几何计算前沿第三次作业报告

利用点云卷积神经网络进行 3D classification 算法的复现

沈千帆 2200013220

2024 年 5 月 23 日

1 PAConv 的复现 [2]

1.1 算法介绍

常规的基于点云的卷积神经网络算法可以分为这么几种：（1）根据点云的无序性，邻域不变性和旋转平移不变形进行特征变换，然后输入到 MLP 网络中训练，最后通过池化来聚合全局特征，典型的代表是 **PointNet** 和 **PointNet++**。但是这种算法的局限性在于不能提取出细节性的局部特征。（2）这类算法注重于设计点卷积核，代表作如 **PointCNN**。但这类算法会导致大量的计算和空间消耗。

PAConv 算法提出了位置自适应卷积算法，通过动态组装存储在权重库中的基本权重矩阵来构建卷积核，而并不是从点的位置来推断卷积核。

1.1.1 动态核组装

首先定义一个权重库 $B = \{B_m | m = 1, 2, ..., M\}$，其中每个 $B_m \in R^{c_{in} \times c_{out}}$，其中 c_{in} 和 c_{out} 分别为输入和输出的通道数。接着设置了 Scorenet 来将输入的相对位置与权重库中的权重矩阵相关联。其中 $S_{ij} = \alpha(\theta(p_i, p_j))$ 是经过 softmax 归一化后的 MLP 预测出来的得分函数，由此来加权求和 M 个权重矩阵，也就是 $K(p_i, p_j) = \sum_{m=1}^{M}(S_{ij}^m B_m)$，其中 $K(p_i, p_j)$ 为 p_i 的卷积核关于 p_j 的权重。

```
# 卷积核的计算
point1, center1 = feat_trans_dgcnn(point_input=x, kernel=self.matrice1, m=self.m1)
score1 = self.scorenet1(xyz, calc_scores=self.calc_scores, bias=0.5)
# ...接着进行CNN训练
```

特别地，对于 scorenet 的输入，算法进行了分析并得到如下输入是最佳的：将与邻居点（用 KNN 选取）的坐标差和邻居点的坐标 concat 到一起作为输入，具体代码如下：

```
neighbor = x.view(batch_size * num_points, -1)[idx, :]
neighbor = neighbor.view(batch_size, num_points, k, num_dims)
x = x.view(batch_size, num_points, 1, num_dims).repeat(1, 1, k, 1)
xyz = torch.cat((neighbor - x, neighbor), dim=3).permute(0, 3, 1, 2)  # b,6,n,k
return xyz
```

1.1.2 权重正则化

由于丰富的权重矩阵库是随机生成的，可能导致有一些权重矩阵较为类似，所以计算损失的时候要加上权重正则化项作为惩罚：$L_{corr} = \sum_{B_i, B_j \in B, i \neq j} \frac{|<B_i, B_j>|}{||B_i||_2 ||B_j||_2}$。

1.2 算法代码运行和结果

运行`python main.py --config config/dgcnn_paconv_train.yaml`使用 DGCNN 模型运行在 ModelNet40 数据集上的分类。使用 7 张 RTX3090 共计运行 5 小时 30 分钟左右，在测试集上达到最好的 Accuracy 为 **91.37%**。Loss 和 Accuracy 的曲线如下图所示：

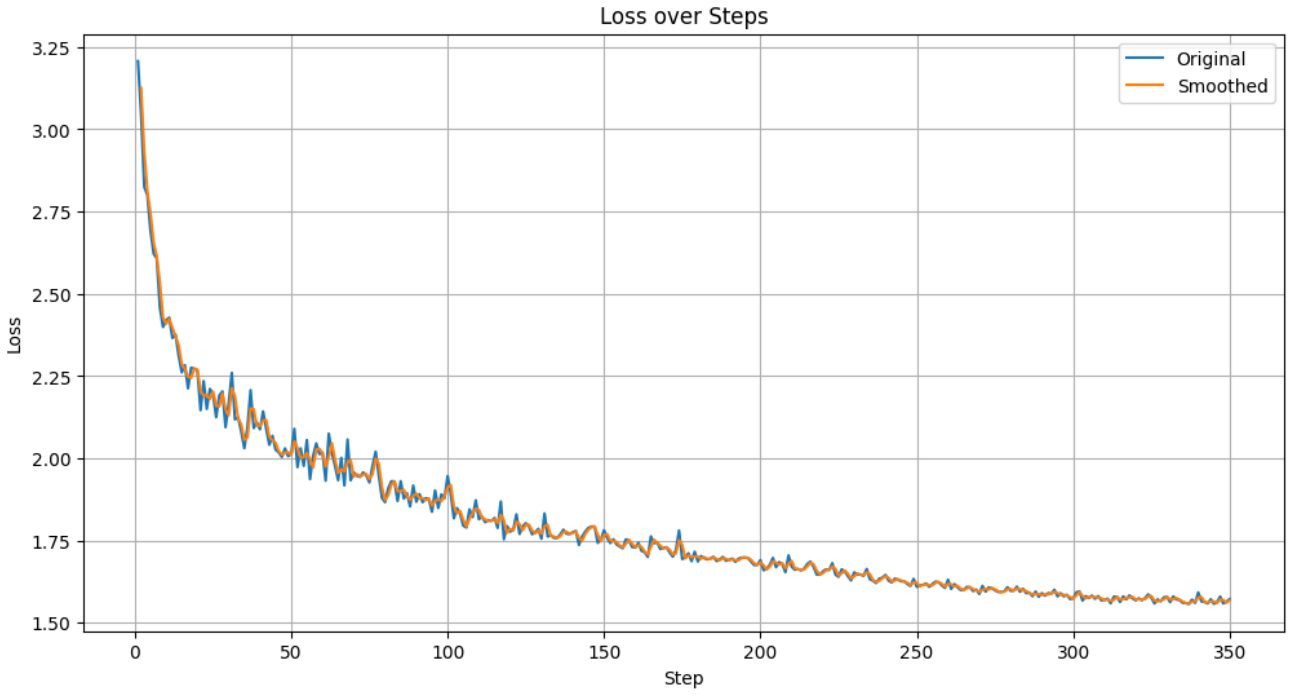

图 1: Loss

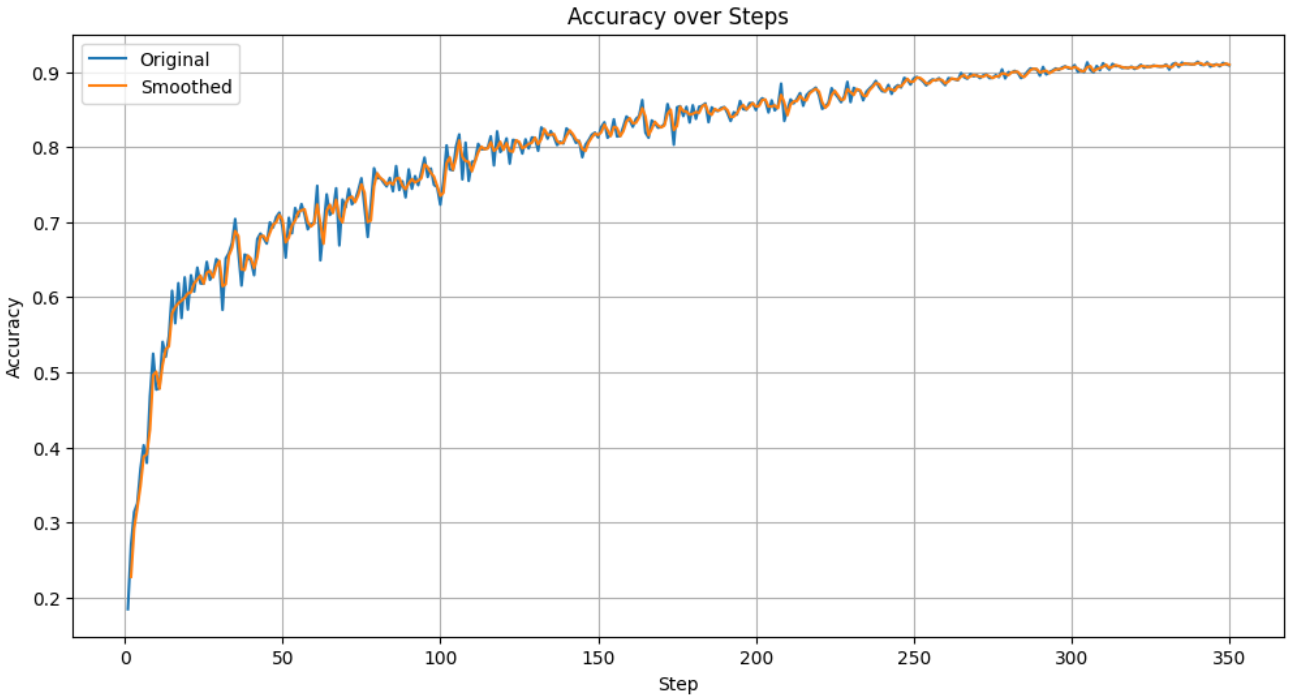

图 2: Accuracy

2 O-CNN 的复现 [1]

2.1 算法介绍

O-CNN 是一种基于八叉树数据结构的卷积神经网络，用于三维形状分析。传统的基于体素的 3D 卷积神经网络需要在密集的体素上进行采样，并输入到 CNN 中，这使得在高分辨率的体素上代价非常高昂。于是 O-CNN 的作者充分利用了八叉树这一数据结构的特性，区分场景中密集和稀疏的区域进行特征的提取。其大致思想是对于密集的区域，不断地构建子八叉树进行采样，对于稀疏的区域就跳过。同时利用八叉树 “shuffled key” 这一在 GPU 并行计算高效的特性，可以非常好地实现 coarse-to-fine 的策略。这样的算法可以将全空间体素 CNN 的 $\mathcal{O}(N^3)$ 的复杂度降低到 $\mathcal{O}(N^2)$。下面，我们就根据一些关键代码来解释这个算法的核心内容。

2.1.1 Shuffled Key

O-CNN 利用了八叉树 z 字形遍历网格的特性是用 shuffled key，也就是 x,y,z 的二进制表示进行交错排列来实现，在源代码中函数`def xyz2key(self, x, y, z, depth)`如下所示：

```
for i in range(depth):
      mask = 1 << i
      key = (key | ((x & mask) << (2 * i + 2)) |
                   ((y & mask) << (2 * i + 1)) |
                   ((z & mask) << (2 * i + 0)))
```

2.1.2 Build an Octree

该算法对于给定的点云进行间八叉树的算法。八叉树的建立与二叉树的建立类似。在对每一个点云中的点按照深度进行标准化后，利用上述函数获取每个点的 shuffled key 值并排序（会进行去重）。接下来对于每一层构建八叉树的所有结点：计算节点数，赋 key 值并更新子结点。这在函数`def octree_grow_full(self , depth: int, update_neigh: bool = True)`中实现：

```
# node number
num = 1 << (3 * depth)
self.nnum[depth] = num * self.batch_size
self.nnum_nempty[depth] = num * self.batch_size
# update key
key = torch.arange(num, dtype=torch.long, device=self.device)
bs = torch.arange(self.batch_size, dtype=torch.long, device=self.device)
key = key.unsqueeze(0) | (bs.unsqueeze(1) << 48)
self.keys[depth] = key.view(-1)
# update children
self.children[depth] = torch.arange(
num * self.batch_size, dtype=torch.int32, device=self.device)
```

随之构建节点之间的父子关系，方便后续查询。其中用到的是八叉树的 shuffled key 一个特别好的性质就是`pkey = node_key >> 3`即兄弟结点的父结点为他们的共同前缀。

2.1.3 CNN Model

在源代码中采用的是 LeNet 结构进行训练。算法将平均法向量作为输入，经过连续 3 个阶段的卷积加池化后输出到一个全连接层中得到最后的输入（40 维），全连接层的结构如下：

```
self.header = torch.nn.Sequential(
        torch.nn.Dropout(p=0.5),                    # drop1
        ocnn.modules.FcBnRelu(64 * 64, 128),        # fc1
        torch.nn.Dropout(p=0.5),                    # drop2
        torch.nn.Linear(128, out_channels))         # fc2
```

2.2 算法代码运行和结果

运行`python classification.py --config configs/cls_m40.yaml SOLVER.alias time`运行在 ModelNet40 数据集上的分类。使用单张 RTX3090 共计运行 2 小时 12 分钟，在测试集上达到最好的 Accuracy 为 **92.07%**。

Loss 和 Accuracy 的曲线如下图所示：

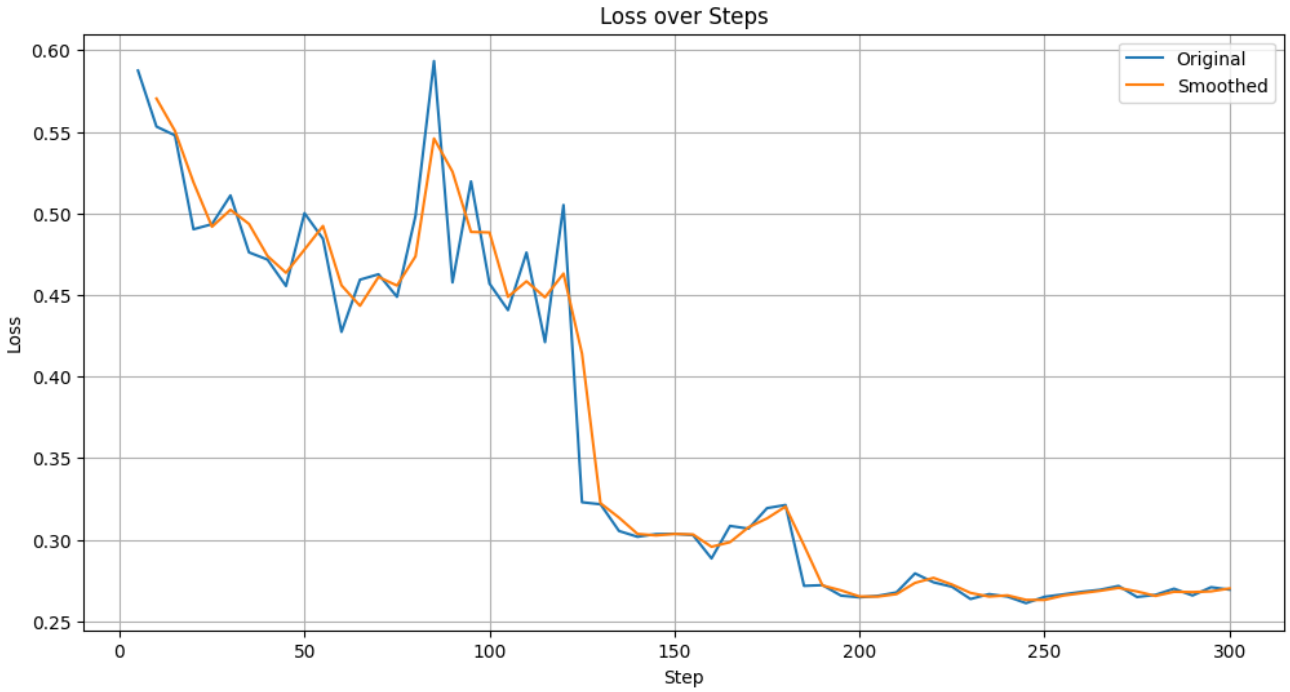

图 3: Loss

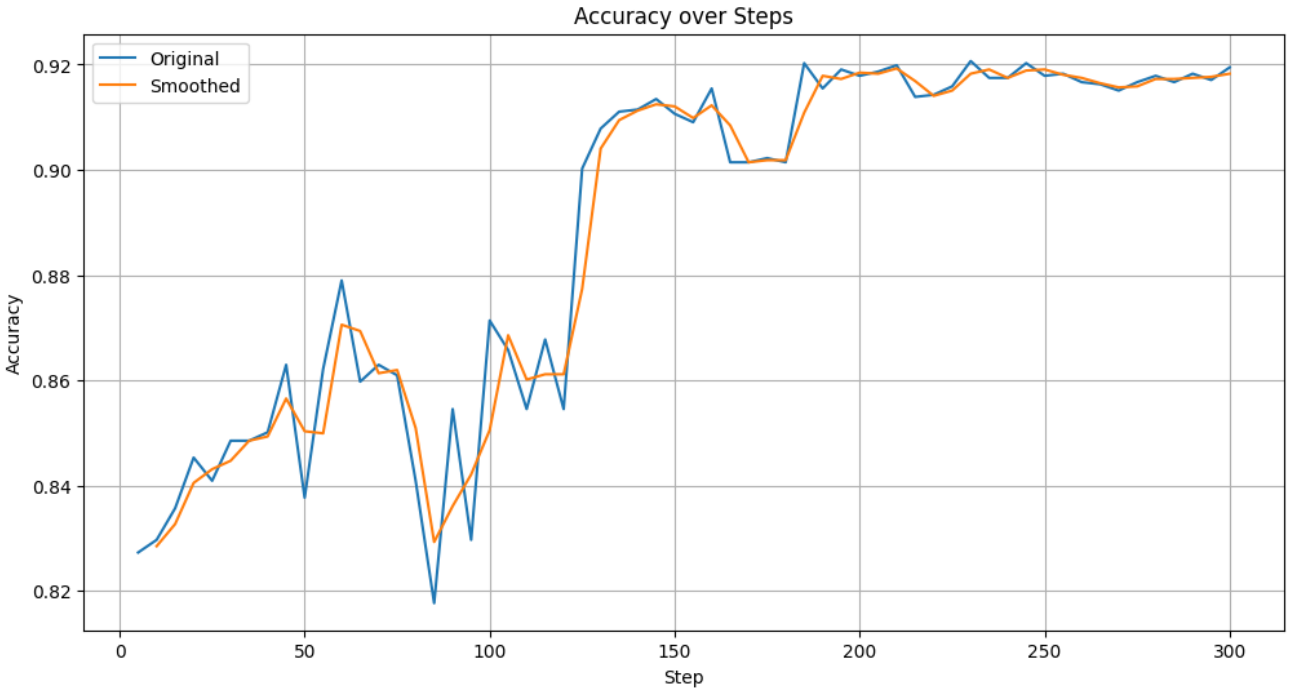

图 4: Accuracy

References

[1] Peng-Shuai Wang et al. "O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), p. 72.

[2] M. Xu et al. "PAConv: Position Adaptive Convolution with Dynamic Kernel Assembling on Point Clouds". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2021, pp. 3172–3181. DOI: 10.1109/CVPR46437.2021.00319. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00319.