01 - 从PointNet到PointNet++理论及pytorch代码

2021年1月1日 20:28

文章地址： https://blog.csdn.net/weixin_39373480/article/details/88878629

PointNet出现之前，点云上的深度学习模型，大概分成3类：

1. 基于3DCNN的体素模型；
2. 将点云映射到2D空间，利用CNN分类；
3. 利用传统的人工点云特征分类

- normal 法向量
- intensity 激光雷达的采样的时候一种特性强度信息的获取是激光扫描仪接受装置采集到的回波强度，此强度信息与目标 的表面材质、粗糙度、入射角方向，以及仪器的发射能量，激光波长有关
- local density 局部稠密度
- local curvature 局部曲率
- linearity, planarity and scattering propesed by Dimensionality based scale selection in 3D lidar point clouds
- verticality feature proposed by Weakly supervised segmentation-aided classification of urban scenes from 3d LiDAR point clouds

启示：各种不同的的传统角度的改进，大概就是从上面这些方向中来的。

点云的两个重要特征：

Permutation Invariance

- Point cloud is a set of unordered points

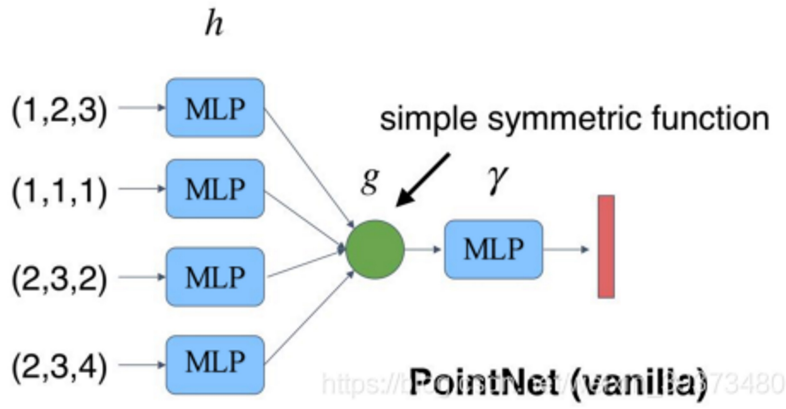
Transformation Invariance

- Point cloud rotations should not alter classification results

置换不变性：点的排序不影响物体的性质；
旋转不变性：给点云一个旋转，所有的坐标都变了，但表示的还是同一个物体。

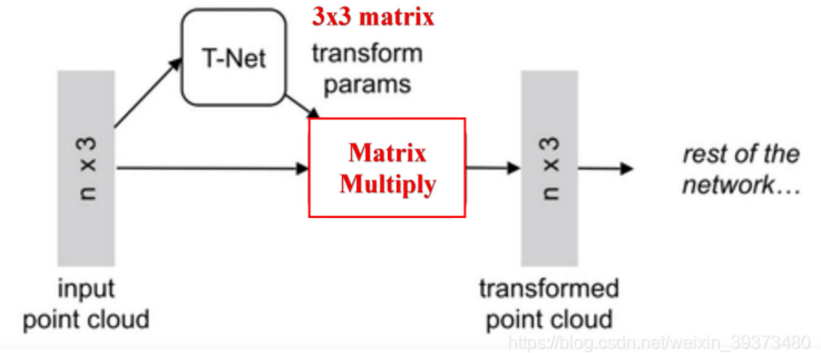
针对PI，设计的网络必须是一个对称函数，比如max或者sum函数
但是仅仅是max函数的话，每个点损失的特征就太多了，所以我们不妨先将点云上的每一个点映射到一个高维空间里，比如1024维，然后再做max操作，这样损失的信息就不会这么多。

“冗余的高维空间” -- PointNet(vanilla)



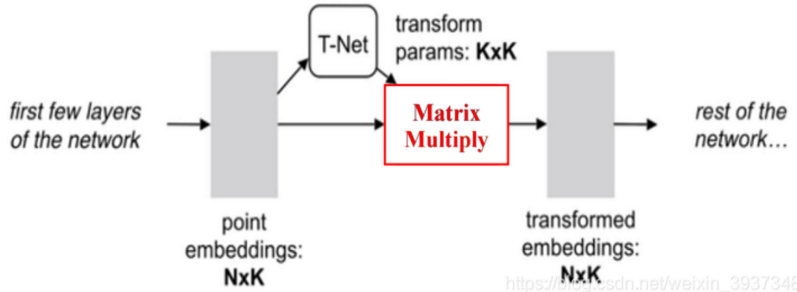
理论证明的大致意思是：任意一个再Hausdorff空间上连续的函数，都可以呗这样的PointNet无限逼近。

针对旋转不变性的解决方式就是T-Net，来学习点云的旋转，将物体进行校准，剩下的PointNet只需要对校准后的物体进行分类或者分割即可：



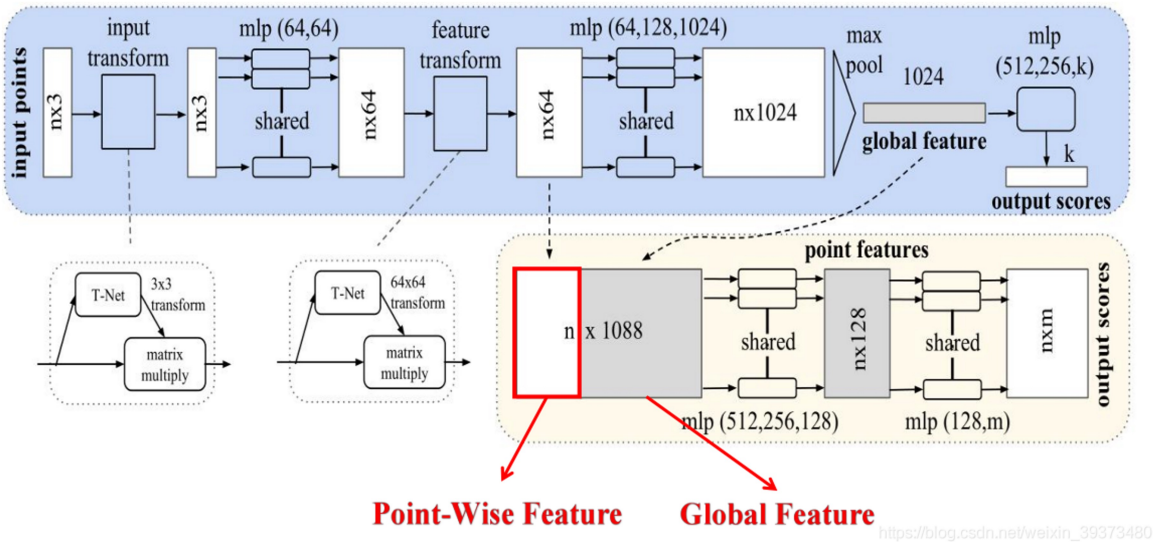
点云的旋转非常简单，只需要对一个N * D的点云矩阵乘以一个 D* D的旋转矩阵即可
因此对输入点云学习一个3×3的矩阵，就可以将其矫正；

类似的，将点云映射到K维的冗余空间后，再对K维的特征进行一次校对，同时引入一个正这话惩罚项目，希望其可以尽可能接近一个正交矩阵。



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

PointNet的整体结构



具体来看就是，

对于每一个N×3的点云输入，首先通过一个T-Net将其再空间上对其（旋转到正面）；
然后通过MLP，将其映射到64维空间，再对其；
最后映射到1024维的空间上。

这时，对于每一个点，都有一个1024维的向量表征。
这样的向量表征对于一个3维点云是冗余的，因此需要引入最大池化操作，保留1024维所有通道上都只保留最大的那个，这样得到一个1 * 1024的向量就是N个点云的全局特征。

如果是分类问题的话，直接将这个全局特征再进入MLP去输出每一类的概率即可；
如果是分割问题，由于需要输出的是点的类别，因此将全局特征拼接再点云64维的驻点特征上，最后通过MLP。输出住店的分类概率。

代码解析，PointNet代码实际上仅由两部分组成，就是一个T-Net和Encoder-Decoder结构

这里的代码就可以直接使用准备做出来的那个脚本来处理一下。

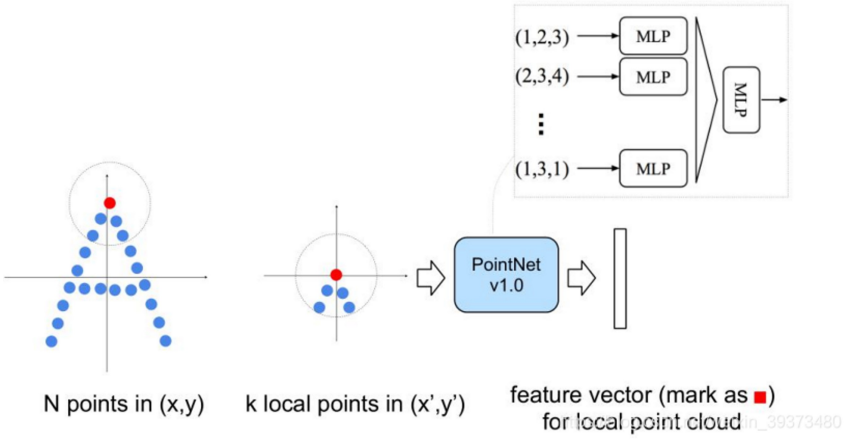
所以，这里先暂时空着，等着后面的图做出来了，再补充上。

PointNet++

PN++的提出主要是针对PN的缺点的：缺失局部特征。

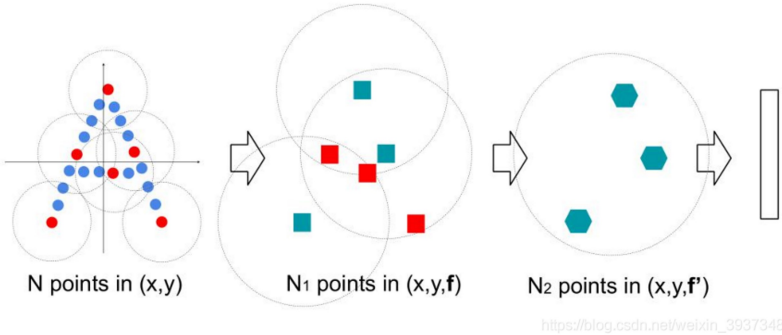
从很多试验结果可以看出来，PN对于场景的分割效果比较一般。
因为其网络是直接暴力地将所有地点最大池化为了一个全局特征，因此局部点之间联系没有呗学习到。
再Part Segmentation中，这样地问题还可以通过去中心化物体地坐标轴部分解决，但是再场景分割中，这样的效果就很一般了

第二代PN中主要借鉴了CNN的多层感受野的思想。
CNN通过分层不断地积累卷积核扫描图像的像素并做内积，使得得到的后面的特征图感受野变大。
同时，每个像素包含的信息也越多。
PN++就是模仿了这样的结构：



首先通过在整个点云的局部采样并划一个范围，将里面的点作为局部特征。
用PN进行一次特征的提取。
因为，通过了多次的这样操作之后，原本的点的个数变越来越少了。
而每个点都是由上一层更多的点通过PN提取出来的局部特征。
也就是每个点包含的信息变多了。

文章将这样的一层成为Set Abstraction.



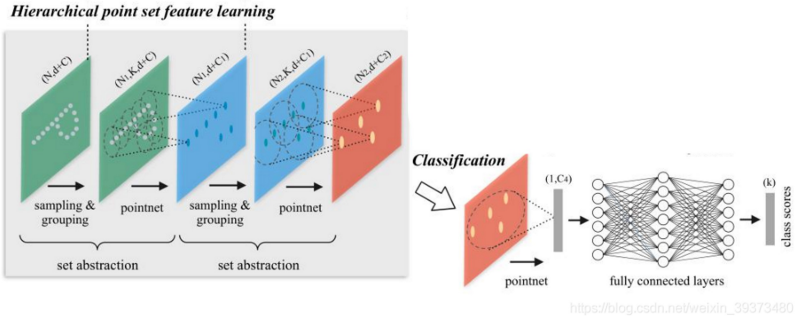
Set Abstraction的实现细节

组成部分：

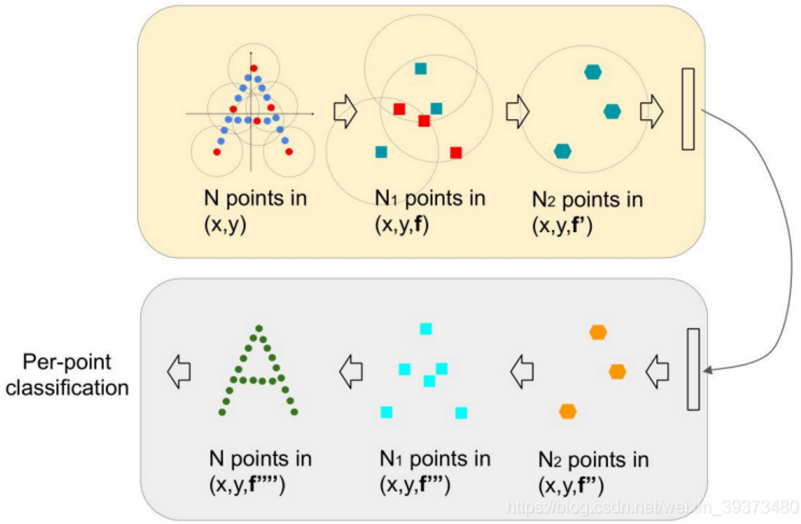
- Sampling：利用FPS随机采样点
- Grouping：利用Ball Query划一个R为半径的圆，将每个圆里面的点云作为一簇
- PointNet：对于前面两步以后的点云进行局部的全局特征提取

其实也就是三个函数~

分类网络：逐层提取局部特征，最后总结出全局特征就可以输出分类结果了。

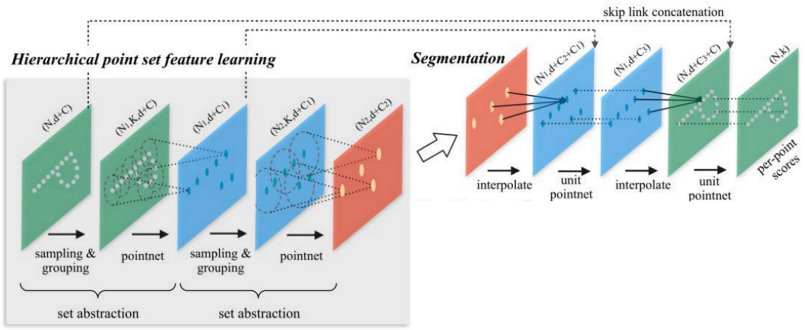


分割网络：先将点云提取一个全局特征，再通过这个全局特征逐步上采样，大致的流程如下：



说到上采样的方法，最简单的就是广播复制：将每个点附近的点和特征都变成和这个点一样。
但没办法处理一些范围相互冲突的点，或者范围没有覆盖的点：

在论文中，作者使用的是线性差值的方法，简单来说，就是距离越远，点的权重越小。
最后再对每一个点的权重做一个全局的归一化，最后分割的网络结构如下：

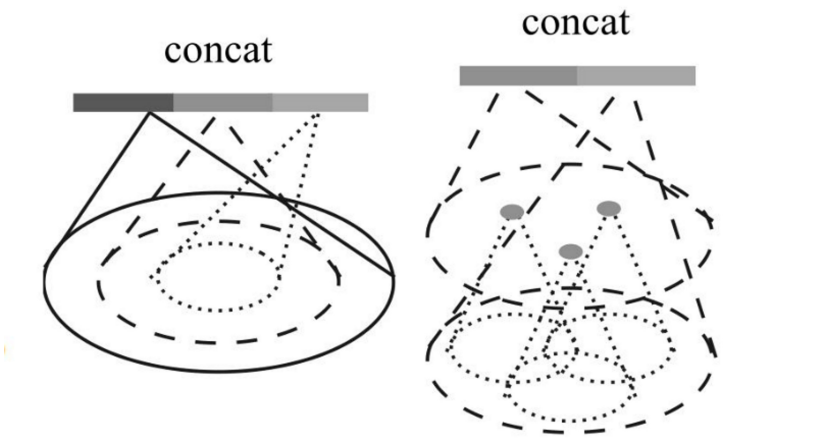


需要注意的是，和图像分割U-Net类似，对于相同的点个数的层，作者采用了直连的方式，
将Encoder里面的特征直接拼接到Decoder的特征后面。

原始方法的不足和改进

PN++在点云缺失的鲁棒性上似乎变得更差了。
因为激光搜集点云的时候，总是在近的地方密集，在远的地方稀疏。
因此，当sampling和grouping的操作在稀疏的地方进行的时候，
一个点可能代表了很多局部特征，因此一旦损失，网络的性能就会受到很大的影响。

作者对其的改进是通过引入 “不同分辨率/尺度的Grouping”去对局部做PN求局部的全局特征，
最后再将不同尺度的特征拼接起来；
同时也通过在训练的时候，随机删除一部分的点，来增加模型的缺失鲁棒性。



总结，这是基本近两年来所有分割网络的baseline。
优点明显，就是参数量小。
缺点是对局部特征的抓取不完善。

接下来最好就是直接把这个代码什么的运行下！