# Python图像的读取

## PIL (Python Image Library)

PIL是免费的图像处理工具包, 该软件包提供了基本的图像处理功能，如：改变图像大小，旋转图像，图像格式转换，色场空间转换，图像增强，直方图处理，插值和滤波等等。

## PIL库的功能

Pillow 库能够很轻松的读取和保存各种格式的图片；

Pillow 库提供了简洁易用的 API 接口，可以让您轻松地完成许多图像处理任务；

Pillow 库能够配合 GUI（图形用户界面） 软件包一起使用；

Pillow 库中的 Image 对象能够与 NumPy ndarray 数组实现相互转换。

http://c.biancheng.net/pillow/image-objects.html

## 安装

Jupyter展示所用的环境：  
conda create -n p\_p python=3.8

conda activate p\_p

pip install jupyter notebook

pip install pillow

pip install numpy opencv-python

pip install matplotlib

jupyter notebook

conda install ipykernel

python -m ipykernel install --user --name p\_p --display-name "Python (p\_p)"

## 导入

from PIL import Image

## open()

使用 Image 类的 open() 方法，可以创建一个 Image 对象，语法格式如下：

im = Image.open(fp,mode="r")

参数说明：

fp：即 filepath 的缩写，表示文件路径，字符串格式；

mode：可选参数，若出现该参数，则必须设置为 "r"，否则会引发 ValueError 异常。

示例：

from PIL import Image

#打开一图片文件

im = Image.open("C:/Users/Administrator/Desktop/c-net.png")

#要显示图像需要调用 show()方法

im.show()

## new()

使用 Image 类提供的 new() 方法可以创建一个新的 Image 对象，语法格式如下:

im=Image.new(mode,size,color)

参数说明如下：

mode：图像模式，字符串参数，比如 RGB（真彩图像）、L（灰度图像）、CMYK（色彩图打印模式）等；

size：图像大小，元组参数（width, height）代表图像的像素大小；

color：图片颜色，默认值为 0 表示黑色，参数值支持（R,G,B）三元组数字格式、颜色的十六进制值以及颜色英文单词。

#使用颜色的十六进制格式

im\_1=Image.new(mode='RGB',(260,100),color="#ff0000")

im\_1.show()

## save()

save() 方法用于保存图像，当不指定文件格式时，它会以默认的图片格式来存储；如果指定图片格式，则会以指定的格式存储图片。

save() 的语法格式如下：

Image.save(fp, format=None)

参数说明如下：

fp：图片的存储路径，包含图片的名称，字符串格式；

format：可选参数，可以指定图片的格式。

示例：

from PIL import Image

im = Image.open("C:/Users/Administrator/Desktop/c-net.png")

# 去噪

# 分割

im.save('C:/Users/Administrator/Desktop/c.biancheng.net.bmp')

## split()

分离通道

im=Image.open("C:/Users/Administrator/Desktop/1.jpg")

#修改图像大小，以适应图像处理

image=im.resize((450,400))

image.save("C:/Users/Administrator/Desktop/2.jpg")

#分离颜色通道，产生三个 Image对象

r,g,b = image.split()

r.show()

# 操作1

g.show()

# 操作2

b.show()

## merge()

合并通道

from PIL import Image

im=Image.open("C:/Users/Administrator/Desktop/1.jpg")

#修改图像大小，以适应图像处理

image=im.resize((450,400))

image.save("C:/Users/Administrator/Desktop/2.jpg")

#分离颜色通道，产生三个 Image对象

r,g,b = image.split()

#重新组合颜色通道，返回先的Image对象

image\_merge=Image.merge('RGB',(b,g,r))

image\_merge.show()

#保存图像至桌面

image\_merge.save("C:/Users/Administrator/Desktop/3.jpg")

## crop()

图像裁剪

crop(box=None)

box：表示裁剪区域，默认为 None，表示拷贝原图像。

注意：box 是一个有四个数字的元组参数 (x\_左上,y\_左下,x1\_右上,y1\_右下)，分别表示被裁剪矩形区域的左上角 x、y 坐标和右下角 x，y 坐标。默认 (0,0) 表示坐标原点，宽度的方向为 x 轴，高度的方向为 y 轴，每个像素点代表一个单位。

"""

裁剪图像

"""

im = Image.open("C:/Users/Administrator/Desktop/C语言中文网.png")

box =(0,0,200,100)

im\_crop = im.crop(box)

im\_crop.show()

## transpose()

图像旋转操作

Image.transpose(method)

method 参数决定了图片要如何翻转，参数值如下：

Image.FLIP\_LEFT\_RIGHT：左右水平翻转；

Image.FLIP\_TOP\_BOTTOM：上下垂直翻转；

Image.ROTATE\_90：图像旋转 90 度；

Image.ROTATE\_180：图像旋转 180 度；

Image.ROTATE\_270：图像旋转 270 度；

Image.TRANSPOSE：图像转置；

Image.TRANSVERSE：图像横向翻转。

im = Image.open("C:/Users/Administrator/Desktop/c-net.png")

#返回一个新的Image对象

im\_out=im.transpose(Image.FLIP\_LEFT\_RIGHT)

im\_out.show()

im\_out.save("C:/Users/Administrator/Desktop/水平翻转.png")

# 图像的降噪处理

由于成像设备、传输媒介等因素的影响，图像总会或多或少的存在一些不必要的干扰信息，我们将这些干扰信息统称为“噪声”，比如数字图像中常见的“椒盐噪声”，指的是图像会随机出现的一些白、黑色的像素点。图像噪声既影响了图像的质量，又妨碍人们的视觉观赏。因此，噪声处理是图像处理过程中必不可少的环节之一，我们把处理图像噪声的过程称为“图像降噪”。

Pillow 通过 ImageFilter 类达到图像降噪的目的，该类中集成了不同种类的滤波器，通过调用它们从而实现图像的平滑、锐化、边界增强等图像降噪操作。常见的降噪滤波器如下表所示：

ImageFilter.BLUR 模糊滤波，即均值滤波

ImageFilter.CONTOUR 轮廓滤波，寻找图像轮廓信息

ImageFilter.DETAIL 细节滤波，使得图像显示更加精细

ImageFilter.FIND\_EDGES 寻找边界滤波（找寻图像的边界信息）

ImageFilter.EMBOSS 浮雕滤波，以浮雕图的形式显示图像

ImageFilter.EDGE\_ENHANCE 边界增强滤波

ImageFilter.EDGE\_ENHANCE\_MORE 深度边缘增强滤波

ImageFilter.SMOOTH 平滑滤波

ImageFilter.SMOOTH\_MORE 深度平滑滤波

ImageFilter.SHARPEN 锐化滤波

ImageFilter.GaussianBlur() 高斯模糊

ImageFilter.UnsharpMask() 反锐化掩码滤波

ImageFilter.Kernel() 卷积核滤波

ImageFilter.MinFilter(size) 最小值滤波器，从 size 参数指定的区域中选择最小像素值，然后将其存储至输出图像中。

ImageFilter.MedianFilter(size) 中值滤波器，从 size 参数指定的区域中选择中值像素值，然后将其存储至输出图像中。

ImageFilter.MaxFilter(size) 最大值滤波器

ImageFilter.ModeFilter() 模式滤波

## 图像转为numpy数组

from PIL import Image

import numpy as np

img = Image.open("C:/Users/Administrator/Desktop/大熊猫.png")

img.show()

#Image图像转换为ndarray数组

img\_2 = np.array(img)

print(img\_2)

#ndarray转换为Image图像

arr\_img = Image.fromarray(img\_2)

#显示图片

arr\_img.show()

#保存图片

arr\_img.save("C:/Users/Administrator/Desktop/arr\_img.png")

## 噪声处理

Matplotlib 是一款用于数据可视化的 Python 软件包，支持跨平台运行，它能够根据 NumPy ndarray 数组来绘制 2D 图像，它使用简单、代码清晰易懂，深受广大技术爱好者喜爱。

pip install matplotlib

opencv

pip install opencv-python

pip install open3d

### 形态学转换

形态变换是一些基于图像形状的简单操作。通常在二进制图像上执行。它需要两个输入，一个是我们的原始图像，第二个是决定\*\*操作性质的结构元素\*\*或\*\*内核\*\*。两种基本的形态学算子是侵蚀和膨胀。然后，它的变体形式（如“打开”，“关闭”，“渐变”等）也开始起作用。

### 侵蚀

侵蚀的基本思想就像土壤侵蚀一样，它侵蚀前景物体的边界(尽量使前景保持白色)。它是做什么的呢?原始图像中的一个像素(无论是1还是0)只有当内核下的所有像素都是1时才被认为是1，否则它就会被侵蚀(变成0)。

结果是，根据内核的大小，边界附近的所有像素都会被丢弃。因此，前景物体的厚度或大小减小，或只是图像中的白色区域减小。它有助于去除小的白色噪声(正如我们在颜色空间章节中看到的)，分离两个连接的对象等。

在这里，作为一个例子，我将使用一个5x5内核，它包含了所有的1。让我们看看它是如何工作的:

import cv2 as cv

import numpy as np

img = cv.imread('j.png',0)

kernel = np.ones((5,5),np.uint8)

erosion = cv.erode(img,kernel,iterations = 1)

### 扩张

它与侵蚀正好相反。如果内核下的至少一个像素为“ 1”，则像素元素为“ 1”。因此，它会增加图像中的白色区域或增加前景对象的大小。通常，在消除噪音的情况下，腐蚀后会膨胀。因为腐蚀会消除白噪声，但也会缩小物体。因此，我们对其进行了扩展。由于噪音消失了，它们不会回来，但是我们的目标区域增加了。在连接对象的损坏部分时也很有用。

dilation = cv.dilate(img,kernel,iterations = 1)

### 开运算

开放只是\*\*侵蚀然后扩张\*\*的另一个名称。如上文所述，它对于消除噪音很有用。在这里，我们使用函数\*\*cv.morphologyEx\*\*()

### 闭运算

闭运算与开运算相反，先扩张然后再侵蚀。在关闭前景对象内部的小孔或对象上的小黑点时很有用。

closing = cv.morphologyEx(img, cv.MORPH\_CLOSE, kernel)

## 边缘检测

Canny Edge Detection是一种流行的边缘检测算法。这是一个多阶段算法，我们将经历每个阶段。

降噪，由于边缘检测容易受到图像中噪声的影响，因此第一步是使用5x5高斯滤波器消除图像中的噪声。

查找图像的强度梯度,然后使用Sobel核在水平和垂直方向上对平滑的图像进行滤波，以在水平方向(Gx)和垂直方向(Gy)上获得一阶导数。

非极大值抑制 在获得梯度大小和方向后，将对图像进行全面扫描，以去除可能不构成边缘的所有不需要的像素。为此，在每个像素处，检查像素是否是其在梯度方向上附近的局部最大值。查看下面的图片：



点A在边缘（垂直方向）上。渐变方向垂直于边缘。点B和C在梯度方向上。因此，将A点与B点和C点进行检查，看是否形成局部最大值。如果是这样，则考虑将其用于下一阶段，否则将其抑制（置为零）。 简而言之，你得到的结果是带有“细边”的二进制图像。

该阶段确定哪些边缘全部是真正的边缘，哪些不是。为此，我们需要两个阈值minVal和maxVal。强度梯度大于maxVal的任何边缘必定是边缘，而小于minVal的那些边缘必定是非边缘，因此将其丢弃。介于这两个阈值之间的对象根据其连通性被分类为边缘或非边缘。如果将它们连接到“边缘”像素，则将它们视为边缘的一部分。否则，它们也将被丢弃。

边缘A在maxVal之上，因此被视为“确定边缘”。尽管边C低于maxVal，但它连接到边A，因此也被视为有效边，我们得到了完整的曲线。但是边缘B尽管在minVal之上并且与边缘C处于同一区域，但是它没有连接到任何“确保边缘”，因此被丢弃。因此，非常重要的一点是我们必须相应地选择minVal和maxVal以获得正确的结果。

## 图像分割

如果我们想从图像的其余部分中提取或定义某些内容, 例如。从背景中检测到物体, 我们可以将图像分解为多个片段, 在其中可以进行更多处理。通常称为分割.

为了进行处理, 我们将使用OTSU的阈值算法, 该算法可消除由于噪声或图像中任何其他不规则性造成的过度分割结果, 并使用OpenCV实现。

Otsu算法之所以称为最大类间方差法，该方法主要是通过阈值进行前后背景分割，而该方法确定最佳阈值的方法是该值使类间方差最大，它是按图像的灰度特性,将图像分成背景和前景两部分，使类间方差最大的分割意味着错分概率最小。

方法：

用一种颜色(或强度)标记我们确定为前景或对象的区域, 用另一种颜色标记我们确定为背景或非对象的区域。

最后, 我们不确定任何区域, 将其标记为0。这就是我们的标记。然后应用分水岭算法。

然后, 我们的标记将使用给定的标签进行更新, 并且对象的边界的值为-1。

让我们从一个示例开始, 考虑硬币图像。

输入图片：



# Python program to transform an image using

# threshold.

import numpy as np

import cv2

from matplotlib import pyplot as plt

# Image operation using thresholding

img = cv2.imread( 'c4.jpg' )

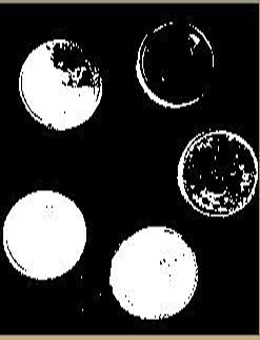
gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(gray, 0 , 255 , cv2.THRESH\_BINARY\_INV +

cv2.THRESH\_OTSU)

cv2.imshow( 'image' , thresh)

输出：



此输出显示图像通过阈值运算进行了变换, 其中前景仍然包含一些噪声。

现在, 我们需要去除图像中所有小的白噪声, 即前景。为此, 我们可以使用形态学封闭。要去除前景对象中的任何小孔, 我们可以使用形态学封闭。为了获得背景, 我们对图像进行了放大。膨胀将对象边界增加到背景。

kernel = np.ones(( 3 , 3 ), np.uint8)

closing = cv2.morphologyEx(thresh, cv2.MORPH\_CLOSE, kernel, iterations = 2 )

# Background area using Dialation

bg = cv2.dilate(closing, kernel, iterations = 1 )

# Finding foreground area

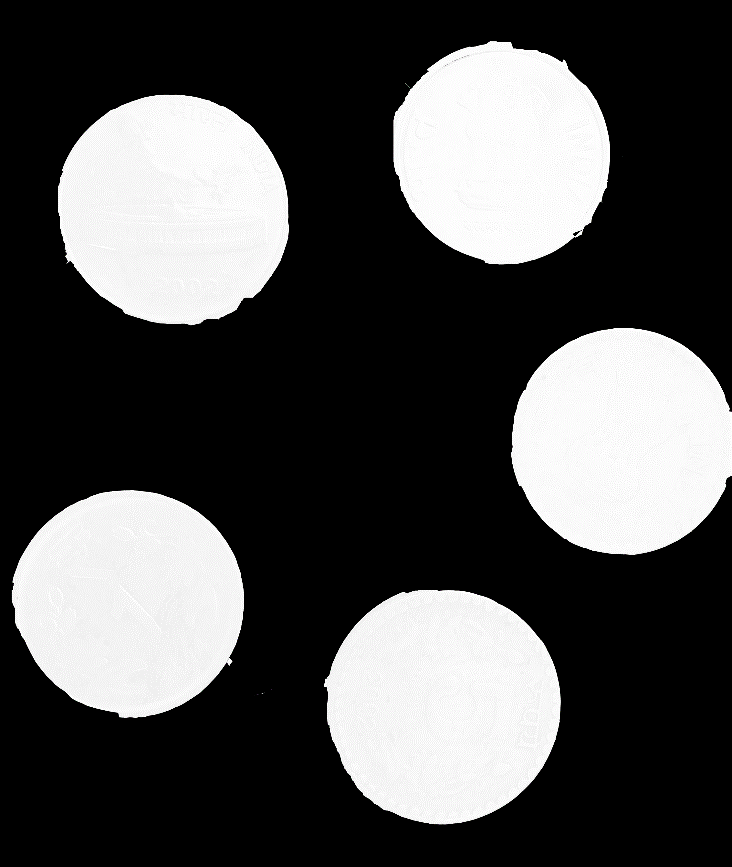
dist\_transform = cv2.distanceTransform(closing, cv2.DIST\_L2, 0 )

ret, fg = cv2.threshold(dist\_transform, 0.02

\* dist\_transform. max (), 255 , 0 )

cv2.imshow( 'image' , fg)

输出：



从输出图像中, 我们可以得出结论, 使用闭合操作可以去除小孔, 并确保结果中前景中的任何区域确实是前景。

# 点云处理

## Open3d

N \* [x,y,z,r,]

<https://github.com/isl-org/Open3D-ML>

Open3D-ML 是 3D 机器学习任务 Open3D 的扩展。它建立在 Open3D 核心库之上，并使用机器学习工具扩展 3D 数据处理。Open3D-ML侧重于语义点云细分等应用，并提供可应用于常见任务的预培训模型以及用于训练的管道。

### 安装

pip install open3d

#### 依赖

PyTorch 1.8.2

TensorFlow 2.5.2

CUDA 10.1, 11.\* (On GNU/Linux x86\_64, optional)

### 数据集

SemanticKITTI (project page)

Toronto 3D (github)

Semantic 3D (project-page)

S3DIS (project-page)

Paris-Lille 3D (project-page)

Argoverse (project-page)

KITTI (project-page)

Lyft (project-page)

nuScenes (project-page)

Waymo (project-page)

ScanNet(project-page)

数据集：<https://blog.csdn.net/Zeal510/article/details/121602764>

SemanticKITTI 由德国波恩大学的研发团队开发的大型户外场景数据集，是自动驾驶领域的权威数据集。它基于 KITTI 数据集，对 KITTI Vision Odometry Benchmark 中的所有序列都进行了标注，同时还为 LiDAR 360 度范围内采集到的所有目标，进行了密集的逐点注释。该数据集包含 28 个标注类别，分为静态对象和动态对象，既包括行人、车辆等交通参与者，也包括停车场、人行道等地面设施。Kitti的点云里程计数据集一共有00-21这22个序列，每个序列都是一段录制的点云包，并且该数据集可做可视化。

### 任务一

读取SemanticKITTI数据集并将其可视化。

import open3d.ml.torch as ml3d # or open3d.ml.tf as ml3d

# 通过指定datasetpath构造数据集

dataset = ml3d.datasets.SemanticKITTI(dataset\_path='/path/to/SemanticKITTI/')

# 获取全部数据集

all\_split = dataset.get\_split('all')

# 打印第一个数据的属性

print(all\_split.get\_attr(0))

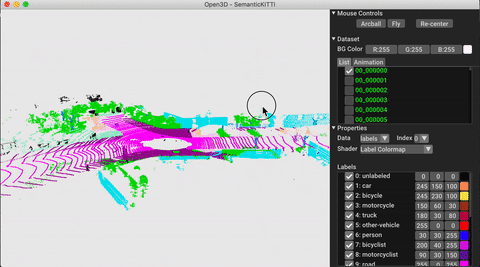
# 打印第一个点云的形状

print(all\_split.get\_data(0)['point'].shape)

# 显示100帧点云

vis = ml3d.vis.Visualizer()

vis.visualize\_dataset(dataset, 'all', indices=range(100))



### 任务二

加载一个配置文件.

import open3d.ml as \_ml3d

import open3d.ml.torch as ml3d # or open3d.ml.tf as ml3d

framework = "torch" # or tf

cfg\_file = "ml3d/configs/randlanet\_semantickitti.yml"

cfg = \_ml3d.utils.Config.load\_from\_file(cfg\_file)

# fetch the classes by the name

Pipeline = \_ml3d.utils.get\_module("pipeline", cfg.pipeline.name, framework)

Model = \_ml3d.utils.get\_module("model", cfg.model.name, framework)

Dataset = \_ml3d.utils.get\_module("dataset", cfg.dataset.name)

# use the arguments in the config file to construct the instances

cfg.dataset['dataset\_path'] = "/path/to/your/dataset"

dataset = Dataset(cfg.dataset.pop('dataset\_path', None), \*\*cfg.dataset)

model = Model(\*\*cfg.model)

pipeline = Pipeline(model, dataset, \*\*cfg.pipeline)

任务三

语义分割

在前一个示例的基础上，我们可以实例化一个带有用于语义分割的预训练模型的管道，并在数据集的点云上运行它。

import os

import open3d.ml as \_ml3d

import open3d.ml.torch as ml3d

cfg\_file = "ml3d/configs/randlanet\_semantickitti.yml"

cfg = \_ml3d.utils.Config.load\_from\_file(cfg\_file)

model = ml3d.models.RandLANet(\*\*cfg.model)

cfg.dataset['dataset\_path'] = "/path/to/your/dataset"

dataset = ml3d.datasets.SemanticKITTI(cfg.dataset.pop('dataset\_path', None), \*\*cfg.dataset)

pipeline = ml3d.pipelines.SemanticSegmentation(model, dataset=dataset, device="gpu", \*\*cfg.pipeline)

# download the weights.

ckpt\_folder = "./logs/"

os.makedirs(ckpt\_folder, exist\_ok=True)

ckpt\_path = ckpt\_folder + "randlanet\_semantickitti\_202201071330utc.pth"

randlanet\_url = "https://storage.googleapis.com/open3d-releases/model-zoo/randlanet\_semantickitti\_202201071330utc.pth"

if not os.path.exists(ckpt\_path):

cmd = "wget {} -O {}".format(randlanet\_url, ckpt\_path)

os.system(cmd)

# load the parameters.

pipeline.load\_ckpt(ckpt\_path=ckpt\_path)

test\_split = dataset.get\_split("test")

data = test\_split.get\_data(0)

# run inference on a single example.

# returns dict with 'predict\_labels' and 'predict\_scores'.

result = pipeline.run\_inference(data)

# evaluate performance on the test set; this will write logs to './logs'.

pipeline.run\_test()

训练一个语义分割模型

# use a cache for storing the results of the preprocessing (default path is './logs/cache')

dataset = ml3d.datasets.SemanticKITTI(dataset\_path='/path/to/SemanticKITTI/', use\_cache=True)

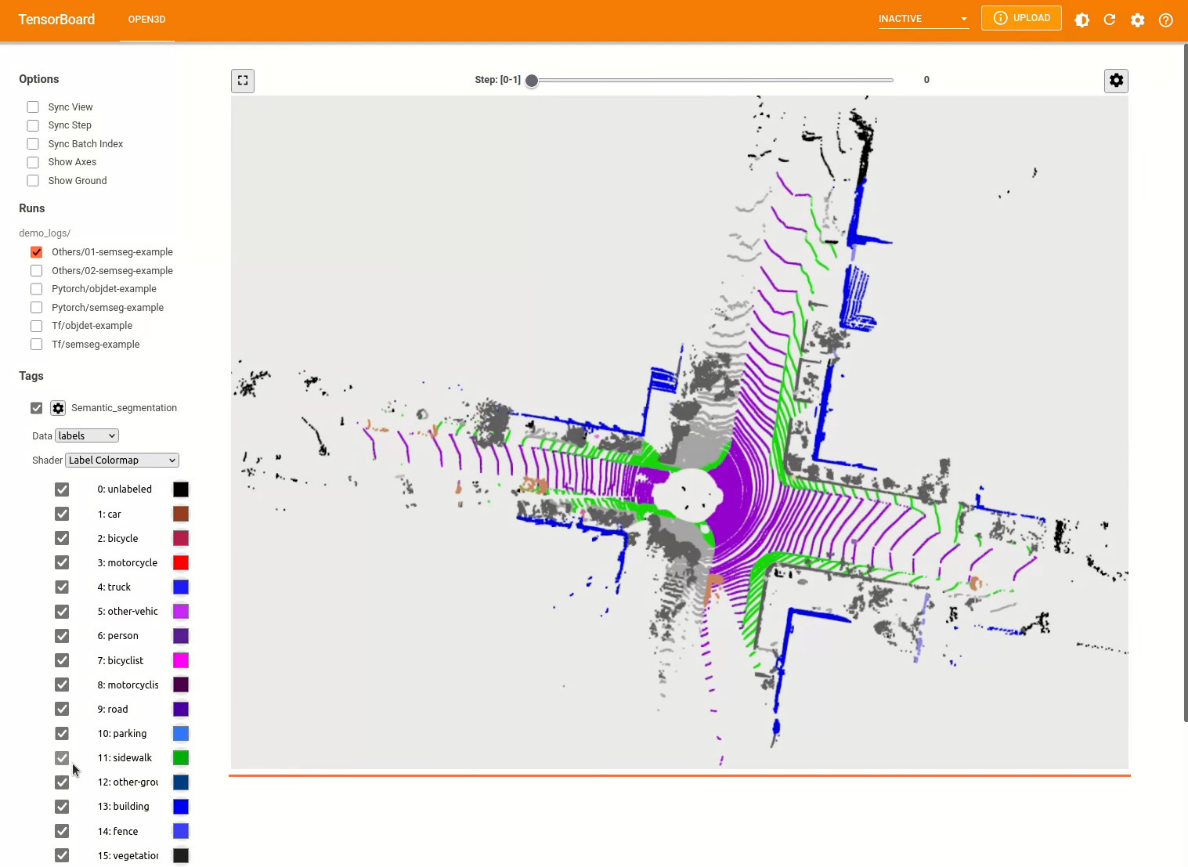
# create the model with random initialization.

model = RandLANet()

pipeline = SemanticSegmentation(model=model, dataset=dataset, max\_epoch=100)

# prints training progress in the console.

pipeline.run\_train()



### 任务三

3d目标检测

3D对象检测模型类似于语义分割模型。我们可以使用对象检测的预训练模型实例化管道，并在数据集的点云上运行它。请参见模型动物园以获取预训练模型的权重。

import os

import open3d.ml as \_ml3d

import open3d.ml.torch as ml3d

cfg\_file = "ml3d/configs/pointpillars\_kitti.yml"

cfg = \_ml3d.utils.Config.load\_from\_file(cfg\_file)

model = ml3d.models.PointPillars(\*\*cfg.model)

cfg.dataset['dataset\_path'] = "/path/to/your/dataset"

dataset = ml3d.datasets.KITTI(cfg.dataset.pop('dataset\_path', None), \*\*cfg.dataset)

pipeline = ml3d.pipelines.ObjectDetection(model, dataset=dataset, device="gpu", \*\*cfg.pipeline)

# download the weights.

ckpt\_folder = "./logs/"

os.makedirs(ckpt\_folder, exist\_ok=True)

ckpt\_path = ckpt\_folder + "pointpillars\_kitti\_202012221652utc.pth"

pointpillar\_url = "https://storage.googleapis.com/open3d-releases/model-zoo/pointpillars\_kitti\_202012221652utc.pth"

if not os.path.exists(ckpt\_path):

cmd = "wget {} -O {}".format(pointpillar\_url, ckpt\_path)

os.system(cmd)

# load the parameters.

pipeline.load\_ckpt(ckpt\_path=ckpt\_path)

test\_split = dataset.get\_split("test")

data = test\_split.get\_data(0)

# run inference on a single example.

# returns dict with 'predict\_labels' and 'predict\_scores'.

result = pipeline.run\_inference(data)

# evaluate performance on the test set; this will write logs to './logs'.

pipeline.run\_test()

训练模型

# use a cache for storing the results of the preprocessing (default path is './logs/cache')

dataset = ml3d.datasets.KITTI(dataset\_path='/path/to/KITTI/', use\_cache=True)

# create the model with random initialization.

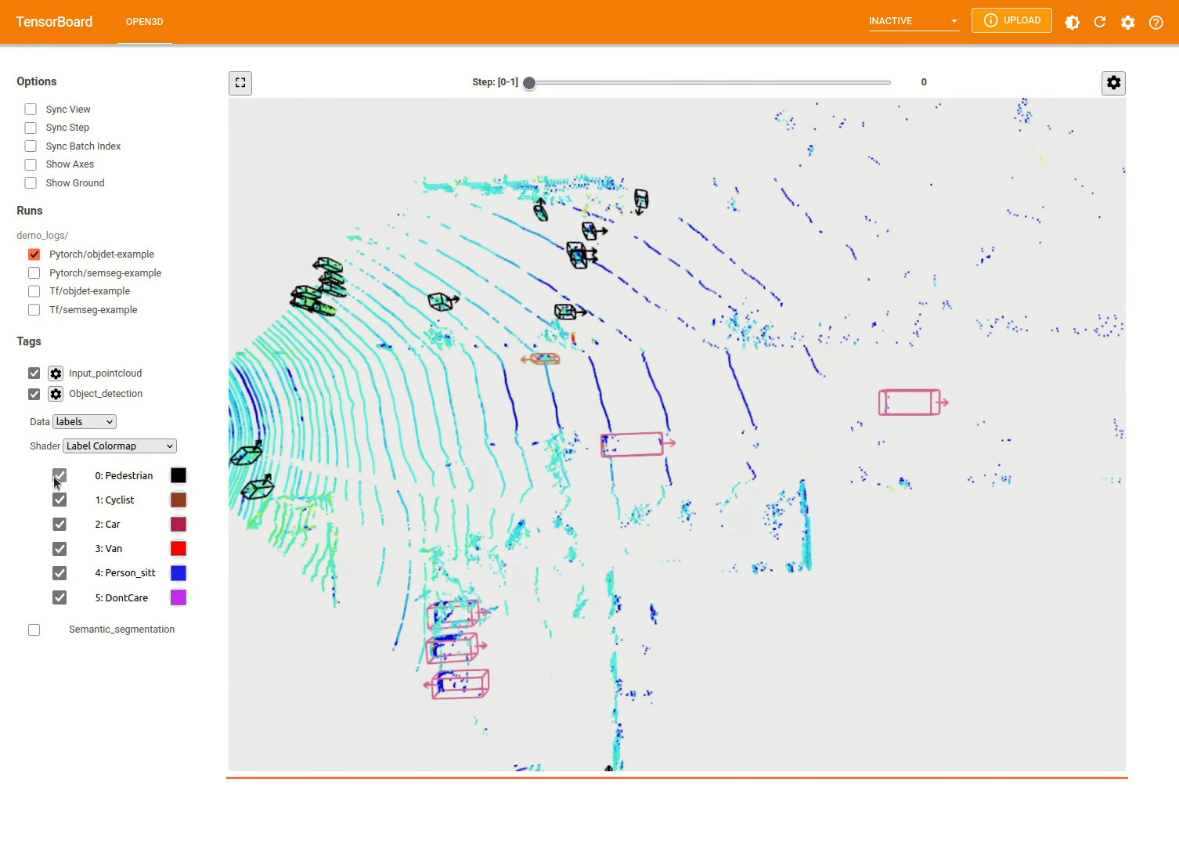
model = PointPillars()

pipeline = ObjectDetection(model=model, dataset=dataset, max\_epoch=100)

# prints training progress in the console.

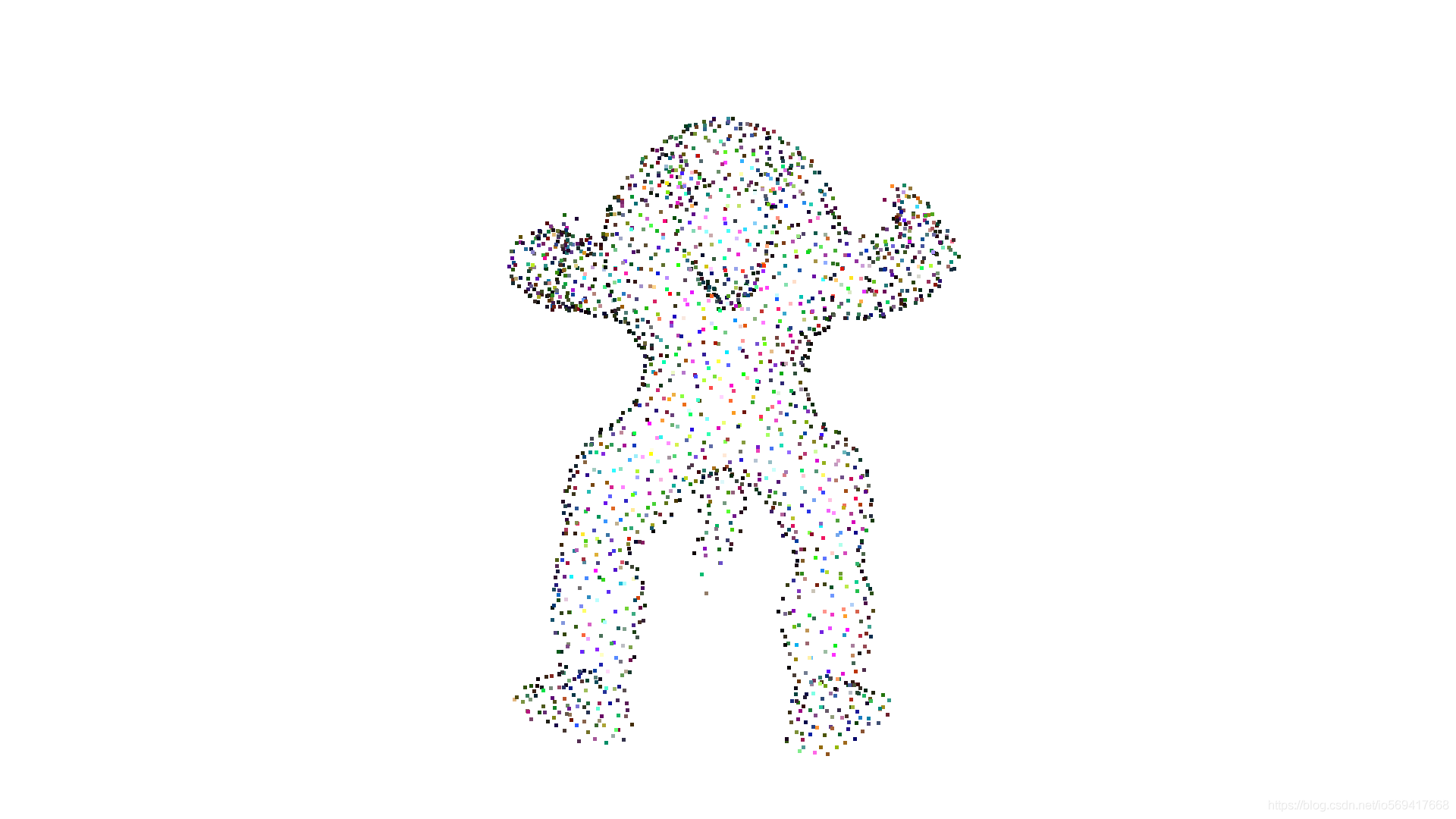
pipeline.run\_train()

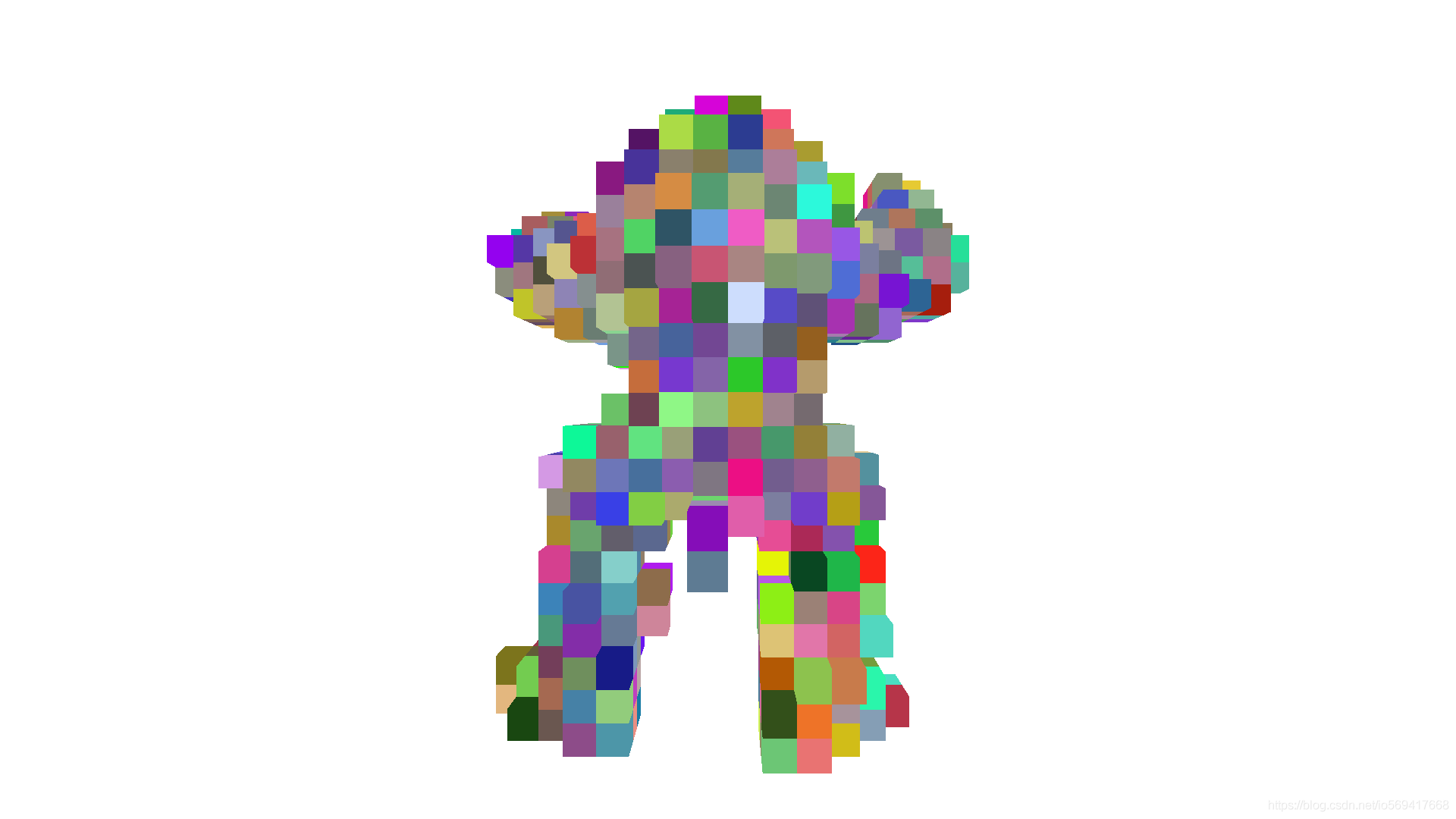
结果



## 点云的体素化

使用create\_from\_point\_cloud函数从点云中生成体素网格。如果点云中至少有一个点在体素网格内，则该网格被占用。颜色表示的是该体素中点的平均值。参数voxel\_size用来定义网格分辨率。





N = 2000

pcd = o3dtut.get\_armadillo\_mesh().sample\_points\_poisson\_disk(N)

# fit to unit cube

pcd.scale(1 / np.max(pcd.get\_max\_bound() - pcd.get\_min\_bound()), center=pcd.get\_center())

pcd.colors = o3d.utility.Vector3dVector(np.random.uniform(0,1,size=(N,3)))

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

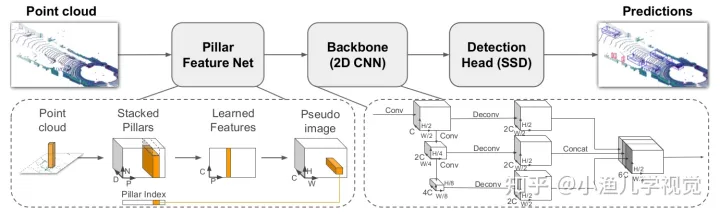
voxel\_grid = o3d.geometry.VoxelGrid.create\_from\_point\_cloud(pcd, voxel\_size=0.05)

o3d.visualization.draw\_geometries([voxel\_grid])

## 点云的pillars化

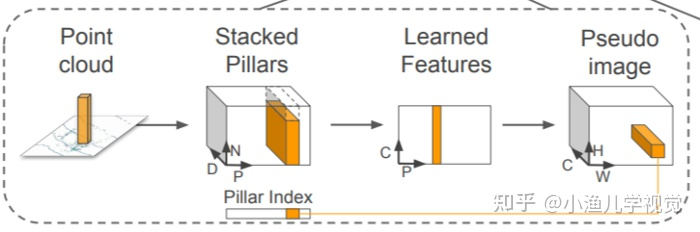
《PointPillars: Fast Encoders for Object Detection from Point Clouds》是一篇发表在CVPR 2019上关于激光点云3D目标检测的文章，其中提出了一种新的点云编码方法用于给PointNet提取点云特征，再将提取的特征映射为2D伪图像以便用2D目标检测的方式进行目标检测。

PointPillars的网络结构如下图所示：

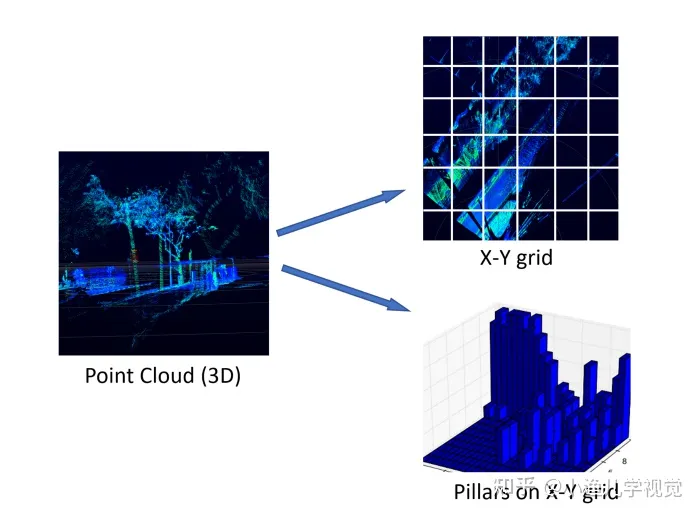


第一部分：Pillar Feature Net (PFN)

该部分网络结构的作用是将点云转换为伪图像，转换过程的示意图如下：



首先，输入点云被分割成多个Pillar单元，每个Pillar是在XY平面上（笛卡尔坐标系）以一定的步长对点云进行划分得到的一个3维的小单元格。



然后每个Pillar中的每个点云被编码成一个9维的向量D：(x,y,z,r,xc,yc,zc,xp,yp)，其中x,y,z,r分别表示点云在三维空间中的3个坐标和反射强度；xc,yc,zc表示到该Pillar中所有点的算术平均值点的距离，xp,yp表示该点到该Pillar的x,y中心的偏移值。由于点云数据的稀疏性，可能很多Pillar都不含点云或者包含的点云数量比较少，考虑到计算复杂度的问题，会对Pillar的数量进行限制，最多处理P个非空的Pillar，同时每个Pillar中最多包含N个点云特征向量，如果点云数大于N，则采用随机采样的方法从中选取N个，反之，如果点云的数量少于N，则用零填充的方法填充到N个。通过上述方法，就将一帧点云数据编码成了一个维度为(D,P,N)的稠密张量。

接下来，用一个简化版的PointNet网络进行处理。首先将每个包含D维特征的点用一个线性层+BatchNorm+ReLU激活函数处理后，生成维度为(C,P,N)的张量；然后对每个Pillar单元进行最大池化操作，得到维度为(C,P)的张量。

最后一步是通过一个scatter算子生成伪图像。这一步的方法比较简单，就是通过每个点的Pillar索引值将上一步生成的(C,P)张量转换回其原始的Pillar坐标用来创建大小为(C,H,W)的伪图像。这里需要解释一下伪图像的高度H和宽度W是怎么来的：在第一步对点云进行Pillar划分的时候会设置XY平面上点云坐标的范围和每个Pillar的大小，假设X轴的范围是[0,69.12]，Y轴的范围是[-39.68,39.68]，每个Pillar的大小是``0.16x0.16，那么以X轴表示宽，Y轴表示高，一个Pillar表示一个像素的话，那么这个伪图像的宽W = (69.12 - 0) / 0.16 = 432，高H = (39.68 -(-39.68)) / 0.16 = 496。

## 点云的滤波

半径滤波与统计滤波器类似，根据空间点半径范围中临近点数量是否满足给定值来滤波。该滤波算法比统计滤波更加简单，计算速度更快。

在点云数据中以某点为中心画一个圆计算落在该圆中点的数量，当数量大于给定值时，则保留该点，数量小于给定值则剔除该点。此算法运行速度快，依序迭代留下的点一定是最密集的，但是圆的半径和圆内点的数目都需要人工指定。

open3d实现半径滤波：

import open3d as o3d

import numpy as np

pcd = o3d.io.read\_point\_cloud('013205.pcd',remove\_nan\_points = True,remove\_infinite\_points = True)

print('原点云个数为：',np.array(pcd.points).shape[0])

o3d.visualization.draw\_geometries([pcd],window\_name = '源数据',width = 1080,height = 720,top = 30,left = 20)

cl,index = pcd.remove\_radius\_outlier(nb\_points=10,radius=0.2)

new\_pcd = pcd.select\_by\_index(index)

print('半径滤波后的点云个数为：',np.array(new\_pcd.points).shape[0])

o3d.visualization.draw\_geometries([new\_pcd],window\_name= '半径滤波',width = 1080,height = 720,top = 20,left = 30)

一、无穷值和非数点的剔除

该功能可筛选出点云中坐标值为以下两种类型的点：

坐标值为NaN(not a number，非数)的点，通常为未定义或不可表示的坐标值点。

坐标值为Infinite(无穷大)的点，包含正向和负向的无穷大坐标值点。

import open3d as o3d

pcd = o3d.io.read\_point\_cloud(path) # path为文件路径

pcd\_new = o3d.geometry.PointCloud.remove\_non\_finite\_points(

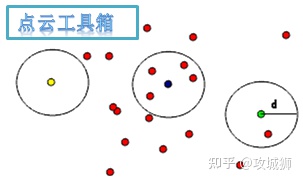
pcd, remove\_nan = True, remove\_infinite = False)

# remove\_nan和remove\_infinite参数均包含两个类型值，True和False

# 分别为剔除非数点和无穷值点

二、孤点滤波

此处孤点滤波就是半径滤波，在点云数据中，指定每个的点一定范围内周围至少要有足够多的近邻。例如，如果指定至少要有1个邻居，只有黄色的点会被删除，如果指定至少要有2个邻居，黄色和绿色的点都将被删除。



pcd\_new = o3d.geometry.PointCloud.remove\_radius\_outlier(pcd, knn, radius)

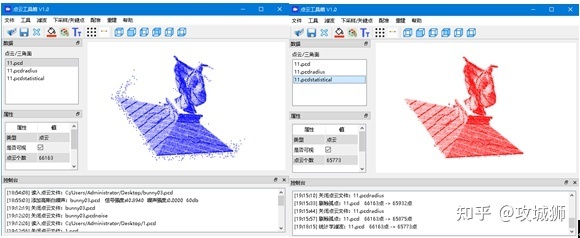
# knn参数为选择邻近点个数，radius参数为设置的半径

三、统计学滤波

统计学滤波用于去除明显离群点。离群点特征是在空间中分布稀疏，考虑到离群点的特征，则可以定义某处点云小于某个密度，既点云无效。对每个点的邻域进行一个统计分析，并修剪掉一些不符合标准的点。具体方法为在输入数据中对点到临近点的距离分布的计算，对每一个点，计算它到所有临近点的平均距离（假设得到的结果是一个高斯分布，其形状是由均值和标准差决定），那么平均距离在标准范围之外的点，可以被定义为离群点并从数据中去除。特点：主要是根据密度去除离群点，对密度差异较大的离群点去除效果较好。

pcd\_new = o3d.geometry.PointCloud.remove\_statistical\_outlier(pcd, knn, std)

# knn参数为选择邻近点个数，std参数为设置的标准差阈值



四、RANSAC分割平面

RANSAC(Random Sample Consensus)是根据一组包含异常数据的样本数据集，计算出数据的数学模型参数，得到有效样本数据的算法。

RANSAC算法的基本假设是样本中包含正确数据(inliers，可以被模型描述的数据)，也包含异常数据(outliers，偏离正常范围很远、无法适应数学模型的数据)，即数据集中含有噪声。这些异常数据可能是由于错误的测量、错误的假设、错误的计算等产生的。同时RANSAC也假设，给定一组正确的数据，存在可以计算出符合这些数据的模型参数的方法。

在拟合平面（地面）这一需求上，平面的凹凸点（小的坑洼）是有效数据，但对所需平面来说有一定的偏移。而大的凹凸，比如地面上的障碍物、地面的深坑，这些都是偏移量过大的无效数据。最小二乘拟合，旨在照顾所有人的想法，对所有数据进行拟合，在无效数据多且偏移量大的情况下，拟合效果不好。而RANSAC拟合，旨在照顾多数人的意愿，对主体数据进行拟合，手动设置一个阈值，同拟合平面的距离超过阈值的点，就被判定为无效数据。随机拟合多个平面，选取平面内数据点最多的平面，或者说，无效数据最少的平面，作为拟合出的结果。根据如上思路，RANSAC在拟合平面这一需求上，可以得到更准确的结果。

plane\_model, inliers = pcd.segment\_plane(distance\_threshold=dis, ransac\_n=rnn, num\_iterations=n)

# distance\_threshold为距离阈值参数，ransac\_n为RANSAC迭代的点数，num\_iterations为最大迭代次数

pcd\_in = pcd.select\_by\_index(inliers) # RANSAC分割后的内部点云(拟合平面点)

pcd\_out = pcd.select\_by\_index(inliers, invert=True) # RANSAC分割后的外部点云(拟合平面之外的点)

