PointNet论文复现及代码详解

写在前面

本文主要对PointNet（之前有解读论文）的代码进行了分析和解读，有助于进一步理解其思想。可以发现，PointNet的结构并不复杂，比起CNN还要简单一些。理解PointNet关键在于理解一维卷积在网络中的作用，本文对该部分进行了详细的说明。另外，可以看到，PointNet最大的缺陷就是没有考虑周围的局部信息，所有的卷积操作都是针对单个点的进行的。这一点在PointNet++中得到了关注，后面的文章会对其进行解读。

1. 代码下载

这部分很简单啦，github上作者放出了TensorFlow的版本，这里使用的是Pytorch的版本，链接如下：PointNet-Pytorch代码。

按照页面的指示把代码和数据集下载到本地。

Download data and running

```
git clone https://github.com/fxia22/pointnet.pytorch
cd pointnet.pytorch
pip install -e .
```

Download and build visualization tool

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


2. 数据集

首先看一下数据集到底是什么样的，这里用的包含16类样本的ShapeNet。里面有好多个文件夹，每个文件夹里面放着同一类的样本，每个文件夹对应类别如下：

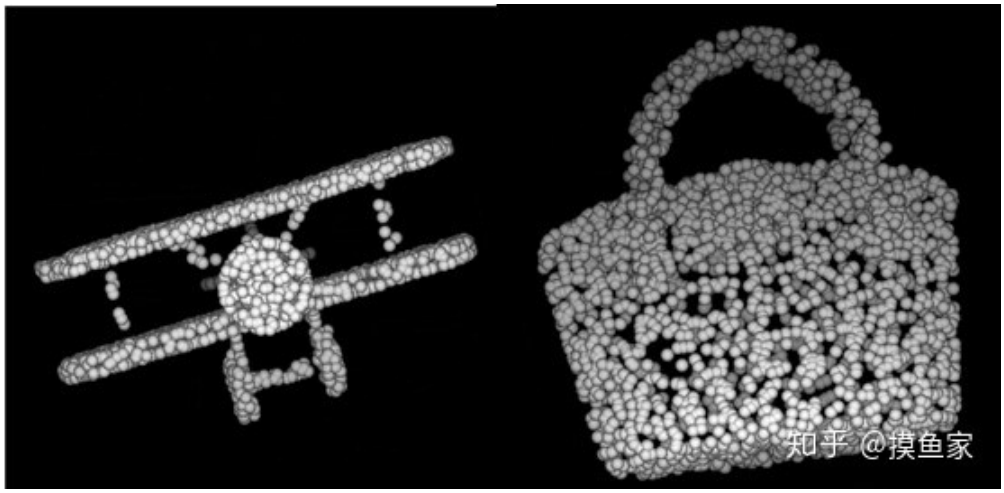
```
Airplane        02691156
Bag     02773838
Cap     02954340
Car     02958343
Chair   03001627
Earphone        03261776
Guitar  03467517
Knife   03624134
Lamp    03636649
Laptop  03642806
Motorbike       03790512
Mug     03797390
Pistol  03948459
Rocket  04099429
Skateboard      04225987
Table   04379243
```


打开第一个Airplane的文件夹，里面很多.pts格式的文件，这就是不同飞机模型的点云格式，

里面放的就是一个个坐标点，坐标是经过归一化的。

```
0.09869 0.16609 -0.15352
0.09074 -0.13189 -0.08192
0.08266 0.12005 -0.08345
0.15621 0.16975 0.19691
0.09576 -0.12031 -0.08290
0.04417 -0.02100 0.23305
0.07133 -0.02100 0.26592
0.07855 -0.02100 0.26991
0.06474 0.09330 0.06324
0.08311 -0.12156 0.04090
0.02842 0.15944 0.28260
0.02905 0.15084 0.33599
0.02913 0.15681 0.34314
0.06747 -0.11991 -0.06018
0.14266 -0.02221 -0.01147
0.23003 -0.02897 -0.01491
0.26063 -0.03138 -0.01572
0.22522 -0.02797 -0.01947
0.00521 -0.01066 -0.01306
```

然后使用下载好的文件里面的一个可视化代码，稍微修改一下，看一看数据到底长什么样。下图是一个飞机和一个包。



针对分类问题，在训练时直接读取点数据及类别。在dataloader里面，对每个样本的坐标进行了中心化及随机增强。另外，**值得注意的一点是**，在输入网络训练之前，还对样本进行了特定数量的随机重采样，目的就是为了保证输入到网络的样本的点的数量都是一样的，所以严格来说，网络对于样本中点的数量并不是没有要求。

3. 模型及实现

a. 关于一维卷积conv1d

在PointNet中，用到的卷积操作都是conv1d，和卷积神经网络用到的conv2d有些不同，这里先介绍一下conv1d操作原理，这样能更好的理解数据在PointNet中是怎么变化的。

让图像变成更高分辨率，处理非均衡数据

首先，在实验中对所有样本进行了2500个点的重新采样，那么每个样本的尺寸就是2500*3，为了进行卷积操作，输入网络前进行了转置，同时为了方便理解，假定batch size = 1，那么最终输入到网络的样本尺寸就是1*3*2500，其中1是batch size，即一个样本，如下图。

0.1	0.5	...	0.4	0.5
0.3	0.3	...	0.6	0.3
0.5	0.2	...	0.1	0.2

假设做的第一步卷积操作是conv1 = torch.nn.Conv1d(3, 64, 1)，即输入通道=3，输出通道=64，卷积核大的大小为1，卷积核第二个维度是由in_channels来决定的，所以实际上卷积大小为in_channels*kerner_size，这里为3*1。

进一步理解一下，在第一个卷积层中，使用conv1对x进行卷积的时候，实际上是使用64个3*1的卷积核进行卷积，输出尺寸应该是1*64*2500，其中1还是batch size。

后面还会接着进行多层的卷积，可以看到，样本的坐标从刚开始的3维（xyz），经过多次卷积之后会变为1024维。

这里可以稍微联想一下，在卷积神经网络中，因为图像矩阵本身包含了位置信息，每一次卷积都会考虑周边像素的值，所以每一次卷积后提取出的特征一般都是综合了像素值和周围邻近像素的信息从而得到更高维的特征。比如图像第一层卷积后一般都会提取到边缘，纹理等特征。但在这里的卷积，可以看到，它只是对一个点的坐标进行卷积，完全没有考虑其他点的信息（很难做到），所以我们也很难感性的理解一个3维的坐标卷积之后得到的是什么特征。专门看了一下在文本分类中的一维卷积，和这里还有点不同，挺有意思，具体的分析放在最后的延伸部分。

b. 网络细节

在了解了一维卷积之后，网络就变得很简单了。对于分类问题，如果把batch size记为n，样本在网络中的变化就是n*3*2500 ——> n*64*2500 ——> n*128*2500 ——> n*1024*2500 ——> n*1024*1(max pooling后) ——> n*512*1 ——> n*256*1 ——> n*16*1 (本次实验样本共有16类)。

在整体之外，还有两个分支网络做Transform（T-Net)。第一个T-Net是在数据刚输入的时候，结构和上面的过程基本一致，该分支对样本进行一系列卷积及FC后输出是一个3*3的矩阵，然后先用样本和该矩阵相乘，根据论文的说法是对样本进行旋转对齐。第二个T-Net是加在第一次卷积之后的（即n*64*2500），得到一个64*64的变换矩阵，对n*64*2500的样本进行变换。

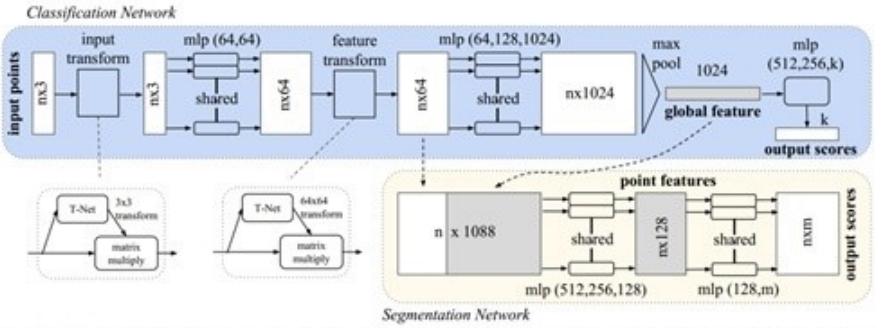
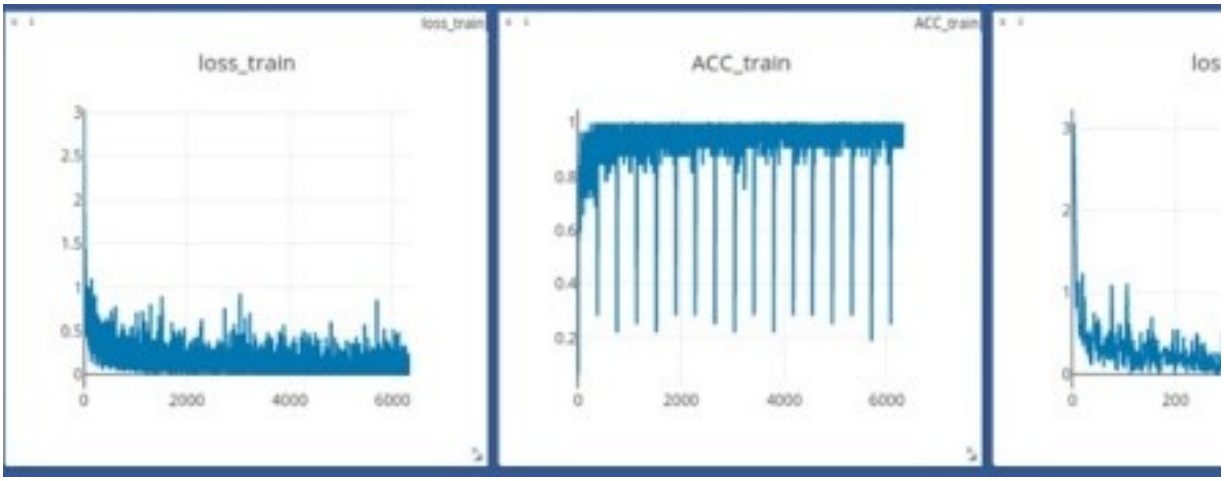


Figure 2. **PointNet Architecture.** The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

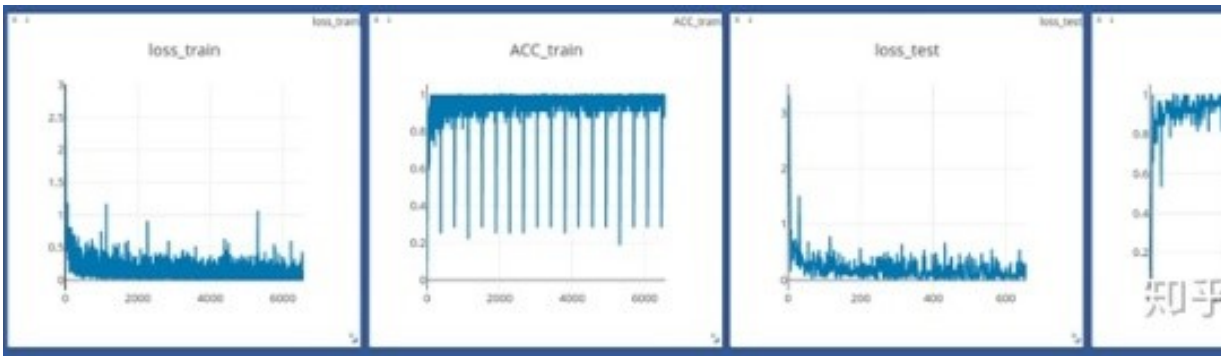
通过实验发现, 对于所用的16类的ShapeNet数据集，在分类的时候，T-Net并没有太大的作用，是否有T-Net对于网络的最终结果几乎没有影响。具体可以参见结果部分。

4. 结果

a. 使用T-Net分类



b. 未使用T-Net分类

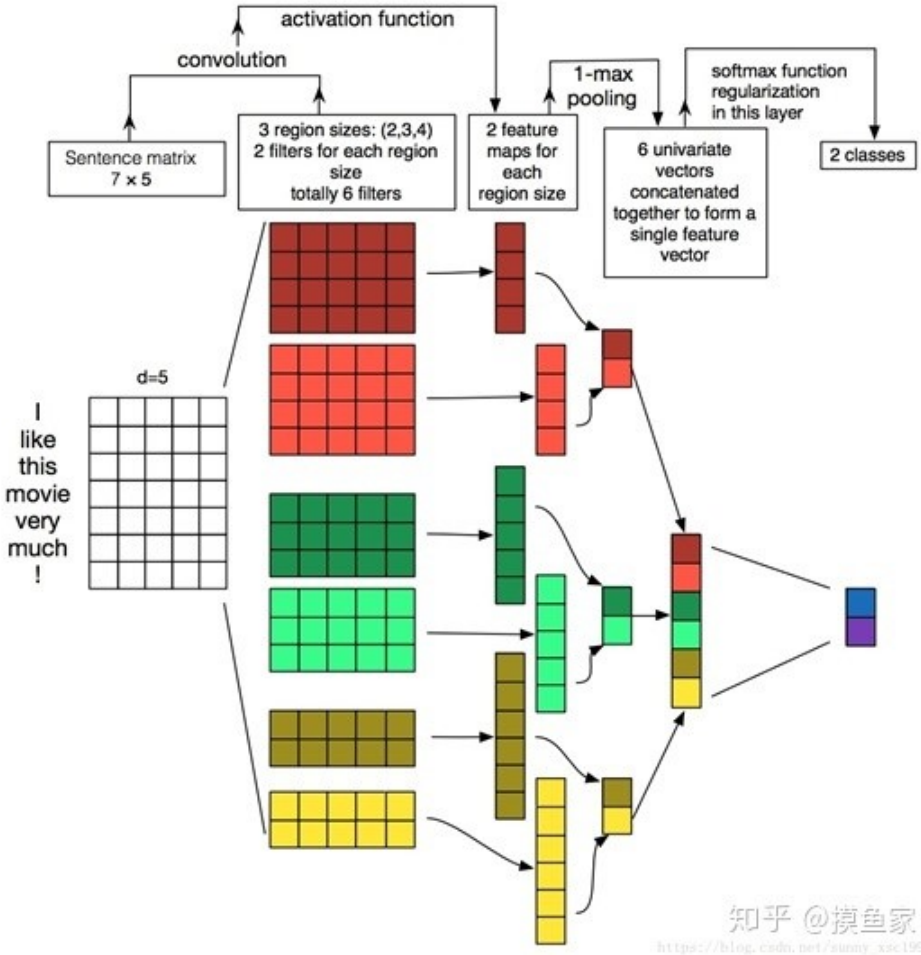


5. 延伸

在NLP的文本分类中，用的也是一维卷积，但和PointNet中卷积核大小总是1不同，文本分类网络中卷积核大小很多时候是大于1的。如下图例子所示，样本大小是7*5,每个词向量大小是5，对应点云中点的坐标维度是3。7代表有7个词，对应点云样本有2500个点。上面提到，在PointNet中所有卷积核的大小都是1， 也就是每层卷积都是对单个点样本进行处理，并没有考虑别的点。但在下面的例子的一维卷积中，卷积核可以不同。比如可以是2,3,4等。

下图深红色的就是一个尺寸为4的卷积核，结果是一个4*1的输出。在这个卷积过程中，每一个输出值都是同时考虑了4个词向量的值，所以会提取到词之间的关系特征。但类似的做法并不能在PointNet中实现。原因也很简单，在文本分类中，每个词向量的关系是可以确定了，相邻

的词向量本身代表了一种现实中的前后关系，但对于点云样本，在存储时位置相邻的点并不意味这在实际的空间中是相邻的，这一点是很难保证的。



但不可否认的是，在卷积过程中无法考虑周围点的信息会严重减低网络的能力，尽管在我们的例子中分类精度很高，但那只是因为样本比较简单而已，一旦加入了背景噪声，难度就很大了，这个问题应该是点云网络一个比较大的挑战了。

值得注意的是，在PointNet++中，已经关注了到了局部信息的问题，并尝试去解决，在下一篇中，会进行解读。

——>《PointNet++论文及代码详解》

Reference

https://towardsdatascience.com/understanding-machine-learning-on-point-clouds-through-pointnet-f8f3f2d53cc3
https://www.qwertee.io/blog/deep-learning-with-point-clouds/