Point Cloud Transformer 的实现以及效率改进

摘要

由于点云数据中存在不规则的域并且缺少顺序性，设计用于点云处理的深度神经网络极具挑战性。这篇工作提出了一种名为点云变换器（Point Cloud Transformer，PCT）的新框架，用于点云学习。PCT 基于在自然语言处理中取得巨大成功的变换器模型，并在图像处理中展现出巨大潜力。它天生适用于处理点序列，因而非常适合点云学习。为了更好地捕捉点云内的局部上下文，我们通过最远点采样和最近邻搜索增强了输入嵌入。实验表明，PCT 在形状分类、部件分割、语义分割和法线估计任务上实现了良好的性能，这项创新不仅提高了点云处理的效率和精度，也为后续的研究提供了新的视角和方法。

关键词：Transformer；点云处理

1 引言

与二维图像不同，点云是无序且非结构化的，这使得设计用于处理它们的神经网络面临挑战。文中提到了 Qi 等人开创性的 PointNet 模型，该模型通过使用多层感知器（MLP）、最大池化和刚性变换来确保对置换和旋转的不变性，以实现点云的特征学习。随后，文章讨论了许多研究工作试图定义可以汇聚点云局部特征的卷积运算符。这些方法要么重新排序输入点序列，要么将点云体素化以获得卷积的规范域。

Transformer 是一个包含输入的词嵌入、位置编码和自注意力的解码器-编码器结构。其中，自注意力模块是核心组件，基于全局上下文为其输入特征生成精细的注意力特征。首先，自注意力将输入嵌入和位置编码的总和作为输入，并为每个词计算出三个向量：查询、键和值。然后，通过匹配（点积）它们的查询和键向量，可以获得任意两个词之间的注意力权重。最后，注意力特征被定义为所有值向量的加权和。显然，每个词的输出注意力特征与所有输入特征相关，使其能够学习全局上下文。所有 Transformer 的操作都是可并行化且与顺序无关的。理论上，它可以替代卷积神经网络中的卷积操作，并具有更好的通用性。

这篇工作提出了 point Cloud Transformer 框架，该框架基于传统 Transformer 的原理，使用 Transformer 的固有顺序不变性来避免定义点云数据的顺序，并通过注意力机制进行特征学习。PCT 框架对于点云和自然语言这两种不同类型的数据进行了几项调整，包括基于坐标的输入嵌入模块、优化的偏移注意力模块和邻域嵌入模块。偏移注意力模块是原始自注意力的有效升级，它通过将自注意力模块的输入与注意力特征之间的偏移替换注意力特征来工作。而邻域嵌入模块则强调了在捕获全局特征的同时，不应忽视局部几何信息，这对于点云学习同样重要。

选择这篇工作进行复现的动机是：这篇工作提出了一种基于 Transformer 的新框架 PCT，适用于非结构化、无序点云数据，基于全局注意力的类似工作较少；作者在全局注意力的基础上提出了偏移注意力和隐式拉普拉斯算子及其规范化改进，这些改进天然地对置换不变，更适合点云学习；并通过广泛的实验表明，PCT 在形状分类、部件分割和法线估计等任务上实现了优良的性能。

2 相关工作

2.1 点云中的传统深度学习模型

关键方法和进展。首先，Qi 等人提出的 PointNet 模型是点云学习的先驱作品，它通过使用多层感知器（MLP）和最大池化来提取全局特征，为点云数据的深度学习奠定了基础 [1]。为了进一步改进，Qi 等人提出了 PointNet++，该模型通过查询球分组和分层网络结构来捕获点云的局部结构特征，增强了对细微局部变化的感知能力 [2]。

随后，研究者们开始探索如何在点云数据上定义卷积操作，这一探索主要分为两个方向。一方面，一些工作将点云转换为规则的体素阵列，以便应用传统的卷积操作。例如，Tchapmi 等人提出的 SEGCloud 通过三线性插值将 3D 体素的卷积特征映射回点云，并使用全连接条件随机场来保持全局一致性 [3]。

Atzmon 等人的 PCNN 框架则通过扩展和限制操作在点基表示和体素基表示之间进行映射，实现了在体素上进行点特征提取的体积卷积 [4]。Hermosilla 等人的 MCCNN 将卷积视为蒙特卡罗积分问题，使其适用于非均匀采样的点云 [5]。Wu 等人的 PointConv 则通过蒙特卡罗估计和重要性采样在点云上执行 3D 卷积 [6]。

另一方面，一些研究则重新定义了卷积操作，使其直接适用于不规则的点云数据。Li 等人的 PointCNN 通过训练 $\mathcal{X}$ 变换来确定卷积的一维点顺序 [7]。Tatarchenko 等人提出的切线卷积利用投影的虚拟切线图像来学习表面几何特征 [8]。Landrieu 等人的 SPG 通过将扫描场景划分为相似元素并建立超点图结构，来学习对象部件之间的上下文关系 [9]。此外，Wang 等人设计的 EdgeConv 操作符适用于动态图，能够通过恢复局部拓扑结构来支持点云学习 [10]。

2.2 基于注意力机制的深度学习模型

除此之外，还有一些方法利用了注意力机制和 Transformer 技术。Yan 等人提出的 PointASNL 利用自注意力机制来处理点云中的噪声问题，通过更新局部点群的特征来提高处理效果 [11]。Hertz 等人的 PointGMM 则使用多层感知器和注意力分割来进行形状插值 [12]。

这些研究不仅展示了点云数据处理技术的多样性和复杂性，也为本文复现的点云变换器（PCT）框架提供了重要的背景和启发。与现有方法相比，PCT 更全面地利用了 Transformer 的原理，不仅仅将全局自注意力作为一个辅助模块，而是构建了一个更通用的用于各种点云任务的框架。

3 本文方法

3.1 本文方法概述

Point Cloud Transformer 的总体架构如图 1所示，PCT 旨在将输入点编码为一个新的高维特征空间，这个空间能够基于点之间的语义亲和性，支持各种点云处理任务。PCT 编码器通过将输入坐标嵌入到新的特征空间来启动，嵌入的特征随后被传入四个堆叠的注意力模块，以学习每个点的语义丰富和有区别性的表示。然后，通过一个线性层来生成输出特征。总体来说，PCT 的编码器在设计思想上与原始的 Transformer 类似，除了放弃位置嵌入，因为点的坐标已经包含了这些信息。

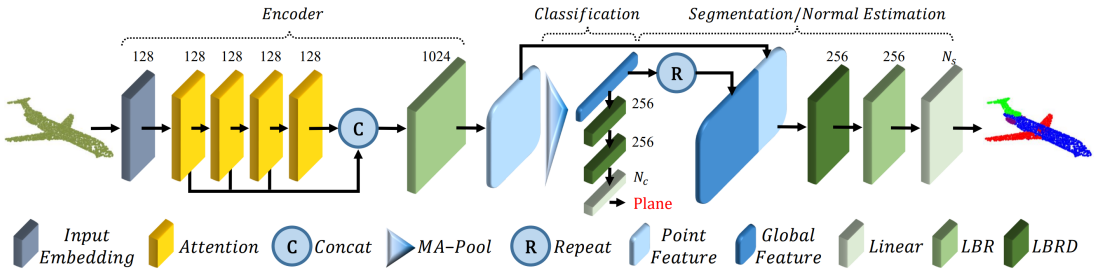

图 1. PCT 的总体架构

给定一个输入点云 $P \in \mathbb{R}^{N\times d}$，其中 N 是点的数目，每个点有一个 d-维的特征描述。首先通过输入嵌入模块学习一个 d_e 维的嵌入特征 $F_e \in \mathbb{R}^{N\times d_e}$。然后，这些嵌入的特征被输入到四个堆叠的注意力模块中以学习每个点的语义丰富和区分性的表示，接着通过一个线性层来生成输出特征 F_o:

$$F_1 = \mathrm{AT}^1(F_e),$$

$$F_i = \mathrm{AT}^i(F_{i-1}), \quad i = 2, 3, 4,$$

$$F_o = \mathrm{concat}(F_1, F_2, F_3, F_4) \cdot W_o,$$

其中 AT^i 表示第 i 个注意力层，W_o 是线性层的权重。为了提取一个有效的全局特征向量 F_g 来代表点云，选择将两个池化操作符的输出连接起来：一个最大池化（MP）和一个平均池化（AP）在学习到的逐点特征表示上。

在分类任务中，为了将点云 P 分类到 N_c 个对象类别（例如桌子、椅子），将全局特征 F_g 馈送到分类解码器，该解码器包括两个级联的前馈神经网络 LBRs（结合线性层、批量归一化和 ReLU 层），每个都有 0.5 的 dropout 概率，最终通过一个线性层来预测最终的分类分数 $C \in \mathbb{R}^{N\times c}$。点云的类别标签由具有最大分数的类别确定。

对于分割任务，为了将点云分割成 N_s 个部分（例如桌面、桌腿；部分不必连续），我们必须预测每个点的部分标签。首先，将全局特征 F_g 与逐点特征在 F_o 中连接起来。为了学习适用于各种物体的通用模型，还对一个 one-hot 编码对象类别向量进行 64 维的特征编码，并将其与全局特征连接起来，这遵循了大多数其他点云分割网络的做法。分割网络解码器的架构几乎与分类网络相同，除了仅在第一个 LBR 上执行 dropout。然后预测输入点云的最终逐点分割分数 $S \in \mathbb{R}^{N\times N_s}$，最终，点的部分标签也是确定为得分最高的标签。

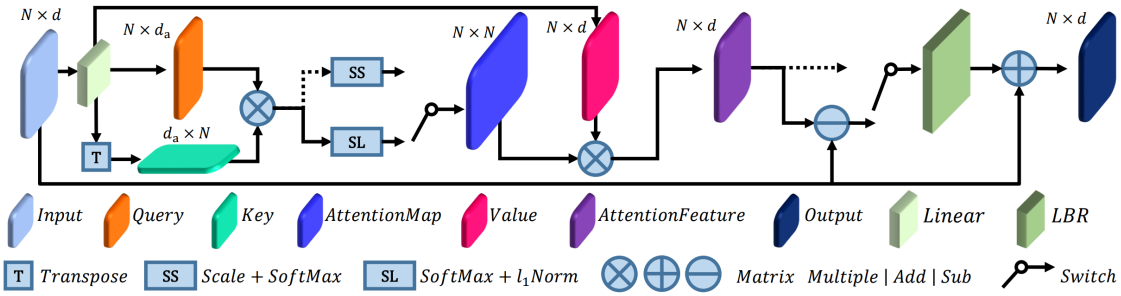

图 2. PCT 的自注意力机制

3.2 自注意力模块

在 PCT 的简易实现中，采用了如原始 Transformer 中介绍的自注意力（SA）。自注意力，也称为内部注意力，是一个计算数据序列中不同项之间语义亲和度的机制。自注意力层的架构如图 2所示，通过改变点数据流的方向实现。假设点云 P 已经通过编码嵌入到一个 d_e 维的空间 $F_e \in \mathbb{R}^{N\times d_e}$ 中，设置 $d_a = d_e/4$；令 Q,K,V 分别为查询、键和值矩阵，通过线性变换输入特征 F_{in} 生成：

$$(Q, K, V) = F_{\text{in}} \cdot (W_q, W_k, W_v)$$

$$Q, K \in \mathbb{R}^{N\times d_a}, V \in \mathbb{R}^{N\times d_e}$$

$$W_q, W_k \in \mathbb{R}^{d_e\times d_a}, W_v \in \mathbb{R}^{d_e\times d_e}$$

其中 W_q,W_k,W_v 是共享的可学习线性变换权重，d_a 可能不等于 d_e。然后使用查询和键矩阵计算注意力权重：

$$\hat{A} = \alpha_{ij} = Q \cdot K^T$$

这些权重随后被标准化，自注意力输出特征 F_{sa} 是通过相应的注意力权重加权值向量计算得到：

$$F_{\text{sa}} = A \cdot V$$

查询、键和值矩阵是由输入特征 F_{in} 经过共享的线性变换矩阵确定的，因此它们都是与顺序无关的。因此，整个自注意力过程是排列不变的，使其适用于点云所表示的无序、不规则域。最后，自注意力特征 F_{sa} 和输入特征 F_{in} 结合生成最终的输出特征 F_{out}，整个自注意力层通过一个 LBR 网络实现:

$$F_{\text{out}} = \text{SA}(F_{\text{in}}) = \text{LBR}(F_{\text{sa}}) + F_{\text{in}}$$

进一步的，作者发现如果用偏移注意力（OA）模块替换原始的自注意力（SA）模块，可以提高网络性能。偏移注意力层计算自注意力特征和输入特征之间的偏移（差异），这个偏移代替了简易版本中使用的 SA 特征。具体来说，偏移注意力功能如下：

$$F_{\text{out}} = \text{OA}(F_{\text{in}}) = \text{LBR}(F_{\text{in}} - F_{\text{sa}}) + F_{\text{in}}$$

偏移注意力锐化了注意力权重并减少了噪声的影响，这对下游任务有益。

3.3 邻域编码

邻域嵌入编码是 PCT 为了提升点云处理性能而增强局部特征表示的方法。利用邻域嵌入策略，可以优化点嵌入，以捕获点云学习中的邻域信息，这对于捕捉点云中的局部结构和细节至关重要。如图 3所示，邻域嵌入模块结合了两个 LBR 层和两个 SG（采样和分组）层，以此作为基础点嵌入。通过使用两个级联的 SG 层，可以逐渐扩大特征聚合过程中的感受野，类似于 CNN 中的做法。SG 层通过 kNN 搜索，基于每个点的局部邻居的欧几里得距离来聚合特征。

更具体地说，SG 层接受一个有 N 个点及其对应特征 F 的点云 P，并输出一个采样后的点云 P_s 与 N_s 个点及其相应的聚合特征 F_s。首先，采用最远点采样（FPS）算法选择点云中的点，然后，对于每个采样点 p，通过 k-NN 搜索在 P 中找到它的 k 个最近邻居。然后计算输出特征 F_s 如下所示：

$$\Delta F(p) = \text{concat}_{knn(p,P)}(F(q) - F(p))$$

$$\hat{F}(p) = \text{concat}(\Delta F(p), RP(F(p), k))$$

$$F_s(p) = \text{MP}(\text{LBR}(\text{LBR}(\hat{F}(p))))$$

其中，$F_{(\text{p})}$ 是点 p 的输入特征，F_s(p) 是采样点 p 的输出特征，MP 是最大池化操作，RP 是将向量 x 重复 k 次形成矩阵的操作，来自 EdgeConv 算法。在这个过程中，不需要确定每个点的部分标签或法向量，因此这个过程仅用于局部特征提取，输出大小在每个阶段仍然是 N。

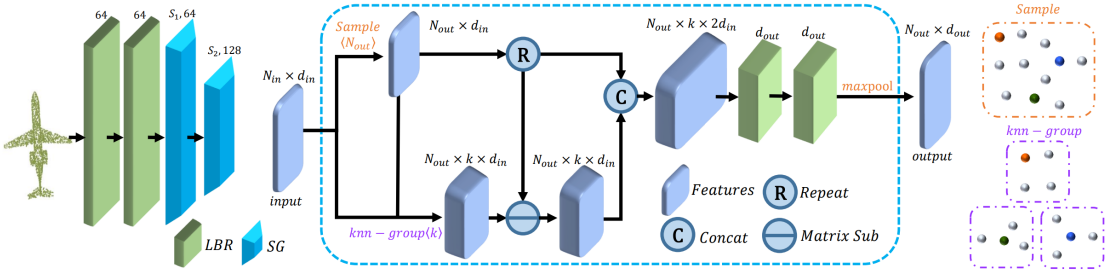

图 3. PCT 的邻域编码

4 复现细节

4.1 与已有开源代码对比

PointNet2-ops-lib 是一个与 3D 点云处理相关的库，通常用于支持 PointNet++ 这类深度学习模型。PointNet++ 是一个用于处理 3D 点云数据的神经网络模型，它能从点云中提取特征，用于各种任务，如物体分类、分割和检测。PointNet2-ops-lib 包含了实现 PointNet++ 模型所需的一系列操作和函数。在复现过程中，我使用了该库中提供的 FPS 算法 (最远点采样) 实现，用于对点云的下采样，如图 4。

```
# FPS sampling
fps_idx = pointnet2_utils.furthest_point_sample(coords, s).long()
new_coords = index_points(coords, fps_idx)
new_features = index_points(features, fps_idx)
```

图 4. 最远点采样

4.2 创新点

Transformer 模型使用注意力机制取得了良好的效果，但是全局注意力机制存在复杂度过高的问题，达到了二次复杂度，Transformer 模型中的全局注意力机制的二次复杂度主要是由于每一对输入元素之间都要计算注意力得分。序列中的 N 个元素，每个元素都要与 N 个元素（包括自身）计算注意力得分，这就导致了 $O(N^2)$ 的复杂度。

由于计算每个 Q 和所有 K 的点积需要 N 次操作，并且我们需要为序列中的每个元素做这样的计算，所以得分计算的复杂度是 $O(N^2)$。由于每个元素的表示是 D 维的，所以整体的复杂度变成了 $O(N^2 \cdot D)$。

这种二次复杂度使得 Transformer 模型在处理长序列时效率较低，因为计算量随着序列长度的平方而增加。为了解决这个问题，研究者们提出了各种方法，比如稀疏注意力机制，它只计算序列中某些特定元素对之间的注意力得分，或者是近似方法，如低秩近似，这可以显著减少计算量，使得模型能够处理更长的序列。

本文的改进采用的是稀疏注意力机制，并对 PCT 的基本架构进行了重新设计，命名为 Sparse PCT，如图 5所示。Sparse PCT 采用双路 Transformer 的设计，分别处理高分辨率的原始点云与低分辨率的点云，其中低分辨率的点云由原始点云的最远点采样得到。低分辨率的点云进行全局的自注意力计算，得到的注意力矩阵将被用于恢复原始点云的注意力矩阵，需要注意的是，最终得到的原始点云注意力矩阵是稀疏的，这与本文设计的复原方式有关。

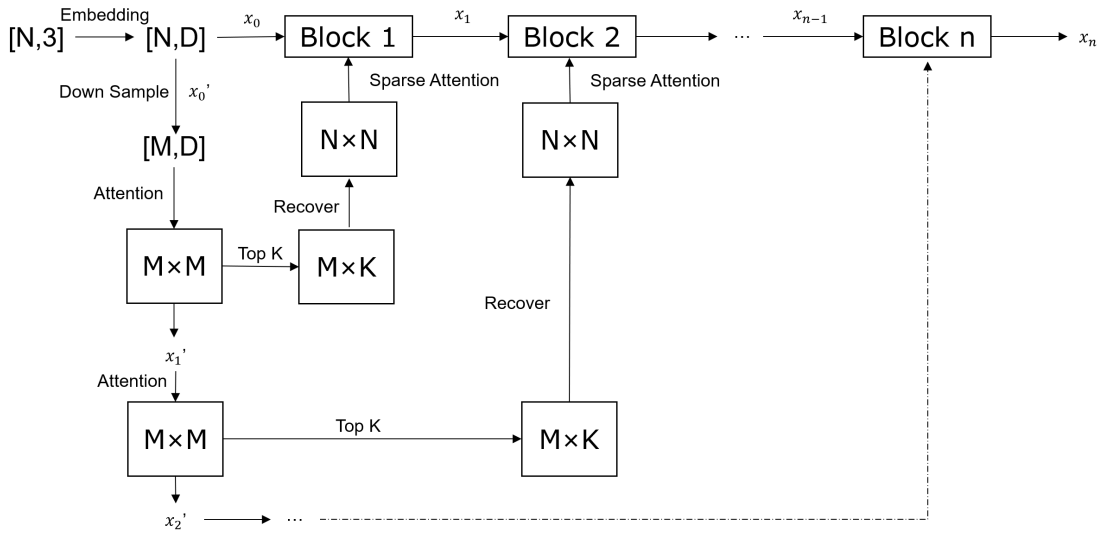

图 5. Sparse PCT 架构

得到低分辨率点云的注意力矩阵后，对于低分辨率点云中的一个点 x_0，本文只关注 x_0 的前 K 个最高注意力点 (实验中设置为 16)；假设 x_i 属于 x_0 的前 K 个最高注意力点。则本文

将 x_0 对 x_i 的注意力赋予 x_i 以及 x_i 在原始点云的 kNN 邻居 (实验设置为 16)。同理对低分辨率下的所有点进行该操作，即可得到原始点云的稀疏注意力矩阵，用于原始点云的特征学习。

图 6为低分辨率点云全局注意力计算模块的实现，低分辨率点云特征经过该模块后，将会返回新的点云特征以及中间得到的注意力矩阵，该矩阵将被作为参数送往高分辨率模块。

```python
class low_attention(nn.Module):
    def __init__(self, channels):
        super(low_attention, self).__init__()
        self.q_conv = nn.Conv1d(channels, channels // 4, 1, bias=False)
        self.k_conv = nn.Conv1d(channels, channels // 4, 1, bias=False)
        self.v_conv = nn.Conv1d(channels, channels, 1)
        self.trans_conv = nn.Conv1d(channels, channels, 1)
        self.after_norm = nn.BatchNorm1d(channels)
        self.act = nn.ReLU()
        self.softmax = nn.Softmax(dim=-1)

    def forward(self,x):
        """
        Input  x: [B, D, low_N]
        Output x: [B, D, low_N]
        """
        x_q = self.q_conv(x).permute(0, 2, 1)#change shape
        x_k = self.k_conv(x)
        x_v = self.v_conv(x)

        energy = torch.bmm(x_q, x_k)
        attention = self.softmax(energy)
        attention = attention / (1e-9 + attention.sum(dim=1, keepdims=True))  # [B, low_n, low_n]

        x_r = torch.bmm(x_v, attention)
        x_r = self.act(self.after_norm(self.trans_conv(x - x_r)))
        x = x + x_r
        return x,attention
```

图 6. Low Attention

图 7为高分辨率点云计算模块的实现，该模块的参数为原始点云特征 x，低分辨率点云注意力矩阵 $low_{\text{attention}}$，低分辨率点云在原始点云的索引 fps_{idx}，以及低分辨率点云在原始点云的 kNN 邻居索引 knn_{idx}。

```python
def forward(self, x,low_attention,fps_idx,knn_idx):
    x_v = self.v_conv(x)
    topk_values, topk_indices = torch.topk(low_attention, k=16, dim=-1)
    #topk_values:[B,low_n,topk] #topk_indices:[B,low_n,topk]
    #top_k=KNN
    attention1=torch.zeros((x.shape[0],(x.shape[2]),(x.shape[2]))).to('cuda')#[B,N,top k]
    B, _ = x.shape[0], x.shape[2]
    low_n, top_k = topk_values.shape[1], topk_values.shape[2]
    # 将 fps_idx 扩展为与 topk_indices 相同的形状
    expanded_fps_idx = fps_idx.unsqueeze(-1).expand(-1, -1, topk_indices.shape[2])
    # 现在, fps_idx 和 topk_indices 具有相同的维度数
    # 使用 expanded_fps_idx 和 topk_indices 来转换全局索引
    global_topk_indices = torch.gather(expanded_fps_idx, 1, topk_indices)
    # 批量更新 attention
    # 使用 global_topk_indices 作为列索引, expanded_fps_idx 作为行索引
    attention1[torch.arange(B)[:, None, None], expanded_fps_idx, global_topk_indices] = topk_values
    # 获取 KNN 索引
    # 首先为每个 topk_indices 找到对应的 KNN 索引
    knn_indices = torch.gather(knn_idx, 1, topk_indices)
    # 扩展 topk_values 以匹配 KNN 索引
    expanded_topk_values = topk_values.unsqueeze(-1).expand(-1, -1, -1, knn_idx.size(-1))
    # 扩展 batch 索引以匹配其他维度
    batch_indices = torch.arange(B, device=attention1.device)[:, None, None, None].expand(-1, low_n, top_k, knn_idx.size(-1))
    # 扩展 fps 索引以匹配 KNN 的数量
    expanded_fps_idx = expanded_fps_idx.unsqueeze(-1).expand(-1, -1, -1, knn_idx.size(-1))
    # 扩展 topk_values 以匹配 KNN 索引
    expanded_topk_values = expanded_topk_values.expand(-1, -1, -1, knn_idx.size(-1))

    knn_indices = knn_indices.unsqueeze(-1).expand(-1, -1, -1, knn_idx.size(-1))
    # 批量更新 attention
    attention1[batch_indices, expanded_fps_idx, knn_indices] = expanded_topk_values

    x_r = torch.bmm(x_v, attention1)
    x_r = self.act(self.after_norm(self.trans_conv(x - x_r)))
    x = x + x_r
    return x
```

图 7. High Attention

图 8为 Sparse PCT 的主体代码，实现上遵循架构图的设计。预处理部分为原始点云的特征编码，用 FPS 采样低分辨率点云以及进行 kNN 索引。随后为 4 层的 Sparse PCT，每层由两个模块组成：低分辨率模块以及高分辨率模块。

```python
#Sparse PCT
def forward(self, x):
    #embedding
    coords,x = self.neighbor_embedding(x)# [B, D, N]
    coords = coords.contiguous()
    # FPS sampling
    low_n=64
    fps_idx = pointnet2_utils.furthest_point_sample(coords, low_n).long()  # [B, low_n]
    low_coords = index_points(coords, fps_idx)           # [B, low_n, 3]
    low_x = index_points(x.permute(0, 2, 1), fps_idx)    #low_x: [B, low_n, D]
    low_x=low_x.permute(0, 2, 1)
    # K-nn grouping
    k=16
    knn_idx = knn_point(k, coords,low_coords)            # [B, low_n, k]

    low_x1,atten = self.low_oa1(low_x)#block 1
    x1=self.high_oa1(x,atten,fps_idx,knn_idx)

    low_x2,atten = self.low_oa2(low_x1)#block 2
    x2=self.high_oa2(x1,atten,fps_idx,knn_idx)

    low_x3,atten = self.low_oa3(low_x2)#block 3
    x3=self.high_oa3(x2,atten,fps_idx,knn_idx)

    _ ,atten = self.low_oa4(low_x3)#block 4
    x4=self.high_oa4(x3,atten,fps_idx,knn_idx)

    x = torch.cat([x, x1, x2, x3, x4], dim=1)
    x = self.linear(x)

    # x = F.adaptive_max_pool1d(x, 1).view(batch_size, -1)
    x_max = torch.max(x, dim=-1)[0]
    x_mean = torch.mean(x, dim=-1)
    return x, x_max, x_mean
```

图 8. Sparse PCT

5 实验结果分析

在实验部分，本文针对物体分类任务 (Classification) 进行测试，使用的数据集为 ModelNet40。ModelNet40 是一个广泛使用的三维物体识别和分类的数据集。这个数据集是从大型 3D CAD 模型库 ModelNet 中精选出来的，专门用于训练和测试三维物体识别算法。ModelNet40 包含 40 种不同类别的物体，总共包含大约 12,311 个模型，其中 9,843 个用于训练，2,468 个用于测试。

在训练上，本文采用了与 PCT 相同的采样策略，从每个物体的点云中随机采样 2,048 个点，以及用于评估的 2,468 个模型。训练过程中使用了范围在 [0.02, 0.024] 之间的随机采样，以及范围在 [0.67, 1.5] 之间的随机输入丢弃，来增强输入数据。为了增加输入数据的难度，测试时没有应用数据增强。采用了 32 的 mini-batch 大小和 250 个训练周期，每个周期都使用一个退火调度来调整学习率。最后，本文使用与 PCT 相同的软交叉熵损失函数和带动量 0.9 的随机梯度下降（SGD）优化器进行训练。

物体分类任务的实验结果如表 1所示，本文复现的 PCT 方法准确率为 92.7%，在 PointNet 基准上提高了 3.5% 的准确率，总体准确率比 PCT 的论文数据低 0.5%；而改进的 Sparse PCT

准确率为 92.8%，在 PointNet 基准上提高了 3.6%，该方法在总体准确率上比复现的 PCT 方法提高了 0.1%。

Method	**Input**	**Points**	**Accuracy**
PointNet	P	1k	89.2%
PointNet++	P	1k	90.7%
Naive_PCT	P	1k	91.0%
PointNet++	P, N	5k	91.9%
PointGrid	P	1k	92%
PCT(复现)	P	1k	92.7%
Sparse PCT	P	1k	92.8%
PCT(论文数据)	P	1k	93.2%

表 1. 点云物件分类任务的性能对比

进一步的，本文测试了 PCT 与 Sparse PCT 在 ModelNet 数据集上的效率，测试方法为记录两个方法在相同数据集中训练时的计算消耗，而计算消耗记录的是各自注意力模块的计算时长，得到两次测试的结果如图 9，图 10所示。

从结果可以看出，Sparse PCT 的计算消耗总体小于 PCT，但效果并没有非常显著，这是由于采用的数据集中点云点数不算多，在将来工作中，本文将会进一步测试两种方法在大型室内场景数据集中的性能与效率表现。

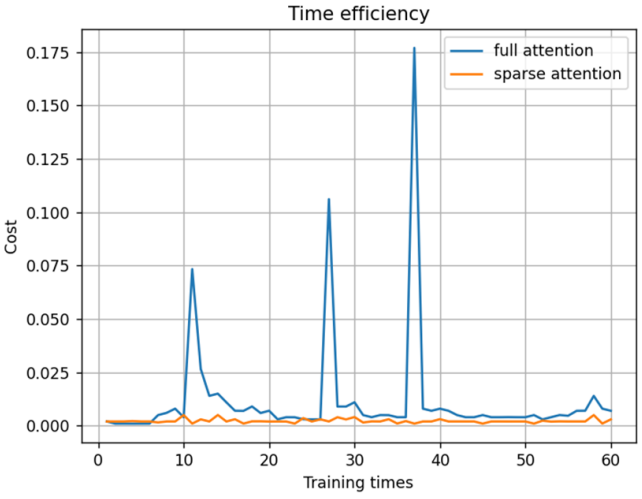

图 9. 测试 1

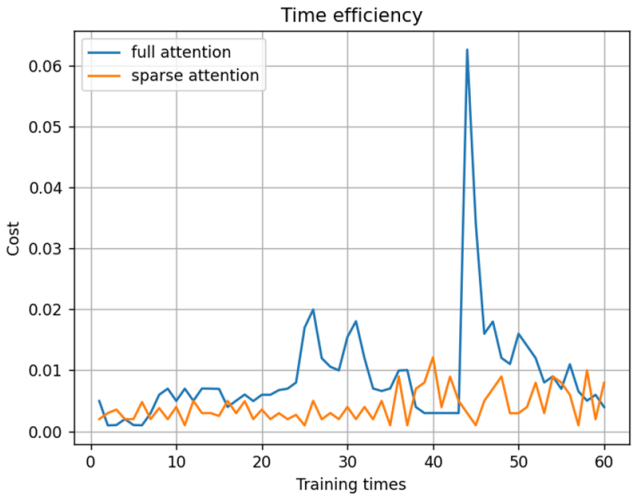

图 10. 测试 2

6 总结与展望

本文成功实现并测试了 Point Cloud Transformer（PCT），一种基于 Transformer 架构的新点云处理框架。PCT 通过特殊的注意力机制和邻域嵌入策略，优化了点云的局部和全局特征表示。进一步的，本文设计并实现了基于稀疏注意力机制的 Sparse PCT，对 PCT 进行了效率上的改进。实验结果表明，在 ModelNet40 数据集上，PCT 及其改进版本 Sparse PCT 在形状分类任务上均优于现有的 PointNet 基准，显示出 PCT 框架的有效性。

最后，本文的未来工作将继续优化 Sparse PCT 的设计，目前的构思是在低分辨率点云的选取上不只是进行随机选取，而是考虑语义信息与空间信息；同时在实验上，本文将集中于评估 PCT 与 Sparse PCT 在处理大型室内场景数据集方面的性能和效率。

参考文献

[1] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[2] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[3] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *2017 international conference on 3D vision (3DV)*, pages 537–547. IEEE, 2017.

[4] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018.

[5] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (TOG)*, 37(6):1–12, 2018.

[6] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 9621–9630, 2019.

[7] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018.

[8] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4558–4567, 2018.

[9] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[10] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

[11] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5589–5598, 2020.

[12] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. Pointgmm: A neural gmm network for point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12054–12063, 2020.