

题目：OctFormer: Octree-based Transformers for 3D Point Clouds

摘要

摘要：本文提出了基于八叉树的 transformer，命名为 OctFormer，用于 3D 点云学习。OctFormer 不仅可以作为三维点云分割和目标检测的通用和有效的骨干，而且具有线性复杂性和可扩展性，适用于大规模点云。将变压器应用于点云的关键挑战是降低二次元的计算复杂度。为了解决这个问题，一些研究将点云划分为不重叠的窗口，并在每个局部窗口中约束注意力。然而，每个窗口中的点数变化很大，阻碍了在 GPU 上的高效执行。观察到注意力对局部窗口形状的鲁棒性，本文提出了一种新的八叉树注意力，该注意力利用八叉树的排序 Z-order 曲线将点云划分为包含固定数量点的局部窗口，同时允许窗口形状自由变化。本文还引入 dilated 分区来进一步扩大感受野。

关键词：transformer；八叉树

1 引言

以往基于点云的 transformer 方法是将点云进行体素化然后在一个体素中的点云当作一个 patch，再进行 embedding，这样做的缺陷是每个体素中的点云数相差很多，有的体素中包含稀疏的点云而有的体素中包含稠密的点云，这将导致 GPU 极大的计算复杂度，同时效率低下，为了降低其计算复杂度，这篇文章介绍了一种名为 OctFormer 的基于新型八叉树的 transformer，用于 3D 点云处理。为此，OctFormer 采用了一种新颖的八叉树注意力机制，通过将点云划分为包含固定点数的局部窗口，使得窗口的形状可以自由变化。此外，文章还提出了基于 dilated 分区的八叉树注意力机制以进一步增强模型的感受野。OctFormer 展示了出色的性能，特别是在 ScanNet200 [3] 数据集上，相比于现有的稀疏体素基 CNN 和基于点云的 transformer，其效率和有效性都有显著提升。而选择本篇论文进行复现可以加深我对基于点云 transformer 方法的理解，同时在最前沿的方法上跑基于本人课题的数据集是很有参考价值的。

2 相关工作

2.1 基于体素的 CNNs

基于体素的 CNN [6] [19] [22] 是将 2D 的 CNN 方法推广到 3D 数据上，而基于体素的 CNN 是通过均匀采样体素表示 3D 数据的。然而，由于体素分辨率的立方计算和内存成本都

很高，这些方法只能采用 32^3 这样低分辨率的体素作为输入。基于稀疏体素的 CNN 通过将 CNN 操作约束为非空稀疏体素并采用八叉树或哈希表来查找临近点提高基于全体素 CNN 的效率，而基于稀疏体素的 CNN 主要使用小卷积核，而实践证明大卷积核可以大大提高网络性能。本文的 OctFormer 可以很容易地扩大窗口大小并且具有基于稀疏体素 CNN 更大的感受野。

2.2 基于点的神经网络方法

基于点的神经网络方法 [14] [15] [16] 是将原始点云作为输入，而不是转化成 3D 体素，但是基于点的方法为了查询临近点，将输入点云构造一个 k 近邻图，同时为了网络能够提取层次特征，基于点的方法采用最远点采样和基于网格的采样来对点云进行逐步下采样，而本文所提出的 OctFormer 也以点云作为输入，但是基于八叉树的注意力机制完全避免了昂贵的 k 近邻搜索和最远点采样。因此，OctFormer 的效率要高得多，并且优于以前的基于点的网络。

2.3 基于视觉的 transformer

受到 transformer 在自然语言处理中 [17] 的巨大成功的启发，ViT [1] 使用基于 transformer 的网络进行视觉识别。ViT 将输入图像划分为不重叠的规则小块，将每个小块视为一个标记，并对这些标记进行关注。为了将 ViT 扩展到图像分割和检测等密集预测任务，PVT [20] 和 Swin Transformer [10] 引入了从 CNN 到视觉 transformer 的分层网络架构。PVT 提出在下采样特征映射上应用关注模块，以提高 ViT 处理大图像时的效率。另一方面，Swin Transformer 引入了移位窗口关注来限制非重叠局部窗口的关注。

2.4 基于点云的 transformer

继基于视觉的 transformer 之后，探索 transformer 对点云理解的扩展是很自然的。点云 transformer(PCT) [7] 将偏移关注应用于所有点特征，用于点云分类和分割。point-bert [24] 和 point-mae [12] 利用在点云上训练的标准 transformer 进行无监督预训练。3DETR [11] 提出了一种使用标准 transformer 进行点云检测的端到端方案。然而，由于全局关注带来的高计算和内存成本，这些方法仅限于仅包含几千个点的点云。

2.5 基于点的 transformer

基于点的 transformer(PT) [4] 将向量关注应用于每个点的局部邻域。虽然 PT 比 PCT 具有更低的内存开销，但由于在池化操作下采样特征映射时使用昂贵的最远点采样，其计算开销仍然很高。而 PTv2 [21] 通过用基于网格的采样代替最远点采样来提高 PT 的效率，并通过利用分组向量关注和额外的位置编码乘法器进一步提高其性能。PT 和 PTv2 的注意力模块都以滑动窗口的方式独立应用于每个点的局部邻域。由于重叠邻域之间没有计算共享，浪费了大量的计算资源。为了扩大点云 transformer 的规模，SST [5]、SWFormer [18] 和分层 transformer [9] 通过将注意力模块限制在点云的非重叠窗口，将 Swin Transformer 用于图像理解扩展到点云。由于每个局部窗口的点数差异较大，所以 SST 和 SWFormer 将点数相近的局部窗口组合在一起，进行批处理；而分层 transformer 利用复杂的 GPU 程序来解决这个问题。与 PVT 的缩放策略类似，PatchFormer [2] 将注意力模块应用于补丁特征而不是点特

征;Fast Point Transformer [13] 将点云采样为低分辨率体素，并将注意力模块限制在这些体素上。然而，PatchFormer 和 Fast Point Transformer 的性能比其他当代点云 transformer 差。与之前的点云 transformer 不同，本文提出的 OctFormer 将输入点云分成包含相同数量点的组，使其易于并行化和扩展。同时 OctFormer 还在大规模基准测试中实现了最先进的语义分割和对象检测性能。

3 本文方法

3.1 本文方法概述

图 1 显示了 OctFormer 的概述。给定一个输入点云，首先用指定的比例因子对其进行归一化，并将其转换为八叉树。初始特征包括存储在非空八叉树叶节点中的平均点位置、颜色和点法线（如果提供的话）。然后使用嵌入模块下采样并将初始特征投影到高维空间中。接下来，交替应用一系列 OctFormer 块和下采样模块来生成高级层次特征，这些特征可以被轻量级特征金字塔网络（FPN）[8] 用作语义分割和目标检测。OctFormer 的核心是一种新型的八叉树注意力模块，如图 1-(b) 所示，在 3.2 节中详细阐述。其他网络组件，包括嵌入和下采样模块，OctFormer 块，以及详细的网络配置，将在 3.3 节中介绍。

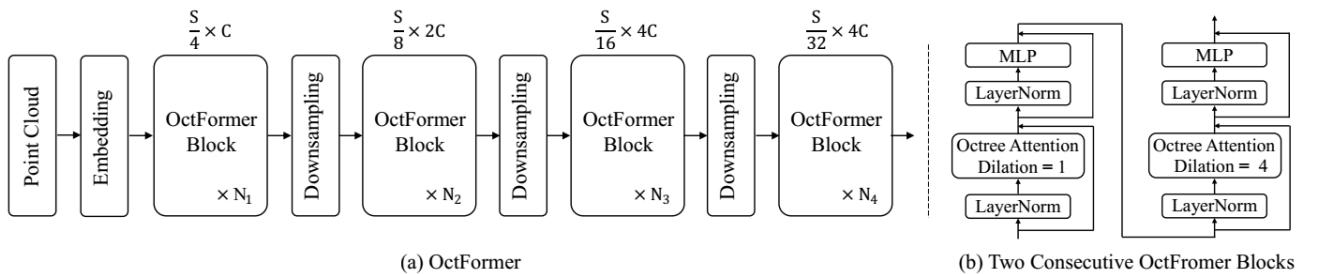


图 1. (a): OctFormer 的架构。OctFormer 由嵌入模块、一系列 OctFormer 块和下采样模块组成。 S 和 C 代表特征的空间分辨率和通道数， N_i 表示对应的 OctFormer 块的数量。(b): 两个连续的 OctFormer 块。每个 OctFormer 块由一个八叉树注意力机制、一个 MLP 和两个层规范化 (LayerNorm) 组成。两个连续的 OctFormer 块对各自的八叉树关注使用 1 和 4 的扩展。

3.2 基于八叉树的注意力机制

注意力机制: 八叉树注意力建立在缩放点积注意力的基础上，该注意力被广泛应用于 NLP 和视觉中的 transformer。本文首先回顾注意模块，然后介绍了激发八叉树注意的观察结果。将一个输入特征映射表示为 $X \in \mathbb{R}^{N \times C}$ ，其中 N 是空间号，而 C 是通道号。其中三个可学习的权重矩阵 W_q 、 W_k 和 W_v 用于将 X 映射到查询、键和值。那么注意力可以定义如下：

$$\text{Attention}(X) = \text{softmax} \left(\frac{(XW_q)(XW_k)^T}{\sqrt{d_k}} \right) (XW_v). \quad (1)$$

八叉树: 本文采用八叉树来生成所需的窗口分区，便于分层的网络架构。对于每个点云，本文采用 Zhou 等人提出的并行算法，在 GPU 上构建八叉树。在构建八叉树之后，使用打乱

键对相同深度的八叉树节点进行排序。将一个八叉树节点的整数坐标表示为 (x, y, z) , 每个坐标的 i^{th} 位分别表示为 x_i , y_i 和 z_i , 二元表达式中的打乱键定义如下:

$$\text{Key}(x, y, z) = x_1y_1z_1x_2y_2z_2\dots x_dy_dz_d \quad (2)$$

其中 d 是八叉树的深度。打乱键的值表示在 3Dz-order 曲线上的位置。我们观察到八叉树节点属于一个父节点, 更一般地说, 属于一个子树的八叉树节点按照 z 顺序连续存储在内存中。八叉树的这个属性是我们高效窗口划分的关键。图 2 显示了一个 2D 插图。对于由图 2-(a) 中用红色标记的点云构建的八叉树, 该八叉树深度 3 处对应的 z 阶曲线如图 2-(b) 所示。由于八叉树节点稀疏, 完整的 z 阶曲线如图 2-(e) 所示, 供参考

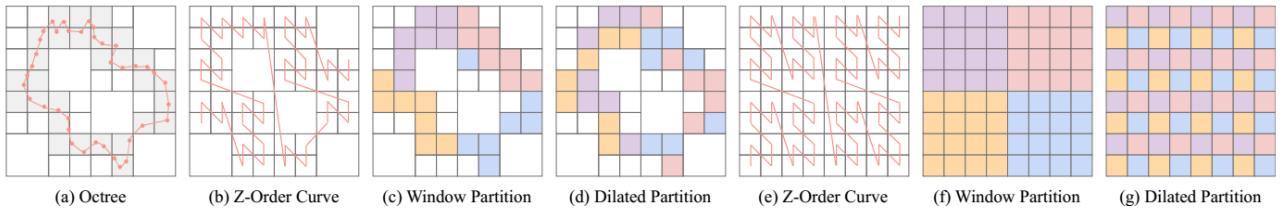


图 2. 这里显示的是 2D 图像, 以便更好地说明。(a): 从红色形状和相应的八叉树(四叉树)中采样的输入点云。非空的八叉树节点以灰色突出显示。(b): 八叉树深度 3 处的 z -order 曲线。(c): 与 (b) 对应的张量重塑转置生成的窗口分区, 点号为 7。特征按照 z -order 曲线上非空八叉树节点的顺序存储在张量中。(d): 点数为 7, 扩张系数为 2 的扩张分区。(e): 覆盖整个空间的 z 阶曲线。(f): 与 (e) 对应的窗口分区, 点号为 16。(g): 与 (e) 对应的扩展分区, 点数为 16, 扩展为 4。

基于八叉树的注意力机制: 通过对八叉树的键进行排序, 我们可以很容易地通过张量重塑和转置来生成窗口分区。注意, 只有非空的八叉树节点包含特征向量。同时可以利用原始点云构建八叉树提供的信息过滤掉非空结点, 如图 2-(a) 所示, 其中非空的八叉树节点用灰色标记。我们将非空八叉树节点中的所有特征按照排序后的打乱键的顺序堆叠成一个二维张量。将得到的特征张量表示为 X , 形状为 (N, C) , 我们首先填充零来创建一个形状为 (\hat{N}, C) 的新张量 \hat{X} , 以便二进制可以被指定的点数 K 整除, 在我们的实验中通常设置为 32。然后我们通过将 \hat{X} 重塑为 (B, K, C) 来生成窗口分区, 其中 B 等于 \hat{N}/K , 表示窗口的总数。有了这个划分, 我们可以通过将等式 1 中的注意力并行地应用于这些变量变量窗口来实现我们的八叉树注意力, 同时屏蔽填充元素。图 2-(c) 显示了一个示例, 其中 K 和 N 分别等于 7 和 28。位置编码: 位置编码对于区分在不同位置定义的特征至关重要。一种广泛使用的策略是添加相对位置偏差到公式 1 定义的注意力机制中。假设表示最大窗口大小为 W , 任意两个点在同一窗口内的相对位置为 $[-W+1, W-1]$ 。将相对位置偏差扩展到 3D 需要 $H * (2W-1)^3$ 个可训练参数, 其中 H 是注意力的头部数。对于我们的八叉树, 当膨胀大于 4 时, 最大窗口大小至少为 50, 这大大增加了我们的 OctFormer 的参数。因此本文采用条件位置编码 (CPE)

$$X = X + \text{batch_norm}(\text{depth_wise_conv}(X)) \quad (3)$$

使用 CPE, 使用比相对位置偏差更少的参数时, 网络性能显著提高。

总结: 基于八叉树的注意力机制非常容易实现, 比以前基于点云的注意力机制更有效。使用 Web 上的开源库, 我们可以在 10 行代码实现基于八叉树的注意力机制, 正如算法 1 中所

Algorithm 1 Pseudocode of Octree Attention in a PyTorch-like style.

- 1: x : input tensor with a shape of (N, C)
- 2: D : dilation for attention, set to 1 or 4 by default
- 3: P : partition number in each window, set to 32 by default
- 4: $attention$: an object of `torch.nn.MultiheadAttention`
- 5:
- 6: Apply conditional positional encoding
- 7: $x = x + \text{batch_norm}(\text{depth_wise_conv}(x))$
- 8:
- 9: Window partition
- 10: $C, N = x.\text{shape}$
- 11: $Nz = (P * D) - N \% (P * D)$ # number of zeros for padding
- 12: $x = \text{torch.cat}([x, x.\text{new_zeros}((Nz, C))], 1)$ # pad zeros
- 13: $x = x.\text{reshape}(-1, P, D, C).\text{transpose}(1, 2).\text{flatten}(0, 1)$
- 14:
- 15: Attention mask
- 16: $m = \text{torch.cat}([x.\text{new_zeros}(N), x.\text{new_ones}(Nz)]).\text{bool}()$
- 17: $m = m.\text{reshape}(-1, P, D).\text{transpose}(1, 2).\text{flatten}(0, 1)$
- 18:
- 19: Apply attention
- 20: $x = attention(\text{query} = x, \text{key} = x, \text{value} = x, \text{key_padding_mask} = m)$
- 21:
- 22: Reverse window partition
- 23: $x = x.\text{reshape}(-1, D, P, C).\text{transpose}(1, 2).\text{reshape}(-1, C)$
- 24:
- 25: **return** $x[:N]$ # remove the padded elements

示。我们的八叉树注意力的核心可以通过 PyTorch - torch.nn.MultiheadAttention 提供的多头注意力模块来实现，该模块基于 GPU 上的一般矩阵乘法例程进行了高度优化。

3.3 网络细节

嵌入模块：本文选择了几个具有小核尺寸的卷积层，这已经被证明能够稳定 transformer 的训练过程。在 OctFormer 中，本文为嵌入模块使用了一系列 5 个基于八叉树的卷积模块。每个模块由一个八叉树卷积，一个归一化层，一个批处理层和一个 ReLU 激活层，这些八叉树卷积的核大小和步长分别为 {3,2,3,2,3} 和 {1,2,1,2,1}。步幅为 2 的卷积下采样张量的空间分辨率为 2 倍。

OctFormer 块：按照标准 transformer 块设计，OctFormer 块由第 3.1 节介绍的八叉树注意力机制、多层次感知器 (MLP) 和 residual 连接组成，如图 1-(b) 所示。MLP 有两个完全连接的层，中间有一个 GELU 激活函数，MLP 隐藏通道的扩展比设置为 4。在每个注意力模块和每个 MLP 之前使用层归一化 (LayerNorm) 来稳定训练。在两个连续的 OctFormer 块中，八叉树注意力的扩展分别设置为 1 和 4。

下采样层：采样模块实现为核大小为 2，步长为 2 的八叉树卷积，然后是批处理归一化层，该层降低了空间分辨率，并将特征映射的通道增加了 2 倍。

网络参数配置：默认情况下 C 设置为 96，块数设置为 {2,2,18,2}，八叉树关注头数设置为通道数的 1/16，窗口划分点数设置为 32。得到的 OctFormer 具有与 MinkowskiNet (38M) 相似的可训练参数数量 (39M)，OctFormer 的输出是四个空间分辨率的层次特征，即 { $S/4$, $S/8$, $S/16$, $S/32$ }，可以方便地与特征金字塔网络 (FPN) 集成来用于分割和检测，最后的特征映射可以取均值作为形状分类的全局特征。

4 复现细节

4.1 与已有开源代码对比

本文所提及的代码已经开源，但是原文只是在 shapenet 和 scannet 数据集上进行实验，更大的室外场景，城市场景本文并没用进行实验，因此该部分需要自己写对应的脚本文件进行实验。由于本实验使用的是 Urbanbis 数据集 [23]，该数据集中一个场景的点云数量可以达到千万级别，与本文用到的数据集的点云数根本不是一个量级，再加上数据格式与原文用到的数据集文件格式不是相似的，所以处理起来十分困难，但是在逐步剖析代码原理和了解 Urbanbis 数据集格式的情况下，对 Urbanbis 数据集进行 0.5 的随机下采样，最后实现了在此数据集上的 600Epoch 的训练，训练结果也跟原文相差不是很大。

4.2 实验环境搭建

1. 安装 Conda 并创建一个 Conda 环境

```
conda create --name octformer python=3.8
conda activate octformer
```

2. 根据官方文件使用 conda 安装 PyTorch-1.12.1

```
conda install pytorch==1.12.1 torchvision torchaudio cudatoolkit=11.3 -c pytorch
```

3. 克隆此存储库并安装 requirement

```
git clone https://github.com/octree-nn/octformer.git  
cd octformer  
pip install -r requirements.txt
```

4. 安装基于八叉树的深度卷积库。

```
git clone https://github.com/octree-nn/dwconv.git  
pip install ./dwconv
```

4.3 八叉树的构建以及 Z-order 曲线的构造

本部分对实验中一些概念的可视化展示，比如八叉树的构建以及 Z-order 的构建，同时包括将 Z-order 曲线进行切分不同 patch 的可视化展示。

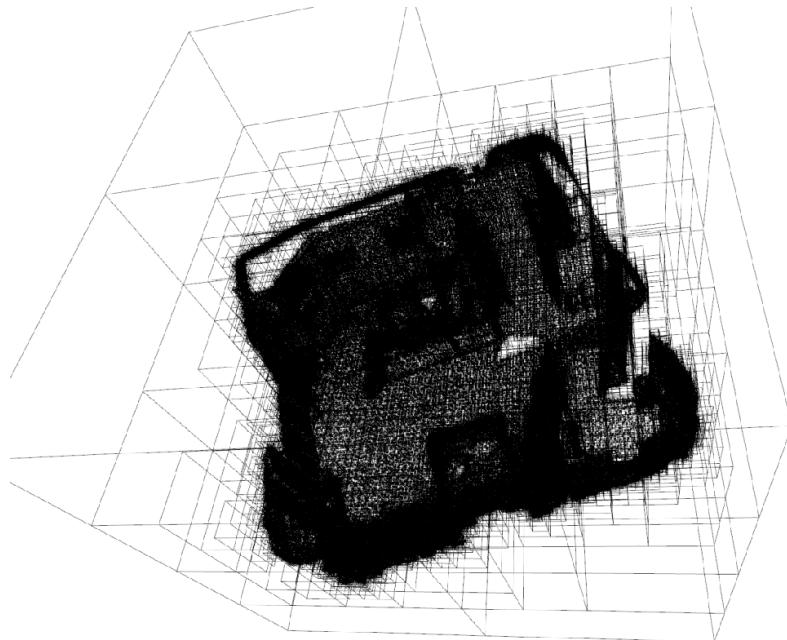


图 3. 基于 Scannet 数据集的八叉树的构建

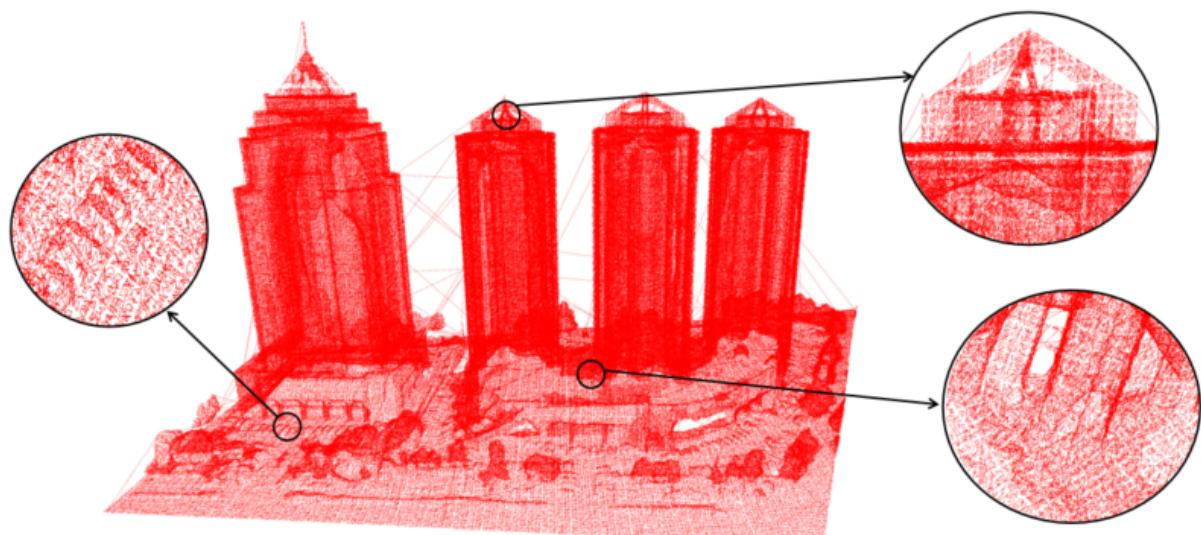


图 4. 基于 Urbanbis 数据集 Z-order 曲线的构建

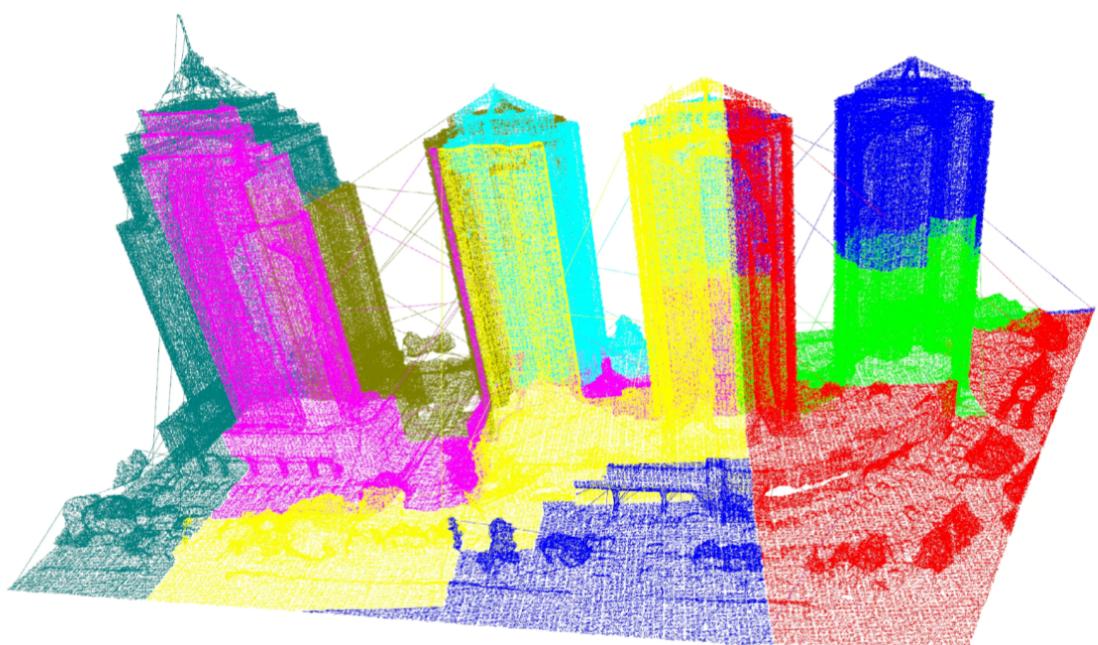


图 5. 基于 Urbanbis 数据集 Z-order 曲线的不同 patch 构建（本图分为 8 个 patch）



图 6. 根据 patch 构建的点云分区（本图分为 8 个 patch）

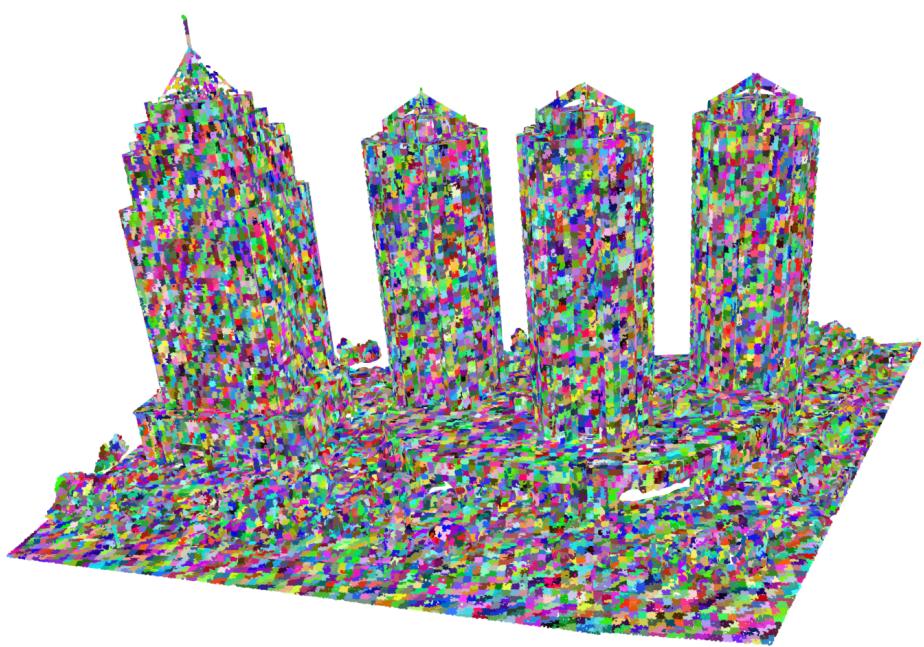


图 7. 按照文章中的 patch size (32) 进行点云分区

4.4 创新点

将此方法应用在城市场景，并对城市场景的数据集在传入网络前进行了特殊处理。

5 实验结果分析

Epoch: 365	test/loss: 0.776	test/accu: 0.747	test/mIoU: 0.505
Epoch: 370	test/loss: 0.671	test/accu: 0.765	test/mIoU: 0.514
Epoch: 375	test/loss: 0.636	test/accu: 0.794	test/mIoU: 0.543
Epoch: 380	test/loss: 0.546	test/accu: 0.801	test/mIoU: 0.548
Epoch: 385	test/loss: 0.573	test/accu: 0.798	test/mIoU: 0.545
Epoch: 390	test/loss: 0.530	test/accu: 0.792	test/mIoU: 0.545
Epoch: 395	test/loss: 0.532	test/accu: 0.803	test/mIoU: 0.561
Epoch: 400	test/loss: 0.414	test/accu: 0.840	test/mIoU: 0.604
Epoch: 405	test/loss: 0.505	test/accu: 0.820	test/mIoU: 0.567
Epoch: 410	test/loss: 0.436	test/accu: 0.839	test/mIoU: 0.601
Epoch: 415	test/loss: 0.552	test/accu: 0.794	test/mIoU: 0.539
Epoch: 420	test/loss: 0.434	test/accu: 0.833	test/mIoU: 0.594
Epoch: 425	test/loss: 0.566	test/accu: 0.783	test/mIoU: 0.514
Epoch: 430	test/loss: 0.566	test/accu: 0.799	test/mIoU: 0.573
Epoch: 435	test/loss: 0.444	test/accu: 0.814	test/mIoU: 0.535
Epoch: 440	test/loss: 0.537	test/accu: 0.815	test/mIoU: 0.575
Epoch: 445	test/loss: 0.427	test/accu: 0.823	test/mIoU: 0.551
Epoch: 450	test/loss: 0.583	test/accu: 0.796	test/mIoU: 0.559
Epoch: 455	test/loss: 0.425	test/accu: 0.824	test/mIoU: 0.552
Epoch: 460	test/loss: 0.629	test/accu: 0.788	test/mIoU: 0.546
Epoch: 465	test/loss: 0.479	test/accu: 0.821	test/mIoU: 0.549
Epoch: 470	test/loss: 0.561	test/accu: 0.814	test/mIoU: 0.579
Epoch: 475	test/loss: 0.507	test/accu: 0.810	test/mIoU: 0.557
Epoch: 480	test/loss: 0.546	test/accu: 0.805	test/mIoU: 0.564
Epoch: 485	test/loss: 0.466	test/accu: 0.817	test/mIoU: 0.567
Epoch: 490	test/loss: 0.528	test/accu: 0.821	test/mIoU: 0.572
Epoch: 495	test/loss: 0.436	test/accu: 0.831	test/mIoU: 0.581
Epoch: 500	test/loss: 0.540	test/accu: 0.809	test/mIoU: 0.557
Epoch: 505	test/loss: 0.493	test/accu: 0.801	test/mIoU: 0.554
Epoch: 510	test/loss: 0.521	test/accu: 0.806	test/mIoU: 0.563
Epoch: 515	test/loss: 0.407	test/accu: 0.846	test/mIoU: 0.609
Epoch: 520	test/loss: 0.608	test/accu: 0.785	test/mIoU: 0.534
Epoch: 525	test/loss: 0.458	test/accu: 0.839	test/mIoU: 0.598
Epoch: 530	test/loss: 0.607	test/accu: 0.789	test/mIoU: 0.537
Epoch: 535	test/loss: 0.466	test/accu: 0.838	test/mIoU: 0.598
Epoch: 540	test/loss: 0.642	test/accu: 0.782	test/mIoU: 0.513
Epoch: 545	test/loss: 0.606	test/accu: 0.800	test/mIoU: 0.573
Epoch: 550	test/loss: 0.463	test/accu: 0.822	test/mIoU: 0.544
Epoch: 555	test/loss: 0.579	test/accu: 0.812	test/mIoU: 0.571
Epoch: 560	test/loss: 0.475	test/accu: 0.815	test/mIoU: 0.545
Epoch: 565	test/loss: 0.601	test/accu: 0.803	test/mIoU: 0.566
Epoch: 570	test/loss: 0.431	test/accu: 0.834	test/mIoU: 0.564
Epoch: 575	test/loss: 0.632	test/accu: 0.796	test/mIoU: 0.553
Epoch: 580	test/loss: 0.455	test/accu: 0.829	test/mIoU: 0.559
Epoch: 585	test/loss: 0.587	test/accu: 0.799	test/mIoU: 0.562
Epoch: 590	test/loss: 0.524	test/accu: 0.813	test/mIoU: 0.560
Epoch: 595	test/loss: 0.630	test/accu: 0.780	test/mIoU: 0.538
Epoch: 600	test/loss: 0.535	test/accu: 0.805	test/mIoU: 0.553

图 8. MIoU 和 Acc 的结果

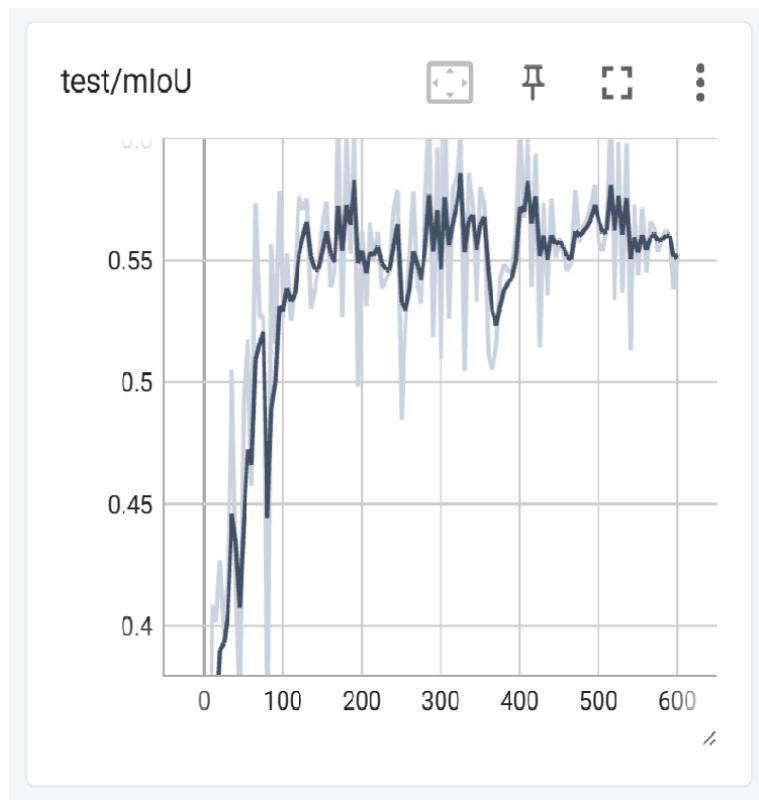


图 9. MIoU 曲线

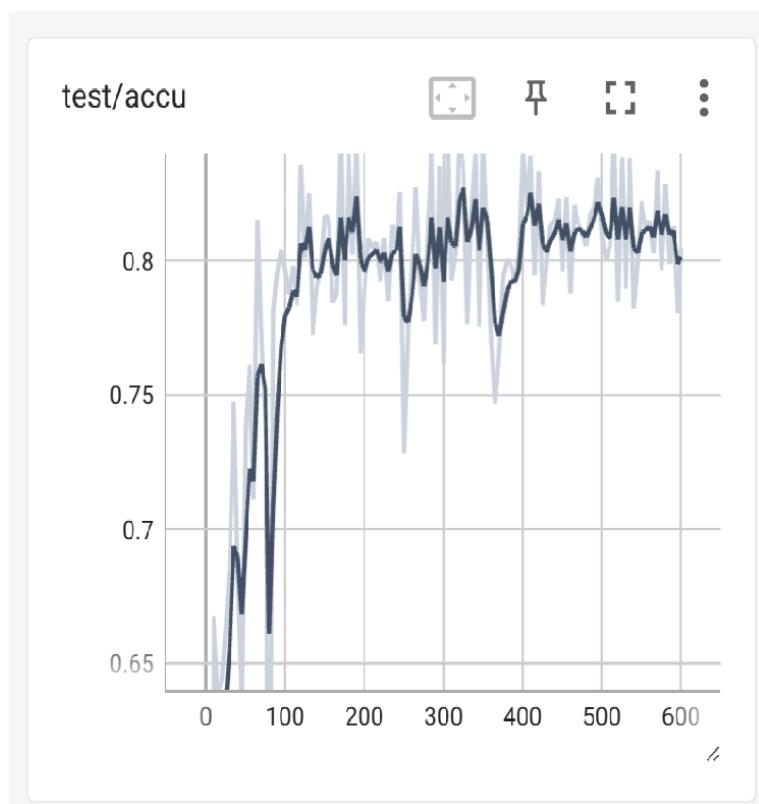


图 10. Acc 曲线

```

- Epoch: 123, train/loss: 0.182, train/accu: 0.936, iter: 0
-> Epoch: 123, train/loss: 0.327, train/accu: 0.877, time/data: 0.186, time/bat
  ch: 5.182, time: 2023/12/01 06:28:46, duration: 31.10s
- Epoch: 124, train/loss: 0.481, train/accu: 0.776, iter: 0
-> Epoch: 124, train/loss: 0.474, train/accu: 0.840, time/data: 0.006, time/bat
  ch: 6.123, time: 2023/12/01 06:29:23, duration: 36.74s
- Epoch: 125, train/loss: 0.267, train/accu: 0.874, iter: 0
-> Epoch: 125, train/loss: 0.432, train/accu: 0.837, time/data: 0.007, time/bat
  ch: 6.028, time: 2023/12/01 06:29:59, duration: 36.17s
=> Epoch: 125, test/mIoU_part: 0.218958
-> Epoch: 125, test/loss: 0.563, test/accu: 0.801, test/mIoU: 0.571, test/intsc
  _0: 0.000, test/intsc_1: 5785.778, test/intsc_2: 0.000, test/intsc_3: 0.000,
  test/intsc_4: 0.000, test/intsc_5: 0.000, test/intsc_6: 1443.889, test/union
  _0: 0.000, test/union_1: 7208.445, test/union_2: 0.000, test/union_3: 0.389
  , test/union_4: 53.056, test/union_5: 0.000, test/union_6: 2825.278, test/mI
  oU_part: 0.219, time: 2023/12/01 06:30:01, duration: 2.36s
- Epoch: 126, train/loss: 0.288, train/accu: 0.904, iter: 0
-> Epoch: 126, train/loss: 0.448, train/accu: 0.842, time/data: 1.132, time/bat
  ch: 3.807, time: 2023/12/01 06:30:25, duration: 22.84s
- Epoch: 127, train/loss: 0.623, train/accu: 0.751, iter: 0
-> Epoch: 127, train/loss: 0.528, train/accu: 0.777, time/data: 1.928, time/bat
  ch: 5.823, time: 2023/12/01 06:31:00, duration: 34.94s
- Epoch: 128, train/loss: 0.419, train/accu: 0.891, iter: 0
-> Epoch: 128, train/loss: 0.463, train/accu: 0.814, time/data: 4.782, time/bat
  ch: 6.599, time: 2023/12/01 06:31:48, duration: 39.60s
- Epoch: 129, train/loss: 0.365, train/accu: 0.889, iter: 0
-> Epoch: 129, train/loss: 0.427, train/accu: 0.832, time/data: 2.788, time/bat
  ch: 4.653, time: 2023/12/01 06:32:08, duration: 27.92s
- Epoch: 130, train/loss: 0.655, train/accu: 0.730, iter: 0
-> Epoch: 130, train/loss: 0.477, train/accu: 0.807, time/data: 2.945, time/bat
  ch: 5.289, time: 2023/12/01 06:32:40, duration: 31.74s
=> Epoch: 130, test/mIoU_part: 0.208940
-> Epoch: 130, train/loss: 0.481, test/accu: 0.825, test/mIoU: 0.575, test/intsc
  _0: 0.000, test/intsc_1: 5760.111, test/intsc_2: 0.000, test/intsc_3: 0.000,
  test/intsc_4: 0.000, test/intsc_5: 0.000, test/intsc_6: 1403.611, test/union
  _0: 0.000, test/union_1: 7374.667, test/union_2: 0.000, test/union_3: 0.389
  , test/union_4: 53.056, test/union_5: 0.000, test/union_6: 2970.167, test/mI
  oU_part: 0.209, time: 2023/12/01 06:32:42, duration: 2.33s
- Epoch: 131, train/loss: 0.274, train/accu: 0.887, iter: 0
-> Epoch: 131, train/loss: 0.394, train/accu: 0.844, time/data: 0.766, time/bat
  ch: 5.221, time: 2023/12/01 06:33:15, duration: 31.33s
- Epoch: 132, train/loss: 0.292, train/accu: 0.895, iter: 0
-> Epoch: 132, train/loss: 0.402, train/accu: 0.838, time/data: 3.932, time/bat
  ch: 5.504, time: 2023/12/01 06:33:48, duration: 33.03s
- Epoch: 133, train/loss: 0.520, train/accu: 0.785, iter: 0
-> Epoch: 133, train/loss: 0.472, train/accu: 0.815, time/data: 2.043, time/bat
  ch: 5.115, time: 2023/12/01 06:34:18, duration: 30.69s
- Epoch: 134, train/loss: 0.325, train/accu: 0.870, iter: 0
22% | 133/600 [1:13:47<4:11:03, 32.26s/it]
17% | 1/6 [00:05<00:28, 5.65s/it]

```

图 11. 训练时间展示

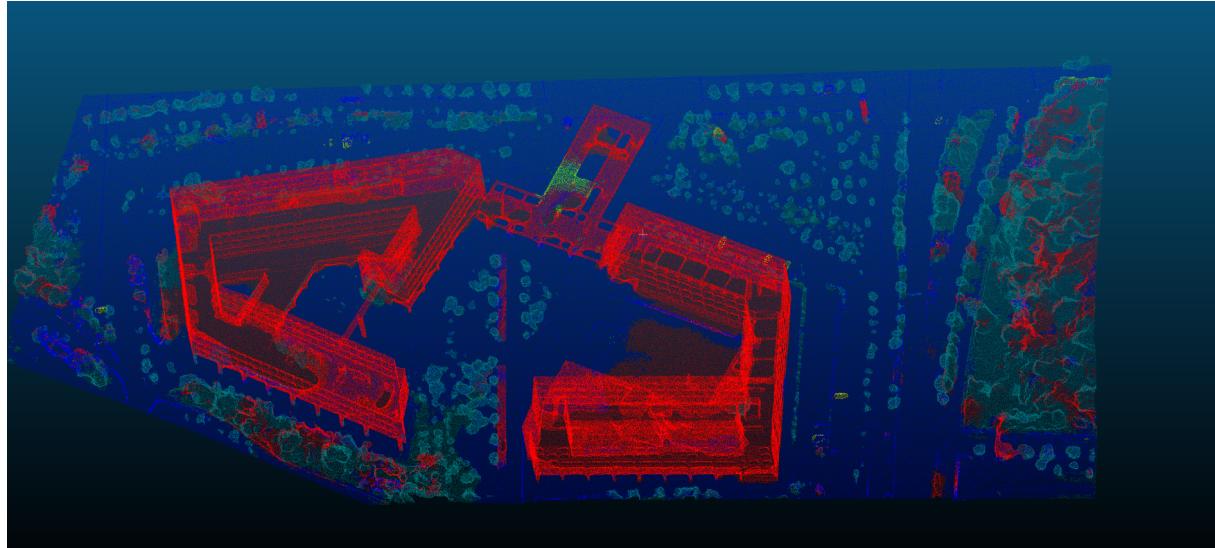


图 12. 分割结果

在本实验中，我用的是 3 张 3090 进行的实验并将 batch size 设置为 4，从图 11 可以看到训练总时间为 4 小时 11 分钟，可见该网络具有极高的训练效率，而从图 8 可以看到最好的 MIoU 可以达到 0.609，Acc 可以达到 0.86，原文中用 Scannet 数据集 MIoU 可以达到 0.7，虽然未达到文中的 MIoU，但是我用的数据集比 Scannet 数据集大，而且相差不是很大。

6 总结与展望

从本实验可看出虽然该网络具有极高的训练速度，但是分割效果不是很好，与其他全监督的网络的精准度还是相差很大的，同时，本人对按照 Z-order 划分 patch 的合理性和可解释性存在疑惑，未来的工作可以考虑往更好的切分方式来思考同时也可以考虑如何提高精准度。

参考文献

- [1] An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.
- [2] Zhang Cheng, Haocheng Wan, Xinyi Shen, and Zizhao Wu. Patchformer: An efficient point transformer with patch attention. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11789–11798, 2021.
- [3] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2432–2443, 2017.
- [4] Nico Engel, Vasileios Belagiannis, and Klaus C. J. Dietmayer. Point transformer. *IEEE Access*, 9:134826–134840, 2020.
- [5] Lue Fan, Ziqi Pang, Tianyuan Zhang, Yu-Xiong Wang, Hang Zhao, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. Embracing single stride 3d object detector with sparse transformer. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8448–8458, 2021.
- [6] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2017.
- [7] Meng-Hao Guo, Junxiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph Robert Martin, and Shimin Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7:187 – 199, 2020.
- [8] Alexander Kirillov, Ross B. Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6392–6401, 2019.
- [9] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified transformer for 3d point cloud segmentation. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8490–8499, 2022.

- [10] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021.
- [11] Ishan Misra, Rohit Girdhar, and Armand Joulin. An end-to-end transformer model for 3d object detection. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2886–2897, 2021.
- [12] Yatian Pang, Wenxiao Wang, Francis E. H. Tay, W. Liu, Yonghong Tian, and Liuliang Yuan. Masked autoencoders for point cloud self-supervised learning. In *European Conference on Computer Vision*, 2022.
- [13] Chunghyun Park, Yoonwoo Jeong, Minsu Cho, and Jaesik Park. Fast point transformer. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16928–16937, 2021.
- [14] C. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas. Deep hough voting for 3d object detection in point clouds. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9276–9285, 2019.
- [15] C. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2016.
- [16] C. Qi, L. Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Neural Information Processing Systems*, 2017.
- [17] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021.
- [18] Pei Sun, Mingxing Tan, Weiyue Wang, Chenxi Liu, Fei Xia, Zhaoqi Leng, and Drago Anguelov. Swformer: Sparse window transformer for 3d object detection in point clouds. In *European Conference on Computer Vision*, 2022.
- [19] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn. *ACM Transactions on Graphics (TOG)*, 36:1 – 11, 2017.
- [20] Wenhui Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvt v2: Improved baselines with pyramid vision transformer. *Computational Visual Media*, 8:415 – 424, 2021.
- [21] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. *ArXiv*, abs/2210.05666, 2022.

- [22] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3 d shapenets : A deep representation for volumetric shape modeling. 2015.
- [23] Guoqing Yang, Fuyou Xue, Qi Zhang, Ke Xie, Chi-Wing Fu, and Hui Huang. Urbanbis: a large-scale benchmark for fine-grained urban building instance segmentation. *ACM SIGGRAPH 2023 Conference Proceedings*, 2023.
- [24] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19291–19300, 2021.