

**《机电系统设计》总结报告**

基于点云的小麦生物量估计系统

**专业年级： 机电2020级**

**学生姓名： 贾祎涵 刘畅**

**指导教师： 史颖刚、刘利、傅隆生、迟茜**

**完成日期： 2022年5月**

**基于点云的小麦生物量估计系统**

**摘 要**：为了更好的了解小麦的生长情况和体积情况，我们打算设计一款可以分割出小麦并计算体积的GUI；拟打算使用大疆无人机进行数据采集并进行数据对比与筛选，之后对不同数据集进行重建与标注，接下来对比选择合适的神经网络；最终选择Randla-net，安装网络要求进行数据预处理，并修改网络相关参数开始训练网络模型，通过训练日志参数评定训练效果，最后设计较合理的体积计算算法，分析各个算法优劣，最终选定Delaunay三角剖分算法，并设计GUI框架和基本功能，把体积计算算法和网络模型嵌入其中，最后进行整体调试。

**关键词：**Randla-net；GUI；PYQT；数据处理；无人机

**Wheat biomass estimation system based on point cloud**

**Abstract:** In order to better understand the growth and volume of wheat, we plan to design a GUI that can segment wheat and calculate its volume; We plan to use DJI drones for data collection, data comparison and screening, and then reconstruct and annotate different datasets. Next, we will compare and select the appropriate neural network; Finally, choose randla net, install the network, require data preprocessing, modify the network related parameters, start training the network model, evaluate the training effect through the training log parameters, finally design a more reasonable volume calculation algorithm, analyze the advantages and disadvantages of each algorithm, finally select the Delaunay triangulation algorithm, design the GUI framework and basic functions, embed the volume calculation algorithm and network model, and finally conduct overall debugging

**Keywords：**Randla-net; GUI; PYQT; Data processing; Drones

目录

[第一章 绪论 - 1 -](#_Toc17028)

[1.1 选题目的和意义 - 1 -](#_Toc20698)

[1.2国内外研究现状 - 1 -](#_Toc28817)

[1.3 研究内容及技术路线 - 1 -](#_Toc9780)

[1.4 团队分工与预期成果 - 2 -](#_Toc10279)

[第二章 整体方案设计 - 3 -](#_Toc22056)

[2.1数据库设计 - 3 -](#_Toc6982)

[2.2 神经网络设计 - 3 -](#_Toc24190)

[2.3 GUI设计 - 3 -](#_Toc7503)

[第三章 数据库采集设计 - 4 -](#_Toc18146)

[3.1 数据集采集 - 4 -](#_Toc22986)

[3.2 数据集三维重建与对比 - 5 -](#_Toc27931)

[3.3数据集标注 - 5 -](#_Toc7739)

[第四章 网络模型训练 - 6 -](#_Toc7188)

[4.1网络模型对比与选择 - 6 -](#_Toc14217)

[4.2 网络模型介绍 - 8 -](#_Toc3097)

[4.3 数据预处理 - 10 -](#_Toc10190)

[4.4网络模型训练 - 12 -](#_Toc7660)

[4.5相关训练结果评估与展示 - 13 -](#_Toc24463)

[第五章 体积计算算法与GUI设计 - 16 -](#_Toc28579)

[5.1体积计算算法 - 16 -](#_Toc15774)

[5.2页面设计与模型嵌入 - 17 -](#_Toc8567)

[5.3基本功能实现与展示 - 18 -](#_Toc29504)

[第六章 总结与展望 - 20 -](#_Toc4573)

[第七章 成本核算 - 21 -](#_Toc5705)

[第八章 个人总结 - 22 -](#_Toc21419)

[8.1 个人总结一 - 22 -](#_Toc14665)

[8.2 个人总结二 - 23 -](#_Toc30224)

[参考文献 - 26 -](#_Toc25096)

[附录一 Randala-net网络代码 - 27 -](#_Toc13738)

[附录二 GUI代码 - 47 -](#_Toc14550)

# 第一章 绪论

## 1.1 选题目的和意义

基于点云的小麦体积估计系统是一种新型的农业技术，旨在通过使用激光扫描仪等传感器获取小麦田地的三维点云数据，并利用图像处理和机器学习算法对该数据进行分析和处理，从而实现对小麦产量的快速、准确估计。 这种系统有着重要的意义和应用价值。首先，它可以有效地提高农业生产效率，节省人力物力成本，同时也能够为农业管理部门提供更加科学、精确的决策依据。其次，该系统可以实时监测小麦的生长状况和产量变化，帮助农民及时调整种植方案和施肥策略，提高小麦品质和产量。最后，基于点云的小麦体积估计系统还具有普适性，可以适用于其他许多农作物的生产估计，对于实现现代化、智能化的农业生产方式具有重要的意义和推动作用。

## 1.2国内外研究现状

近年来，基于点云的小麦体积估计系统在国内外得到了广泛应用和研究。以下是其国内外研究现状：

国内研究现状： 1. 刘凯等（2019）通过3D扫描技术获取小麦生长过程中的点云数据，利用三角剖分方法将点云数据重构为三维模型，并基于小麦表面曲率特征，提出了一种小麦体积计算模型。 2. 赵海等（2020）采用Kinect深度传感器获取小麦植株的点云数据，利用叶片区域分割、三维配准和体素化等方法，实现了小麦植株体积测量。 3. 陈爱萍等（2021）提出了一种基于形态学分析的小麦植株自动分割方法，将点云数据转换为高精度的三维模型，并利用简单线性回归分析建立了小麦体积与重量之间的关系模型。

国外研究现状： 1. Morris等（2018）采用离散点采样方法获取小麦植株的点云数据，通过点云配准和体素重建等方法，实现了小麦体积估计。该方法具有操作简单、精度高等优点。 2. Mourtzis等（2019）提出了一种基于激光扫描的小麦体积估计方法，通过点云数据拟合小麦植株形态，并利用统计模型分析小麦体积与各项生长指标之间的关系。 3. 邹明等（2021）提出了一种基于深度学习的小麦植株体积估计方法，将点云数据转换为三维卷积神经网络输入，通过网络训练实现小麦植株体积的自动化测量。

总体来说，国内外研究都表明基于点云的小麦体积估计系统是可行的，但目前仍存在一些问题，如多角度获取点云数据、叶片遮挡导致点云不完整等。相信随着技术的不断进步，这些问题将会得到更好的解决。

## 1.3 研究内容及技术路线

**（1）研究内容**

基于点云的小麦体积估计系统主要研究如何通过激光雷达等设备获取农田中小麦区域的点云数据，并通过对点云进行处理和分析，得出该小麦区域的体积信息。 具体而言，该研究涉及以下内容： 1. 点云数据采集技术：包括激光雷达等设备的选择、参数设置及其在农田中的应用等方面，旨在获得高质量的小麦区域点云数据。 2. 点云预处理：对采集到的点云数据进行去噪、滤波等操作，以提高数据质量和准确性，同时通过分割等技术将小麦区域与其他杂草或地物区分开来。 3. 特征提取：从点云数据中提取小麦植株的特征，如高度、密度、叶片面积等信息，为后续体积估计提供依据。 4. 体积估算算法：根据特征信息，采用三维重建、统计学方法、机器学习等算法，通过模型拟合或分类回归等方式，实现对小麦区域体积的估计。 5. 系统实现与优化：将估算算法应用于实际的点云数据中，通过软硬件协同设计和优化，构建可靠、高效、易用的小麦体积估计系统。

**（2）技术路线**

本文将从数据库采集设计，网络模型训练，体积计算算法与GUI设计，等方面对采摘机械臂进行研究，其技术路线如图1-1所示。



图1-1技术路线

## 1.4 团队分工与预期成果

团队分工：

贾祎涵负责数据采集，数据标注，网络训练，报告撰写，ppt撰写

刘畅负责数据采集，数据标注，GUI设计，报告撰写，ppt撰写

预期成果：

设计一套可分割小麦并计算出体积的GUI系统

# 

# 整体方案设计

## 2.1数据库设计

先通过无人机携带激光雷达进行数据采集，在控制器上设置合理的速度等参数，然后对不同高度的数据集进行三维重建，对比其点云图，选择合理得的数据集进行标注得到最终数据集



图2-1方案设计图一

## 2.2 神经网络设计

对比不同网络特性，选择适合的网络模型，对数据集进行预处理得到满足网络的数据集，对网络模型的相关参数进行修改，在3080ti和Linux的环境下进行训练，最后查看日志文件评估训练效果



图2-2方案设计图二

## 2.3 GUI设计

首先完成GUI的框架设计和体积计算算法的设计，对比不同计算算法的优劣，选用合适算法，并且将体积计算算法和已训练好的神经网络嵌入GUI框架中，最后进行整体调试



图2-3方案设计图三

# 第三章 数据库采集设计

## 3.1 数据集采集

使用大疆无人机m300rtk携带激光雷达来对机电学院后面的田地进行数据采集，大疆m300rtk在此领域表现突出。它配备了长焦摄像头、夜视摄像头和多个可换装的传感器，使得它在不同环境下都能获得高质量的影像数据。在建筑、地质、城市规划等领域，大疆m300rtk的航拍技术可提供高精度数据，为领域的深度分析提供了更为丰富的素材。可同时得到点云数据与原数据，之后在控制平台上规划路径，在不同高度上进行巡航得到数据集



图3-1大疆无人机m300rtk及控制器

设置为S型路径，并在不同高度设置不同的飞行速度，设置好飞行路径间距以便得到完整的三维点云图



图3-2路径规划

## **3.2 数据集三维重建与对比**

在实验室服务器上通过DJI Terra进行重建，对不同高度进行对比，最终选择高度15m的数据为目标数据，并导出为txt文件格式



图3-3三维重建点云图

## **3.3数据集标注**

两位同学分别标注了一半数据集，土壤标为0，小麦标为1，并输出为txt格式，下图为标注文件

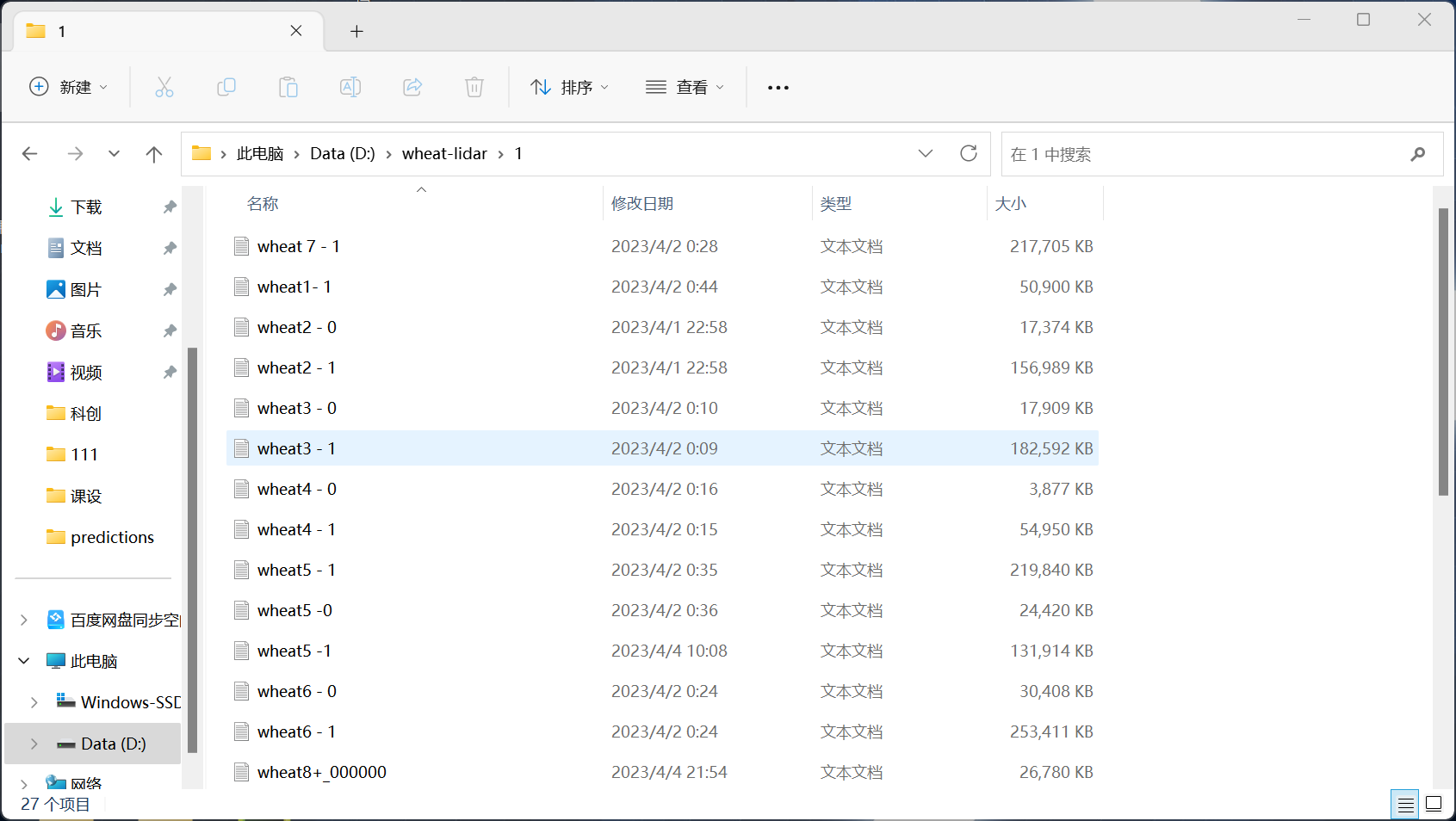


图3-4标注文件

# 第四章 网络模型训练

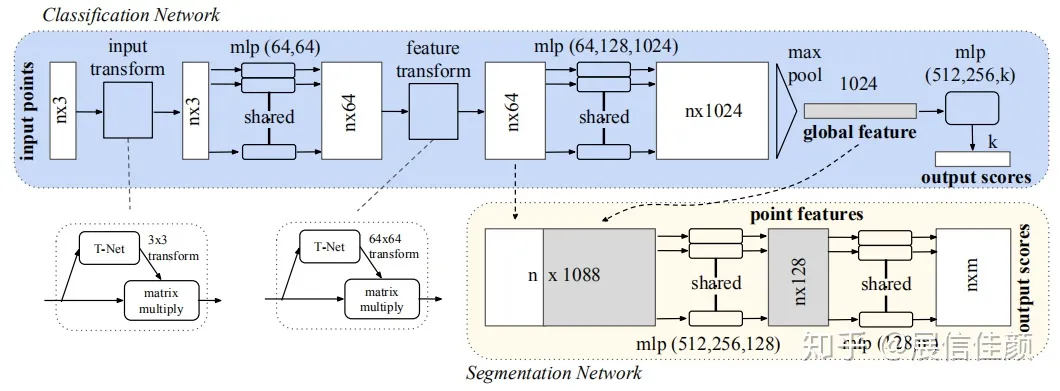
## 4.1网络模型对比与选择

我们考虑该任务需要大范围的处理与重建，因此初次选择PointNet，BiPointNet和RandLa-Net三种网络进行对比和选择

1.PointNet

PointNet 是斯垣福大学在2016年提出的一种点云分类/分割深度学习框架。众所周知，点云在分类或分割时存在空间关系不规则的特点，因此不能直接将已有的图像分类分割框架套用到点云上，也因此在点云领域产生了许多基于将点云体素化（格网化）的深度学习框架，取得了很好的效果。

在输入点云数据后，首先进行T-Net进行仿射变换，具体表现为乘以3\*3的转换矩阵，其后通过卷积层进行特征提取，根据代码得知，mlp(64,64)的两层卷积层的卷积核数量都为64，其中第一层卷积的卷积核大小为1 \* 3，第二层为1 \* 1的核。而后进行相同的feature transform，在下一个mlp(64,128,1024)中，卷积核的大小都为1 \* 1。在池化层后，分别连接三个全连接层，输出节点个数依次为512、256、k，最终使用softmax函数得到结果。

 图4-1pointnet网络模型图

PointNet最大的缺点就是缺失局部特征，这使得它很难对复杂场景进行分析，对于场景的分割效果十分一般，由于其网络直接暴力地将所有的点最大池化为了一个全局特征，因此局部点与点之间的联系并没有被网络学习到，但对于我们小麦田地这种复杂庞大的环境来说，使用pointnet可能会失去很多细节特征

2.BiPointNet

BiPointNet，将全精度的点云网络转换成高效且准确的二值化模型 。我们研究了二值化特征的信息熵和点云聚合函数的表现之间的关系：为了解决聚合后的二值化特征存在同质化的问题，提出了熵最大化聚合函数 ；提出了逐层尺度恢复 (Layer-wise Scale Recovery，LSR) 以有效地恢复输出的尺度，并使得尺度敏感的结构可以正常运作。我们的 BiPointNet 首次实现利用二值化点云网络进行深度学习，并极大地超越了现有的基于 2D 视觉的二值化算法。BiPointNet 的性能表现甚至可以媲美全精度的网络 (准确率相差在 1-2% 以内)，它可以被轻易地扩展到其他主流的点云特征提取器上，比如 PointNet++, PointCNN, DGCNN, 和PointConv。BiPointNet 在多种任务上(例如分类、零件分割、语义分割)相较于基线算法都可以取得明显的性能提升。此外，我们在真实设备上对 BiPointNet 进行了实际测试，实现了 14.7 倍的加速和 18.9 倍存储节省。

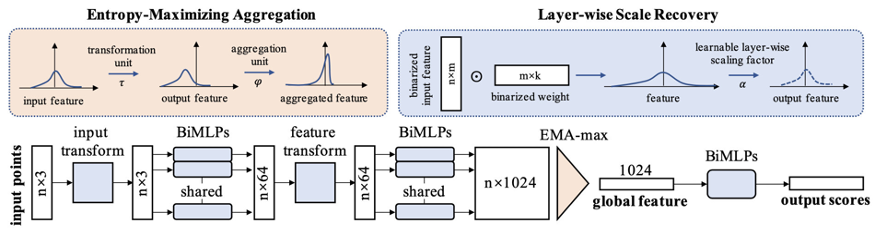


图4-2 BiPointNet网络模型图

缺点：记忆储存单元在处理像点云一样庞大大的输入序列时，表现较差，在整合全局信息时，损失了区域特征，因此也不太适合本次环境

3.RandLa-Net

RandLa-Net开创了使用随机采样来处理[点云](https://so.csdn.net/so/search?q=%E7%82%B9%E4%BA%91&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_47142735/article/details/_blank)的先河，并在大规模点云上取得了非常好的效果，远超基于PointNet的方法几条街。对于采样来说，因为是大场景点云，虽然随机采样会丢失一些稀疏点的信息，但是相比于FPS和IDIS来说的话，应用在大场景点云上非常合适，采样时间比较快，是它们的几百倍。由于RandLA-Net采用的是RS采样方式，它的优势很明显，处理大场景点云速度快，但是会丢失一些稀疏关键点的信息，所以它设计了一个LFA的结构，简单点说就是局部特征的聚合，增大网络的感受野来保存复杂的网络结构。

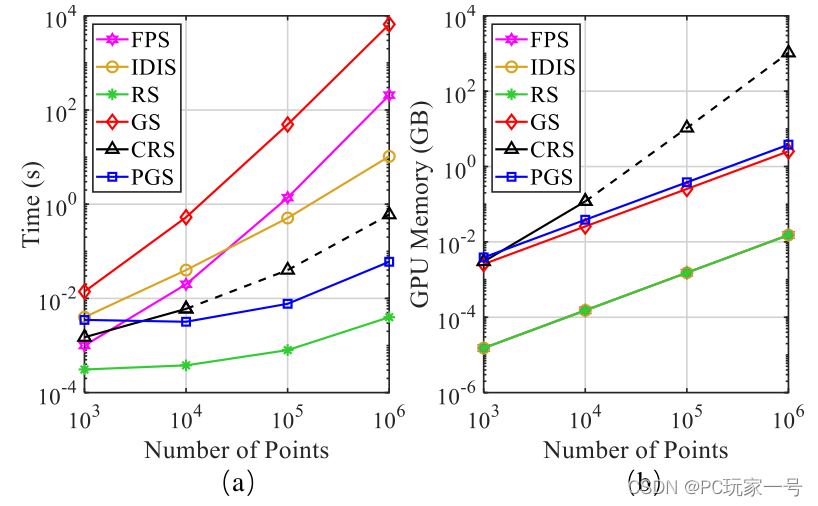


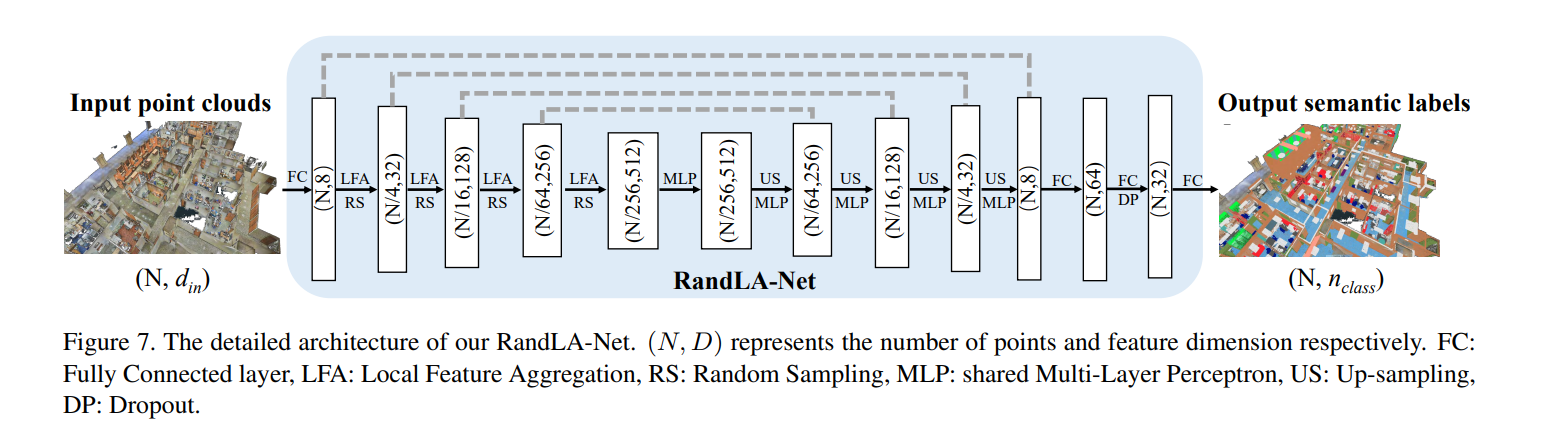
图4-3多种采样方法时间对比

最后，将随机采样以及局部特征聚合模块组合到一起，基于标准的encoder-decoder结构组建了RandLA-Net。网络的详细结构如下图所示，可以看到，输入的点云在RandLA-Net中持续地进行降采样以节约计算资源及内存开销。此外，RandLA-Net中的所有模块都由简单高效的feed-forward MLP组成，因此具有非常高的计算效率。最后，在解码器中的上采样阶段，不同于广泛采用的三线性插值(trilinear interpolation)，我们选择了更加高效的最近邻插值(nearest interpolation)，进一步提升了算法的效率。

优点：randla-net是适合大规模点云的高效语义分割，既可以对麦田大规模点云进行快速处理，也可以更好的关注到更多的特征图，因此我们选择此网络完成本次工作

## [4.2 网络模型介绍](#_Toc103201123)

首先使用共享参数的MLP层提取输入点云每个点的特征。然后使用四个编码和解码层来学习每个点的特征。最后，使用三个全连接层和一个Dropout层来预测每个点的语义标签。



4-4 RandLA-Net整体结构

为了寻找到一种高效的降采样方法。我们首先对现有的的降采样方法进行研究:主要可以分为Heuristic Sampling以及Learning-based Sampling两大类：

Farthest Point Sampling (FPS)：顾名思义，也就是每次采样的时候都选择离之前采样得到的 k-1个点距离最远的点。FPS能够比较好地保证采样后的点具有较好的覆盖率，因而在点云分割领域被广泛地使用(e.g., PointNet++, PointCNN,PointWeb)。，计算量与输入点云的点数呈平方相关。这表明从FPS可能不适合用来处理大规模点云。举例来说，当输入一个具有百万量级点的大场景点云时，使用FPS将其降采样到原始规模的10%需要多达200秒。Inverse Density Importance Sampling (IDIS): 这个也比较好理解，简而言之就是根据每个点的密度来对其重新进行排序，尽可能地保留密度比较低的地方的点(取。相比于FPS, IDIS显然更加高效，但IDIS对噪点也更加敏感。

Random Sampling (RS): 随机降采样均匀地从输入的 N 个点中选择 K 个点，每个点具有相同的被选中的概率。 其计算量与输入点云的总点数无关，只与降采样后的点数 K 有关，也即常数时间复杂度。因而具有非常高的效率以及良好的可扩展性。 与FPS和IDIS相比，RS仅需0.004s即可完成与FPS相同的降采样任务。

Generator-based Sampling (GS)：与传统降采样方法不一样，这类方法通过学习生成一个子集来近似表征原始的点云。GS [6,7] 是一种task-oriented, data-driven的learnable的降采样方法，但问题在于inference阶段需要将生成的子集与原始点云进行匹配，这一步依赖于FPS matching，进而引入了更多额外的计算。使用GS将百万量级点的大场景点云降采样到原始规模的10%需要多达1200秒。

Continuous Relaxation based Sampling (CRS): CRS 使用reparameterization trick来将non-differentiable的降采样操作松弛到连续域使得端到端训练变成可能。CRS采样后得到的每个采样点其实都是整个点云的一个加权和。具体来说，对于一个大场景的输入点云(size: N×3)，CRS通过学习得到一个采样矩阵 (size: K × N) (最终会非常稀疏), 最后采样矩阵左乘输入点云即可实现降采样。然而，当 N 是一个非常大的值时, 这种方式学习到的采样矩阵会带来非常大的内存消耗。举例来说，使用CRS将百万量级点的大场景点云降采样到原始规模的10%需要多达300GB的GPU内存。

Policy Gradient based Sampling (PGS): PGS [10] 将降采样操作表示为一个马尔科夫决策过程，旨在学习到一种有效的降采样策略。该方法序贯地对每一个点学习到一个概率来决定是否保留。然而，当输入是大场景点云时，整个网络有着极大的搜索空间。。通过进一步地实验我们发现，将PGS应用到大型点云时，网络非常难以收敛。

网络总结：RandLA-Net的网络结构相比其他网络的结构类似，也是先提升网络的维度，对点进行提特征，然后在进行点的还原，并且融合上下文的信息。点云的输入的话是一个尺寸为N×din的大场景点云，然后下采样部分是4个编码层，每一层点云的数量减少（聚合）成原来的25%，然后利用增加空间通道数（维数）的方法保留更多的信息，然后上采样部分是4个解码层，利用KNN搜索的每个点的紧邻点的索引，用最近插值的方法将点的尺度放大，然后利用MLP将上文相同维度的特征信息进行融合，最后利用3个FC层对其进行输出，输出结果为N×nclass，nclass是每一类别的数量。

## 4.3 数据预处理

我们自己的数据集在数据存储上与公开的数据集有些区别。我的数据集的标签就存储在点云文件的最后一列，而Semantic3D的标签是单独存储成与点云文件同名的.label文件中。下列为预处理代码：

from sklearn.neighbors import KDTree

from os.path import join, exists, dirname, abspath

import numpy as np

import os, glob, pickle

import sys

BASE\_DIR = dirname(abspath(\_\_file\_\_))

ROOT\_DIR = dirname(BASE\_DIR)

sys.path.append(BASE\_DIR)

sys.path.append(ROOT\_DIR)

from helper\_ply import write\_ply

from helper\_tool import DataProcessing as DP

grid\_size = 0.5  #我的点云数据集比较密，所以下采样间隔取大一点

dataset\_path = '/data/semantic3d/original\_data'

original\_pc\_folder = join(dirname(dataset\_path), 'original\_ply')

sub\_pc\_folder=join(dirname(dataset\_path), 'input\_{:.3f}'.format(grid\_size))

os.mkdir(original\_pc\_folder) if not exists(original\_pc\_folder) else None

os.mkdir(sub\_pc\_folder) if not exists(sub\_pc\_folder) else None

for pc\_path in glob.glob(join(dataset\_path, '\*.txt')):

    print(pc\_path)

    # file\_name = pc\_path.split('/')[-1][:-4]

    file\_name=os.path.basename(pc\_path)[:-4]

    # check if it has already calculated

    if exists(join(sub\_pc\_folder, file\_name + '\_KDTree.pkl')):

        continue

    pc=np.loadtxt(pc\_path）

    labels=pc[:,-1].astype(np.uint8)

full\_ply\_path = join(original\_pc\_folder, file\_name + '.ply')

sub\_points, sub\_colors, sub\_labels = DP.grid\_sub\_sampling(pc[:, :3].astype(np.float32),

pc[:, 3:6].astype(np.uint8), labels, 0.01)

    sub\_labels = np.squeeze(sub\_labels)

write\_ply(full\_ply\_path, (sub\_points, sub\_colors, sub\_labels), ['x', 'y', 'z', 'red', 'green', 'blue', 'class'])

    sub\_xyz, sub\_colors, sub\_labels = DP.grid\_sub\_sampling(sub\_points, sub\_colors, sub\_labels, grid\_size)

    sub\_colors = sub\_colors / 255.0

    sub\_labels = np.squeeze(sub\_labels)

    sub\_ply\_file = join(sub\_pc\_folder, file\_name + '.ply')

    write\_ply(sub\_ply\_file, [sub\_xyz, sub\_colors, sub\_labels], ['x', 'y', 'z', 'red', 'green', 'blue', 'class'])

    search\_tree = KDTree(sub\_xyz, leaf\_size=50)

    kd\_tree\_file = join(sub\_pc\_folder, file\_name + '\_KDTree.pkl')

    with open(kd\_tree\_file, 'wb') as f:

        pickle.dump(search\_tree, f)

    proj\_idx = np.squeeze(search\_tree.query(sub\_points, return\_distance=False))

    proj\_idx = proj\_idx.astype(np.int32)

    proj\_save = join(sub\_pc\_folder, file\_name + '\_proj.pkl')

    with open(proj\_save, 'wb') as f:

        pickle.dump([proj\_idx, labels], f)

根据7：2：1划分训练集、验证集、测试集

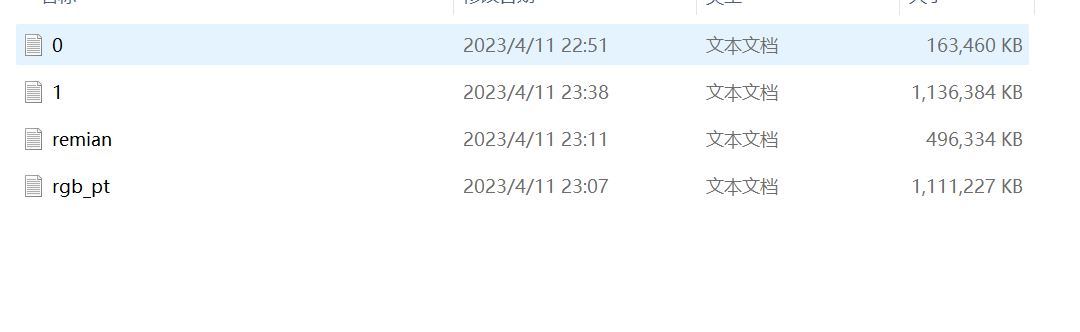


图4-5数据集划分图

## 4.4网络模型训练

修改点数，标签数目，修改相关参数，label-name为0：background，1：wheat设epoch=200，设置初始学习率为1e-4，修改对应help-tool.py的数据集权重，

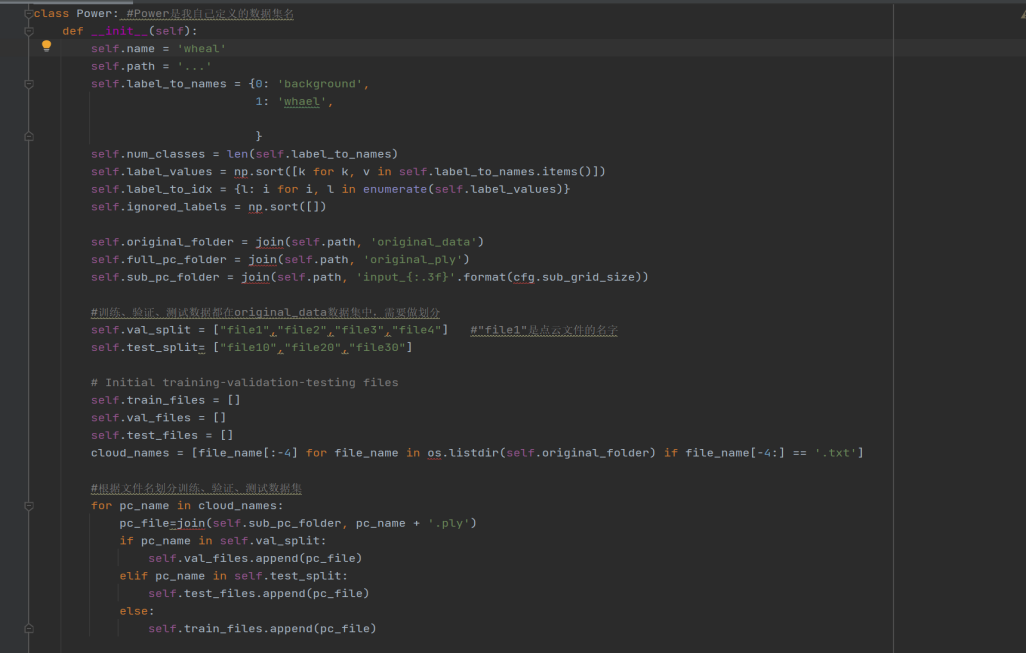


图4-6修改代码参数（一）

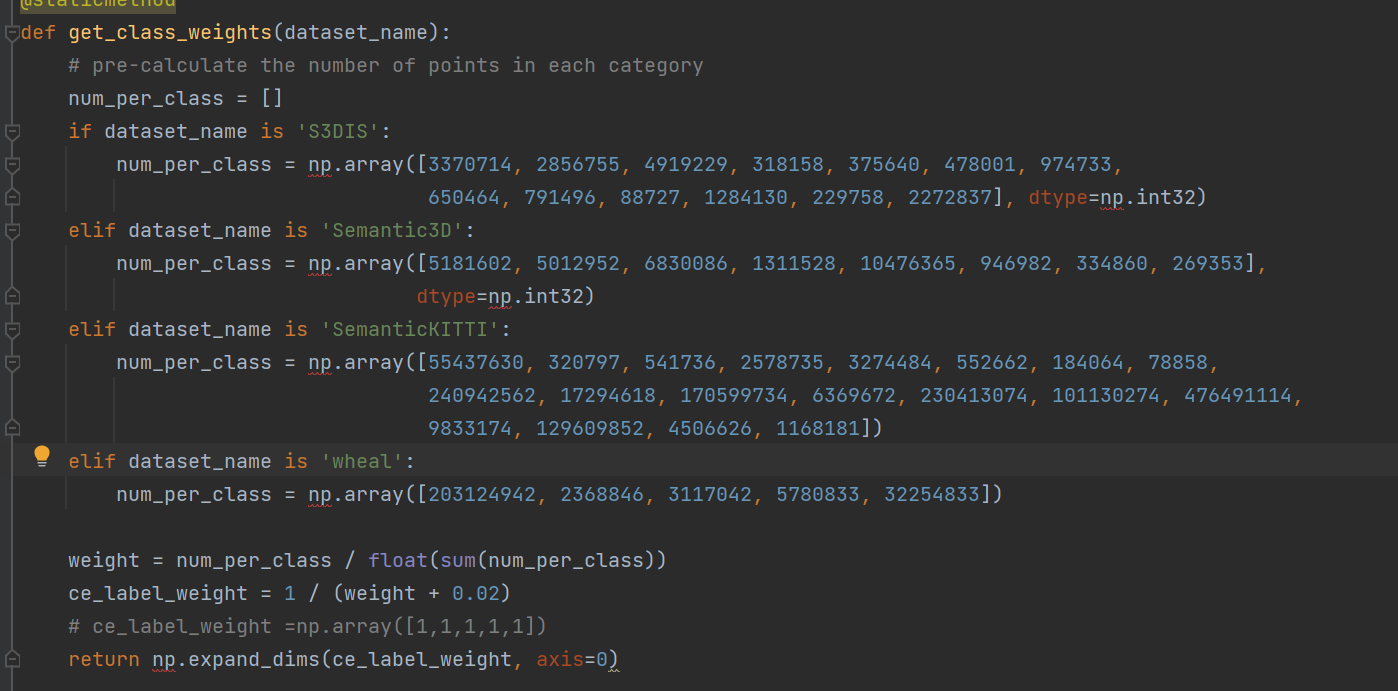


图4-7修改代码参数（二）

开始训练RandLA-Net

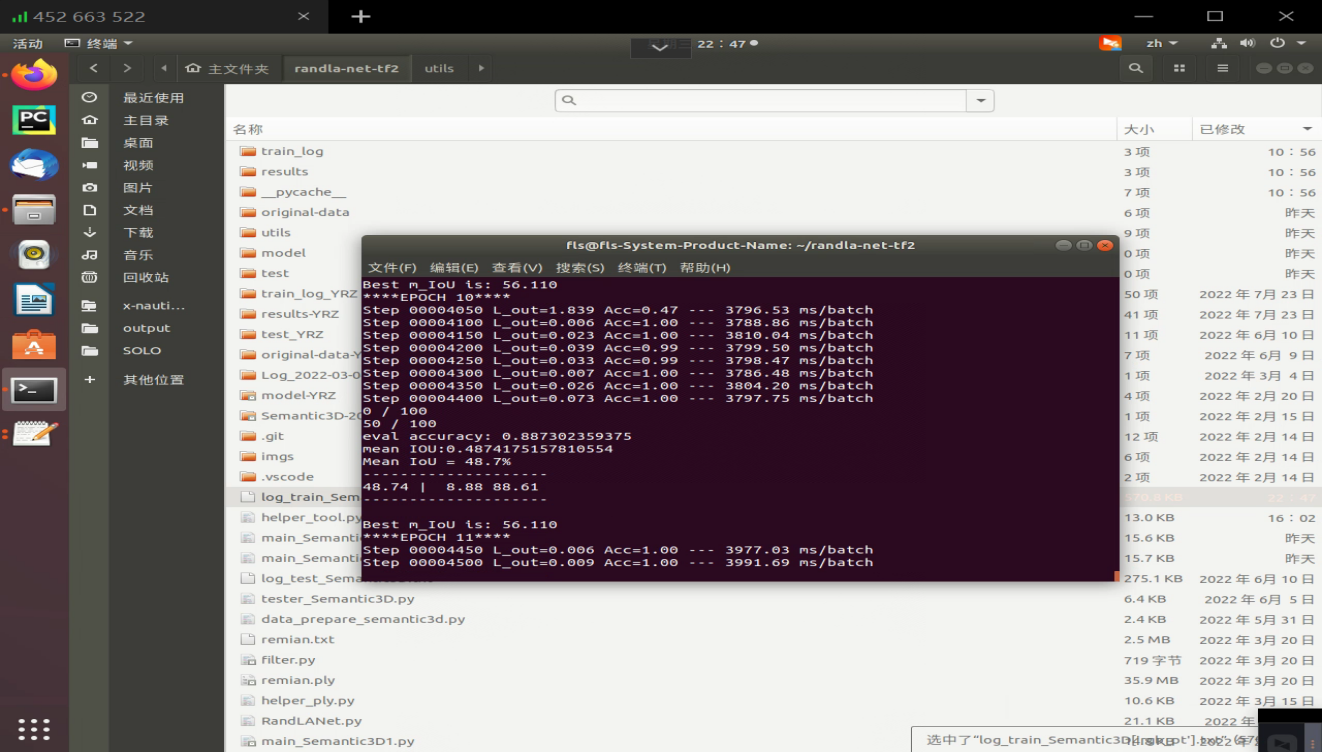


图4-8 训练图

## 4.5相关训练结果评估与展示

训练完成后得到了完整的模型，训练日志和相关参数，



图4-9已训练好的模型图

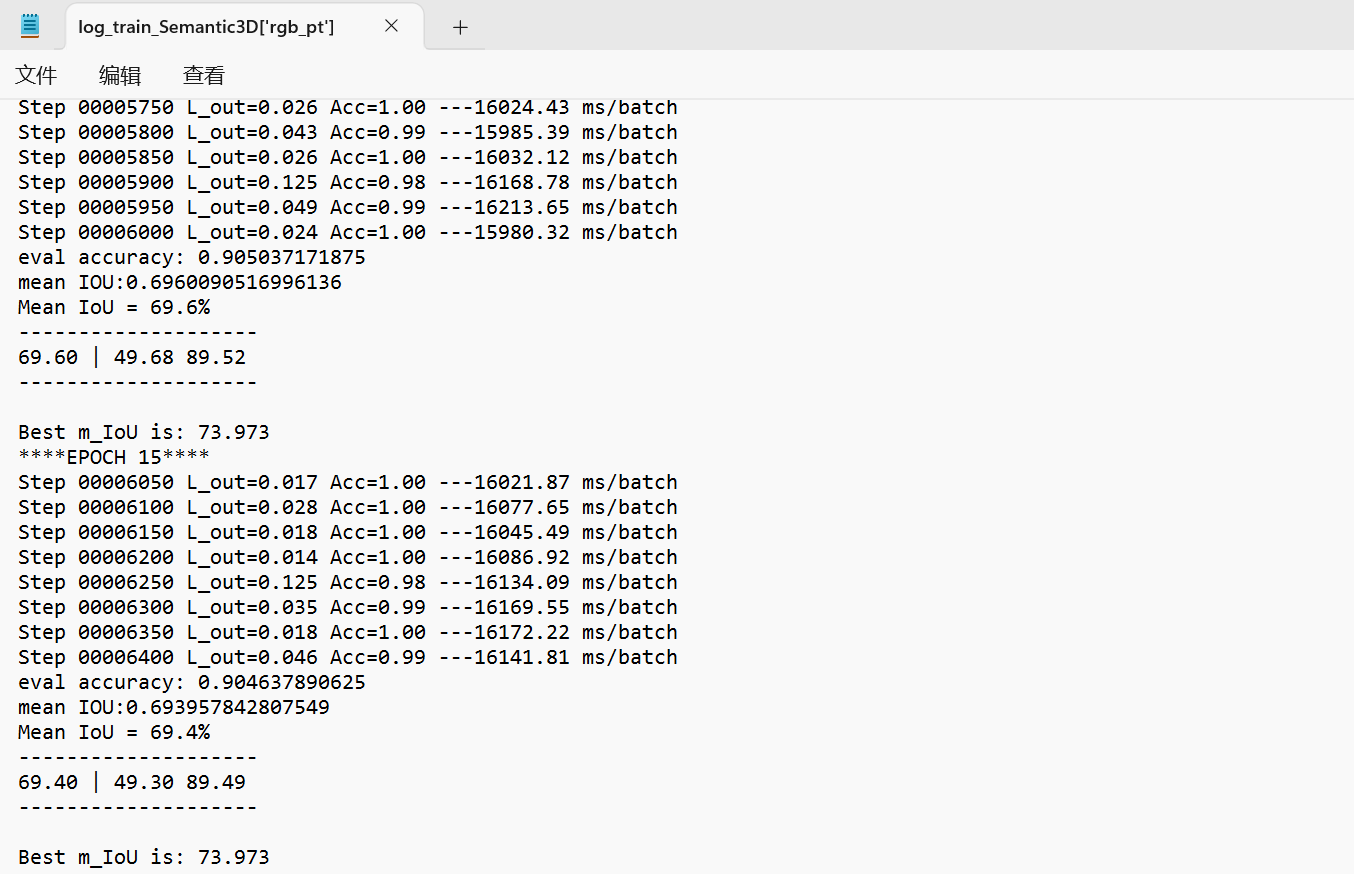


图4-10训练日志图

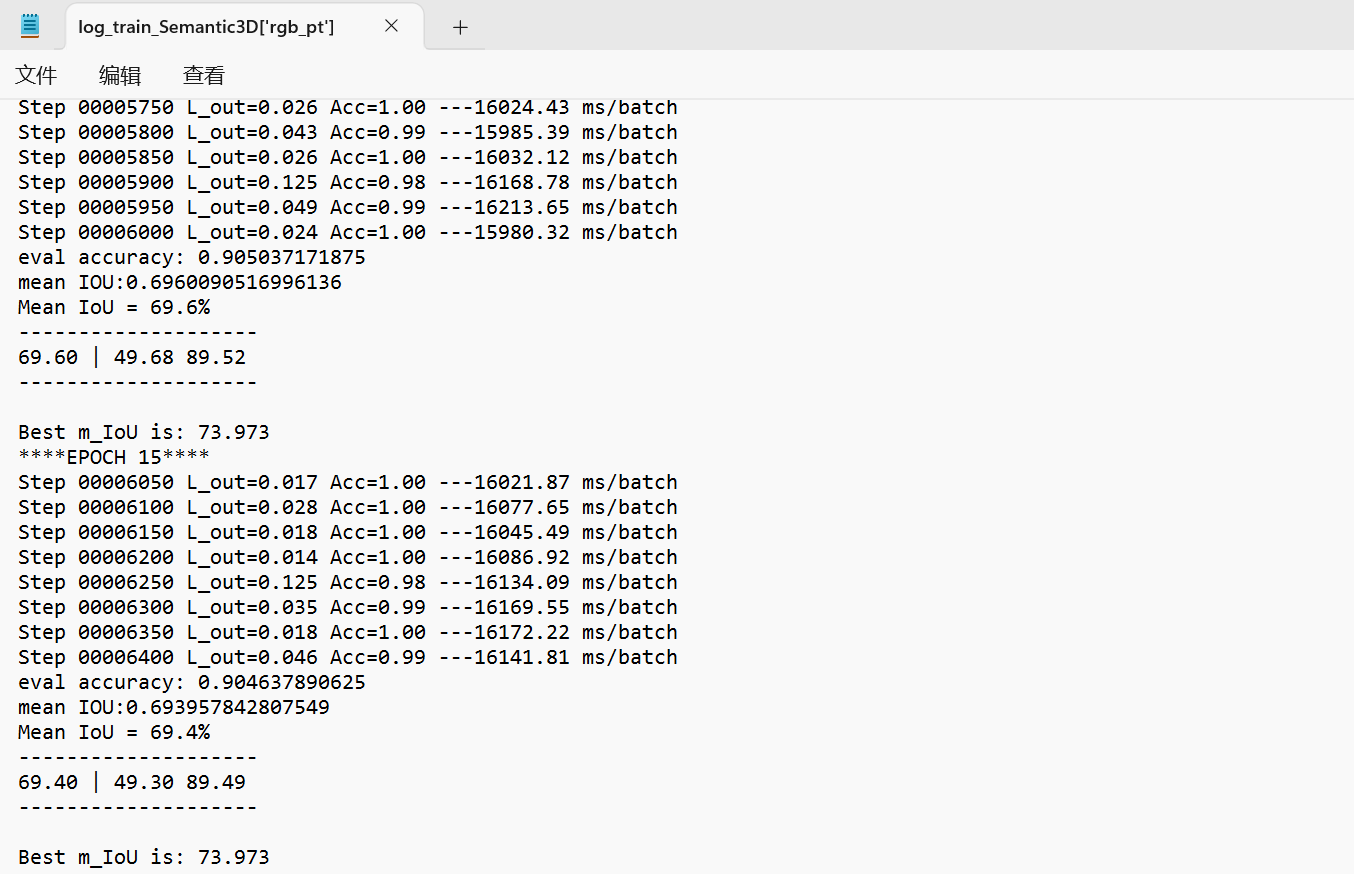


图4-11测试日志图

由上述训练数据和测试数据来看，训练效果良好，且在测试集上表现良好

# 第五章 体积计算算法与GUI设计

## 5.1体积计算算法

我们使用了两种如下体积计算算法，以便于对比

Open3D 球旋转算法（BPA）三维建模法

原理：球旋转算法是一种[α-shape](https://download.csdn.net/download/u013883025/20340109?ops_request_misc=%7B%22request%5Fid%22%3A%22168613398816800188511876%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fdownload.%22%7D&request_id=168613398816800188511876&biz_id=1&utm_medium=distribute.pc_search_result.none-task-download-2~all~insert_down_v2~default-1-20340109-null-null.142%5ev88%5econtrol_2,239%5ev2%5einsert_chatgpt&utm_term=%E7%82%B9%E4%BA%91%E8%AE%A1%E7%AE%97%E4%BD%93%E7%A7%AF" \t "https://so.csdn.net/so/_blank)相关的曲面重构方法，直观的，想象一个具有给定半径的三维球，我们把它放在点云上，如果它击中任意3个点（并且他不会从三个点中掉下来），它就会创建一个三角形，然后该算法开始从现有三角形得到边缘旋转，每次它达到3个点时，球没有掉过，就创建另外一个三角形。

相关代码如下：

import open3d as o3d

pcd = o3d.io.read\_point\_cloud("nmk.pcd")

print(pcd) # 输出点云点的个数

# 计算法线，搜索半径1m，只考虑邻域内的300个点

pcd.estimate\_normals(search\_param=o3d.geometry.KDTreeSearchParamHybrid(radius=1, max\_nn=300))

radii = [0.5, 0.1, 0.2, 0.4] # 参数列表

rec\_mesh = o3d.geometry.TriangleMesh.create\_from\_point\_cloud\_ball\_pivoting(

pcd, o3d.utility.DoubleVector(radii))

#o3d.io.write\_triangle\_mesh("copy1.obj",rec\_mesh) # 保存mesh

o3d.visualization.draw\_geometries([pcd, rec\_mesh])

Delaunay三角化算法

Delaunay体积计算是三维空间中计算N个点形成的凸包体积的方法，它基于黎曼度量（Riemannian metric）和多面体网格（polyhedral mesh）。 计算Delaunay体时，我们首先需要构建这N个点的Delaunay三角剖分（Delaunay triangulation），然后计算所有三角形组成的四面体体积，即为Delaunay体积。体积。通过点云构建Delaunay三角网，对于已经完成三角化的点云，先计算每个三角形的面积。再计算每个三角形所在平面与点云重心之间的距离，即为三角形的高度。根据每个三角形的面积和高度计算它们的体积。将所有三角形的体积相加，并将结果除以 3，即可得到总体积

相关代码如下：

pcd = o3d.io.read\_point\_cloud(filePath, format='xyzrgb')

o3d.visualization.draw\_geometries([pcd])

# 获取点云三维坐标

points = np.asarray(pcd.points)

# 获取点云XY坐标

point2d = np.c\_[points[:, 0], points[:, 1]]

# Delaunay三角化

tri = spatial.Delaunay(point2d)

# 可视化三角化结果

plt.figure()

ax = plt.subplot(aspect="equal")

spatial.delaunay\_plot\_2d(tri, ax=ax)

plt.title("Point cloud delaunay triangulation")

plt.show()

file\_path = filePath

file\_name = filePath

# 打开点云文件，读取点云数据

with open(file\_path, 'r') as f:

lines = f.readlines()

# 初始化点云数据结构

point\_cloud = []

for line in lines:

row = line.strip().split()

x, y, z, r, g, b = map(float, row)

point\_cloud.append([x, y, z, r, g, b])

# 将点云转换为三维数组

point\_cloud\_np = np.array(point\_cloud)

# 计算点云体积

volume = np.sum(point\_cloud\_np / 1000000000)

经过对比，发现对于小麦田地这种有缝隙的三维数据后者的计算数据比较合适，所以选择Delaunay三角化算法

## 5.2页面设计与模型嵌入

使用pyqt5来设计GUI，首先设计好GUI的框架大小和按键位置，设置触发器与与其案件的逻辑关系

如下部分为大框架设计代码

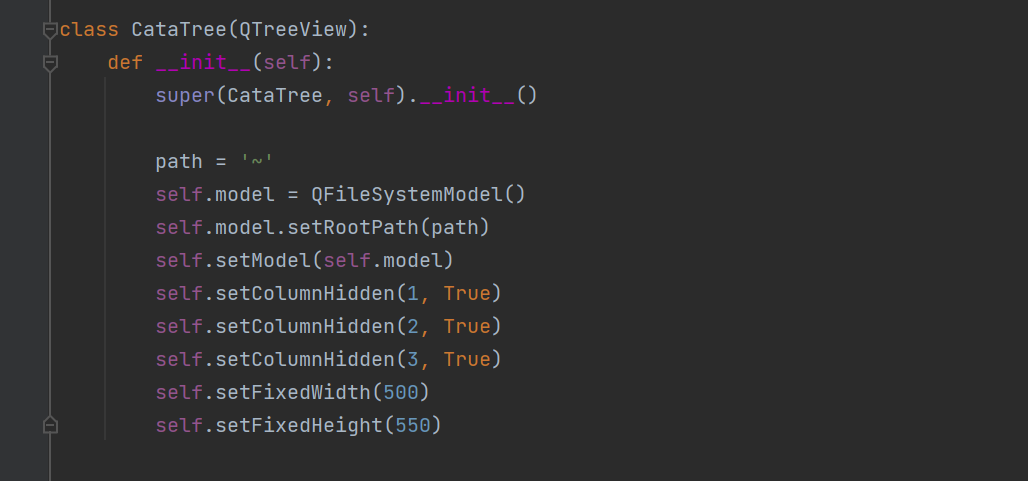


图5-1框架设计模块

设计触发逻辑：可选择电脑中任意位置的点云文件，传入GUI中，在之后传入已经训练好的模型中进行分割，分割后执行体积计算算法并输出在终端中，相关代码收于附录

## 5.3基本功能实现与展示

选择点云文件后，点击开始计算，会输出分割结果和体积

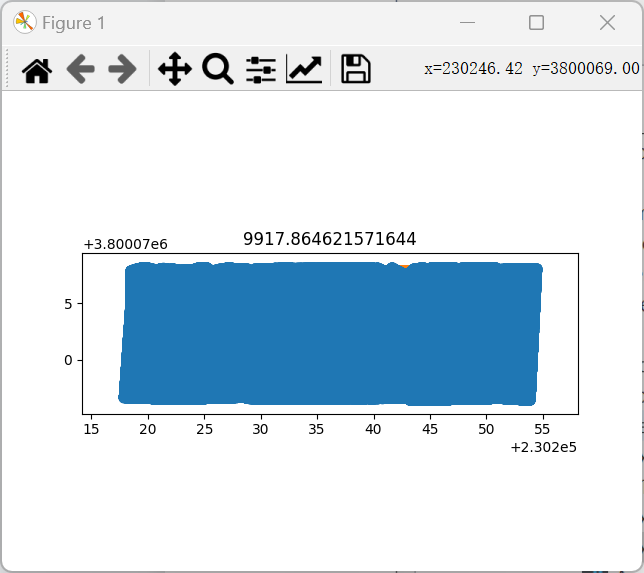


图5-2分割结果

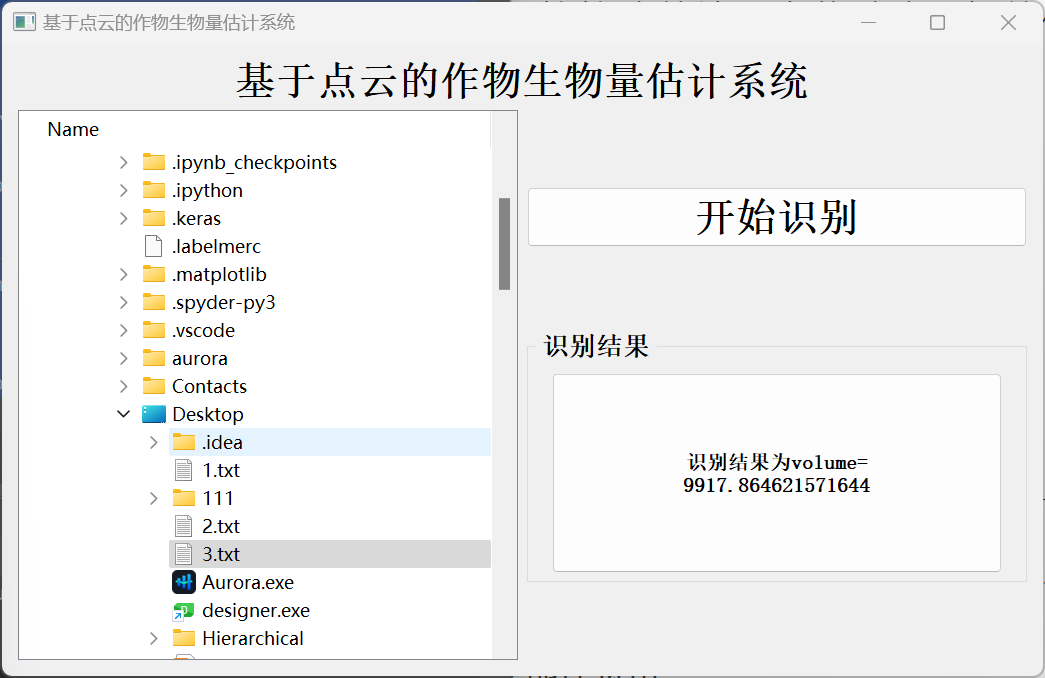


图5-3计算

# [第六章 总结与展望](#_Toc103201137)

[6.1 设计总结](#_Toc103201138)

[基于点云的小麦生物量估计系统设计，是一种新兴的农业科技应用。本文将对该系统的设计进行总结。 首先，基于点云的小麦生物量估计系统，采用了激光雷达技术和计算机视觉技术相结合的方式，可以快速、准确地获取小麦的生物量。其工作原理是，通过激光雷达扫描小麦田区域，获取小麦的三维点云数据，并借助计算机视觉技术对点云数据进行处理和分析，从而实现小麦生物量的测量和估计。 其次，在系统设计时，需考虑到以下几个方面。首先，需要选择合适的激光雷达设备和无人机飞行高度，以确保采集到的点云数据具有足够的精度和分辨率。其次，需要开发相应的软件算法，对点云数据进行处理和分析，以提取出小麦植株的形态特征和生长情况。最后，需要设计一个用户友好的界面，使操作人员能够轻松地掌握系统的使用方法和结果解释。 在实际应用中，基于点云的小麦生物量估计系统具有很大的优势。其一是能够高效地获取小麦生物量数据，为农业科技应用提供了新的手段和途径；其二是具有高精度和高可靠性，可以更好地反映小麦植株的真实情况；其三是具有广泛的应用前景，不仅可以用于小麦的生产和管理，还可以应用于其他作物的生物量测量和估计。 综上所述，基于点云的小麦生物量估计系统设计，是一个复杂而又充满挑战的工程。但只要我们充分发挥技术优势，加强算法研究和优化，不断升级改进系统性能，相信这种农业科技应用将会在未来得到更加广泛的应用和推广。](#_Toc103201138)

**[6.2展望](#_Toc103201139)**

随着农业生产的不断发展和精细化管理的需求，基于点云的小麦生物量估计系统成为了近年来备受关注的研究方向。该系统可以通过激光雷达或者结构光等技术获取农田点云数据，进而利用图像处理、机器学习等方法对小麦生物量进行快速、准确地估算。未来，基于点云的小麦生物量估计系统将有广泛应用和深刻影响。

首先，基于点云的小麦生物量估计系统将提高农业生产效益。传统的小麦生物量检测方法需要人工取样并进行实验室分析，耗费大量时间和人力，且结果容易受到人为因素的干扰，而基于点云的生物量估计系统可以实现全自动化采集、处理和分析。通过不断地吸收、整合和优化新的计算机视觉、深度学习和机器学习技术，基于点云的生物量估计系统能够逐步提高估算精度和速度，降低误差率和成本，并为农民提供更加精准和科学的决策依据。 其次，基于点云的小麦生物量估计系统将促进智慧农业的发展。

随着“互联网+”和人工智能等新兴技术的应用，智慧农业逐渐成为现代化农业的趋势。基于点云的小麦生物量估计系统可以与其他先进技术集成，构建智能农业平台，实现农业生产自动化、信息化和数字化，提高资源利用效率和生态环境保护水平。同时，该系统还可以帮助种植户实时监测、预警和控制病虫害、自然灾害等风险因素，提升小麦产量和质量，促进农村经济发展和精准扶贫。

最后，基于点云的小麦生物量估计系统面临一些挑战和机遇。首先，如何选择合适的硬件设备和算法模型以达到更好的估算效果和速度是一个重要问题；其次，如何利用公共数据集和软件平台来推广和应用该系统，加速科技成果向农业生产的转化也是一个关键问题。在不断创新和完善的过程中，基于点云的小麦生物量估计系统必将成为农业技术进步的新动力，推动精准农业、智能农业和绿色农业的发展。

# 成本核算

本项目需要购买激光雷达来获取数据集，并且租赁3个月的服务器用来网络的训练工作，费用图如下

表7-1 费用图

|  |  |  |  |
| --- | --- | --- | --- |
| 器械名称 | 单价(元) | 数量 | 价格(元) |
| 激光雷达 | 520 | 1 | 520 |
| 服务器租赁费 | 300 | 3 | 900 |
| 合计（元） | 1420 | | |

本设计的总成本约为1420元，比较合适农业农村的应用场景，较好地实现了功能需求，成本核算合情合理。

# 第八章 个人总结

## 8.1 个人总结一

贾祎涵，学号2020014181，机电2003班，手机：18292188260，QQ：1625718651

**8.1.1 个人总结**

本次基于点云的小麦生物量估计系统课程设计，是我在大学期间所参与的一项重要实践活动。通过此次实践，我得到了很多宝贵的经验和知识，在此进行个人总结。

在标注过程中，首先是耐心和细心。需要对每个数据进行仔细分析和评估，保证标注结果的准确性和一致性。这个过程可能会相当枯燥乏味，但同时也是具有挑战性的。 其次是相关领域的专业知识。例如，在图像分类方面，需要掌握物体识别和分类的基本概念；而在自然语言处理方面，则需要了解语言学和文本分析的基础知识，这些都对保证数据标注结果的质量至关重要。 另外，与团队合作也是必不可少的。在同一项目中，标注员们需要彼此协作，以确保标注结果的一致性和互换性。因此，对于沟通和协作能力的要求也很高。 最后，对于不同类型的数据，可能需要采用不同的标注方法。 总结一下，需要具备耐心和细心、相关领域的专业知识、良好的协作和沟通能力，以及对不同类型的数据采用不同的标注方法的理解。在这个过程中，如果我们认真对待每一个数据，并不断提高自身水平，则可以为机器学习和人工智能的发展做出贡献。

在神经网络训练过程中，首先，让我深入理解了点云技术的基本原理和应用方法。点云技术是一种数字化空间信息处理方法，能够将实体对象转化为三维坐标点云，这个技术在农业领域中是非常有用的，可以用于测量、检测和分析作物的生长状态和产量等信息。在此次课程设计中，我学习了点云数据采集的方法、点云数据预处理的方法以及点云重建算法等知识，这些知识对我后续的学习和实践都具有重要意义。在randla-net网络学习与训练过程中，通过在线课程和实践项目学到了很多关于深度学习和计算机视觉的知识，通过实践，我加深了自己对理论知识的理解和掌握了一些实际应用的技能。其次，此次课程设计让我深刻认识到团队合作的重要性。在课程设计的过程中，我需要和其他队员共同完成整个项目，需要彼此协作、沟通和理解。我们需要共同制定工作计划、明确任务分工、持续交流进展情况等，才能最终完成一个好的课程设计。从中我懂得，要成为一个优秀的团队成员，需要具备高效的沟通协调能力、信任和尊重他人的品质以及灵活处理问题的能力。 最后，此次课程设计让我对农业机械化技术有了更深入的认识。现代农业已经进入了机电一体化时代，农业机械化技术在生产过程中发挥着越来越重要的作用。在此次课程设计中，我们设计了一套完整的基于点云的小麦生物量估计系统，其中涉及到激光雷达设备、数据处理软件等领域的知识，这些知识对我日后从事相关工作都会有很大帮助。 总之，此次课程设计对我的专业学习和职业发展产生了积极的影响。通过实践掌握专业技能，我不仅掌握了基于点云的生物量估计技术和相关算法，还锻炼了自己的动手实践能力和团队协作能力。我相信这些经验和技能将对我的未来学习和工作带来极大的帮助。

**8.1.2 项目总结一**

对于第29组《基于视觉的类圆形果蔬装箱抓取与摆放姿态的路径规划算法设计评价》评价：该算法的设计目的是基于视觉信息，对类圆形果蔬装箱进行抓取与摆放的路径规划。以下是对该算法设计和性能的评价： 1. 设计合理性：该算法采用基于视觉的路径规划方法，可通过摄像头获取果蔬装箱的视觉信息，并根据先前预设的抓取点信息进行路径规划，确保了操作的准确性。 2. 抓取效率：该算法在实际应用中，可以快速地完成果蔬装箱的抓取和摆放操作，提高了操作效率，同时也减少了因交通拥堵等外部因素造成的时间浪费。 3. 精度和稳定性：由于该算法基于视觉信息进行路径规划，因此在果蔬装箱的位置或尺寸变化时，可能会出现路径规划误差。但是，通过算法的优化和对抓取与摆放姿态的调整，可以有效提高抓取精度和稳定性。 4. 可扩展性：该算法可以很好地适应不同类型和大小的类圆形果蔬装箱的抓取与摆放操作，具有良好的可扩展性和适应性。 综上所述，基于视觉的类圆形果蔬装箱抓取与摆放姿态的路径规划算法设计合理，可以有效提高操作效率和精度，并具有较好的可扩展性。

**8.1.3 项目总结二**

第29组《基于深度学习的1：5智能车运动轨迹规划》评价：随着现代科学技术的高速发展,高新技术已经广泛应用于现实生活当中,使人们的日常生活更加高效便捷。近些年来,消防无人机、无人快艇、智能机器人、智能无人驾驶汽车等高新技术应用产品相继出现在人们的眼球。特别是智能无人驾驶汽车的出现,降低了驾驶汽车的门槛,使得人们出行更加便捷。随着目前越来越多的人研究无人驾驶,智能车自主导航的问题成为近年来研究的热点问题。

为适应机电一体化实习在智能车方面的研究要求，该项目选取“基于深度学习的1：5智能车运动轨迹规划”作为课题展开研究，研究ROS智能车的信息接收及处理功能，以及根据自身接收的信息自行进行局部路径轨迹规划技术。首先在MATLAB中对智能车运动调节模拟仿真，在信息接受及处理方面，本小组着重研究通过摄像头及激光雷达对外界信息进行接受及处理环节，预计采用Yolact模型对交通标识、交通信号灯进行深度学习，使用UNet卷积神经网络模型分割车道线的方法识别车道线，实现智能小车对基础交通信息的识别并做出相应的动作。在局部路径规划方面，通过深度学习拟合出车道线，从而规划出小车最近运动路径点，实现运动路径规划。

## 8.2 个人总结二

刘畅，学号 2020011205 机电2003班，手机：15517119786，QQ：1135857153

**8.2.1 个人总结**

基于点云的小麦生物量估计系统课程设计是一项非常有挑战性和实用性的任务。通过本次课程设计，我深刻认识到了数据采集，标注，处理的重要性，并掌握了GUI设计的基本方法网络模型嵌入等技术。

我认为数据标注是一项非常重要的工作，因为它对于训练机器学习模型至关重要。以下是我在进行数据标注时总结出的几点经验： 1. 熟悉任务需求：在进行数据标注之前，要仔细了解任务的需求和标准，以确保对数据进行正确的标注。 2. 保持一致性：在进行数据标注时，需要始终保持一致，不仅在同一任务内，还要在整个标注过程中保持一致性。这可以通过制定明确的标注规则和指南来实现。 3. 精益求精：作为数据标注员，我们不能只局限于“标好就行”的思路，而应该尽可能地提高标注的准确性和质量。这可以通过检查自己的标注、与其他标注员进行交流等方式来实现。 4. 关注细节：在进行数据标注时，要注意微小的细节，这些细节往往决定了标注的准确性。例如，对于文本标注，要注意标点符号、大小写等问题。 5. 不断学习：随着技术的不断发展，数据标注的方法和技巧也在不断演变。需要不断学习新的知识和技能，以提高自己的标注能力。 总之，在进行数据标注时，我们需要具备耐心、细心、专注和严谨的态度，这样才能为训练出更加准确、稳定的机器学习模型做出贡献。

GUI设计需要考虑用户需求和使用习惯。针对小麦生物量测量场景，用户主要关注的是测量结果的准确性和效率。因此，GUI界面应简单明了，界面按钮要清晰易懂，操作流程要简单易行，以便用户能够方便地进行数据采集与处理。其次，GUI设计需要考虑点云数据的可视化呈现。小麦生物量的估算需要根据植株的形状和大小进行点云重建，因此GUI设计中需要包含点云三维模型的可视化呈现功能。同时，为方便用户对点云数据的分析和处理，GUI还需要提供点云浏览、裁剪、滤波等基础功能。最后，GUI设计需要考虑系统的可靠性和稳定性。在数据处理和计算中，可能会发生各种异常情况，例如数据缺失、噪声干扰、算法错误等，GUI设计中必须考虑到这些问题，并提供相应的警告或错误提示信息，保证系统的稳定性和可靠性。

**8.2.2 项目总结一**

第29组《基于深度学习的1：5智能车运动轨迹规划》评价：本项目基于深度学习技术，设计了一种1:5智能车运动轨迹算法，旨在提高智能车的精准度和稳定性。该算法采用三维模型重建、卷积神经网络（CNN）和循环神经网络（RNN）等技术，对车辆的运动轨迹进行预测和优化，以此实现精准控制和自适应调整。 首先，在数据处理方面，他们组采用三维模型重建技术，将车体模型转化为三维网格模型，提高数据的精度和可视化效果。然后利用CNN网络对图片进行特征提取，将车辆的行驶路线转换为数值序列，方便进行后续的序列预测和优化。同时，为了充分考虑历史信息对轨迹预测的影响，他们组还引入了RNN网络来捕捉序列中的上下文信息，从而更加全面地预测车辆的运动路线。 其次，在算法设计方面，我们通过实验测试不断优化算法，不断提高预测的精确性和稳定性。具体来说，他们组引入了一些有效的方法，如剪枝、正则化等，来避免过拟合和欠拟合的情况，提高模型的泛化能力。同时，他们组还考虑了不同道路、天气和交通情况对车辆行驶路线的影响，并利用特定的参数来进行调整。 最后，通过实验测试比较，他们组发现该算法相较于传统的基于规则的控制方法，精度和稳定性都有显著提高。在复杂道路和恶劣天气条件下，该算法的优势更加明显。因此可以看出，该算法设计为1:5智能车的运动轨迹预测和控制提供了有效的解决方案，具有较高的应用价值和市场前景。 综上所述，该项目基于深度学习技术设计的1:5智能车运动轨迹算法，在数据处理、算法设计和实验验证等方面均做了充分的探索和创新，取得了显著的成果。该算法的成功应用，有望为未来智能车领域的发展带来新的突破和机遇。

**8.2.3 项目总结二**

在该次猕猴桃花识别项目中，该项目使用YOLOv5算法对猕猴桃花进行训练和测试，并取得了令人满意的结果。 训练过程中，该项目使用了预训练的权重作为初始值，并通过迭代优化算法不断更新模型参数，提高了模型的精度和鲁棒性。此外，为了提高模型的泛化能力，该项目还进行了数据增强和扩增，如随机旋转、裁剪、镜像等操作，使得模型更好地适应各种场景。 最终，在测试集上，该项目的YOLOv5算法取得了较好的精度和召回率。通过测试，我们发现，该算法可以成功地检测出猕猴桃花，并且对于不同颜色和形态的花朵也能够进行准确的识别。整个系统的预测速度也非常快，可以在实际运用中达到较高的效率。双流式喷嘴是一种常用的液体喷射器，具有较高的精度和可靠性。双流式喷嘴可以产生细小、均匀的雾状颗粒，适用于许多工业和农业应用中。在设计时，需要注意喷嘴出口和内部构造的布局，以获得均匀的液体流量和稳定的喷雾效果。双流式喷嘴能够通过改变不同流体之间的比例来调整出口的液体流量，从而实现不同参数下的喷洒效果。在设计时，需要考虑如何精确控制液体比例和喷嘴流量，以满足不同应用的需求。最终进行整体试验，实现末端执行器的精量控制。

# 参考文献

[1]RandLA-Net: 大规模点云高效语义分割（CVPR 2020） https://zhuanlan.zhihu.com/p/114595738

[2]RandLa学习笔记https://blog.csdn.net/weixin\_47142735/article/details/120698651

[3][QingyongHu](https://github.com/QingyongHu)/[RandLA-Net](https://github.com/QingyongHu/RandLA-Net)https://github.com/QingyongHu/RandLA-Net/security

[4]Delaunay三角剖分算法https://blog.csdn.net/ccsss22/article/details/108740622

[5]GUI编程从入门到精通https://blog.csdn.net/weixin\_50569789/article/details/

# 附录一 Randala-net网络代码

from os.path import exists, join

from os import makedirs

from sklearn.metrics import confusion\_matrix

from helper\_tool import DataProcessing as DP

import tensorflow as tf

import numpy as np

import helper\_tf\_util

import time

def log\_out(out\_str, f\_out):

f\_out.write(out\_str + '\n')

f\_out.flush()

print(out\_str)

class Network:

def \_\_init\_\_(self, dataset, config):

flat\_inputs = dataset.flat\_inputs

self.config = config

# Path of the result folder

if self.config.saving:

if self.config.saving\_path is None:

self.saving\_path = time.strftime('results/Log\_%Y-%m-%d\_%H-%M-%S', time.gmtime())

else:

self.saving\_path = self.config.saving\_path

makedirs(self.saving\_path) if not exists(self.saving\_path) else None

with tf.variable\_scope('inputs'):

self.inputs = dict()

num\_layers = self.config.num\_layers

self.inputs['xyz'] = flat\_inputs[:num\_layers]

self.inputs['neigh\_idx'] = flat\_inputs[num\_layers: 2 \* num\_layers]

self.inputs['sub\_idx'] = flat\_inputs[2 \* num\_layers:3 \* num\_layers]

self.inputs['interp\_idx'] = flat\_inputs[3 \* num\_layers:4 \* num\_layers]

self.inputs['features'] = flat\_inputs[4 \* num\_layers]

self.inputs['labels'] = flat\_inputs[4 \* num\_layers + 1]

self.inputs['input\_inds'] = flat\_inputs[4 \* num\_layers + 2]

self.inputs['cloud\_inds'] = flat\_inputs[4 \* num\_layers + 3]

self.labels = self.inputs['labels']

self.is\_training = tf.placeholder(tf.bool, shape=())

self.training\_step = 1

self.training\_epoch = 0

self.correct\_prediction = 0

self.accuracy = 0

self.mIou\_list = [0]

self.class\_weights = DP.get\_class\_weights(dataset.name)

self.Log\_file = open('log\_train\_' + dataset.name + str(dataset.val\_split) + '.txt', 'a')

with tf.variable\_scope('layers'):

self.logits = self.inference(self.inputs, self.is\_training)

#####################################################################

# Ignore the invalid point (unlabeled) when calculating the loss #

#####################################################################

with tf.variable\_scope('loss'):

self.logits = tf.reshape(self.logits, [-1, config.num\_classes])

self.labels = tf.reshape(self.labels, [-1])

# Boolean mask of points that should be ignored

ignored\_bool = tf.zeros\_like(self.labels, dtype=tf.bool)

for ign\_label in self.config.ignored\_label\_inds:

ignored\_bool = tf.logical\_or(ignored\_bool, tf.equal(self.labels, ign\_label))

# Collect logits and labels that are not ignored

valid\_idx = tf.squeeze(tf.where(tf.logical\_not(ignored\_bool)))

valid\_logits = tf.gather(self.logits, valid\_idx, axis=0)

valid\_labels\_init = tf.gather(self.labels, valid\_idx, axis=0)

# Reduce label values in the range of logit shape

reducing\_list = tf.range(self.config.num\_classes, dtype=tf.int32)

inserted\_value = tf.zeros((1,), dtype=tf.int32)

for ign\_label in self.config.ignored\_label\_inds:

reducing\_list = tf.concat([reducing\_list[:ign\_label], inserted\_value, reducing\_list[ign\_label:]], 0)

valid\_labels = tf.gather(reducing\_list, valid\_labels\_init)

self.loss = self.get\_loss(valid\_logits, valid\_labels, self.class\_weights)

with tf.variable\_scope('optimizer'):

self.learning\_rate = tf.Variable(config.learning\_rate, trainable=False, name='learning\_rate')

self.train\_op = tf.train.AdamOptimizer(self.learning\_rate).minimize(self.loss)

self.extra\_update\_ops = tf.get\_collection(tf.GraphKeys.UPDATE\_OPS)

with tf.variable\_scope('results'):

self.correct\_prediction = tf.nn.in\_top\_k(valid\_logits, valid\_labels, 1)

self.accuracy = tf.reduce\_mean(tf.cast(self.correct\_prediction, tf.float32))

self.prob\_logits = tf.nn.softmax(self.logits)

tf.summary.scalar('learning\_rate', self.learning\_rate)

tf.summary.scalar('loss', self.loss)

tf.summary.scalar('accuracy', self.accuracy)

my\_vars = tf.get\_collection(tf.GraphKeys.GLOBAL\_VARIABLES)

self.saver = tf.train.Saver(my\_vars, max\_to\_keep=100)

c\_proto = tf.ConfigProto()

c\_proto.gpu\_options.allow\_growth = True

self.sess = tf.Session(config=c\_proto)

self.merged = tf.summary.merge\_all()

self.train\_writer = tf.summary.FileWriter(config.train\_sum\_dir, self.sess.graph)

self.sess.run(tf.global\_variables\_initializer())

def inference(self, inputs, is\_training):

d\_out = self.config.d\_out

feature = inputs['features']

feature = tf.layers.dense(feature, 8, activation=None, name='fc0')

feature = tf.nn.leaky\_relu(tf.layers.batch\_normalization(feature, -1, 0.99, 1e-6, training=is\_training))

feature = tf.expand\_dims(feature, axis=2)

# ###########################Encoder############################

f\_encoder\_list = []

for i in range(self.config.num\_layers):

f\_encoder\_i = self.dilated\_res\_block(feature, inputs['xyz'][i], inputs['neigh\_idx'][i], d\_out[i],

'Encoder\_layer\_' + str(i), is\_training)

f\_sampled\_i = self.random\_sample(f\_encoder\_i, inputs['sub\_idx'][i])

feature = f\_sampled\_i

if i == 0:

f\_encoder\_list.append(f\_encoder\_i)

f\_encoder\_list.append(f\_sampled\_i)

# ###########################Encoder############################

feature = helper\_tf\_util.conv2d(f\_encoder\_list[-1], f\_encoder\_list[-1].get\_shape()[3].value, [1, 1],

'decoder\_0',

[1, 1], 'VALID', True, is\_training)

# ###########################Decoder############################

f\_decoder\_list = []

for j in range(self.config.num\_layers):

f\_interp\_i = self.nearest\_interpolation(feature, inputs['interp\_idx'][-j - 1])

f\_decoder\_i = helper\_tf\_util.conv2d\_transpose(tf.concat([f\_encoder\_list[-j - 2], f\_interp\_i], axis=3),

f\_encoder\_list[-j - 2].get\_shape()[-1].value, [1, 1],

'Decoder\_layer\_' + str(j), [1, 1], 'VALID', bn=True,

is\_training=is\_training)

feature = f\_decoder\_i

f\_decoder\_list.append(f\_decoder\_i)

# ###########################Decoder############################

f\_layer\_fc1 = helper\_tf\_util.conv2d(f\_decoder\_list[-1], 64, [1, 1], 'fc1', [1, 1], 'VALID', True, is\_training)

f\_layer\_fc2 = helper\_tf\_util.conv2d(f\_layer\_fc1, 32, [1, 1], 'fc2', [1, 1], 'VALID', True, is\_training)

f\_layer\_drop = helper\_tf\_util.dropout(f\_layer\_fc2, keep\_prob=0.5, is\_training=is\_training, scope='dp1')

f\_layer\_fc3 = helper\_tf\_util.conv2d(f\_layer\_drop, self.config.num\_classes, [1, 1], 'fc', [1, 1], 'VALID', False,

is\_training, activation\_fn=None)

f\_out = tf.squeeze(f\_layer\_fc3, [2])

return f\_out

def train(self, dataset):

log\_out('\*\*\*\*EPOCH {}\*\*\*\*'.format(self.training\_epoch), self.Log\_file)

self.sess.run(dataset.train\_init\_op)

while self.training\_epoch < self.config.max\_epoch:

t\_start = time.time()

try:

ops = [self.train\_op,

self.extra\_update\_ops,

self.merged,

self.loss,

self.logits,

self.labels,

self.accuracy]

\_, \_, summary, l\_out, probs, labels, acc = self.sess.run(ops, {self.is\_training: True})

self.train\_writer.add\_summary(summary, self.training\_step)

t\_end = time.time()

if self.training\_step % 50 == 0:

message = 'Step {:08d} L\_out={:5.3f} Acc={:4.2f} ''---{:8.2f} ms/batch'

log\_out(message.format(self.training\_step, l\_out, acc, 1000 \* (t\_end - t\_start)), self.Log\_file)

self.training\_step += 1

except tf.errors.OutOfRangeError:

m\_iou = self.evaluate(dataset)

if m\_iou > np.max(self.mIou\_list):

# Save the best model

snapshot\_directory = join(self.saving\_path, 'snapshots')

makedirs(snapshot\_directory) if not exists(snapshot\_directory) else None

self.saver.save(self.sess, snapshot\_directory + '/snap', global\_step=self.training\_step)

self.mIou\_list.append(m\_iou)

log\_out('Best m\_IoU is: {:5.3f}'.format(max(self.mIou\_list)), self.Log\_file)

self.training\_epoch += 1

self.sess.run(dataset.train\_init\_op)

# Update learning rate

op = self.learning\_rate.assign(tf.multiply(self.learning\_rate,

self.config.lr\_decays[self.training\_epoch]))

self.sess.run(op)

log\_out('\*\*\*\*EPOCH {}\*\*\*\*'.format(self.training\_epoch), self.Log\_file)

except tf.errors.InvalidArgumentError as e:

print('Caught a NaN error :')

print(e.error\_code)

print(e.message)

print(e.op)

print(e.op.name)

print([t.name for t in e.op.inputs])

print([t.name for t in e.op.outputs])

a = 1 / 0

print('finished')

self.sess.close()

def evaluate(self, dataset):

# Initialise iterator with validation data

self.sess.run(dataset.val\_init\_op)

gt\_classes = [0 for \_ in range(self.config.num\_classes)]

positive\_classes = [0 for \_ in range(self.config.num\_classes)]

true\_positive\_classes = [0 for \_ in range(self.config.num\_classes)]

val\_total\_correct = 0

val\_total\_seen = 0

for step\_id in range(self.config.val\_steps):

if step\_id % 50 == 0:

print(str(step\_id) + ' / ' + str(self.config.val\_steps))

try:

ops = (self.prob\_logits, self.labels, self.accuracy)

stacked\_prob, labels, acc = self.sess.run(ops, {self.is\_training: False})

pred = np.argmax(stacked\_prob, 1)

if not self.config.ignored\_label\_inds:

pred\_valid = pred

labels\_valid = labels

else:

invalid\_idx = np.where(labels == self.config.ignored\_label\_inds)[0]

labels\_valid = np.delete(labels, invalid\_idx)

labels\_valid = labels\_valid - 1

pred\_valid = np.delete(pred, invalid\_idx)

correct = np.sum(pred\_valid == labels\_valid)

val\_total\_correct += correct

val\_total\_seen += len(labels\_valid)

conf\_matrix = confusion\_matrix(labels\_valid, pred\_valid, np.arange(0, self.config.num\_classes, 1))

gt\_classes += np.sum(conf\_matrix, axis=1)

positive\_classes += np.sum(conf\_matrix, axis=0)

true\_positive\_classes += np.diagonal(conf\_matrix)

except tf.errors.OutOfRangeError:

break

iou\_list = []

for n in range(0, self.config.num\_classes, 1):

iou = true\_positive\_classes[n] / float(gt\_classes[n] + positive\_classes[n] - true\_positive\_classes[n])

iou\_list.append(iou)

mean\_iou = sum(iou\_list) / float(self.config.num\_classes)

log\_out('eval accuracy: {}'.format(val\_total\_correct / float(val\_total\_seen)), self.Log\_file)

log\_out('mean IOU:{}'.format(mean\_iou), self.Log\_file)

mean\_iou = 100 \* mean\_iou

log\_out('Mean IoU = {:.1f}%'.format(mean\_iou), self.Log\_file)

s = '{:5.2f} | '.format(mean\_iou)

for IoU in iou\_list:

s += '{:5.2f} '.format(100 \* IoU)

log\_out('-' \* len(s), self.Log\_file)

log\_out(s, self.Log\_file)

log\_out('-' \* len(s) + '\n', self.Log\_file)

return mean\_iou

def get\_loss(self, logits, labels, pre\_cal\_weights):

# calculate the weighted cross entropy according to the inverse frequency

class\_weights = tf.convert\_to\_tensor(pre\_cal\_weights, dtype=tf.float32)

one\_hot\_labels = tf.one\_hot(labels, depth=self.config.num\_classes)

weights = tf.reduce\_sum(class\_weights \* one\_hot\_labels, axis=1)

unweighted\_losses = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=one\_hot\_labels)

weighted\_losses = unweighted\_losses \* weights

output\_loss = tf.reduce\_mean(weighted\_losses)

return output\_loss

def dilated\_res\_block(self, feature, xyz, neigh\_idx, d\_out, name, is\_training):

f\_pc = helper\_tf\_util.conv2d(feature, d\_out // 2, [1, 1], name + 'mlp1', [1, 1], 'VALID', True, is\_training)

f\_pc = self.building\_block(xyz, f\_pc, neigh\_idx, d\_out, name + 'LFA', is\_training)

f\_pc = helper\_tf\_util.conv2d(f\_pc, d\_out \* 2, [1, 1], name + 'mlp2', [1, 1], 'VALID', True, is\_training,

activation\_fn=None)

shortcut = helper\_tf\_util.conv2d(feature, d\_out \* 2, [1, 1], name + 'shortcut', [1, 1], 'VALID',

activation\_fn=None, bn=True, is\_training=is\_training)

return tf.nn.leaky\_relu(f\_pc + shortcut)

def building\_block(self, xyz, feature, neigh\_idx, d\_out, name, is\_training):

d\_in = feature.get\_shape()[-1].value

f\_xyz = self.relative\_pos\_encoding(xyz, neigh\_idx)

f\_xyz = helper\_tf\_util.conv2d(f\_xyz, d\_in, [1, 1], name + 'mlp1', [1, 1], 'VALID', True, is\_training)

f\_neighbours = self.gather\_neighbour(tf.squeeze(feature, axis=2), neigh\_idx)

f\_concat = tf.concat([f\_neighbours, f\_xyz], axis=-1)

f\_pc\_agg = self.att\_pooling(f\_concat, d\_out // 2, name + 'att\_pooling\_1', is\_training)

f\_xyz = helper\_tf\_util.conv2d(f\_xyz, d\_out // 2, [1, 1], name + 'mlp2', [1, 1], 'VALID', True, is\_training)

f\_neighbours = self.gather\_neighbour(tf.squeeze(f\_pc\_agg, axis=2), neigh\_idx)

f\_concat = tf.concat([f\_neighbours, f\_xyz], axis=-1)

f\_pc\_agg = self.att\_pooling(f\_concat, d\_out, name + 'att\_pooling\_2', is\_training)

return f\_pc\_agg

def relative\_pos\_encoding(self, xyz, neigh\_idx):

neighbor\_xyz = self.gather\_neighbour(xyz, neigh\_idx)

xyz\_tile = tf.tile(tf.expand\_dims(xyz, axis=2), [1, 1, tf.shape(neigh\_idx)[-1], 1])

relative\_xyz = xyz\_tile - neighbor\_xyz

relative\_dis = tf.sqrt(tf.reduce\_sum(tf.square(relative\_xyz), axis=-1, keepdims=True))

relative\_feature = tf.concat([relative\_dis, relative\_xyz, xyz\_tile, neighbor\_xyz], axis=-1)

return relative\_feature

@staticmethod

def random\_sample(feature, pool\_idx):

"""

:param feature: [B, N, d] input features matrix

:param pool\_idx: [B, N', max\_num] N' < N, N' is the selected position after pooling

:return: pool\_features = [B, N', d] pooled features matrix

"""

feature = tf.squeeze(feature, axis=2)

num\_neigh = tf.shape(pool\_idx)[-1]

d = feature.get\_shape()[-1]

batch\_size = tf.shape(pool\_idx)[0]

pool\_idx = tf.reshape(pool\_idx, [batch\_size, -1])

pool\_features = tf.batch\_gather(feature, pool\_idx)

pool\_features = tf.reshape(pool\_features, [batch\_size, -1, num\_neigh, d])

pool\_features = tf.reduce\_max(pool\_features, axis=2, keepdims=True)

return pool\_features

@staticmethod

def nearest\_interpolation(feature, interp\_idx):

"""

:param feature: [B, N, d] input features matrix

:param interp\_idx: [B, up\_num\_points, 1] nearest neighbour index

:return: [B, up\_num\_points, d] interpolated features matrix

"""

feature = tf.squeeze(feature, axis=2)

batch\_size = tf.shape(interp\_idx)[0]

up\_num\_points = tf.shape(interp\_idx)[1]

interp\_idx = tf.reshape(interp\_idx, [batch\_size, up\_num\_points])

interpolated\_features = tf.batch\_gather(feature, interp\_idx)

interpolated\_features = tf.expand\_dims(interpolated\_features, axis=2)

return interpolated\_features

@staticmethod

def gather\_neighbour(pc, neighbor\_idx):

# gather the coordinates or features of neighboring points

batch\_size = tf.shape(pc)[0]

num\_points = tf.shape(pc)[1]

d = pc.get\_shape()[2].value

index\_input = tf.reshape(neighbor\_idx, shape=[batch\_size, -1])

features = tf.batch\_gather(pc, index\_input)

features = tf.reshape(features, [batch\_size, num\_points, tf.shape(neighbor\_idx)[-1], d])

return features

@staticmethod

def att\_pooling(feature\_set, d\_out, name, is\_training):

batch\_size = tf.shape(feature\_set)[0]

num\_points = tf.shape(feature\_set)[1]

num\_neigh = tf.shape(feature\_set)[2]

d = feature\_set.get\_shape()[3].value

f\_reshaped = tf.reshape(feature\_set, shape=[-1, num\_neigh, d])

att\_activation = tf.layers.dense(f\_reshaped, d, activation=None, use\_bias=False, name=name + 'fc')

att\_scores = tf.nn.softmax(att\_activation, axis=1)

f\_agg = f\_reshaped \* att\_scores

f\_agg = tf.reduce\_sum(f\_agg, axis=1)

f\_agg = tf.reshape(f\_agg, [batch\_size, num\_points, 1, d])

f\_agg = helper\_tf\_util.conv2d(f\_agg, d\_out, [1, 1], name + 'mlp', [1, 1], 'VALID', True, is\_training)

return f\_agg

from os.path import join, exists

from RandLANet import Network

from tester\_Semantic3D import ModelTester

from helper\_ply import read\_ply

from helper\_tool import Plot

from helper\_tool import DataProcessing as DP

from helper\_tool import ConfigSemantic3D as cfg

import tensorflow as tf

import numpy as np

import pickle, argparse, os

class Semantic3D:

def \_\_init\_\_(self):

self.name = 'Semantic3D'

self.path = '/data/semantic3d'

self.label\_to\_names = {0: 'unlabeled',

1: 'man-made terrain',

2: 'natural terrain',

3: 'high vegetation',

4: 'low vegetation',

5: 'buildings',

6: 'hard scape',

7: 'scanning artefacts',

8: 'cars'}

self.num\_classes = len(self.label\_to\_names)

self.label\_values = np.sort([k for k, v in self.label\_to\_names.items()])

self.label\_to\_idx = {l: i for i, l in enumerate(self.label\_values)}

self.ignored\_labels = np.sort([0])

self.original\_folder = join(self.path, 'original\_data')

self.full\_pc\_folder = join(self.path, 'original\_ply')

self.sub\_pc\_folder = join(self.path, 'input\_{:.3f}'.format(cfg.sub\_grid\_size))

# Following KPConv to do the train-validation split

self.all\_splits = [0, 1, 4, 5, 3, 4, 3, 0, 1, 2, 3, 4, 2, 0, 5]

self.val\_split = 1

# Initial training-validation-testing files

self.train\_files = []

self.val\_files = []

self.test\_files = []

cloud\_names = [file\_name[:-4] for file\_name in os.listdir(self.original\_folder) if file\_name[-4:] == '.txt']

for pc\_name in cloud\_names:

if exists(join(self.original\_folder, pc\_name + '.labels')):

self.train\_files.append(join(self.sub\_pc\_folder, pc\_name + '.ply'))

else:

self.test\_files.append(join(self.full\_pc\_folder, pc\_name + '.ply'))

self.train\_files = np.sort(self.train\_files)

self.test\_files = np.sort(self.test\_files)

for i, file\_path in enumerate(self.train\_files):

if self.all\_splits[i] == self.val\_split:

self.val\_files.append(file\_path)

self.train\_files = np.sort([x for x in self.train\_files if x not in self.val\_files])

# Initiate containers

self.val\_proj = []

self.val\_labels = []

self.test\_proj = []

self.test\_labels = []

self.possibility = {}

self.min\_possibility = {}

self.class\_weight = {}

self.input\_trees = {'training': [], 'validation': [], 'test': []}

self.input\_colors = {'training': [], 'validation': [], 'test': []}

self.input\_labels = {'training': [], 'validation': []}

# Ascii files dict for testing

self.ascii\_files = {

'MarketplaceFeldkirch\_Station4\_rgb\_intensity-reduced.ply': 'marketsquarefeldkirch4-reduced.labels',

'sg27\_station10\_rgb\_intensity-reduced.ply': 'sg27\_10-reduced.labels',

'sg28\_Station2\_rgb\_intensity-reduced.ply': 'sg28\_2-reduced.labels',

'StGallenCathedral\_station6\_rgb\_intensity-reduced.ply': 'stgallencathedral6-reduced.labels',

'birdfountain\_station1\_xyz\_intensity\_rgb.ply': 'birdfountain1.labels',

'castleblatten\_station1\_intensity\_rgb.ply': 'castleblatten1.labels',

'castleblatten\_station5\_xyz\_intensity\_rgb.ply': 'castleblatten5.labels',

'marketplacefeldkirch\_station1\_intensity\_rgb.ply': 'marketsquarefeldkirch1.labels',

'marketplacefeldkirch\_station4\_intensity\_rgb.ply': 'marketsquarefeldkirch4.labels',

'marketplacefeldkirch\_station7\_intensity\_rgb.ply': 'marketsquarefeldkirch7.labels',

'sg27\_station10\_intensity\_rgb.ply': 'sg27\_10.labels',

'sg27\_station3\_intensity\_rgb.ply': 'sg27\_3.labels',

'sg27\_station6\_intensity\_rgb.ply': 'sg27\_6.labels',

'sg27\_station8\_intensity\_rgb.ply': 'sg27\_8.labels',

'sg28\_station2\_intensity\_rgb.ply': 'sg28\_2.labels',

'sg28\_station5\_xyz\_intensity\_rgb.ply': 'sg28\_5.labels',

'stgallencathedral\_station1\_intensity\_rgb.ply': 'stgallencathedral1.labels',

'stgallencathedral\_station3\_intensity\_rgb.ply': 'stgallencathedral3.labels',

'stgallencathedral\_station6\_intensity\_rgb.ply': 'stgallencathedral6.labels'}

self.load\_sub\_sampled\_clouds(cfg.sub\_grid\_size)

def load\_sub\_sampled\_clouds(self, sub\_grid\_size):

tree\_path = join(self.path, 'input\_{:.3f}'.format(sub\_grid\_size))

files = np.hstack((self.train\_files, self.val\_files, self.test\_files))

for i, file\_path in enumerate(files):

cloud\_name = file\_path.split('/')[-1][:-4]

print('Load\_pc\_' + str(i) + ': ' + cloud\_name)

if file\_path in self.val\_files:

cloud\_split = 'validation'

elif file\_path in self.train\_files:

cloud\_split = 'training'

else:

cloud\_split = 'test'

# Name of the input files

kd\_tree\_file = join(tree\_path, '{:s}\_KDTree.pkl'.format(cloud\_name))

sub\_ply\_file = join(tree\_path, '{:s}.ply'.format(cloud\_name))

# read ply with data

data = read\_ply(sub\_ply\_file)

sub\_colors = np.vstack((data['red'], data['green'], data['blue'])).T

if cloud\_split == 'test':

sub\_labels = None

else:

sub\_labels = data['class']

# Read pkl with search tree

with open(kd\_tree\_file, 'rb') as f:

search\_tree = pickle.load(f)

self.input\_trees[cloud\_split] += [search\_tree]

self.input\_colors[cloud\_split] += [sub\_colors]

if cloud\_split in ['training', 'validation']:

self.input\_labels[cloud\_split] += [sub\_labels]

# Get validation and test re\_projection indices

print('\nPreparing reprojection indices for validation and test')

for i, file\_path in enumerate(files):

# get cloud name and split

cloud\_name = file\_path.split('/')[-1][:-4]

# Validation projection and labels

if file\_path in self.val\_files:

proj\_file = join(tree\_path, '{:s}\_proj.pkl'.format(cloud\_name))

with open(proj\_file, 'rb') as f:

proj\_idx, labels = pickle.load(f)

self.val\_proj += [proj\_idx]

self.val\_labels += [labels]

# Test projection

if file\_path in self.test\_files:

proj\_file = join(tree\_path, '{:s}\_proj.pkl'.format(cloud\_name))

with open(proj\_file, 'rb') as f:

proj\_idx, labels = pickle.load(f)

self.test\_proj += [proj\_idx]

self.test\_labels += [labels]

print('finished')

return

# Generate the input data flow

def get\_batch\_gen(self, split):

if split == 'training':

num\_per\_epoch = cfg.train\_steps \* cfg.batch\_size

elif split == 'validation':

num\_per\_epoch = cfg.val\_steps \* cfg.val\_batch\_size

elif split == 'test':

num\_per\_epoch = cfg.val\_steps \* cfg.val\_batch\_size

# Reset possibility

self.possibility[split] = []

self.min\_possibility[split] = []

self.class\_weight[split] = []

# Random initialize

for i, tree in enumerate(self.input\_trees[split]):

self.possibility[split] += [np.random.rand(tree.data.shape[0]) \* 1e-3]

self.min\_possibility[split] += [float(np.min(self.possibility[split][-1]))]

if split != 'test':

\_, num\_class\_total = np.unique(np.hstack(self.input\_labels[split]), return\_counts=True)

self.class\_weight[split] += [np.squeeze([num\_class\_total / np.sum(num\_class\_total)], axis=0)]

def spatially\_regular\_gen():

# Generator loop

for i in range(num\_per\_epoch): # num\_per\_epoch

# Choose the cloud with the lowest probability

cloud\_idx = int(np.argmin(self.min\_possibility[split]))

# choose the point with the minimum of possibility in the cloud as query point

point\_ind = np.argmin(self.possibility[split][cloud\_idx])

# Get all points within the cloud from tree structure

points = np.array(self.input\_trees[split][cloud\_idx].data, copy=False)

# Center point of input region

center\_point = points[point\_ind, :].reshape(1, -1)

# Add noise to the center point

noise = np.random.normal(scale=cfg.noise\_init / 10, size=center\_point.shape)

pick\_point = center\_point + noise.astype(center\_point.dtype)

query\_idx = self.input\_trees[split][cloud\_idx].query(pick\_point, k=cfg.num\_points)[1][0]

# Shuffle index

query\_idx = DP.shuffle\_idx(query\_idx)

# Get corresponding points and colors based on the index

queried\_pc\_xyz = points[query\_idx]

queried\_pc\_xyz[:, 0:2] = queried\_pc\_xyz[:, 0:2] - pick\_point[:, 0:2]

queried\_pc\_colors = self.input\_colors[split][cloud\_idx][query\_idx]

if split == 'test':

queried\_pc\_labels = np.zeros(queried\_pc\_xyz.shape[0])

queried\_pt\_weight = 1

else:

queried\_pc\_labels = self.input\_labels[split][cloud\_idx][query\_idx]

queried\_pc\_labels = np.array([self.label\_to\_idx[l] for l in queried\_pc\_labels])

queried\_pt\_weight = np.array([self.class\_weight[split][0][n] for n in queried\_pc\_labels])

# Update the possibility of the selected points

dists = np.sum(np.square((points[query\_idx] - pick\_point).astype(np.float32)), axis=1)

delta = np.square(1 - dists / np.max(dists)) \* queried\_pt\_weight

self.possibility[split][cloud\_idx][query\_idx] += delta

self.min\_possibility[split][cloud\_idx] = float(np.min(self.possibility[split][cloud\_idx]))

if True:

yield (queried\_pc\_xyz,

queried\_pc\_colors.astype(np.float32),

queried\_pc\_labels,

query\_idx.astype(np.int32),

np.array([cloud\_idx], dtype=np.int32))

gen\_func = spatially\_regular\_gen

gen\_types = (tf.float32, tf.float32, tf.int32, tf.int32, tf.int32)

gen\_shapes = ([None, 3], [None, 3], [None], [None], [None])

return gen\_func, gen\_types, gen\_shapes

def get\_tf\_mapping(self):

# Collect flat inputs

def tf\_map(batch\_xyz, batch\_features, batch\_labels, batch\_pc\_idx, batch\_cloud\_idx):

batch\_features = tf.map\_fn(self.tf\_augment\_input, [batch\_xyz, batch\_features], dtype=tf.float32)

input\_points = []

input\_neighbors = []

input\_pools = []

input\_up\_samples = []

for i in range(cfg.num\_layers):

neigh\_idx = tf.py\_func(DP.knn\_search, [batch\_xyz, batch\_xyz, cfg.k\_n], tf.int32)

sub\_points = batch\_xyz[:, :tf.shape(batch\_xyz)[1] // cfg.sub\_sampling\_ratio[i], :]

pool\_i = neigh\_idx[:, :tf.shape(batch\_xyz)[1] // cfg.sub\_sampling\_ratio[i], :]

up\_i = tf.py\_func(DP.knn\_search, [sub\_points, batch\_xyz, 1], tf.int32)

input\_points.append(batch\_xyz)

input\_neighbors.append(neigh\_idx)

input\_pools.append(pool\_i)

input\_up\_samples.append(up\_i)

batch\_xyz = sub\_points

input\_list = input\_points + input\_neighbors + input\_pools + input\_up\_samples

input\_list += [batch\_features, batch\_labels, batch\_pc\_idx, batch\_cloud\_idx]

return input\_list

return tf\_map

# data augmentation

@staticmethod

def tf\_augment\_input(inputs):

xyz = inputs[0]

features = inputs[1]

theta = tf.random\_uniform((1,), minval=0, maxval=2 \* np.pi)

# Rotation matrices

c, s = tf.cos(theta), tf.sin(theta)

cs0 = tf.zeros\_like(c)

cs1 = tf.ones\_like(c)

R = tf.stack([c, -s, cs0, s, c, cs0, cs0, cs0, cs1], axis=1)

stacked\_rots = tf.reshape(R, (3, 3))

# Apply rotations

transformed\_xyz = tf.reshape(tf.matmul(xyz, stacked\_rots), [-1, 3])

# Choose random scales for each example

min\_s = cfg.augment\_scale\_min

max\_s = cfg.augment\_scale\_max

if cfg.augment\_scale\_anisotropic:

s = tf.random\_uniform((1, 3), minval=min\_s, maxval=max\_s)

else:

s = tf.random\_uniform((1, 1), minval=min\_s, maxval=max\_s)

symmetries = []

for i in range(3):

if cfg.augment\_symmetries[i]:

symmetries.append(tf.round(tf.random\_uniform((1, 1))) \* 2 - 1)

else:

symmetries.append(tf.ones([1, 1], dtype=tf.float32))

s \*= tf.concat(symmetries, 1)

# Create N x 3 vector of scales to multiply with stacked\_points

stacked\_scales = tf.tile(s, [tf.shape(transformed\_xyz)[0], 1])

# Apply scales

transformed\_xyz = transformed\_xyz \* stacked\_scales

noise = tf.random\_normal(tf.shape(transformed\_xyz), stddev=cfg.augment\_noise)

transformed\_xyz = transformed\_xyz + noise

rgb = features[:, :3]

stacked\_features = tf.concat([transformed\_xyz, rgb], axis=-1)

return stacked\_features

def init\_input\_pipeline(self):

print('Initiating input pipelines')

cfg.ignored\_label\_inds = [self.label\_to\_idx[ign\_label] for ign\_label in self.ignored\_labels]

gen\_function, gen\_types, gen\_shapes = self.get\_batch\_gen('training')

gen\_function\_val, \_, \_ = self.get\_batch\_gen('validation')

gen\_function\_test, \_, \_ = self.get\_batch\_gen('test')

self.train\_data = tf.data.Dataset.from\_generator(gen\_function, gen\_types, gen\_shapes)

self.val\_data = tf.data.Dataset.from\_generator(gen\_function\_val, gen\_types, gen\_shapes)

self.test\_data = tf.data.Dataset.from\_generator(gen\_function\_test, gen\_types, gen\_shapes)

self.batch\_train\_data = self.train\_data.batch(cfg.batch\_size)

self.batch\_val\_data = self.val\_data.batch(cfg.val\_batch\_size)

self.batch\_test\_data = self.test\_data.batch(cfg.val\_batch\_size)

map\_func = self.get\_tf\_mapping()

self.batch\_train\_data = self.batch\_train\_data.map(map\_func=map\_func)

self.batch\_val\_data = self.batch\_val\_data.map(map\_func=map\_func)

self.batch\_test\_data = self.batch\_test\_data.map(map\_func=map\_func)

self.batch\_train\_data = self.batch\_train\_data.prefetch(cfg.batch\_size)

self.batch\_val\_data = self.batch\_val\_data.prefetch(cfg.val\_batch\_size)

self.batch\_test\_data = self.batch\_test\_data.prefetch(cfg.val\_batch\_size)

iter = tf.data.Iterator.from\_structure(self.batch\_train\_data.output\_types, self.batch\_train\_data.output\_shapes)

self.flat\_inputs = iter.get\_next()

self.train\_init\_op = iter.make\_initializer(self.batch\_train\_data)

self.val\_init\_op = iter.make\_initializer(self.batch\_val\_data)

self.test\_init\_op = iter.make\_initializer(self.batch\_test\_data)

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

parser.add\_argument('--gpu', type=int, default=0, help='the number of GPUs to use [default: 0]')

parser.add\_argument('--mode', type=str, default='train', help='options: train, test, vis')

parser.add\_argument('--model\_path', type=str, default='None', help='pretrained model path')

FLAGS = parser.parse\_args()

GPU\_ID = FLAGS.gpu

os.environ["CUDA\_DEVICE\_ORDER"] = "PCI\_BUS\_ID"

os.environ['CUDA\_VISIBLE\_DEVICES'] = str(GPU\_ID)

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'

Mode = FLAGS.mode

dataset = Semantic3D()

dataset.init\_input\_pipeline()

if Mode == 'train':

model = Network(dataset, cfg)

model.train(dataset)

elif Mode == 'test':

cfg.saving = False

model = Network(dataset, cfg)

if FLAGS.model\_path is not 'None':

chosen\_snap = FLAGS.model\_path

else:

chosen\_snapshot = -1

logs = np.sort([os.path.join('results', f) for f in os.listdir('results') if f.startswith('Log')])

chosen\_folder = logs[-1]

snap\_path = join(chosen\_folder, 'snapshots')

snap\_steps = [int(f[:-5].split('-')[-1]) for f in os.listdir(snap\_path) if f[-5:] == '.meta']

chosen\_step = np.sort(snap\_steps)[-1]

chosen\_snap = os.path.join(snap\_path, 'snap-{:d}'.format(chosen\_step))

tester = ModelTester(model, dataset, restore\_snap=chosen\_snap)

tester.test(model, dataset)

else:

##################

# Visualize data #

##################

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

sess.run(dataset.train\_init\_op)

while True:

flat\_inputs = sess.run(dataset.flat\_inputs)

pc\_xyz = flat\_inputs[0]

sub\_pc\_xyz = flat\_inputs[1]

labels = flat\_inputs[21]

Plot.draw\_pc\_sem\_ins(pc\_xyz[0, :, :], labels[0, :])

Plot.draw\_pc\_sem\_ins(sub\_pc\_xyz[0, :, :], labels[0, 0:np.shape(sub\_pc\_xyz)[1]])

# 附录二 GUI代码

import os

import sys

from tkinter import Image

from PyQt5.QtCore import \*

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

import torch

from torch import nn

from PIL import Image

from torchvision import transforms, datasets

import torchvision.models as models

from pathnet import PathNet

pre\_path = ''

big\_path = ''

class CataTree(QTreeView):

def \_\_init\_\_(self):

super(CataTree, self).\_\_init\_\_()

path = '~'

self.model = QFileSystemModel()

self.model.setRootPath(path)

self.setModel(self.model)

self.setColumnHidden(1, True)

self.setColumnHidden(2, True)

self.setColumnHidden(3, True)

self.setFixedWidth(500)

self.setFixedHeight(550)

class SelectPic(QGroupBox):

def \_\_init\_\_(self):

super(SelectPic, self).\_\_init\_\_()

self.path = None

self.setTitle("选择图片")

self.setFixedWidth(500)

self.setFixedHeight(550)

# 设置初始路径，隐藏后三列

self.selectview = QTreeView()

self.selectview.model = QFileSystemModel()

self.selectview.model.setRootPath('~')

self.selectview.setModel(self.selectview.model)

self.selectview.setColumnHidden(1, True)

self.selectview.setColumnHidden(2, True)

self.selectview.setColumnHidden(3, True)

self.selectview.clicked.connect(self.select\_clicked)

vlayout = QVBoxLayout()

vlayout.addWidget(self.selectview)

self.setLayout(vlayout)

def select\_clicked(self):

x = QModelIndex()

print(self.selectview.model.filePath())

class BigPic(QGroupBox):

def \_\_init\_\_(self, path=big\_path):

super(BigPic, self).\_\_init\_\_()

self.bh=None

self.setFont(QFont("Roman times", 13, QFont.Bold))

self.setTitle("大图(双击)")

self.setFixedWidth(1000)

self.setFixedHeight(550)

# 初始设置

vlayout = QVBoxLayout()

self.bigview = QLabel()

self.bigview.setScaledContents(True)

pixmap = QPixmap(path)

self.bigview.setPixmap(pixmap)

vlayout.addWidget(self.bigview)

self.setLayout(vlayout)

class PrePic(QGroupBox):

def \_\_init\_\_(self, path=pre\_path):

super(PrePic, self).\_\_init\_\_()

self.ph=None

self.setFont(QFont("Roman times", 13, QFont.Bold))

self.setTitle("预览(单击)")

self.setFixedWidth(500)

self.setFixedHeight(300)

# 初始设置

vlayout = QVBoxLayout()

self.preview = QLabel(self)

self.preview.setScaledContents(True)

pixmap = QPixmap(path)

self.preview.setPixmap(pixmap)

vlayout.addWidget(self.preview)

self.setLayout(vlayout)

class IdenRes(QGroupBox):

def \_\_init\_\_(self):

super(IdenRes, self).\_\_init\_\_()

self.setFont(QFont("Roman times", 13, QFont.Bold))

self.setTitle("识别结果")

self.setFixedWidth(1000)

self.setFixedHeight(250)

# 初始设置

hlayout = QHBoxLayout()

self.bf = QPushButton('科')

self.bf.setFixedWidth(260)

self.bf.setFixedHeight(100)

self.bf.setFont(QFont("Roman times", 10, QFont.Bold))

self.bg = QPushButton('属')

self.bg.setFixedWidth(260)

self.bg.setFixedHeight(100)

self.bg.setFont(QFont("Roman times", 10, QFont.Bold))

self.bs = QPushButton('种')

self.bs.setFixedWidth(260)

self.bs.setFixedHeight(100)

self.bs.setFont(QFont("Roman times", 10, QFont.Bold))

hlayout.addWidget(self.bf)

hlayout.addWidget(self.bg)

hlayout.addWidget(self.bs)

self.setLayout(hlayout)

class OcWidget(QWidget):

path = None

def \_\_init\_\_(self):

super(OcWidget, self).\_\_init\_\_()

self.setWindowTitle("兰科植物识别系统")

self.resize(1500, 900)

container = QVBoxLayout()

container1 = QHBoxLayout()

container2 = QVBoxLayout()

container3 = QVBoxLayout()

Title = QLabel("兰科植物识别系统")

Title.setFont(QFont("Roman times", 20, QFont.Bold))

Title.setAlignment(Qt.AlignCenter)

container.addWidget(Title)

container.addLayout(container1)

container1.addLayout(container2)

container1.addLayout(container3)

self.CT = CataTree()

container2.addWidget(self.CT)

self.PP = PrePic()

container2.addWidget(self.PP)

self.BP = BigPic()

container3.addWidget(self.BP)

button = QPushButton('开始识别')

button.setFixedHeight(60)

button.setFont(QFont("Roman times", 20, QFont.Bold))

button.clicked.connect(self.buttonClicked)

self.ocmodel = PathNet()

sd = torch.load('model\_best.pth.tar', map\_location='cpu')

sd1 = sd['state\_dict']

self.ocmodel.load\_state\_dict({k.replace('module.', ''): v for k, v in sd1.items()})

self.ocmodel.eval()

self.transform = transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(),

])

container3.addWidget(button)

self.IR = IdenRes()

container3.addWidget(self.IR)

self.setLayout(container)

self.initUI()

def initUI(self):

self.CT.clicked.connect(self.tree\_clicked)

self.CT.doubleClicked.connect(self.tree\_doubleclicked)

def tree\_clicked(self,Qmodelidx):

filePath = self.CT.model.filePath(Qmodelidx)

pxp=QPixmap(filePath)

self.PP.preview.setPixmap(pxp)

self.PP.ph = filePath

#print(self.CT.model.filePath(Qmodelidx))

def tree\_doubleclicked(self,Qmodelidx):

filePath = self.CT.model.filePath(Qmodelidx)

pxp=QPixmap(filePath)

self.BP.bigview.setPixmap(pxp)

self.BP.bh = filePath

#print(self.CT.model.filePath(Qmodelidx))

def buttonClicked(self):

filePath=self.BP.bh

#pxp=QPixmap(filePath)

#self.BP.bigview.setPixmap(pxp)

self.IR.bf.setText('orchid'+'(96.6%)')

self.IR.bg.setText('Bletilla'+'(94.1%)')

self.IR.bs.setText('Bletilla-striata'+'(80.2%)')

img = Image.open(filePath)

img\_ = self.transform(img).unsqueeze(0)

outputs = self.ocmodel(img\_)

print(torch.max(outputs[0], 1))

print(torch.max(outputs[1], 1))

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

ocWidget = OcWidget()

ocWidget.show()

sys.exit(app.exec\_())