PointNet

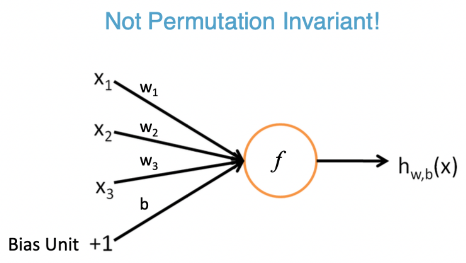
Created time	@September 21, 2022 2:13 PM
Last edited time	@November 24, 2022 1:36 AM
Date	@September 21, 2022
Tags	
Status	Done

PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

> Abstract : We design a novel type of neural network that directly consumes PointClouds, which respects the permutation invariance of points in the input

Problem of MLPs applied on PointClouds

- 点云无序性 (points in the cloud don’t have an explicit order)



$$f(w_1x_3 + w_2x_1 + w_3x_2 + b) \neq h_{W,b}(x)$$

- 从图中可以看出，当我们交换输入点的顺序，点云仍然表示同一个物体，但是MLP的输出会发生变化（有时这个变化是剧烈的）。我们不希望网络受到输入顺序的影响，反之网络对于相同物体应该具有相同输出，即交换不变的（Permutation Invariant）
- From the figure above, we see that when we permute the input points, the output of the MLP layer changes (sometimes intensely), while the PointCloud is still representing the same object. We’d like to eliminate the undesirable impact of input order as the network should output the same result for the same object (Permutation Invariance).

Contributions of PointNet:

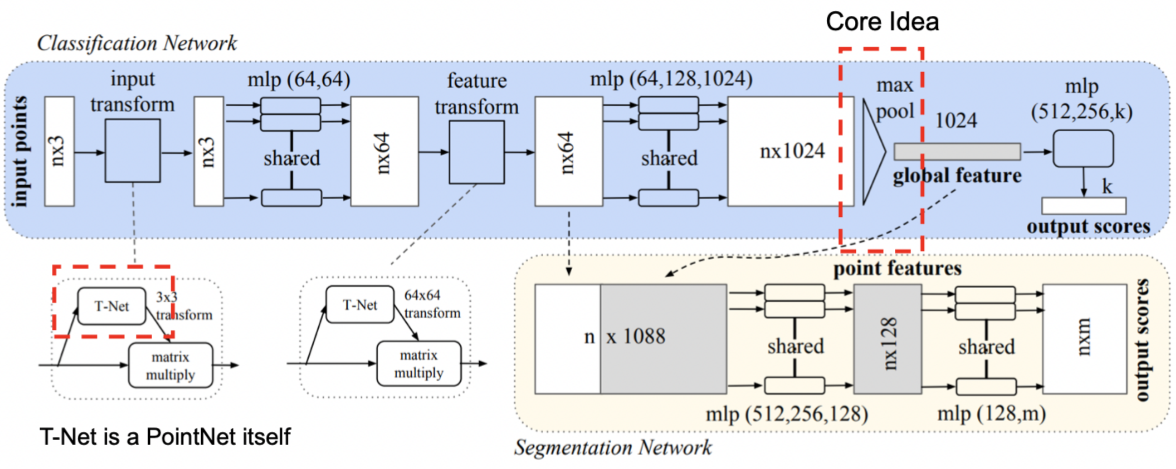
> PointNet provides a unified architecture for applications ranging from object classification, par segmentation, to scene semantic parsing. Though simple, PointNet is highly efficient and

> effective. We provide analysis towards what the network has learnt and why the network is robust with respect to input perturbation and corruption.

1. 我们设计了一个新颖的深层网络架构来处理三维中的无序点集
2. 我们设计的网络表征可以做三维图形分类、图形的局部分割以及场景的语义分割等任务
3. 我们提供了完备的经验和理论分析来证明PointNet的稳定和高效。
4. 充分的消融实验，证明网络各个部分对于表征的有效性。

PointNet

PointNet Architecture



PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. CR Qi et.al.

设计思路

PointNet 的几个特点：

1. 无序性 --> 对称函数设计用于表征 (shared MLP + MaxPool)
2. 点不是孤立的，需要考虑局部结构 --> 局部全局特征结合
3. 仿射变换无关性 --> alignment network

Proposed Solutions :

1. unordered input —> **Symmetry Function for Unordered Input**（shared MLP + MaxPool)
2. points aren't not isolated, neighborhoods can contain infos —> local and global feature aggregation
3. invariant to affine transformation —> alignement network

具体分析：

对点云点排列不变性有几种思路：

1. 直接将点云中的点以某种顺序输入（比如按照坐标轴从小到大这样）

   为什么不这样做？（摘自原文）in high dimensional space there in fact does not exist an ordering that is stable w.r.t. point perturbations in the general sense.简单来说就是很难找到一种稳定的排序方法

2. 作为序列去训练一个RNN，即使这个序列是随机排布的，RNN也有能力学习到排布不变性.

   为什么不这样做？（摘自原文）While RNN has relatively good robustness to input ordering for sequences with small length (dozens), it's hard to scale to thousands of input elements, which is the common size for point sets. RNN很难处理好成千上万长度的这种输入元素（比如点云）

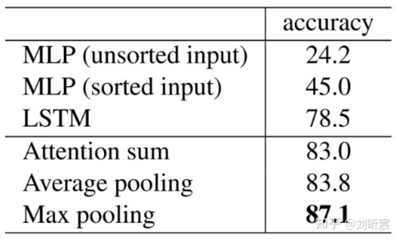
3. 使用一个简单的对称函数去聚集每个点的信息

   $f(x_1, x_2, ..., x_n) = g(h(x_1), h(x_2), ..., h(x_n))$

   目标：左边 f是我们的目标，右边 g是我们期望设计的对称函数。由上公式可以看出，基本思路就是对各个元素（即点云中的各个点）使用 h分别处理，在送入对称函数 g 中处理，以实现排列不变性。

   在实现中 h 就是MLP， g 就是max pooling

   实验比较：



	accuracy
MLP (unsorted input)	24.2
MLP (sorted input)	45.0
LSTM	78.5
Attention sum	83.0
Average pooling	83.8
Max pooling	**87.1**

局部与全局信息聚合 Local and Global Information Aggregation：

对于分割任务，我们需要point-wise feature 。 因此分割网络和分类网络设计局部略有不同，分割网络添加了每个点的**local和global特征的拼接过程**，以此得到同时对局部信息和全局信息感知的point-wise特征，提升表征效果。

仿射变换、刚体变换等变换的无关性 alignment network：

直接的思路：将所有的输入点集对齐到一个统一的点集空间

PN的做法：直接**预测一个变换矩阵**（3*3）来处理输入点的坐标(dim = 3)。因为会有数据增强的操作存在，这样做可以在一定程度上保证网络可以学习到变换无关性。

特征空间 (dim >> 3) 的对齐也可以这么做，但是需要注意：

transformation matrix in the feature space has much higher dimension than the spatial transform matrix, which greatly increases the difficulty of optimization. 对于特征空间的alignment network，由于特征空间维度比较高，因此直接生成的alignment matrix会维度特别大，不好优化，因此这里需要加个loss约束一下。

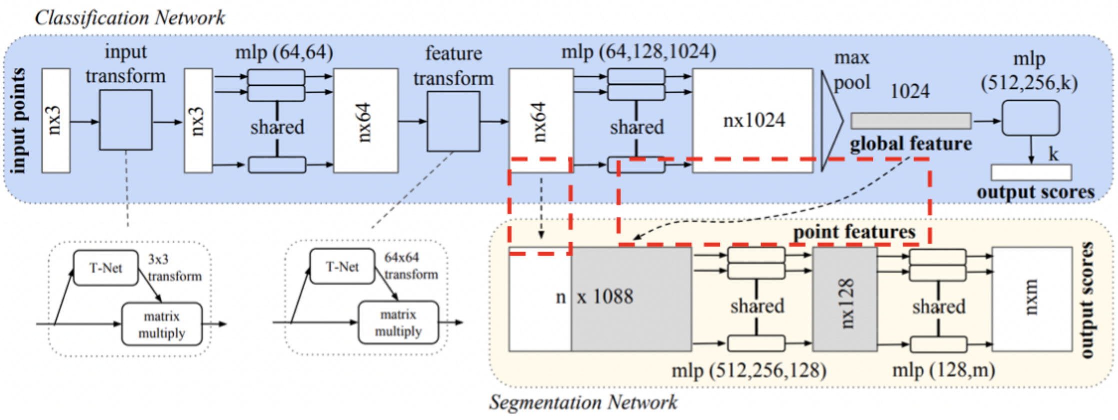
add a regularization term to our softmax training loss：

$L_{reg} = ||I - AA^T||_F^2$

使得特征空间的变换矩阵A尽可能接近正交矩阵

PointNet – Segmentation

- Segmentation is per-point classification
  - MLP on per-point feature, instead of global feature
  - How to get per-point feature?



网络架构与数据流

- 首先忽略 Tnet 和 feature transformation
- 每个点 nx3 独立地经过一个共享的MLP，得到 nx64，nx1024 个特征
- Maxpool 对nx1024 每一列 feature 取element-wise max 得到一个 长度为1024 的全局特征。注意：Max操作不受点的顺序影响
- 对于分割网络，将经过feature transformation 操作的输出 nx64 （每行不同）与全局特征 nx1024（每行都相同）拼接，经过一个两层MLP，降为到nx128，再经过一个MLP 调整输出维度

数学证明

> PointNet 被证明可以拟合任何的作用在点云上的函数

问题转化：对于任意连续方程f，f的输入点云S，都存在一个 **全局MLP** 和一个**共享MLP** 满足如下方程：

PointNet – Proof

- Given continuous $f: \chi \to \mathbb{R}$
- $\forall \epsilon > 0, \exists h: \mathbb{R}^m \to \mathbb{R}^{m'}, and\ \gamma: \mathbb{R}^n \to \mathbb{R}$
- s.t. $\forall S \in \chi, e.g. S = \{x_1, \cdots, x_n\}, x_i \in \mathbb{R}^m$

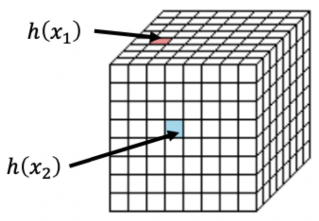
$$\left| f(S) - \underbrace{\gamma}_{\text{MLP for global feature}} \left(MAX\left(\underbrace{h}_{\text{Shared MLP}}(x_1), \cdots, h(x_n) \right) \right) \right| < \epsilon$$

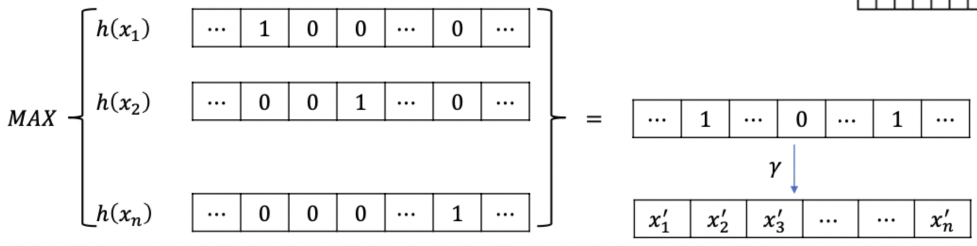
证明步骤：

PointNet – Proof

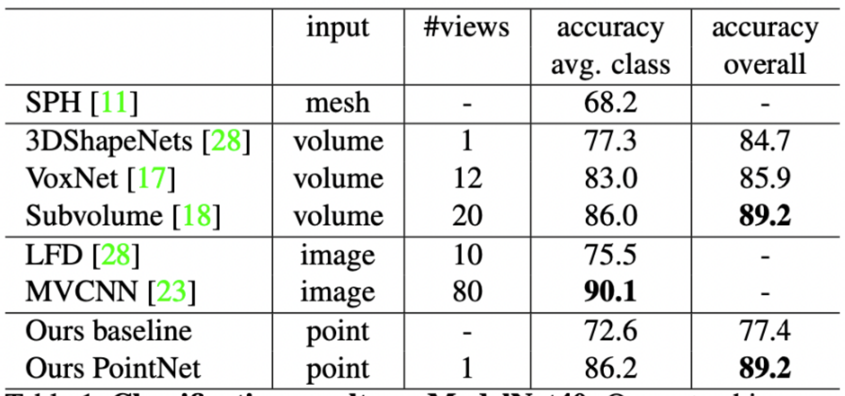
$$\left|f(S) - \gamma\left(MAX\left(h(x_1),\cdots,h(x_n)\right)\right)\right| < \epsilon$$

- $h(\cdot)$ maps x_i to the some deterministic position of a huge vector
  - By Voxel Grid Downsampling
- $MAX\left(h(x_1),\cdots,h(x_n)\right)$ simply builds a voxel grid representation.
  - There will be lots of 0 elements because of empty cells in voxel grid.
- $\gamma(\cdot) = reconstruct\ the\ points + f(\cdot)$
- Done




MAX
$h(x_1)$
$h(x_2)$
$h(x_n)$
$\cdots$ 1 0 0 $\cdots$ 0 $\cdots$
$\cdots$ 0 0 1 $\cdots$ 0 $\cdots$
$\cdots$ 0 0 0 $\cdots$ 1 $\cdots$
=
$\cdots$ 1 $\cdots$ 0 $\cdots$ 1 $\cdots$
γ
x'_1 x'_2 x'_3 $\cdots$ $\cdots$ x'_n

PointNet – Classification



	input	#views	accuracy avg. class	accuracy overall
SPH [11]	mesh	-	68.2	-
3DShapeNets [28]	volume	1	77.3	84.7
VoxNet [17]	volume	12	83.0	85.9
Subvolume [18]	volume	20	86.0	**89.2**
LFD [28]	image	10	75.5	-
MVCNN [23]	image	80	**90.1**	-
Ours baseline	point	-	72.6	77.4
Ours PointNet	point	1	86.2	**89.2**

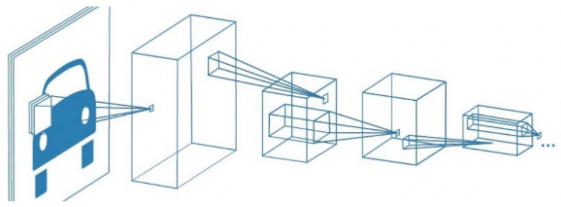
Table 1. **Classification results on ModelNet40.** Our net achieves state-of-the-art among deep nets on 3D input.

Limitations of PointNet

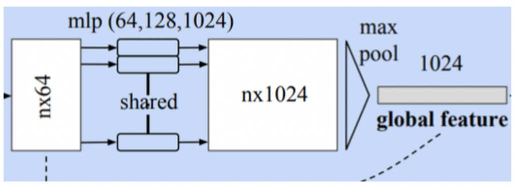
Lack of hierarchical feature aggregation

- CNN has multiple, increasing receptive field
- PointNet has one receptive field – all points

CNN



PointNet



—> PointNet ++

```
input tensor n*3

for each point

layer 1 : MLP
          linear(3,64)
          linear(64,64)
layer 2 : MLP
          linear(64,128)
          linear(128,256)
layer 3 : Maxpool
```

```
from __future__ import print_function
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.utils.data
from torch.autograd import Variable
import numpy as np
import torch.nn.functional as F


class STN3d(nn.Module):
    def __init__(self):
        super(STN3d, self).__init__()
        self.conv1 = torch.nn.Conv1d(3, 64, 1)
        self.conv2 = torch.nn.Conv1d(64, 128, 1)
        self.conv3 = torch.nn.Conv1d(128, 1024, 1)
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 9)
        self.relu = nn.ReLU()

        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(1024)
        self.bn4 = nn.BatchNorm1d(512)
        self.bn5 = nn.BatchNorm1d(256)


    def forward(self, x):
        batchsize = x.size()[0]
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn3(self.conv3(x)))
        x = torch.max(x, 2, keepdim=True)[0] ## returns values and indices
        x = x.view(-1, 1024)

        x = F.relu(self.bn4(self.fc1(x)))
        x = F.relu(self.bn5(self.fc2(x)))
        x = self.fc3(x)

        iden = Variable(torch.from_numpy(np.array([1,0,0,0,1,0,0,0,1]).astype(np.float32))).view(1,9).repeat(batchsize,1)
        if x.is_cuda:
            iden = iden.cuda()
        x = x + iden
        x = x.view(-1, 3, 3)
        return x


class STNkd(nn.Module):
    def __init__(self, k=64):
        super(STNkd, self).__init__()
        self.conv1 = torch.nn.Conv1d(k, 64, 1)
        self.conv2 = torch.nn.Conv1d(64, 128, 1)
        self.conv3 = torch.nn.Conv1d(128, 1024, 1)
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, k*k)
        self.relu = nn.ReLU()

        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(1024)
        self.bn4 = nn.BatchNorm1d(512)
        self.bn5 = nn.BatchNorm1d(256)

        self.k = k

    def forward(self, x):
        batchsize = x.size()[0]
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn3(self.conv3(x)))
        x = torch.max(x, 2, keepdim=True)[0]
        x = x.view(-1, 1024)

        x = F.relu(self.bn4(self.fc1(x)))
        x = F.relu(self.bn5(self.fc2(x)))
        x = self.fc3(x)

        iden = Variable(torch.from_numpy(np.eye(self.k).flatten().astype(np.float32))).view(1,self.k*self.k).repeat(batchsize,1)
        if x.is_cuda:
            iden = iden.cuda()
        x = x + iden
        x = x.view(-1, self.k, self.k)
        return x

class PointNetfeat(nn.Module):
    def __init__(self, global_feat = True, feature_transform = False):
        super(PointNetfeat, self).__init__()
        self.stn = STN3d()
        self.conv1 = torch.nn.Conv1d(3, 64, 1)
        self.conv2 = torch.nn.Conv1d(64, 128, 1)
        self.conv3 = torch.nn.Conv1d(128, 1024, 1)
        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(1024)
        self.global_feat = global_feat
        self.feature_transform = feature_transform
        if self.feature_transform:
            self.fstn = STNkd(k=64)

    def forward(self, x):
        n_pts = x.size()[2]
        trans = self.stn(x)
        x = x.transpose(2, 1)
        x = torch.bmm(x, trans)
        x = x.transpose(2, 1)
        x = F.relu(self.bn1(self.conv1(x)))

        if self.feature_transform:
            trans_feat = self.fstn(x)
            x = x.transpose(2,1)
            x = torch.bmm(x, trans_feat)
            x = x.transpose(2,1)
        else:
            trans_feat = None
        pointfeat = x
        x = F.relu(self.bn2(self.conv2(x)))
        x = self.bn3(self.conv3(x))
        x = torch.max(x, 2, keepdim=True)[0]
        x = x.view(-1, 1024)
        if self.global_feat:
            return x, trans, trans_feat
        else:
            x = x.view(-1, 1024, 1).repeat(1, 1, n_pts)
            return torch.cat([x, pointfeat], 1), trans, trans_feat

class PointNetCls(nn.Module):
    def __init__(self, k=2, feature_transform=False):
        super(PointNetCls, self).__init__()
        self.feature_transform = feature_transform
        self.feat = PointNetfeat(global_feat=True, feature_transform=feature_transform)
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, k)
        self.dropout = nn.Dropout(p=0.3)
        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.relu = nn.ReLU()

    def forward(self, x):
        x, trans, trans_feat = self.feat(x)
        x = F.relu(self.bn1(self.fc1(x)))
        x = F.relu(self.bn2(self.dropout(self.fc2(x))))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1), trans, trans_feat


class PointNetDenseCls(nn.Module):
    def __init__(self, k = 2, feature_transform=False):
        super(PointNetDenseCls, self).__init__()
        self.k = k
        self.feature_transform=feature_transform
        self.feat = PointNetfeat(global_feat=False, feature_transform=feature_transform)
        self.conv1 = torch.nn.Conv1d(1088, 512, 1)
        self.conv2 = torch.nn.Conv1d(512, 256, 1)
        self.conv3 = torch.nn.Conv1d(256, 128, 1)
        self.conv4 = torch.nn.Conv1d(128, self.k, 1)
        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.bn3 = nn.BatchNorm1d(128)

    def forward(self, x):
        batchsize = x.size()[0]
        n_pts = x.size()[2]
        x, trans, trans_feat = self.feat(x)
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn3(self.conv3(x)))
        x = self.conv4(x)
        x = x.transpose(2,1).contiguous()
        x = F.log_softmax(x.view(-1,self.k), dim=-1)
        x = x.view(batchsize, n_pts, self.k)
        return x, trans, trans_feat

def feature_transform_regularizer(trans):
    d = trans.size()[1]
    batchsize = trans.size()[0]
    I = torch.eye(d)[None, :, :]
    if trans.is_cuda:
        I = I.cuda()
    loss = torch.mean(torch.norm(torch.bmm(trans, trans.transpose(2,1)) - I, dim=(1,2)))
    return loss

if __name__ == '__main__':
    sim_data = Variable(torch.rand(32,3,2500))
    trans = STN3d()
    out = trans(sim_data)
    print('stn', out.size())
    print('loss', feature_transform_regularizer(out))

    sim_data_64d = Variable(torch.rand(32, 64, 2500))
    trans = STNkd(k=64)
    out = trans(sim_data_64d)
    print('stn64d', out.size())
    print('loss', feature_transform_regularizer(out))

    pointfeat = PointNetfeat(global_feat=True)
    out, _, _ = pointfeat(sim_data)
    print('global feat', out.size())

    pointfeat = PointNetfeat(global_feat=False)
    out, _, _ = pointfeat(sim_data)
    print('point feat', out.size())

    cls = PointNetCls(k = 5)
    out, _, _ = cls(sim_data)
    print('class', out.size())

    seg = PointNetDenseCls(k = 3)
    out, _, _ = seg(sim_data)
    print('seg', out.size())
```

```
import torch
import torch.nn as nn
import numpy as np
import torch.nn.functional as F

class Tnet(nn.Module):
    def __init__(self, k=3):
        super().__init__()
        self.k=k
        self.conv1 = nn.Conv1d(k,64,1)
        self.conv2 = nn.Conv1d(64,128,1)
        self.conv3 = nn.Conv1d(128,1024,1)
        self.fc1 = nn.Linear(1024,512)
        self.fc2 = nn.Linear(512,256)
        self.fc3 = nn.Linear(256,k*k)

        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(1024)
        self.bn4 = nn.BatchNorm1d(512)
        self.bn5 = nn.BatchNorm1d(256)


    def forward(self, input):
        # input.shape == (bs,n,3)
        bs = input.size(0)
        xb = F.relu(self.bn1(self.conv1(input)))
        xb = F.relu(self.bn2(self.conv2(xb)))
        xb = F.relu(self.bn3(self.conv3(xb)))
        pool = nn.MaxPool1d(xb.size(-1))(xb)
        flat = nn.Flatten(1)(pool)
        xb = F.relu(self.bn4(self.fc1(flat)))
        xb = F.relu(self.bn5(self.fc2(xb)))

class Transform(nn.Module):
    def __init__(self):
        super().__init__()
        self.input_transform = Tnet(k=3)
        self.feature_transform = Tnet(k=64)
        self.conv1 = nn.Conv1d(3,64,1)

        self.conv2 = nn.Conv1d(64,128,1)
        self.conv3 = nn.Conv1d(128,1024,1)


        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(1024)

    def forward(self, input):
        matrix3x3 = self.input_transform(input)
        # batch matrix multiplication
        xb = torch.bmm(torch.transpose(input,1,2), matrix3x3).transpose(1,2)

        xb = F.relu(self.bn1(self.conv1(xb)))

        matrix64x64 = self.feature_transform(xb)
        xb = torch.bmm(torch.transpose(xb,1,2), matrix64x64).transpose(1,2)

        xb = F.relu(self.bn2(self.conv2(xb)))
        xb = self.bn3(self.conv3(xb))
        xb = nn.MaxPool1d(xb.size(-1))(xb)
        output = nn.Flatten(1)(xb)
        return output, matrix3x3, matrix64x64



class PointNet(nn.Module):
    def __init__(self, classes = 40):
        super().__init__()
        self.transform = Transform()
        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(p=0.3)
        self.logsoftmax = nn.LogSoftmax(dim=1)

    def forward(self, input):
        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)
        return self.logsoftmax(output), matrix3x3, matrix64x64
        #initialize as identity
        init = torch.eye(self.k, requires_grad=True).repeat(bs,1,1)
        if xb.is_cuda:
            init=init.cuda()
        matrix = self.fc3(xb).view(-1,self.k,self.k) + init
        return matrix
```