
```

```
path = Path("off_points")
trainDataset = PointCloudData(path, folder='train', sampled=True, norm=True, rotation=True, noise=True)
testDataset = PointCloudData(path, folder='test', sampled=True, norm=True, rotation=False, noise=False)
```

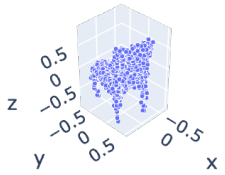
```
inv_classes = {i: cat for cat, i in trainDataset.classes.items()};
inv_classes
```

```
{0: 'bathtub',
 1: 'bed',
 2: 'chair',
 3: 'desk',
 4: 'dresser',
 5: 'monitor',
 6: 'night_stand',
 7: 'sofa',
 8: 'table',
 9: 'toilet'}
```

```
print('Train dataset size: ', len(trainDataset))
print('Test dataset size: ', len(testDataset))
print('Number of classes: ', len(trainDataset.classes))
print('Sample pointcloud shape: ', trainDataset[0]['pointcloud'].shape)
print('Sample pointcloud dtype: ', trainDataset[0]['pointcloud'].dtype)
print('Class: ', inv_classes[trainDataset[0]['category']])
```

```
Train dataset size:  2359
Test dataset size:  588
Number of classes:  10
Sample pointcloud shape:  torch.Size([1024, 3])
Sample pointcloud dtype:  torch.float64
Class:  bathtub
```

```
item = trainDataset[423]
pointcloud = item['pointcloud']
category = item['category']
pcshow(*pointcloud.T)
print(inv_classes[category])
```




```
chair
```

```
trainLoader = DataLoader(dataset=trainDataset, batch_size=32, shuffle=True)
testLoader = DataLoader(dataset=testDataset, batch_size=64)
```

Model Structure

The first transformation network is a mini-PointNet that takes raw point cloud as input and regresses to a 3 × 3 matrix. It's composed of a shared MLP(64, 128, 1024) network (with layer output sizes 64, 128, 1024) on each point, a max pooling across points and two fully connected layers with output sizes 512, 256.

Understanding Shared MLP

An important concept in the paper is Shared-MLP. Shared-MLP takes an input of shape [n, m] and returns a [n, k] matrix. The reason that it's called Shared-MLP is that it does the same as what nn.Linear does except that the matrix used to transform it is the same across the layer. In the following problem, you can understand that Shared-MLP is essentially the same as convolution with kernel size 1.

Implement the one-layer shared MLP and compare it with Conv1d to understand what Shared-MLP is and how Shared-MLP works.

- Implement the Matrix-based Shared-MLP
- Check the Matrix-based Shared-MLP has the same numbers of parameter as Conv1d
- Check the forward output of them are the same.

After that, you are free to use conv1d as shared-MLP in latter design

```
class SharedMLPLayer(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(SharedMLPLayer, self).__init__()
        torch.manual_seed(42)
        ##########################################################################
        # TODO: Implement the __init__ function
        # input: input_dim is the feature dimension of the input x
        #        output_dim is the feature dimension of the output
        #
        # hint: initialize the random params with given torch.randn.
        ##########################################################################
        self.sharedMatrix = nn.Parameter(torch.randn(input_dim, output_dim))
        ########################## END ###################################

    def forward(self, x):
        ##########################################################################
        # input: a tensor of shape [N, input_dim]
        # output a tensor of shape [N, output_dim]
        ##########################################################################
        out = x @ self.sharedMatrix
        ########################## END ###################################
        return out

k = 3
c = 64
sharedMLP = SharedMLPLayer(k, c)

# check the weight shape of sharedMLP
for name, param in sharedMLP.named_parameters():
    print(name, param.shape)
```

```
 sharedMatrix torch.Size([3, 64])
```

```
# An equivalent Conv1d layer implemend, and will be used to compare later
class Conv1d_layer(nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size, stride, bias=False):
        super(Conv1d_layer, self).__init__()
        self.conv1d_layer = nn.Conv1d(in_channel, out_channel, kernel_size, stride, bias=False)

        torch.manual_seed(42)
        self.conv_weights = torch.randn(in_channel, out_channel)
        conv_weights_ = self.conv_weights.t()
        conv_weights_ = torch.unsqueeze(conv_weights_, dim=-1)
        self.conv1d_layer.weight.data = conv_weights_

    def forward(self, x):
        out = self.conv1d_layer(x)
        out = torch.squeeze(out)
        out = torch.transpose(out, 0, 1)
        return out

in_channel = 3
out_channel = 64
kernel_size = 1
stride = 1

# Check the shape of conv1d_layer
conv1d_layer = Conv1d_layer(in_channel, out_channel, kernel_size, stride=1, bias=False)
for name, param in conv1d_layer.named_parameters():
    print(name, param.shape)
```

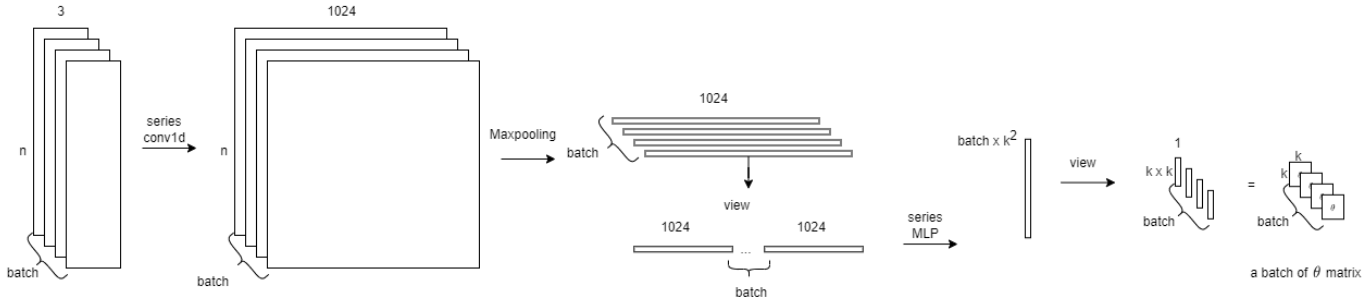
```
 conv1d_layer.weight torch.Size([64, 3, 1])
```

```
# Check the Matrix-based Shared-MLP has the same weights of Conv1d
print("Check the parameter number of sharedMLP and conv1d_layer is the same")
if not torch.allclose(conv1d_layer.conv_weights, sharedMLP.sharedMatrix, atol=1e-5):
    print("Error")
    print("Your sharedMLP weight = \n", sharedMLP.sharedMatrix.shape)
    print("Expected sharedMLP weight should be equal to conv1d_layer \n", conv1d_layer.conv_weights.shape)
    print("Difference = ", torch.norm(conv1d_layer.conv_weights-sharedMLP.sharedMatrix).item())
assert torch.allclose(conv1d_layer.conv_weights, sharedMLP.sharedMatrix, atol=1e-5), "conv_weights and sharedMLP.shar
print("passed")


print("\nCheck the forward output of the sharedMLP and conv1d_layer is the same")
x0 = torch.randn(5, 3)
y0 = sharedMLP(x0)
z0 = conv1d_layer(torch.transpose(torch.unsqueeze(x0, 0), 1, 2))
if not torch.allclose(y0, z0, atol=1e-5):
    print("Error")
    print("y0 = ", y0)
    print("z0 = ", z0)
    print("Difference = ", torch.norm(y0-z0).item())
assert torch.allclose(y0, z0, atol=1e-5), "y0 and z0 are not equal"
print("passed")
```

```
Check the parameter number of sharedMLP and conv1d_layer is the same
passed

Check the forward output of the sharedMLP and conv1d_layer is the same
passed
```

T-Net

The T-Net module is a type of Spatial Transformer Network (STN) that learns a kxk transformation matrix for a given point cloud, which is then used to transform the point cloud to a canonical pose. It consists of two parts: a convolutional network and a fully connected network. The convolutional network maps the input point cloud to a feature space, consisting of a series of convolutional layers with batch normalization and ReLU activation. The fully connected network takes the feature space and learns the transformation matrix, consisting of fully connected layers with batch normalization and ReLU activation. Finally, the T-Net applies the transformation matrix to the input point cloud to transform it to a canonical pose.




```
class Tnet(nn.Module):
    """
    T-Net is a type of spatial transformer network (STN) that learns a kxk transformation matrix
    for a given point cloud. The matrix is then used to transform the point cloud to a canonical
    pose. It consists of two parts: a convolutional network and a fully connected network.
    The convolutional network maps the input point cloud to a feature space and the fully connected
    network learns the transformation matrix from the feature space.
    """
    def __init__(self, hidden_sizes_conv=[64, 128, 1024], hidden_sizes_fc=[512, 256], k=3):
        super().__init__()
        self.k=k
        self.hidden_sizes_conv=hidden_sizes_conv
        self.hidden_sizes_fc=hidden_sizes_fc

        self.conv = self._build_conv()
        self.fc = self._build_fc()
        self.last_fc = nn.Linear(self.hidden_sizes_fc[-1],self.k*self.k)
```

```
    def _build_conv(self):
        ########################################################################
        # TODO: Build the convolutional network that maps the input point cloud
        # to a feature space. The hidden dimension is hidden_sizes_conv
        #
        # Hint: construct a series of convolutional layers with batch
        # normalization and ReLU activation.
        #   The convolution layers follows the following structure:
        #   [conv1d]-> [Batch Norm Layer] -> [ReLU]-> [conv1d]-> ...
        #   Take the integer in hidden_size_conv for convolution1d layer
        #   size and batch_norm layer size
        ########################################################################
        layers = []
        prev_size = self.k
        for layer_id, size in enumerate(self.hidden_sizes_conv):
            bn = nn.BatchNorm1d(size)
            conv = nn.Conv1d(prev_size, size,1)
            layers.append(conv)
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        ################################ END ###################################
        return nn.Sequential(*layers)

    def _build_fc(self):
        '''
        This function implements the "series mlp" part in the above graph
        The fully connected layers that this function build transforms an input
        of shape [bs, 1024] to [bs, k^2] where k is the desired size of the
        transformation matrix
        '''
        layers = []
        prev_size = self.hidden_sizes_conv[-1]
        for layer_id, size in enumerate(self.hidden_sizes_fc):
            bn = nn.BatchNorm1d(size)
            fc = nn.Linear(prev_size, size)
            layers.append(fc)
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        return nn.Sequential(*layers)


    def forward(self, input):
        ########################################################################
        # TODO: Performs the forward pass of the T-Net.
        # It first applies the convolutional network to the input point cloud
        # to obtain a feature space. Then it applies maxpool along the first dimension
        # Then, it applies the fully connected network to the feature space to
        # obtain the kxk transformation matrix. Finally, it applies the
        # transformation matrix to the input point cloud to transform it to a
        # canonical pose.
        # input: [batch, k=3, N], N is the points number in a object
        # output: [batch, k, k]
        #
        # Hint: the forward structure is as follows:
        # [ConvLayers]->[MaxPooling]->[Flatten]->[Fully Connected Layers]->[theta_Matrix + identity]
        # Remember that the identity matrix requires gradient
        ########################################################################
        # input.shape (bs,n,3)
        bs = input.size(0)

        xb = self.conv(input)
        pool = nn.MaxPool1d(xb.size(-1))(xb)
        flat = nn.Flatten(1)(pool)
        xb = self.fc(flat)

        init = torch.eye(self.k, requires_grad=True).repeat(bs,1,1)
        if xb.is_cuda:
          init=init.cuda()
        matrix = self.last_fc(xb).view(-1,self.k,self.k) + init
```

```
        ############################## END ###################################
        return matrix
```

Test your T-Net construction is correct.

- Test the layer parameter is correct, you should follow the comment in the coding part
- Test the T-Net forward is correct.

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters())

torch.manual_seed(42)
test_t_net = Tnet()

print("Getting the T-Net parameters")
if count_parameters(test_t_net)!=803081:
    print("Error")
    print("test_t_net parameters number = ", count_parameters(test_t_net))

assert count_parameters(test_t_net)==803081
print('passed')
```

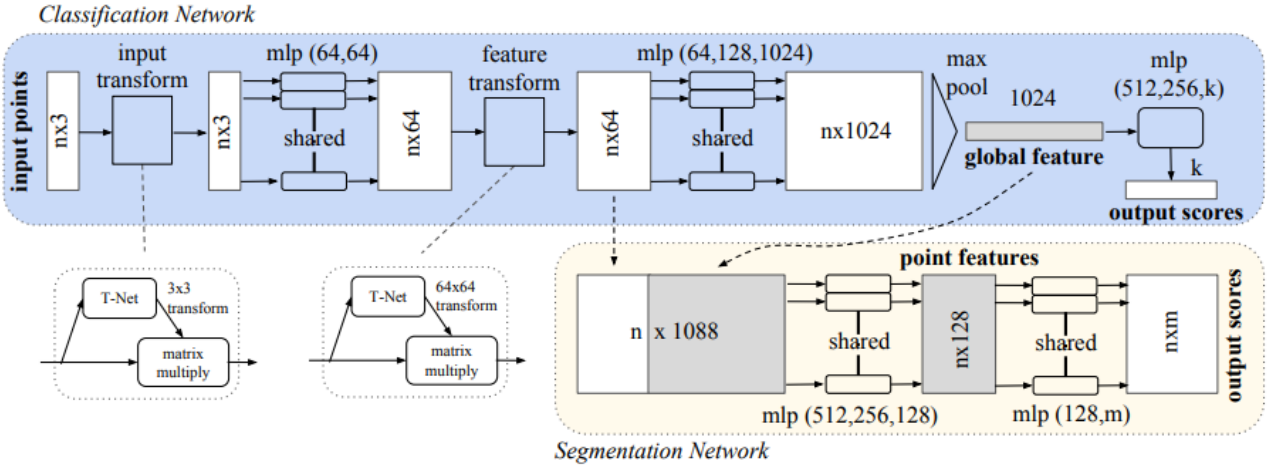
```
Getting the T-Net parameters
passed
```

```
torch.manual_seed(42)
x1 = torch.randn(3, 3, 5)

y1 = torch.tensor([[[ 1.4712e+00,  1.1447e+00,  6.5780e-02],
         [ 2.6862e-01,  1.5355e+00, -7.9635e-01],
         [-3.1744e-01,  4.8485e-01,  1.2669e+00]],

        [[ 1.0652e+00, -2.9729e-02, -9.1289e-04],
         [-2.0753e-01,  1.6646e+00,  5.0989e-01],
         [-2.5312e-01,  7.1402e-01,  8.2575e-01]],

        [[ 1.3445e+00,  6.7090e-01, -4.4554e-01],
         [ 2.4452e-01,  1.1833e+00, -5.8614e-01],
         [-5.3094e-02, -1.3413e-01,  9.4217e-01]]])

pred_y1 = test_t_net(x1)

print("Getting T-Net out:", end = (" "))
if not torch.allclose(y1, pred_y1, rtol=1e-03, atol=1e-03):
    print("Error")
    print("Your answer is:", pred_y1)
    print("The expected answer is:", y1)

assert torch.allclose(y1, pred_y1, rtol=1e-03, atol=1e-03),  "different y_pred and y"
print("passed")
```

```
Getting T-Net out: passed
```

Transform Class

The Transform class is every thing before last MLPs in PointNet. It is a neural network architecture that uses two pairs of spatial transform net (STN) and shared MLP layers to extract global features from a point cloud data of (nx3) shape. The STN is implemented using the T-Net and computes the 3x3 transform matrix, which is then multiplied with the input point cloud to get a transformed point cloud of the same shape. The transformed point cloud is then input into the shared MLP layers along with the feature transform matrix. The output from the shared MLP layers is max pooled along the feature dimension to get a global feature vector. The output also includes the point and feature transform matrices.



Implement the Transform part

```
class Transform(nn.Module):
    def __init__(self, input_size=3, feature_size=64, sharedMLP1_layers=[64, 64], sharedMLP2_layers=[64, 128, 1024],
        """
        Transform class is all the pipeline to get a global feature
                 ____________________                                  _______________                 _________
                |                    |                                |               |               |         |
        x -->   |   input transform  | --> y (canonical point cloud) --> |  shared MLP   | --> feature -->|feature t
                |____________________|                                |_______________|               |_________
        The transform class is a neural network architecture that go throught 2 pairs of spactial transform net and s
        The STN is the T-Net that implement above, and the shared-MLP can be regarded as a one-dimensional convolutic

        the input x as a point cloud data of (nx3) shape first compute the 3x3 transform matrix and multiplied with t

        the last_activate bool is True when you want to add the last layer with activation function.
        """
        super().__init__()
        self.batch_norm = True

        self.input_transform = Tnet(k=3)
        self.feature_transform = Tnet(k=64)

        self.sharedMLP1 = self._build_sharedMLP(input_size, sharedMLP1_layers, last_activate=True)
        self.sharedMLP2 = self._build_sharedMLP(feature_size, sharedMLP2_layers, last_activate=False)

    def _build_sharedMLP(self, input_dim, sharedMLP_layers, last_activate = True):
        ##########################################################################
        # TODO: Build the shared MLP layers. Take the sharedMLP_layers list as
        # sharedMLP_layers is a list of dimensions of hidden layers
        # last_activate represents whether apply activation to the last layer
        # hidden dimension in Conv1d, Batch norm
        #       The structure is [Conv1d]->[Batch Norm]->[ReLU]
        ##########################################################################
        layers = []
        prev_size = input_dim
        for layer_id, size in enumerate(sharedMLP_layers):
            layers.append(nn.Conv1d(prev_size, size, 1))

            if self.batch_norm:
```

```
                layers.append(nn.BatchNorm1d(size))

            if (layer_id < len(sharedMLP_layers)-1) or last_activate:
                layers.append(nn.ReLU())

            prev_size = size
        ############################################################################
        return nn.Sequential(*layers)

    def forward(self, input):     #input:[batch_size, 3, 1024] output:[batch_size, 1024]
        ############################################################################
        # TODO: Implement the code to multiply the transform matrix and the point
        # cloud.
        #   The transformed x should be the same shape of x [batch_size, 3, N]
        #   The transformed feature is [batchsize, 64, N]
        #   The maxpooled feature is [batch_size, 1024, 1]
        # Hint:
        # 1. Get the transform matrix [batch_size, 3, 3] by the T-Net
        # 2. Batch matrix multiply the input x and transform matrix
        # 3. Input the data into the Shared MLP
        # 4. Batch matrix multiply the feature and the feature_transform matrix
        # 5. Input the output into the Shared MLP with feature dimension
        # 6. Maxpooling along the feature dimension
        # 7. output the output data, points transform matrix, and the feature
        #       transform matrix
        ############################################################################
        matrix3x3 = self.input_transform(input)     #[batch_size, 3, 3]
        # batch matrix multiplication
        xb = torch.bmm(torch.transpose(input,1,2), matrix3x3).transpose(1,2)     #[batch_size, 3, N]
        xb = self.sharedMLP1(xb)

        matrix64x64 = self.feature_transform(xb)     #[batch_size, 64, 64]
        xb = torch.bmm(torch.transpose(xb,1,2), matrix64x64).transpose(1,2)     #[batch_size, 64, N]
        xb = self.sharedMLP2(xb)

        xb = nn.MaxPool1d(xb.size(-1))(xb)     #[batch_size, 1024, 1]
        # print(xb.shape)
        output = nn.Flatten(1)(xb)     #[batch_size, 1024]
        ############################################################################
        return output, matrix3x3, matrix64x64
```

```
# Check the default parameter number to see the model is correct
torch.manual_seed(42)
test_transform_net = Transform()

print("Getting the parameter number of the transform net:", end = (" "))
test_tranform_net_param_num = count_parameters(test_transform_net)
if test_tranform_net_param_num!=2812105:
    print("Error")
    print("Your test_transform_net parameters number = ", count_parameters(test_transform_net))
    print("Expected answer = 2812105")
    print("Difference = ", torch.absolute(test_tranform_net_param_num!=2812105))

assert count_parameters(test_transform_net)==2812105
print("passed")

# Check the forward output is correct
torch.manual_seed(42)
x2 = torch.randn(2, 3, 5)
pred_y2, pred_mat1, pred_mat2 = test_transform_net(x2)

mat1 = torch.tensor([[[ 1.0655,  0.4169,  0.1452],
         [ 0.0240,  1.7885, -0.6011],
         [-0.5582,  0.6857,  1.0256]],

        [[ 1.5976,  0.8703, -0.4317],
         [ 0.2260,  1.2361,  0.0457],
         [ 0.0986,  0.0844,  1.0267]]])

print("Getting the correct forward output : ", end = (""))
if not torch.allclose(mat1, pred_mat1, rtol=1e-03, atol=1e-03):
    print("Error")
    print("Your pred_mat1 is \n", pred_mat1)
    print("Expected answer mat1 is \n", mat1)
    print("Difference = ", torch.norm(pred_mat1- mat1))
assert torch.allclose(mat1, pred_mat1, rtol=1e-03, atol=1e-03),  "different pred_mat1 and mat1"
print("passed")
```

```
Getting the parameter number of the transform net: passed
Getting the correct forward output : passed
```

PointNet Classifier

The following code defines a PyTorch module called PointNet for classifying point cloud data. The PointNet module includes a Transform class, which takes in 3D point cloud data as input and generates global features and transformation matrices. The global features are then passed through a multi-layer perceptron (MLP) to generate scores for classification. The MLP consists of linear layers, batch normalization, ReLU activation, and dropout layers. The PointNet module outputs the logsoftmax of the scores along with the 3x3 and 64x64 transformation matrices generated by the Transform class. The PointNet module can be customized with different layer configurations, batch normalization, and dropout rates.

```
class PointNet(nn.Module):
    def __init__(self, sharedMLP1_layers=[64, 64], sharedMLP2_layers=[64, 128, 1024], dropout_rate = 0.3,  classes =
        """
        Point Net the whole neural network for the classification of point cloud data
                      _________                        ___
        input x--->|Transform|---> global feature--->|MLP|---> scores
                   |_________|                       |___|
            args:   sharedMLP1_layers is the first shared MLP in Transform class
                    sharedMLP2_layers is the second shared MLP in Transorm class
                The MLP has the structure of
                    [Linear] -> [Batch Norm] -> [ReLU] -> [Dropout] -> [Linear] -> ... -> [Linear] -> [Batch Norm] ->


        """
        super().__init__()
        self.transform = Transform(input_size=3, feature_size=64, sharedMLP1_layers=sharedMLP1_layers, sharedMLP2_lay
        self.batch_norm = batch_norm
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, classes)
        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.dropout = nn.Dropout(dropout_rate)
        self.logsoftmax = nn.LogSoftmax(dim=1)

    def forward(self, input):
        ########################################################################
        # TODO: get the output y, 3x3 transform matrix and 64x64 transform
        # matrix from self.transform net.
        # Then, y->[fc1]->[bn1]->[relu]->[fc2]->[dropout]->[bn2]->[relu]->[fc3]->z
        # return logsoftmax(z), 3x3 transform matrix and 64x64 transform matrix
        ########################################################################
        xb, matrix3x3, matrix64x64 = self.transform(input)
        xb = F.relu(self.bn1(self.fc1(xb)))
        xb = F.relu(self.bn2(self.dropout(self.fc2(xb))))
        output = self.fc3(xb)
        ########################## END ##########################################
        return self.logsoftmax(output), matrix3x3, matrix64x64
```

```
# Test your implementation
sharedMLP1_layers=[64, 64]
sharedMLP2_layers=[64, 128, 1024]

torch.manual_seed(42)
test_point_net = PointNet(sharedMLP1_layers, sharedMLP2_layers)

print("Getting the parameter number of the pointnet:", end = " ")
test_point_net_param_num = count_parameters(test_point_net)
if test_point_net_param_num!=3472339:
    print("Error")
    print("Your test_transform_net parameters number = ", count_parameters(test_point_net))
    print("The expected answer is 3472339")
    print("Difference = ", torch.absolute(torch.tensor(test_point_net_param_num-3472339)))

assert count_parameters(test_point_net)==3472339
print("passed")

torch.manual_seed(42)
x3 = torch.randn(3, 3, 5)
w, pred_mat3x3, pred_matfxf = test_point_net(x3)

mat3x3 = torch.tensor([[[ 1.4712e+00,  1.1447e+00,  6.5780e-02],
         [ 2.6862e-01,  1.5355e+00, -7.9635e-01],
         [-3.1744e-01,  4.8485e-01,  1.2669e+00]],

        [[ 1.0652e+00, -2.9729e-02, -9.1289e-04],
         [-2.0753e-01,  1.6646e+00,  5.0989e-01],
         [-2.5312e-01,  7.1402e-01,  8.2575e-01]],

        [[ 1.3445e+00,  6.7090e-01, -4.4554e-01],
         [ 2.4452e-01,  1.1833e+00, -5.8614e-01],
         [-5.3094e-02, -1.3413e-01,  9.4217e-01]]])
print("Check the output correctness by mat3x3", end = (" "))
if not torch.allclose(mat3x3, pred_mat3x3, rtol=1e-03, atol=1e-03):
    print("Error")
    print("Your pred_mat3x3 is \n", pred_mat1)
    print("The answer mat3x3 is \n", mat1)
    print("Difference = ", torch.norm(pred_mat3x3- mat3x3))
assert torch.allclose(pred_mat3x3, mat3x3, rtol=1e-03, atol=1e-03),  "different pred_mat1 and mat1"
print("passed")
```

```
Getting the parameter number of the pointnet: passed
Check the output correctness by mat3x3 passed
```

Train the model

Loss formula

One last thing before training the model is defining the loss of this network. Our loss function is formed by two part: softmax classification loss and regularization term:

1. Softmax Classification Loss: Use cross entropy loss to calculate softmax classification loss, the formula is following:

   $H(p, q) = -\sum_{x \in X} p(x) \cdot log\ q(x)$

2. Regularization Term: Constrain the feature transformation matrix learned by T-net to be close to orthogonal matrix, so the regularization term formula is following:

   $\| I - AA^T \|_F^2$

By combining the above two parts, our overall loss function is:

$L = H(p, q) + \alpha \times (\| I - A_1 A_1^T \|_F^2 + \| I - A_2 A_2^T \|_F^2)$

$Note: A_1, A_2$ is 3-dimensional and 64-dimensional feature transform matrix formed by T-net

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

```
class ClsExperiment:
    def __init__(self, train_data, val_data, model: nn.Module,
                 lr: float, save=True, alpha=None):
        self.train_data = train_data
        self.val_data = val_data
        self.optimizer = torch.optim.Adam(model.parameters(), lr=lr)
        self.save=save
        if alpha==None:
          self.alpha=0.0001
        else:
          self.alpha=alpha
        self.model = model.cuda()
        self.loss=[]
        self.acc=[]

    def train(self, epochs):
        epoch_iterator = trange(epochs)
        # validation
        if self.val_data:
            self.evaluate("Initial ")
        for epoch in epoch_iterator:
            self.model.train()
            running_loss = 0.0
            data_iterator = tqdm(self.train_data)
            for i, data in enumerate(data_iterator, 0):
                inputs, labels = data['pointcloud'].to(device).float(), data['category'].to(device)
                self.optimizer.zero_grad()

                outputs, m3x3, m64x64 = self.model(inputs.transpose(1,2))

                loss = self.getloss(outputs, labels, m3x3, m64x64, self.alpha)
                loss.backward()
                self.optimizer.step()

                # print statistics
                running_loss += loss.item()
                if i % 10 == 9:    # print every 10 mini-batches
                    self.loss.append(running_loss / 10)
                    data_iterator.set_postfix(loss=running_loss / 10)
                    running_loss = 0.0
            # validation
            if self.val_data:
                self.evaluate("Epoch:{} ".format(epoch+1))
            if self.save:
                torch.save(self.model.state_dict(), "save_"+str(epoch)+".pth")

    def evaluate(self, ttype):
        self.model.eval()
        correct = total = 0
        with torch.no_grad():
            for data in self.val_data:
                inputs, labels = data['pointcloud'].to(device).float(), data['category'].to(device)
                inputs = inputs.transpose(1,2)
                outputs, __, __ = self.model(inputs)
                num_correct = self.get_num_correct(outputs.data, labels)
                total += labels.size(0)
                correct += num_correct
            val_acc = 100. * correct / total
            self.acc.append(val_acc)
            print(ttype+'Test accuracy: %d %%' % val_acc)
            #epoch_iterator.set_postfix(val_acc=val_acc)

    def get_num_correct(self, outputs, labels):
          ##########################################################################
          # TODO: given outputs of the model, and ground truth labels,
```

```
        # get the number of correct predictions among bs objects
        # Inputs: outputs: [bs, 10]
        #         labels: [bs]
        # Return: the number of correct predictions among bs objects
        # hint: torch.max can be helpful
        ########################################################################
        _, predicted = torch.max(outputs, 1)
        num_correct = (predicted == labels).sum()
        ########################################################################
        return num_correct.item()

    def getloss(self, outputs, labels, m3x3, m64x64, alpha):
        ########################################################################################
        # TODO: Get the loss of the model. The loss of the model consists of three parts:
        # 1. Cross entropy loss between outputs and labels
        # 2. The norm of two T-net transform matrices: I - m3x3^T @ m3x3 and
        #                                              I - m64x64^T @ m64x64
        # 3. The final overall loss is corssEctropy(y, y_hat) + alpha*(|I-m3x3^T@ m3x3|_2 + |I-m64x64^T@ m64x64|_2) /
        #
        # Inputs: outputs: [bs, 10]
        #         labels: [bs]
        #         m3x3: [bs, 3, 3]
        #         m64x64: m3x3: [bs, 64, 64]
        # Return: the loss
        #
        #
        # hint: 1. Remember to add an extra parameter (requires_grad=True) when you create
        # a new matrix using pytorch if the created matrix is interacted with other matrices
        # in the model
        ########################################################################################
        criterion = torch.nn.CrossEntropyLoss()
        bs = outputs.size(0)
        id3x3 = torch.eye(3, requires_grad=True).repeat(bs,1,1)       # identity matrix repeat batch_size times
        id64x64 = torch.eye(64, requires_grad=True).repeat(bs,1,1)
        if outputs.is_cuda:
            id3x3=id3x3.cuda()
            id64x64=id64x64.cuda()
        diff3x3 = id3x3-torch.bmm(m3x3,m3x3.transpose(1,2))         # Loss: I-A.t@A
        diff64x64 = id64x64-torch.bmm(m64x64,m64x64.transpose(1,2))
        loss = criterion(outputs, labels) + alpha * (torch.norm(diff3x3)+torch.norm(diff64x64)) / float(bs)
        ########################################################################################
        return loss

    def plt_loss(self):
      x1 = range(0, len(self.loss))
      y1 = self.loss
      plt.plot(x1, y1, 'o-')
      plt.title('Train loss vs. epoches')
      plt.ylabel('Train loss')
      plt.show()

    def plt_accuracy(self):
      x1 = range(0, len(self.acc))
      y1 = self.acc
      plt.plot(x1, y1, 'o-')
      plt.title('Test accuracy vs. epoches')
      plt.ylabel('Tes accuracy')
      plt.show()

    def get_pred_label(self):
      final_preds = []
      final_labels = []
      with torch.no_grad():
          for data in self.val_data:
              inputs, labels = data['pointcloud'].to(device).float(), data['category'].to(device)
              outputs, __, __ = self.model(inputs.transpose(1,2))
              _, preds = torch.max(outputs.data, 1)
              final_preds += list(preds.cpu().numpy())
              final_labels += list(labels.cpu().numpy())
      return final_preds, final_labels
```

```
    def plot_confusion_matrix_(self, cm, classes, normalize, title='Confusion matrix', cmap=plt.cm.Blues):
      if normalize:
          cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
          print("Normalized confusion matrix")
      else:
          print('Confusion matrix, without normalization')

      plt.figure(figsize=(8,8))
      plt.imshow(cm, interpolation='nearest', cmap=cmap)
      plt.title(title)
      plt.colorbar()
      tick_marks = np.arange(len(classes))
      plt.xticks(tick_marks, classes, rotation=45)
      plt.yticks(tick_marks, classes)

      fmt = '.2f' if normalize else 'd'
      thresh = cm.max() / 2.
      for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
          plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white" if cm[i, j] > thresh else

      plt.tight_layout()
      plt.ylabel('True label')
      plt.xlabel('Predicted label')

    def plt_confusion_matrix(self, normalize=True):
      preds, labels = self.get_pred_label()
      cm = confusion_matrix(labels, preds)
      self.plot_confusion_matrix_(cm, list(classes.keys()), normalize)
```

Test your implementation

```
torch.manual_seed(99)
pointnet_test = PointNet()
clsexp_test = ClsExperiment(trainLoader, testLoader, pointnet_test, lr=0.001, save=False)
# Testing get_num_correct
correct1 = torch.tensor([3])
outputs = torch.randn(8, 4)
labels = torch.tensor([0,2,1,1,3,2,3,0])
res1 = torch.tensor(clsexp_test.get_num_correct(outputs, labels))
print('Testing get_num_correct:')
if not torch.allclose(res1, correct1, rtol=1e-03):
  print("Error")
  print("Your answer is: ", res1)
  print("The expected answer is: ", correct1)
else:
  print('passed')

# Testing get_loss
m3x3 = torch.randn(8, 3,3)
m64x64 = torch.randn(8, 64,64)
res2 = clsexp_test.getloss(outputs, labels, m3x3, m64x64, alpha = 0.0001)
correct2 = torch.tensor(1.4213)
print('Testing getloss:')
if not torch.allclose(res2, correct2, rtol=1e-03):
  print("Error")
  print("Your answer is: ", res2)
  print("The expected answer is: ", correct2)
else:
  print('passed')
```

```
Testing get_num_correct:
passed
Testing getloss:
passed
```

Fine tuning the hyper-parameters to reach a testing accuray of 75 within 15 epochs

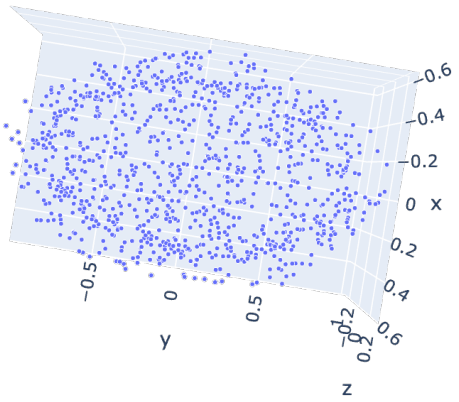
```
lr=0.005
alpha=0.0001
sharedMLP1_layers=[64, 64]
sharedMLP2_layers=[64, 128, 1024]
dropout_rate=0.3


pointnet = PointNet(sharedMLP1_layers, sharedMLP2_layers, dropout_rate)
pointnet.to(device)
clsexp = ClsExperiment(trainLoader, testLoader, pointnet, lr, False, alpha)
clsexp.train(epochs=15)
```

```
100%                                      15/15 [03:44<00:00, 13.86s/it]
Initial Test accuracy: 10 %
100%                                      74/74 [00:14<00:00, 7.38it/s, loss=1.39]
Epoch:1 Test accuracy: 44 %
100%                                      74/74 [00:11<00:00, 7.45it/s, loss=1.04]
Epoch:2 Test accuracy: 63 %
100%                                      74/74 [00:11<00:00, 7.20it/s, loss=0.922]
Epoch:3 Test accuracy: 63 %
100%                                      74/74 [00:11<00:00, 7.49it/s, loss=1]
Epoch:4 Test accuracy: 64 %
100%                                      74/74 [00:11<00:00, 6.80it/s, loss=0.759]
Epoch:5 Test accuracy: 66 %
100%                                      74/74 [00:11<00:00, 7.33it/s, loss=1.02]
Epoch:6 Test accuracy: 67 %
100%                                      74/74 [00:11<00:00, 7.41it/s, loss=0.63]
Epoch:7 Test accuracy: 69 %
100%                                      74/74 [00:12<00:00, 6.05it/s, loss=0.576]
Epoch:8 Test accuracy: 78 %
100%                                      74/74 [00:11<00:00, 5.59it/s, loss=0.603]
Epoch:9 Test accuracy: 63 %
100%                                      74/74 [00:11<00:00, 7.18it/s, loss=0.586]
Epoch:10 Test accuracy: 78 %
100%                                      74/74 [00:11<00:00, 7.29it/s, loss=0.536]
Epoch:11 Test accuracy: 80 %
100%                                      74/74 [00:11<00:00, 6.99it/s, loss=0.537]
Epoch:12 Test accuracy: 79 %
100%                                      74/74 [00:11<00:00, 7.36it/s, loss=0.521]
Epoch:13 Test accuracy: 78 %
100%                                      74/74 [00:11<00:00, 7.52it/s, loss=0.571]
Epoch:14 Test accuracy: 79 %
100%                                      74/74 [00:11<00:00, 6.77it/s, loss=0.565]
Epoch:15 Test accuracy: 72 %
```

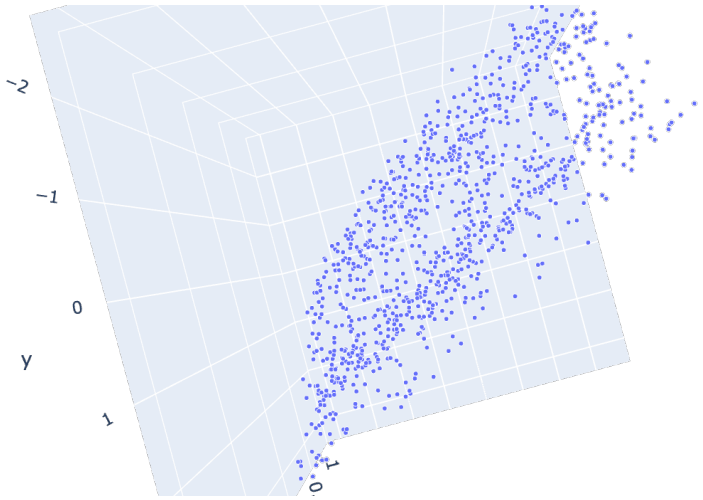
Visualize the pointcloud after applying T-net transformation matrix

```
data=trainDataset[55]
inputs, labels = data['pointcloud'].to(device).float(), data['category']
__, matrix, _=pointnet(inputs.view(1, 1024, 3).transpose(1,2))
index=random.randint(0, len(data['pointcloud']))
inputs = inputs.view(1024, 3).cpu()
print("Before applying transform matrix visualization:\n")
pcshow(*inputs.T)
print("After applying transform matrix visualization:\n")
after_transform=inputs@matrix[0].cpu()
after_transform=after_transform.detach().numpy()
pcshow(*after_transform.T)
print(matrix[0].T@matrix[0])
```

Before applying transform matrix visualization:

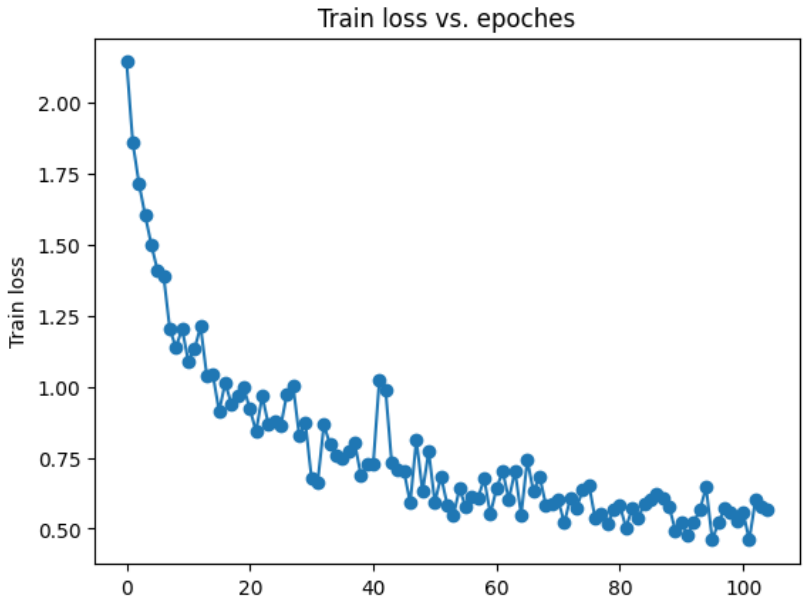


After applying transform matrix visualization:

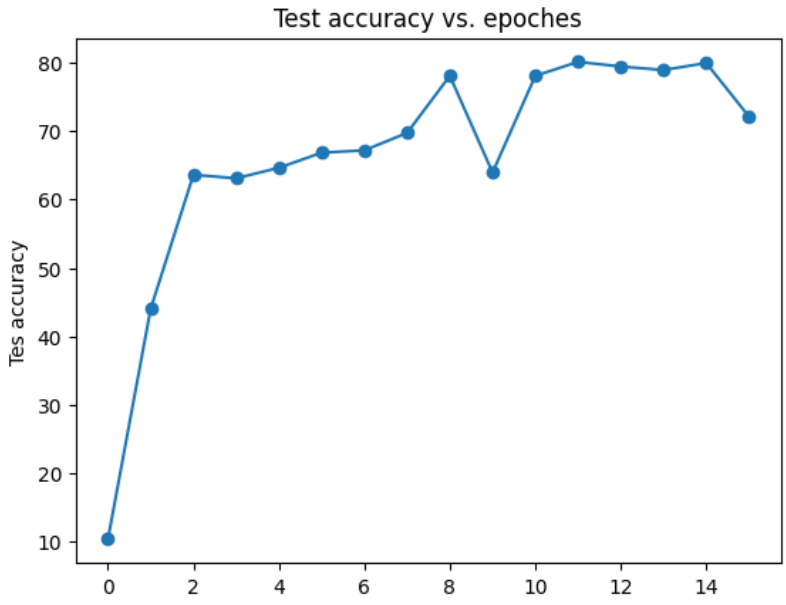



```
tensor([[ 70.4333, -17.6840,  -2.2044],
        [-17.6840,  27.8880,  -5.1411],
        [ -2.2044,  -5.1411,   2.8246]], device='cuda:0',
       grad_fn=<MmBackward0>)
```

```
clsexp.plt_loss()
```

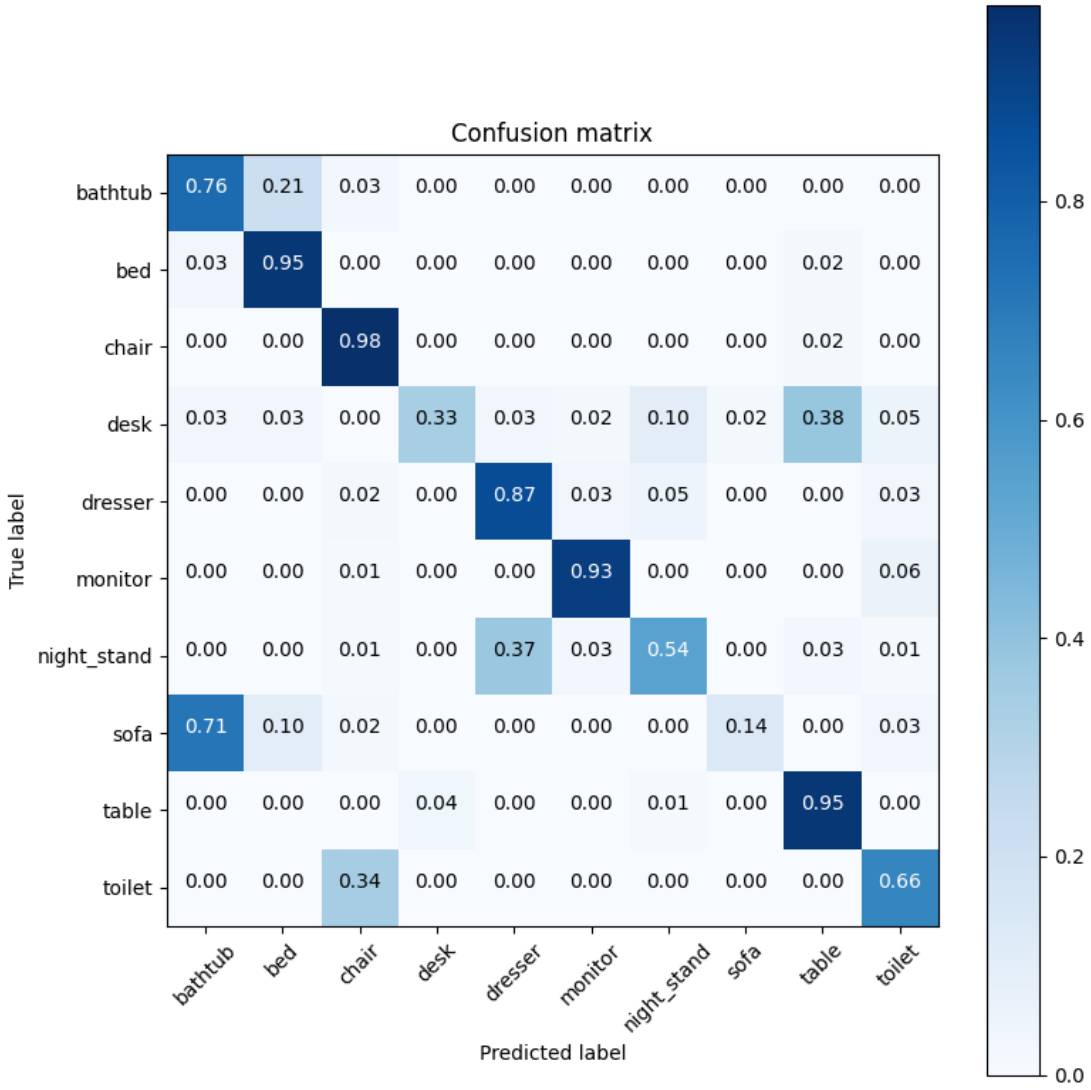



```
clsexp.plt_accuracy()
```



```
clsexp.plt_confusion_matrix()
```

```
Normalized confusion matrix
```



3.3.7 Segmentation_sol.ipynb Code pdf

PointNet Segmentation part

This is a homework based on the paper [PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation]

Dataset

Download the dataset by wget

```
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookie
!unzip -q expert_verified.zip
```

```
--2023-05-05 01:35:00--  https://docs.google.com/uc?export=download&confirm=t&id=1LxpGXSDOI9V5mc5DGuRlOcbulxZoFohC
Resolving docs.google.com (docs.google.com)... 108.177.121.100, 108.177.121.138, 108.177.121.113, ...
Connecting to docs.google.com (docs.google.com)|108.177.121.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-10-4s-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/2rmmufibfk7rrm49qoj5580
Warning: wildcards not supported in HTTP.
--2023-05-05 01:35:18--  https://doc-10-4s-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/2rmmufib
Resolving doc-10-4s-docs.googleusercontent.com (doc-10-4s-docs.googleusercontent.com)... 142.250.128.132, 2607:f8b0:4001:c32:
Connecting to doc-10-4s-docs.googleusercontent.com (doc-10-4s-docs.googleusercontent.com)|142.250.128.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4081865 (3.9M) [application/zip]
Saving to: 'expert_verified.zip'

expert_verified.zip 100%[===================>]   3.89M  --.-KB/s    in 0.02s

2023-05-05 01:35:19 (236 MB/s) - 'expert_verified.zip' saved [4081865/4081865]
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookie
!unzip -q points.zip
```



```
ad&confirm=t&id=1yQapyoGEfdBen75MTwh6AD7GRY-UMVZn
08.177.121.101, 108.177.121.100, ...
443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0o-4s-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/0uisukk05etrdfv5809j2r2
Warning: wildcards not supported in HTTP.
--2023-05-05 01:35:20--  https://doc-0o-4s-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/0uisukk0
Resolving doc-0o-4s-docs.googleusercontent.com (doc-0o-4s-docs.googleusercontent.com)... 142.250.128.132, 2607:f8b0:4001:c32:
Connecting to doc-0o-4s-docs.googleusercontent.com (doc-0o-4s-docs.googleusercontent.com)|142.250.128.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 97711612 (93M) [application/zip]
Saving to: 'points.zip'

points.zip          100%[===================>]  93.18M   166MB/s    in 0.6s

2023-05-05 01:35:20 (166 MB/s) - 'points.zip' saved [97711612/97711612]
```

```
from __future__ import print_function, division
import os
import torch
import pandas as pd
from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
from torch.utils.data.dataset import random_split
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import random
from tqdm.notebook import trange, tqdm

import plotly.graph_objects as go
import plotly.express as px
```

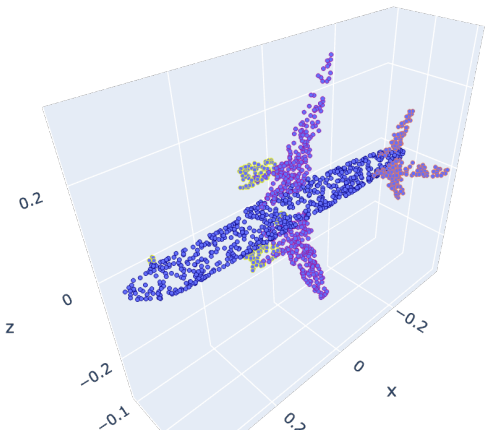
Load the dataset

```
def read_pts(file):
    verts = np.genfromtxt(file)
    #return utils.cent_norm(verts)
    return verts

def read_seg(file):
    verts = np.genfromtxt(file, dtype= (int))
    return verts

def sample_2000(pts, pts_cat):
    res1 = np.concatenate((pts,np.reshape(pts_cat, (pts_cat.shape[0], 1))), axis= 1)
    res = np.asarray(random.choices(res1, weights=None, cum_weights=None, k=2000))
    images = res[:, 0:3]
    categories = res[:, 3]
    categories-=np.ones(categories.shape)
    return images, categories


class Data(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, root_dir, valid=False, transform=None):

        self.root_dir = root_dir
        self.files = []
        self.valid=valid

        newdir = root_dir + '/expert_verified/points_label/'

        for file in os.listdir(newdir):
            if file.find("(")!=-1:
                continue
            o = {}
            o['category'] = newdir + file
            o['img_path'] = root_dir + '/points/'+ file.replace('.seg', '.pts')
            self.files.append(o)


    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        img_path = self.files[idx]['img_path']
        category = self.files[idx]['category']
        with open(img_path, 'r') as f:
            image1 = read_pts(f)
        with open(category, 'r') as f:
            category1 = read_seg(f)
        image2, category2 = sample_2000(image1, category1)
        if not self.valid:
            theta = random.random()*360

        return {'image': np.array(image2, dtype="float32"), 'category': category2.astype(int)}



root_dir="/content"
dset = Data(root_dir , transform=None)
train_num = int(len(dset) * 0.95)
val_num = int(len(dset) *0.05)
if int(len(dset)) - train_num -  val_num >0 :
    train_num = train_num + 1
elif int(len(dset)) - train_num -  val_num < 0:
    train_num = train_num -1
train_dataset, val_dataset = random_split(dset, [train_num, val_num])
val_dataset.valid=True

print('######### Dataset class created #########')
print('Number of images: ', len(dset))
print('Sample image shape: ', dset[0]['image'].shape)

Seg_train_loader = DataLoader(dataset=train_dataset, batch_size=32)
Seg_val_loader = DataLoader(dataset=val_dataset, batch_size=32)
```

```
######### Dataset class created #########
Number of images:  2690
Sample image shape:  (2000, 3)
```

Visualize item in dataset

```
def visualize_rotate(data):
    x_eye, y_eye, z_eye = 1.25, 1.25, 0.8
    frames=[]

    def rotate_z(x, y, z, theta):
        w = x+1j*y
        return np.real(np.exp(1j*theta)*w), np.imag(np.exp(1j*theta)*w), z

    for t in np.arange(0, 10.26, 0.1):
        xe, ye, ze = rotate_z(x_eye, y_eye, z_eye, -t)
        frames.append(dict(layout=dict(scene=dict(camera=dict(eye=dict(x=xe, y=ye, z=ze))))))
    fig = go.Figure(data=data,
                    layout=go.Layout(
                        updatemenus=[dict(type='buttons',
                                    showactive=False,
                                    y=1,
                                    x=0.8,
                                    xanchor='left',
                                    yanchor='bottom',
                                    pad=dict(t=45, r=10),
                                    buttons=[dict(label='Play',
                                                  method='animate',
                                                  args=[None, dict(frame=dict(duration=50, redraw=True),
                                                                   transition=dict(duration=0),
                                                                   fromcurrent=True,
                                                                   mode='immediate'
                                                                   )]
                                                  )
                                             ]
                                    )
                                ]
                    ),
                    frames=frames
            )

    return fig


def segpcshow(xs,ys,zs, category):
    data=[go.Scatter3d(x=xs, y=ys, z=zs,
                                   mode='markers')]
    fig = visualize_rotate(data)
    fig.update_traces(marker=dict(size=2,
                      line=dict(width=2,
                      color=category)),
                      selector=dict(mode='markers'))
    fig.show()


item_visualize = train_dataset[1000] # feel free to change the index to visualize different pointcloud
pointcloud = item_visualize['image']
category = item_visualize['category']
segpcshow(*pointcloud.T, category)
```



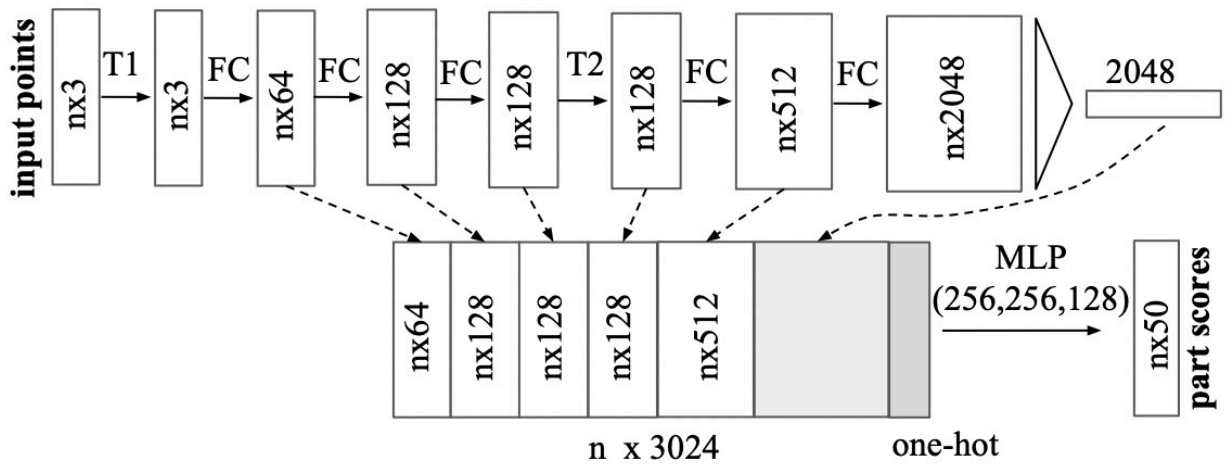
```
def _set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)

def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
```

Model Sturcture

Segmentation Model

Refer the Supplementary C in PointNet paper

The output from the classification global transform section forms a vector [f1 , . . . , fK], which is a global signature of the input set. We can easily train a SVM or multi-layer perceptron classifier on the shape global features for classification. However, point segmentation requires a combination of local and global knowledge. We implement this by using U-Net-Like structure, concatenate the global and local feature together. The feature in every layer is stacked together to form point feature for every point. The architecture can be seen in the following figure. The architecture can be seen in the following figure.



To get started with implementing PointNet segmentation, we need to change `Transform` to `SegTransform`, `PointNet` to `PointNetSeg` in order to correspond with the paper implementation.

To accelerate training, we limit the segmentation dataset to only one category(plane). So we simplify the segmentation model by not concentrating the one-hot vector to point feature. As a result, the resulting point feature will only have 3008 dimensions.

HINT: use `Conv1d` to do shared MLP

T-net part is same as T-net in PointNet classification

```
class Tnet(nn.Module):
    """
    T-Net is a type of spatial transformer network (STN) that learns a kxk transformation matrix
    for a given point cloud. The matrix is then used to transform the point cloud to a canonical
    pose. It consists of two parts: a convolutional network and a fully connected network.
    The convolutional network maps the input point cloud to a feature space and the fully connected
    network learns the transformation matrix from the feature space.
    """
    def __init__(self, hidden_sizes_conv=[64, 128, 1024], hidden_sizes_fc=[512, 256], k=3):
        super().__init__()
        self.k=k
        self.hidden_sizes_conv=hidden_sizes_conv
        self.hidden_sizes_fc=hidden_sizes_fc
        self.fc1 = nn.Linear(hidden_sizes_fc[-1],k*k)
        self.conv = self._build_conv()
        self.fc = self._build_fc()

    def _build_conv(self):
        layers = []
        prev_size = self.k
        for layer_id, size in enumerate(self.hidden_sizes_conv):
            bn = nn.BatchNorm1d(size)
            conv = nn.Conv1d(prev_size, size,1)  #share mlp can be implemented by three 1*1 convolution filter
            layers.append(conv)
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        return nn.Sequential(*layers)

    def _build_fc(self):
        layers = []
        prev_size = self.hidden_sizes_conv[-1]
        for layer_id, size in enumerate(self.hidden_sizes_fc):
            bn = nn.BatchNorm1d(size)
            fc = nn.Linear(prev_size, size)
            layers.append(fc)
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        return nn.Sequential(*layers)


    def forward(self, input):

        # input.shape (bs,n,3)
        bs = input.size(0)

        xb = self.conv(input)   #3->64->128->1024
        #torch.nn.MaxPool1d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
        #bs*n*1024->bs*1024*1
        # print(xb.shape)
        pool = nn.MaxPool1d(xb.size(-1))(xb)
        # print(pool.shape)

        flat = nn.Flatten(1)(pool)   #flatten the tensor bs*1*1024->bs*1024
        #fully connected la m,yer
        #1024->512->256
        xb = self.fc(flat)

        #initialize with identity matrix bs*k*k
        init = torch.eye(self.k, requires_grad=True).repeat(bs,1,1)
        if xb.is_cuda:
          init=init.cuda()
        #b*256->b*(k*k)->b*k*k + init
        matrix = self.fc1(xb).view(-1,self.k,self.k) + init
        #input b*n*k
        return matrix
```

SegTransform, PointNetSeg part

```
class SegTransform(nn.Module):
    """
    SegTransform is a class transform point cloud into point feature. When the data go through
    the network, the features generated by every layer in the network stack together to form the
    point features.
    """
    def __init__(self):
        super().__init__()
        self.input_transform = Tnet(k=3)
        self.feature_transform = Tnet(k=128)
```

```
            self.fc1 = nn.Conv1d(3,64,1)
            self.fc2 = nn.Conv1d(64,128,1)
            self.fc3 = nn.Conv1d(128,128,1)
            self.fc4 = nn.Conv1d(128,512,1)
            self.fc5 = nn.Conv1d(512,2048,1)


            self.bn1 = nn.BatchNorm1d(64)
            self.bn2 = nn.BatchNorm1d(128)
            self.bn3 = nn.BatchNorm1d(128)
            self.bn4 = nn.BatchNorm1d(512)
            self.bn5 = nn.BatchNorm1d(2048)

        def forward(self, input):
            """
            input: bs*3*n tensor
            output:
              outs: List of tensor, contain every output of the fc layer
              matrix3x3: bs*3*3 matrix output by t-net
              matrix128x128: bs*128*128 matrix output by t-net
            """
            #
            n_pts = input.size()[2]
            outs = []
            matrix3x3 = self.input_transform(input)
            xb = torch.bmm(torch.transpose(input,1,2), matrix3x3).transpose(1,2)
            out1 = F.relu(self.bn1(self.fc1(xb)))
            outs.append(out1)
            ########################################################################
            # TODO: Performs the forward pass of the SegTransform.
            # It is used to generate the point feature that is then feeded into the final MLP part.

            # Hint: Make sure adding batch normarlization layer and relu after every fc layer.
            # Hint: The final return output should be like [out1, out2, out3...].
            # Hint: In order to multiply the t-net output with the input, you need to adjust the shape of the input using torch.transpo
            # Hint: The global feature (output after maxpool) is repeated n times to concatenate to every local feature.
            #       You may find torch.repeat and torch.transpose useful here.
            ########################################################################
            #bs*64*n -> bs*128*n
            out2 = F.relu(self.bn2(self.fc2(out1)))
            outs.append(out2)
            out3 = F.relu(self.bn3(self.fc3(out2)))
            # 128->128
            outs.append(out3)
            #bs*128*128
            matrix128x128 = self.feature_transform(out3)
            #bs*n*128 * bs*128*128 -> bs*n*128 -> bs*128*n
            out4 = torch.bmm(torch.transpose(out3,1,2), matrix128x128).transpose(1,2)
            outs.append(out4)
            # 128->512
            out5 = F.relu(self.bn4(self.fc4(out4)))
            outs.append(out5)
            #512->2048
            xb = self.bn5(self.fc5(out5))
            #bs*2048*n -> bs*2048*1
            xb = nn.MaxPool1d(xb.size(-1))(xb)
            #bs*2048->n*bs*2048->2048*bs*n->bs*2048*n
            out6 = nn.Flatten(1)(xb).repeat(n_pts,1,1).transpose(0,2).transpose(0,1)#.repeat(1, 1, n_pts)
            outs.append(out6)
            ########################################################################

            return outs, matrix3x3, matrix128x128


    class PointNetSeg(nn.Module):
        """
        PointNetSeg is a class transform point feature into scores of each point
        In this dataset the category number is 4
        """
        def __init__(self, hidden_sizes_fc=[256, 256, 128, 4]):
            super().__init__()
            self.transform = SegTransform()
            self.hidden_sizes_fc=hidden_sizes_fc
            self.fc = self._build_fc()
            self.logsoftmax = nn.LogSoftmax(dim=1)

        def _build_fc(self):
            layers = []
            prev_size = 3008
            for layer_id, size in enumerate(self.hidden_sizes_fc):
                bn = nn.BatchNorm1d(size)
                fc = nn.Conv1d(prev_size, size, 1)
                layers.append(fc)
```

```
            layers.append(bn)
            layers.append(nn.ReLU())
            prev_size = size
        return nn.Sequential(*layers)

    def forward(self, input):
        """
        input: bs*3*n tensor
        output:
          out: logsoftmax score of each point, of shape bs*category_number*n
          matrix3x3: bs*3*3 matrix output by t-net
          matrix128x128: bs*128*128 matrix output by t-net
        """
        inputs, matrix3x3, matrix128x128 = self.transform(input)
        #bs*3008*n
        stack = torch.cat(inputs,1)
        output = self.fc(stack)

        return self.logsoftmax(output), matrix3x3, matrix128x128
```

Test SegTransform, PointNetSeg

```
_set_seed(2017)
model = SegTransform()

if count_parameters(model)!=6970057:
    print("Error")
    print("test_t_net parameters number = ", count_parameters(model))

assert count_parameters(model)==6970057

_set_seed(2017)

x1 = torch.randn(3, 3, 5)

y1=torch.tensor([0.2305, 0.1496, 0.0000, 0.2544, 0.0000, 2.1451, 0.0000, 0.0000, 0.9137, 0.0000])

pred_y1, _, _ = model(x1)
print(pred_y1[0].view(-1)[35:45])
assert torch.allclose(y1, pred_y1[0].view(-1)[35:45], rtol=1e-03, atol=1e-03),  "different y_pred and y"
print("SegTransform test pass!")
```

```
tensor([0.2305, 0.1496, 0.0000, 0.2544, 0.0000, 2.1451, 0.0000, 0.0000, 0.9137,
        0.0000], grad_fn=<SliceBackward0>)
SegTransform test pass!
```

```
_set_seed(2017)
model = PointNetSeg()

if count_parameters(model)!=7840853:
    print("Error")
    print("test_t_net parameters number = ", count_parameters(model))

assert count_parameters(model)==7840853
_set_seed(2017)
x1 = torch.randn(3, 3, 5)

y1=torch.tensor([-0.6844, -2.1168, -2.7130, -1.6787, -1.5990, -1.8179, -1.4520, -1.5434, -1.8525, -1.4774])
pred_y1, _, _ = model(x1)
assert torch.allclose(y1, pred_y1.view(-1)[35:45], rtol=1e-03, atol=1e-03),  "different y_pred and y"
print("PointNetSeg test pass!")
```

```
PointNetSeg test pass!
```

Training loop for segmentation

```
class ExperimentSeg:
    def __init__(self, train_data, val_data, model: nn.Module,
                 lr: float, save=True):
        self.train_data = train_data
        self.val_data = val_data
        self.model = model.cuda()
        self.optimizer = torch.optim.Adam(pointnet.parameters(), lr=lr)
        self.save=save
        self.loss=[]
        self.acc=[]

    def train(self, epochs):
```

```
            epoch_iterator = trange(epochs)
            # validation
            if self.val_data:
                self.evaluate("Initial ")
            for epoch in epoch_iterator:
                self.model.eval()
                correct = total = 0
                self.model.train()
                running_loss = 0.0
                data_iterator = tqdm(self.train_data)
                for i, data in enumerate(data_iterator, 0):
                    inputs, labels = data['image'].to(device), data['category'].to(device)
                    self.optimizer.zero_grad()
                    outputs, m3x3, m64x64 = self.model(inputs.transpose(1,2))

                    loss = self.getloss(outputs, labels, m3x3, m64x64)
                    loss.backward()
                    self.optimizer.step()

                    # print statistics
                    running_loss += loss.item()
                    if i % 10 == 9:    # print every 10 mini-batches
                        self.loss.append(running_loss / 10)
                        data_iterator.set_postfix(loss=running_loss / 10)
                        running_loss = 0.0

                # validation
                if self.val_data:
                    self.evaluate("Epoch:{} ".format(epoch+1))

                # save the model
                if self.save:
                    torch.save(self.model.state_dict(), "save_"+str(epoch)+".pth")

        def evaluate(self, ttype):
            self.model.eval()
            correct = total = 0
            with torch.no_grad():
                for data in self.val_data:
                    inputs, labels = data['image'].to(device).float(), data['category'].to(device)
                    outputs, __, __ = self.model(inputs.transpose(1,2))
                    _, predicted = torch.max(outputs.data, 1)
                    total += labels.size(0) * labels.size(1)
                    correct += (predicted == labels).sum().item()
                val_acc = 100. * correct / total
                self.acc.append(val_acc)
                print(ttype+'Valid accuracy: %d %%' % val_acc)
                #epoch_iterator.set_postfix(val_acc=val_acc)

        def getloss(self, outputs, labels, m3x3, m128x128, alpha = 0.0001):
            criterion = torch.nn.NLLLoss()
            bs=outputs.size(0)
            id3x3 = torch.eye(3, requires_grad=True).repeat(bs,1,1)
            id128x128 = torch.eye(128, requires_grad=True).repeat(bs,1,1)
            if outputs.is_cuda:
                id3x3=id3x3.cuda()
                id128x128=id128x128.cuda()
            diff3x3 = id3x3-torch.bmm(m3x3,m3x3.transpose(1,2))
            diff128x128 = id128x128-torch.bmm(m128x128,m128x128.transpose(1,2))
            return criterion(outputs, labels) + alpha * (torch.norm(diff3x3)+torch.norm(diff128x128)) / float(bs)

        def plt_loss(self):
          x1 = range(0, len(self.loss))
          y1 = self.loss
          plt.plot(x1, y1, 'o-')
          plt.title('Test loss vs. epoches')
          plt.ylabel('Test loss')
          plt.show()

        def plt_accuracy(self):
          x1 = range(0, len(self.acc))
          y1 = self.acc
          plt.plot(x1, y1, 'o-')
          plt.title('Validation accuracy vs. epoches')
          plt.ylabel('Validation accuracy')
          plt.show()

        def visualize_pred_label(self):
          for data in self.val_data:
              inputs, labels = data['image'].to(device), data['category'].to(device)
              outputs, __, __ = self.model(inputs.transpose(1,2))
              _, predicted = torch.max(outputs.data, 1)
              index=random.randint(0, len(data['image']))
```

```
            inputs, labels, predicted = inputs.cpu(), labels.cpu(), predicted.cpu()
            print("True Label visualization:\n")
            segpcshow(*inputs[index].T, labels[index])
            print("Predicted Label visualization:\n")
            segpcshow(*inputs[index].T, predicted[index])
            break
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

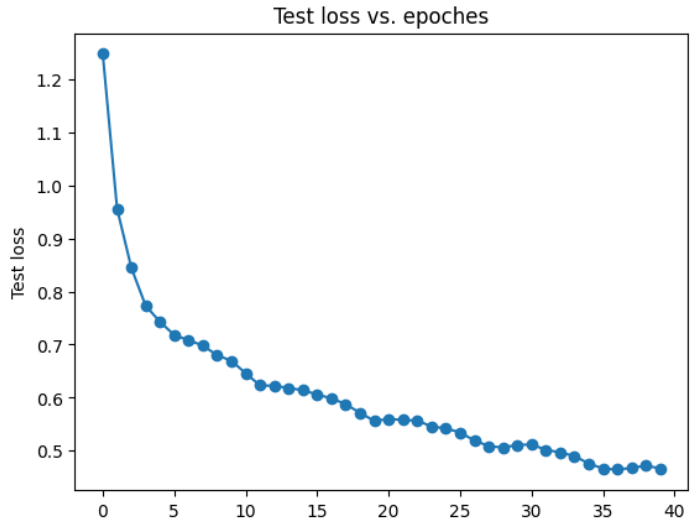
```
    cuda:0
```

```
pointnet = PointNetSeg()
pointnet.to(device)
segexp = ExperimentSeg(Seg_train_loader, Seg_val_loader, pointnet, 0.001, save=False)
segexp.train(5)
```

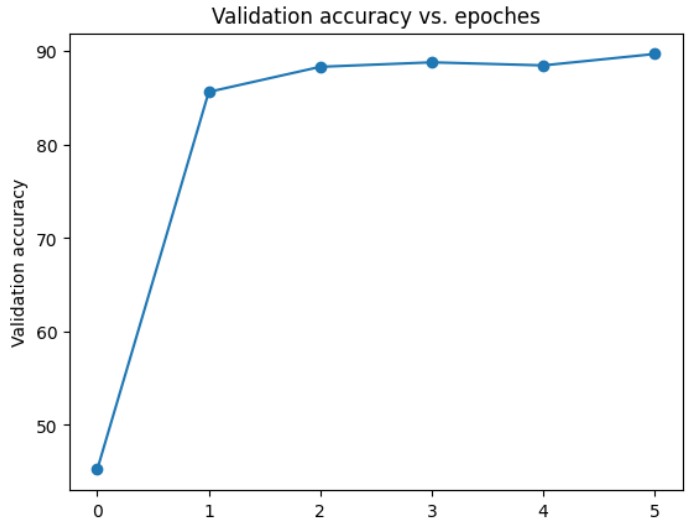
```
100%                                    5/5 [06:27<00:00, 76.38s/it]
Initial Valid accuracy: 45 %
100%                                    80/80 [01:13<00:00, 1.18it/s, loss=0.698]
Epoch:1 Valid accuracy: 85 %
100%                                    80/80 [01:11<00:00, 1.18it/s, loss=0.606]
Epoch:2 Valid accuracy: 88 %
100%                                    80/80 [01:17<00:00, 1.22it/s, loss=0.546]
Epoch:3 Valid accuracy: 88 %
100%                                    80/80 [01:12<00:00, 1.23it/s, loss=0.502]
Epoch:4 Valid accuracy: 88 %
100%                                    80/80 [01:11<00:00, 1.22it/s, loss=0.467]
Epoch:5 Valid accuracy: 89 %
```

Visualize the result

```
segexp.plt_loss()
```

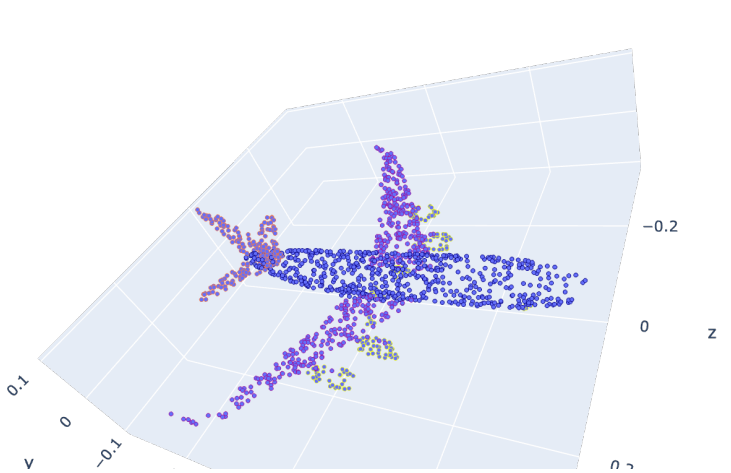


```
segexp.plt_accuracy()
```

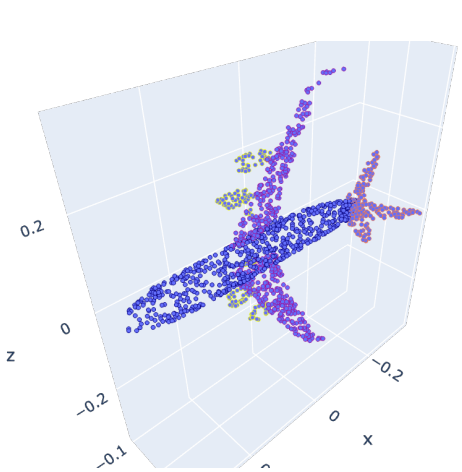


```
segexp.visualize_pred_label()
```

```
True Label visualization:
```




```
Predicted Label visualization:
```



Colab paid products - Cancel contracts here

✓ 0s completed at 6:42 PM

4 Team Member Contribution

Task	Member
DataSet	Songlin Zhao
Classifier Model	Zhen Tong
Segmentation Model	Jianwen Chen, Junzhi Chen
Written Homework	All

5 Additional Links

Github link for additional materials (including .tex sources for written homeworks, coding notebooks with and without solution): https://github.com/thiefCat/CS182-Project