Project-3 of “Neural Network and Deep Learning”

陈亦雄 16307110231

一、前言

这篇报告的目的是探索三维点云数据的存储形式与设计点云分类的深度网络模型。文章包含三个部分：首先尝试如何将 HDF5 数据集文件中的一个三维点云提取出来保存为.obj 文件并进行可视化；第二步使用提取出来的三维数据集进行建模，并训练一个性能良好的深度学习模型；最后讨论一些与 3 维点云相关的问题。

二、探究三维数据的组织形式

2.1 创建 dataloader 用于读取数据

本次实验使用的是 ModelNet40 数据集，训练集和测试集使用的.h5 文件名分别保存在了各自的 txt 文件中。使用预定义好的 Dataset 类迭代读取训练、测试需要的数据文件，并将从文件中提取出来的数据拼接为一个 numpy 数组供 dataloader 进一步读取。读取数据后，发现训练集共包含 9840 个样本，而测试集包含 2468 个数据样本。

2.2 将数据集中的单个样本提取出来，分析数据组织形式

我们将 dataloader 的 batch_size 参数设置为 1，以便提取出单个样本。在得到单个样本后，发现它是一个 python 字典，其包含‘points’，‘label’两个键。‘points’对应的值为点云数据，数据类型为 PyTorch Tensor，维度为$1 \times 3 \times 2048$。经过观察，点云数据为浮点数，每一个值都处于[-1, 1]的值域之内。而‘label’对应的值为样本标签，值域为[0, 39]范围内的整数，表示 40 类对象。

2.3 将提取出来的样本保存为 obj 格式的文件

要将单个点云数据样本（3×2048）保存为 obj 文件，我们首先要了解 obj 文件的存储格式。obj 文件是一种纯文本文件，用于保存 3 维模型的顶点空间坐标、纹理坐标、法线向量等。一般我们按照 v（顶点坐标），vt（纹理坐标），vn(法线向量)的顺序在 obj 文件中保存数据。格式非常简单，即每一行都以[数据类型 坐标]的方式保存，如下图所示：

```
mtllib cube.mtl
g default
v -0.500000 -0.500000 0.500000
v 0.500000 -0.500000 0.500000
.....
v 0.500000 -0.500000 -0.500000
vt 0.001992 0.001992
vt 0.998008 0.001992
.....
vt 0.001992 0.998008
vn 0.000000 0.000000 1.000000
vn 0.000000 0.000000 1.000000
.....
vn 0.000000 0.000000 1.000000
```

图 1.obj 文件的格式

而由于本实验中数据只涉及顶点坐标，所以仅包含 v 开头的数据行。我们将$1 \times 3 \times 2048$的张量数据转化为 ndarray 数据类型，再去掉第一个维度，然后逐行写入空文本文件中即可得到目标文件。

2.4 可视化点云

我们使用专业软件 MeshLab 进行点云可视化。选取一个边缘清晰的点云，并保存为 obj 文件使用 MeshLab 打开，如下图所示。我们可以清楚地看出，这个点云是一个玻璃饮料瓶（在数据集中类别标签为 5，bottle）。

图 2. 点云可视化

三、设计点云分类的深度模型

3.1 使用基础一维卷积神经网络

由于点云数据是以[样本数，样本中点的数量，一个点的 xyz 坐标]形式组织的，我们无法像图像数据一样使用二维卷积将局部信息进行编码。取而代之的是，我们可以利用一维卷积将 3 维对象的表面信息进行编码（直觉上空间上相邻的点也必须在数据集中相邻存储，但真实数据集中相邻的点并不一定在空间上也相邻）。在这一个模块中，首先我们使用卷积核大小为 1 的一维卷积作为核心模块，构建一个简单的神经网络模型对点云进行分类。然后我们在这个模型的基础上，实验几种模型的参数选取方式。基础模型的配置如表 1 所示，学习策略上，默认设置如表 2 所示。实验采用单块 1080Ti 显卡。

layer	type
1	Conv1d, 64 channels, kernel size 1
2	Conv1d, 128 channels, kernel size 1
3	Conv1d, 256 channels, kernel size 1
4	Max pooling
5	Linear 256×40

表 1. 默认的网络结构

parameter(s)	default setting
epochs	150
initial lr	0.1
lr scheduler	multi-step=[80,120], $\gamma = 0.1$
weight decay	0.0005
batch size	128
data augmentation	None
optimizer	SGD, momentum=0.9

表 2. 默认训练设置

默认模型仅仅包含三层一维卷积，以及一层全连接层。值得注意的是，模型中没有 Batch Normalization 操作，这是因为对于浅层网络（少于 10 层）而言只要有了合理的参数初始化（本文中对卷积层选取的是 kaiming 初始化），网络退化（梯度爆炸、消失）的现象就几乎不会发生了。此外 max pooling 层是为了解决点云数据中点的顺序问题，我们将每一个云经过三层卷积变为 256x2048 的维度后，使用该层将其转化为 256 维向量，这样可以使得提取出来的最终特征与数据点顺序无关。

实验的重点在 **1）网络层数、2）训练总 epoch 数量、3）初始学习率、4）batchsize、5）L2 正则化参数以及 6）数据增强**的选取上。实验结果如表 3-表 8 所示。

schedule (total epochs and decay epochs)	max validation accuracy	time consumption
100 epochs with lr decay=[50,75]	85.94%	280s
150 epochs with lr decay=[80,120]	87.32%	420s
240 epochs with lr decay=[135,185]	88.09%	676s
350 epochs with lr decay=[150,250]	**88.21%**	980s

表 3. 不同训练 epoch 对模型验证准确率的影响

如表 3 所示，在训练时长增加的过程中，模型验证准确率也在同向增加。其中训练 350epoch 时模型验证准确率达到最高值。这表明模型在训练的过程中并没有发生太严重的过拟合现象，导致验证准确率随着训练能够持续降低。在模型训练完毕后，训练准确率仅仅比验证准确率高 6%-7%，达到 94%左右。考虑到训练时长，我们最终选取接近最高准确率的设置——总训练 240 epoch 并于 135、185epoch 进行两次学习率衰减进行后续实验。如图 3 所示，当训练达到 240epoch 的时候，确实已经十分接近收敛了，与 350epoch 组的实验结果的差异可以用实验误差来解释。

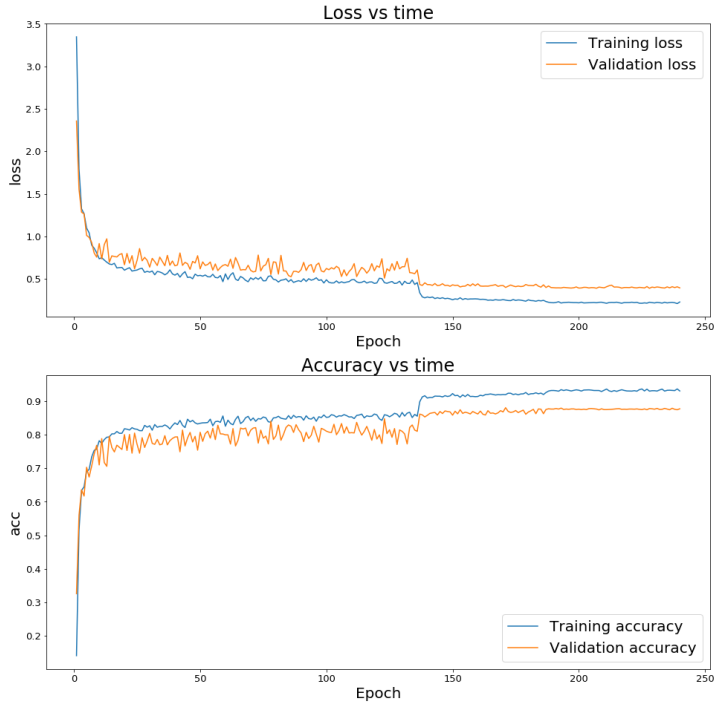

图 3. 训练 240epoch 的模型的准确率、损失曲线

接下来，我们采用 240epoch 的训练轮数来进行网络结构的选择。具体网络层数的设置如下：在通道数为 64、128 的卷积层后分别添加 1/2/3 层同通道的一维卷积。这样我们就可以获得卷积层数为 3/5/7/9 四种设置的网络结构。实验结果如表 4 所示。

从实验结果可以看出，随着网络结构变深，模型训练准确率在上升，然而模型的验证准确率却并没有什么变化。这是由于模型的容量已经超过了当前数据集的需要，过拟合的现象越来越严重所致。接下来的实验都使用训练时间最短却得到最高验证准确率的三层卷积网络。

number of conv layers	max train acc	max val acc	time consumption
3	93.69%	**88.09%**	**676s**
5	95.97%	87.40%	996s
7	**97.39%**	87.93%	1324s
9	96.09%	87.20%	1652s

表 4. 不同层数的网络得到的训练、验证最高准确率

下一步我们实验最佳初始学习率，实验结果如表 5 所示。我们发现默认的初始学习率就已经达到了最佳性能。

initial lr	max train acc	max val acc
0.3	91.56%	86.30%
0.1	**93.69%**	**88.09%**
0.03	93.38%	87.16%
0.01	91.78%	86.06%

表 5. 不同初始学习率下得到的最高训练、验证准确率

第四步测试最佳 batch size，实验结果如表 6 所示。我们发现默认的 batch size 就达到了最佳性能。但是奇怪的是，理论上 batch size 越小越能够充分利用 GPU 的计算能力，越节省训练时间，然而本组实验却有相反的现象。推测这是由于模型较小，数据的组织形式又能够充分发挥出 GPU 的计算能力，所以 batch size 的增大不会导致 GPU 利用率增加（经过观察，利用率在各个设置下都能保持为 90%以上）。而训练一个 epoch 的耗时包括正向传播与反向传播。其中正向传播总计算量与 batch size 无关，GPU 利用率恒定的情况下耗时也固定。反向传播方面由于 batch size 增大导致了一个 epoch 的迭代次数减少，反向传播的次数也减少，所以计算时长降低。然而大 batch size 会增加对 CPU 读取数据的压力。此时 CPU 需要对整个 batch 的数据进行读取并且预处理，才能放进 GPU 计算。由于实验仅使用了单线程 dataloader，数据读取时长较长，这导致了一小段的 GPU 空闲期，增加了计算总时长。综合考虑这几个因素，最终导致了 batch size 与总耗时呈反向依存关系。

batch size	max train acc	max val acc	time consumption
32	91.94%	84.93%	**571s**
64	**93.91%**	87.24%	649s
128	93.69%	**88.09%**	676s
256	93.59%	87.40%	624s
512	92.52%	86.67%	746s

表 6. 不同 batch size 下得到的最高训练、验证准确率

第五步选择最佳 L2 正则化参数，实验结果如表 7 所示。L2 正则化参数的大小反映了模型优化过程中对抗过拟合的强度，参数越大则对抗能力越强，过拟合的程度越小。从实验结果中我们发现参数越大训练准确率越低，而验证准确率则有高有低。我们选取使验证准确率达到最高的最优参数 0.0001 进行下一步的实验。

L2 reg	max train acc	max val acc
0.00001	97.54%	88.49%
0.00005	**97.70%**	88.41%
0.0001	94.76%	**88.78%**
0.0005	93.69%	88.09%
0.001	92.54%	86.71%

表 7. 不同正则化系数下得到的最高训练、验证准确率

第六步本文实验三种不同的数据增强方法。这三种方法分别为：1）对每个点云以 0-0.875 的随机概率取值丢弃部分点，2）以（0.8，1.25）之间的比例随机放缩点云，3）在 xyz 三个轴的方向上对整个点云移动（-0.1，0.1）范围内的距离。实验结果如表 8 所示，我们发现数据增强并不能给模型带来显著的提升，有两种甚至导致模型性能下降了。猜想这是由于这三种增强方式都有极大的自由度，因此对模型容量提出了更高的要求。我们换用 7 层卷积的模型验证这个想法，实验结果如表 9 所示。

random dropout	random scale	random shift	max train acc	max val acc
✓			**96.48%**	88.33%
	✓		96.25%	88.37%
		✓	94.14%	**88.90%**

表 8. 不同数据增强方式下得到的最高训练、验证准确率

如表 9 所示，各种数据增强方式都使得模型有了更好的性能（无数据增强的性能与前文中不同是因为这里采用了 3 层网络的最佳设定而不是默认设定）。相比无数据增强的模型，有数据增强时模型训练准确率都有所下降，这是模型过拟合程度下降的表现。然而在使用了所有三种数据增强方式后，模型性能是不如单种数据增强的。但注意到此时训练准确率不到 92%，这一组实验中训练与验证准确率是最接近的。这说明同时使用了所有数据增强方式后，模型有了更好的泛化潜力，如果恰当选择其他训练参数，模型性能有望有进一步的提升。

random dropout	random scale	random shift	max train acc	max val acc
			98.83%	86.71%
✓			97.70%	87.32%
	✓		96.88%	87.97%
		✓	94.61%	**88.17%**
✓	✓	✓	91.88%	87.28%

表 9. 不同数据增强方式下使用 7 层网络得到的最高训练、验证准确率

现在简单总结上文对简单一维卷积神经网络在 3 维点云分类任务上的探索。首先点云数据对硬件的计算需求较小，如果采用较为简单的模型则可以做到单卡每秒训练数千个样本。其次点云数据的组织方式能够对 GPU 达到比较理想的利用率，节省训练时间。第三，使用层数较少的一维卷积网络即可在 ModelNet40 数据集上达到较好的效果，说明点云分类并不是一个复杂的任务。第四，随机丢弃点云中部分点、随机放缩、随机偏移都是较好的数据增强方式，但是需要更深层的网络并且良好地调参才能取得更好的性能。第五，在实验顺序上本文有一点小瑕疵，应该将数据增强放在各项调参的开头，因为训练参数的调整是建立在模型不变的前提下的，这样调出来的参数才能够最适配原模型。第六，本文使用简单一维卷积神经网络，在仅使用 3 层卷积层，且不包含数据增强的情况下最高达到了 89.90%的验证准确率（此时训练 240epochs，初始学习率 0.1，优化器为 SGD，学习率在 135、185epoch 处下降为 0.1，正则化系数 0.0001，批大小 128）。

3.2 使用 PointNet、PointNet++模型

这一小节本文讨论开源的 PointNet 模型以及 PointNet++模型在 ModelNet40 数据集上的性能。首先对模型做一个简单的介绍。

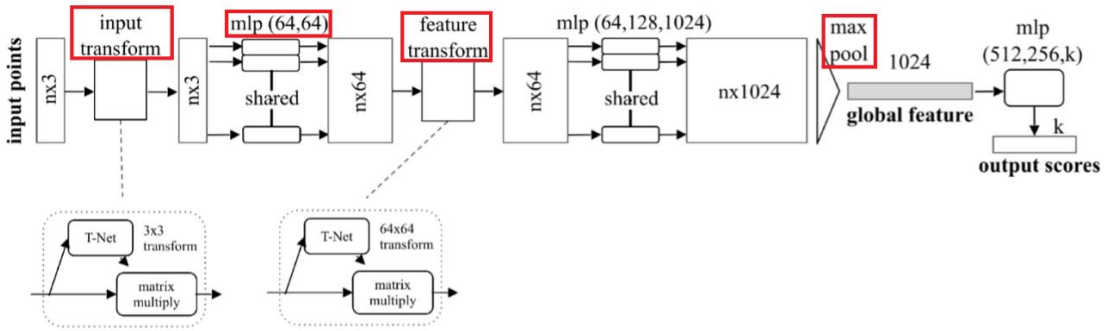

图 4.PointNet 网络结构图

如图 4 所示为 PointNet 的网络结构。PointNet 有两大创新点，首先是 STN 层（spatial transform network），它通过对点云或者点云的特征做仿射变换从而使得其旋转到一个最佳的状态，方便进一步处理。如图 2 所示，如果饮料瓶在点云中是斜放的，经过 STN 层后，其可以旋转到平放的状态。第二个创新点是使用 max pooling 解决点云数据存储的无序性问题。网络对每个点进行了一定程度的特征提取之后，max pooling 可以对点云的整体提取出 global feature，前文的 3 层简单模型也使用了这个思想。

在图 4 中，input transform 与 feature transform 层都是 STN，而前两个 MLP 都是卷积操作（分别拥有 2 层/3 层），最后一个 MLP 是全连接层（3 层），最终将特征变为 k 维用于分类输出。我们发现，PointNet 的结构并不复杂，一个 STN 包含 3 层卷积层与 2 层全连接层，其余主干拥有参数的也仅为 5 层卷积层与 3 层全连接层，整个网络共计 11 层卷积与 7 层全连接层。在论文[1]中，作者使用 PointNet 在 ModelNet40 数据集上达到了 89.2%准确率。

虽然 PointNet 相较于简单的三层网络没有很大的性能提升，但是它的开创性工作为后续的点云深度模型打下了基础，例如简单三层卷积网络中的 max pooling 操作。PointNet 提出于 CVPR 2017，其仍然包含了不少局限性。例如 PointNet 无法很好地捕捉由度量空间引起的局部结构问题，由此限制了网络对精细场景的识别以及对复杂场景的泛化能力。此外 PointNet 是基于均匀采样的点云进行训练的，导致了其在实际场景点云中的准确率下降。

在 PointNet 模型之后，它的作者又提出了 PointNet++模型[2]。作者的基本思想是对输入点云中的每一个点学习其对应的空间编码，之后再利用所有点的特征得到一个全局的点云特征，这样可以使得模型照顾到空间局部信息，而不是仅仅对点云全局描述。这个模型将点集的特征提取分为了几个图层，每个图层又包含了 Sampling layer、Grouping layer、Pointnet layer 三个子阶段。在 Sampling layer 中，模型用迭代最远点采样方法（iterative farthest point sampling, FPS）选择一系列点来定义局部中心；在 Grouping layer 中，由临近点来构建局部区域，进而提取特征，临近点由 neighborhood ball 来进行定义和选取；在 Pointnet layer 中，使用 PointNet 作为特征提取器来进行局部特征的提取。

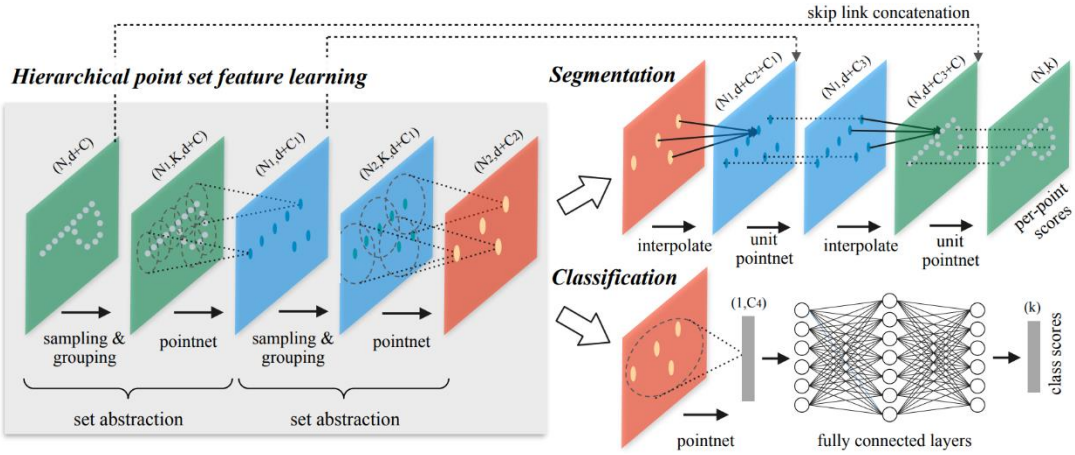

图 5. PointNet++网络结构图

如图 5 所示，图中展示了使用两个图层的网络结构，分类分支处于下方。整个网络相比于 PointNet 更为复杂，也包含了更多参数。此外原论文还测试了不同层级的一些不同聚合方法，以克服采样密度的差异（对于大多数传感器来说这是一个大问题，当物体接近时密集样本，远处时稀疏）。论文中使用 PointNet++在 ModelNet40 上最高达到了 91.9%的准确率。

经过实验，这两个模型的具体训练设置与结果如表 10 所示，它们的准确率曲线如图 6 所示。

configure / result	PointNet	PointNet++
epoch	200	140
data augmentation	all	all
batch size	24	16
optimizer	Adam	Adam
L2 reg	0.0001	0.0001
initial lr	0.001	0.001
lr scheduler	steplr, step=20, γ=0.7	steplr, step=20, γ=0.7
time consumption	7830s	92998s
max train acc	97.87%	97.99%
max valid acc	90.44%	92.42%

表 10. PointNet 与 PointNet++模型的设置与实验结果

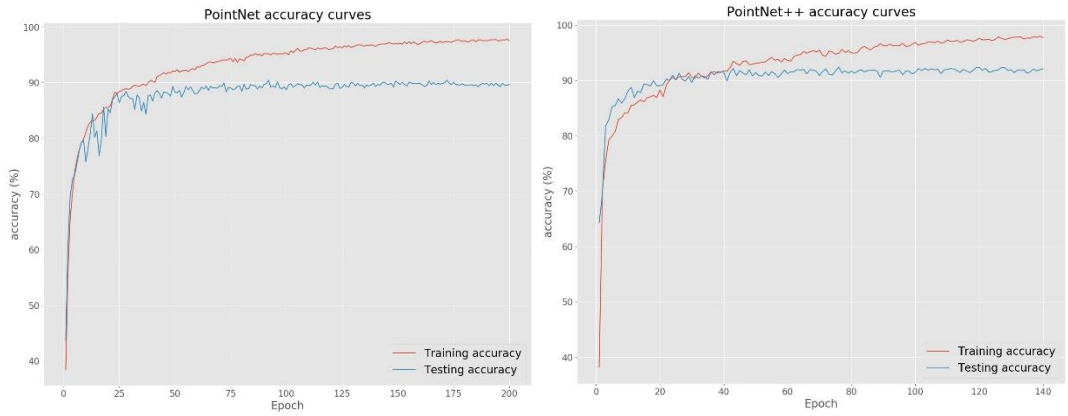

图 6.PointNet 与 PointNet++的准确率曲线

我们发现，PointNet 的最高验证准确率已经超过了经过精细调参的基础版三层卷积神经网络，达到了 90.44%；而 PointNet++则达到了更高的 92.42%。这表明原论文中这两种模型的报告准确率是真实，甚至是保守的。经过针对点云数据的网络结构设计，深度模型能够达到比简单模型更强的性能。然而，利弊总是共同存在的。我们注意到 PointNet 模型的训练时间大大超过简单网络的耗时（240epochs 耗时仅为 600+秒），约为两个小时，而 PointNet++模型的耗时更是达到了 25 个小时。如此惊人的训练时间是这两个模型性能优势背后的代价。

如表 11 所示，不同模型测试时长与训练时长排序关系相同。计算量差异如此之大的模型在 ModelNet40 数据集上却并没有太大的性能差异，这是令人吃惊的。因此，如果当前面临的应用场景较为简单，可以直接使用简单的三层模型，如果场景复杂或者是对精度有严苛的要求，才考虑深层模型。

model	3 layer network	PointNet	PointNet++
testing time	**1.040s**	4.712s	327.140s
inference time on 1 sample	4.2×10^{-4}	1.9×10^{-3}	0.13s

表 11.三种模型的测试/推理时间对比

四、关于 3 维点云的讨论

4.1 obj 文件是以什么格式存储顶点数据的？如果除了顶点坐标（x，y，z），还希望保存每一个顶点的颜色信息，应该怎么做？

如前文提到的，obj 文件以一行一行”v x y x”的格式存储顶点，这里“v”表示当前行是顶点数据，xyz 分别表示三个维度上的坐标。如果想同时保存顶点的颜色，则在这一行后添加 “r g b”即颜色的三通道值（0.0-1.0 之间）。要注意部分软件是不支持这种扩展的 obj 文件的。如图 6 所示，使用 MeshLab 导入扩展的 obj 文件，我们发现添加的颜色信息能够被很好地展示出来。在这个桌子的点云数据中，我们分别使用红色（rgb 值为 1.0、0.0、0.0）以及绿色（rgb 值为 0.0、1.0、0.0）来可视化桌板和桌腿。

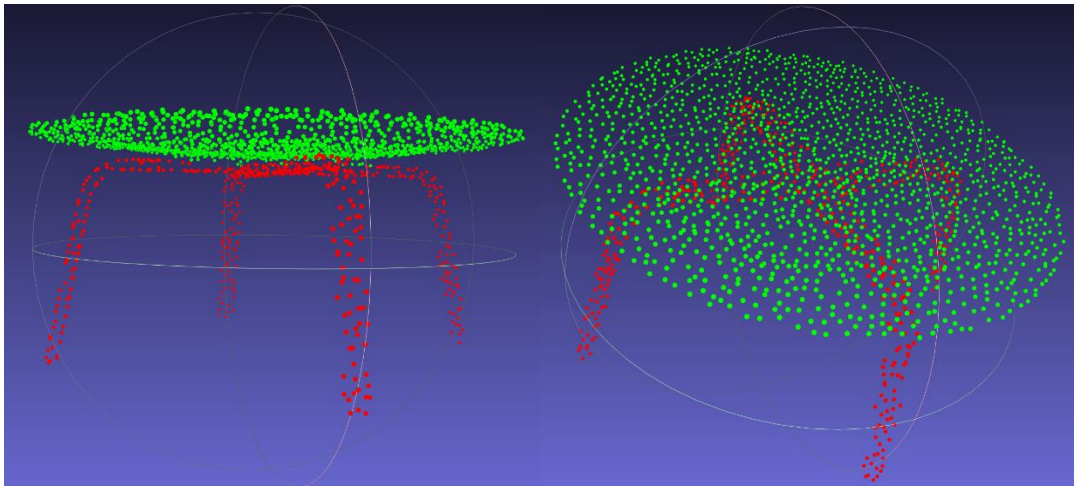

图 6. 使用 MeshLab 可视化带有颜色信息的顶点

4.2 用于点云数据与普通图像数据的神经网络有什么异同？

神经网络最早被用于二维图像上，常见的功能有图像识别、检测、分割等。这些任务都需要神经网络对二维网格数据有强大的编码能力，所以二维卷积作为主要操作用于提取图像中的局部特征成为特征图。经过多层这样的编码，最终我们可以得到一个抽象等级非常高的特征图/特征向量作为图像的表征。有了这样的特征向量，我们用全连接层进行进一步编码为任务所需的向量形式即可。对于分类任务，输出的向量中我们取最大值所在的位置作为模型对图像的类别判断。

使用点云数据与图像数据进行目标识别最大的区别是，点云数据无法形成网格，因此传统的二维卷积在这种数据形式上无法成功提取出局部信息。于是我们选用一维卷积来对点云中的每一个点进行编码。一维卷积一般使用大小为 1 的卷积核，这是因为相邻点之间在空间中不一定相邻，更大的卷积核也无法提取出点与点之间组成的物体形状信息。所以我们不需要相邻层之间不同点信息的交互。在一维卷积提取出了每个点的多维信息之后，一般使用 max pooling 对点云特征进行降维，使得特征与点云中的点数无关，能够接受点数不相同的点云数据。这样的操作还能够保证模型对于点云的点坐标存储顺序无关，得到有代表性且有泛化能力的全局特征。与普通神经网络相同的是，用于点云的网络也使用全连接层将特征向量编码为类别向量进行输出。

综上所述，两类神经网络虽然应用场景不同，但是设计上都必须保证平移不变性与旋转不变性，因此都使用卷积层+全连接层的形式来组成模型。然而点云网络还需保证排列不变性，要求 N 个点，不管以何种顺序（N!种）输入函数中，得到的结果应该是相同的。这就引入了网络中标志性的（卷积核大小为 1 的）一维卷积。这样的一维卷积以及最后的 max pooling 是用于点云的网络与用于图像的网络的最大差异。

4.3 假设你有一张汽车的图像和它对应的世界坐标系下的 3 维模型，如何将三维空间中的坐标转换为与图像像素相对应的坐标？

由于这张汽车的图像必定是由空间中某点的摄像头拍摄的，所以如果我们能够得知摄像头相对物体参考系的位置坐标（以物体中心为原点，球坐标系下距离ρ，以及两个角度α，β）与摄像机焦距大小，则可以在虚拟引擎中模拟一个这样的摄像头，用 3D 图形学的方法将 3 维模型渲染在 2 维平面上。下一步只需将这个 3 维模型的 2 维投影与这张汽车图像进行像素一一对应即可。在配对后，我们再把 2 维投影的像素与 3 维坐标对应起来，就能得到最后汽车图像与 3 维坐标的对应关系了。需要注意的是，最终能够配对的只有 3 维

模型中被摄像头捕捉到的那个表面。现在问题就转化为了如何找到摄像头在汽车参考系下的位置。

为了找到摄像头的 4 个参数，我们可以使用遍历的方法，将每一个参数以一定间隔设设置查找范围，然后生成所有可能的结果图。将所有的结果图与这张汽车图像对比，找到最相似的，对应摄像头参数即为所求。但是这种方法需要渲染巨量的候选 2 维映射图像，计算开销过大，实际应用中不可取。我们可以退而求其次，使用计算机生成训练集并训练一个深度神经网络来估计这 4 个参数。由于 3 维模型以及它的 2 维映射都可以快速生成，网络估计的参数也不复杂，所以初步判断这种解决方法的计算开销会小于暴力搜索。

4.4 如果你想设计一个框架来学习如何利用单个图像的信息给三维模型上色，你会怎么做？

如 4.3 中提到，我们可以使用深度学习的方法将图像像素与 3 维模型中的坐标一一对应。在配对后，为对应的 3 维模型表面的顶点着色即可。然而由于只有被摄像头视角捕捉到的 3 维模型表面能够与彩色图像对应上并着色，我们需要想办法为其余的表面也上色。

为了利用模型的部分色彩信息使三维模型整体都能够上色，我们可以使用自监督学习的方法。将完全着色的三维模型的一部分颜色信息去掉，训练一个深度网络来预测这一部分的颜色。这样，一个给三维模型上色的框架就设计好了。具体而言：第一步使用有颜色信息的 3 维模型来生成一批这个模型的 2 维映射以及相应的摄像头参数，在这样的数据集上训练一个深度模型来根据 2 维图像预测摄像头参数；第二步根据预测得到的摄像头参数渲染得到（可以不含颜色信息）的 3 维模型的相应视角投影，并与给定的图像中像素一一对应；第三步找到 2 维视角投影中的像素与原 3 维模型中的空间坐标的关系，也因此得到了空间坐标与图像像素的对应关系；第四步根据图像像素色彩给 3 维模型上色；第五步用训练好的自监督模型根据 3 维模型部分色彩来预测整个模型的颜色。

五、结论

这份报告主要探究了两点。首先是 obj 文件的内容是如何组织的，以及将 3 维点云数据保存在其中的方法。我们发现 obj 格式是一种对用户友好的数据格式，只需要在行首添加标识字符即可表示三维图形的顶点、面及颜色信息等。此外还探究了在 3 维点云数据上，如何能够训练一个良好的神经网络用于识别分类任务。在 ModelNet40 数据集上，我们仅需一个包含 3 层一维卷积的神经网络就可以达到 89%的验证准确率，更复杂的网络尽管可以少量提升分类性能，但是计算开销却大大增加了。所以综合来看，简单的模型即可又准确又快速地完成 40 类点云的识别任务。最后我们进一步讨论了 obj 文件如何存储颜色信息，以及三维模型的深度网络、模型着色框架如何构建。

六、参考文献

[1] Qi C R, Su H, Mo K, et al. Pointnet: Deep learning on point sets for 3d classification and segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 652-660.

[2] Qi C R, Yi L, Su H, et al. Pointnet++: Deep hierarchical feature learning on point sets in a metric space[C]//Advances in neural information processing systems. 2017: 5099-5108.