这次作业要求我们准备4分类的数据集，训练模型，然后再进行测试。

训练流程如下：

1. 从KITTI的3d目标检测数据集中整理出4分类的数据集
2. 撰写所需的data transform
3. 训练一个PointNet或PointNet++

测试流程如下：

1. 把KITTI数据集点云中的地面移除
2. 做聚类
3. 忽略太大或太小的点云
4. 利用PointNet或PointNet++进行分类
5. 将神经网络的输出转成KITTI的格式

个人觉得本次作业中最难的部分是如何从KITTI数据集中整理出4分类的数据集，以及将神经网络的输出转成KITTI格式，因为网上很少这部分的讨论，最后我是去读了https://github.com/sshaoshuai/PointRCNN里相关的源码才做出来的。

训练的第一步是“从KITTI的3d目标检测数据集中整理出4分类的数据集”，需要读取点云及标签（三维的框）。

读取点云的代码是参考`kitti_rcnn_dataset.py`这个文件，里面有以下几句：

```
calib = self.get_calib(sample_id)
img_shape = self.get_image_shape(sample_id)
pts_lidar = self.get_lidar(sample_id)
pts_rect = calib.lidar_to_rect(pts_lidar[:, 0:3])
pts_intensity = pts_lidar[:, 3]
```

可以看到它不止读取了点云，还读取了calibration file，然后调用`calib.lidar_to_rect`对进行点云做校正。

下面是我改写过的代码：

```
calib = get_calib(calib_dir, sample_id)
pts_lidar = get_lidar(cloud_dir, sample_id)
pts_rect = calib.lidar_to_rect(pts_lidar[:, 0:3])
```

本来`get_calib`及`get_lidar`是`KittiRCNNDataset`类的成员函数，所以可以直接使用`calib_dir`及`cloud_dir`这两个成员变量。我这里则是直接把`get_calib`及`get_lidar`这两个函数搬出来作为独立的函数，然后为他们加上必要的参数（`calib_dir`及`cloud_dir`）。

读取标签的代码是参考自`get_proposal_from_file`：

```
gt_obj_list = self.filtrate_objects(self.get_label(sample_id))
gt_boxes3d = kitti_utils.objs_to_boxes3d(gt_obj_list)
gt_corners = kitti_utils.boxes3d_to_corners3d(gt_boxes3d)
```

我将它改写成：

```
gt_obj_list = filtrate_objects(classes, get_label(label_dir, sample_id))
boxes3d = objs_to_boxes3d(gt_obj_list)
corners3d = boxes3d_to_corners3d(boxes3d)
```

有了点云及标签后，下一步是实际把点云中被框住的点取出来，作为等会训练PointNet的样本。这一步在`kitti_rcnn_dataset.py`的`get_rcnn_sample_info`函数中已经有实现了。但这件事我是在作业完成后经助教提醒才发现的，所以下面还是分享我原来的做法：

先将点云及`corners3d`投影到水平面上，然后判断每个点是否落在长方形内，对于落在长方形内的点，再判断它们在高度方向上是否满足要求。其中"判断每个点是否落在长方形内"的代码是参考自https://stackoverflow.com/questions/21339448/how-to-get-list-of-points-inside-a-polygon-in-python。

```
corner2d = corner3d[:,[0,2]]
corner2d = corner2d[:4]
ymin = corner3d[:,1].min()
ymax = corner3d[:,1].max()
p = Path(corner2d) # make a polygon
in_rect_2d = p.contains_points(pts_rect[:,[0,2]])
in_hull = np.logical_and(in_rect_2d, pts_rect[:,1]>=ymin)
in_hull = np.logical_and(in_hull, pts_rect[:,1]<=ymax)
```

注意"把`corners3d`投影到水平面上"这部分，我用的是`corner2d = corner3d[:,[0,2]]`，这代表保留长方体的x,z坐标，忽略y坐标。这是因为在KITTI数据集中，y方向才是高度方向。

最后得到的`in_hull`是一个bool array，可以判断点云中的哪些点被`corner3d`包围。

有了各类物体的点云后，接下来是把它们切成训练集及测试集。

```
val_size = int(min(map(len, class2samples.values())) * 0.2)

for _class in range(4):
    print("class", _class)
    samples = class2samples[_class]
    full_size = len(samples)

    train_size = full_size - val_size

    random.shuffle(samples)

    samples = ["data_object_box/"+sample[:-4]+"\n" for sample in samples]

    with open("kitti_train.txt", "a") as f:
        f.writelines(samples[:train_size])

    with open("kitti_val.txt", "a") as f:
        f.writelines(samples[train_size:])
```

`class2samples`是一个dict，key是0,1,2,3等四个类别，value是属于该类别的文件名称的list。上面的代码就是将各类别都依照8:2的比例来切成训练集及测试集。

假设我们已经有了数据集跟模型（第五章的作业），在把数据丢进模型内前，还需要做transformation。

首先是坐标系转换：在原来KITTI的坐标系中，高度方向为y方向，这里将它转成以z方向为高度方向：

```
class RotateFromY(object):
    def __init__(self):
        pass

    def __call__(self, sample):
        cloud, _class = sample['cloud'], sample['class']
        cloud[:,1], cloud[:,2] = cloud[:,2], -cloud[:,1]
        return {'cloud': cloud, 'class': _class}
```

然后是Input Dropout：将输入点云随机下采样成1000个点：

```
class InputDropout(object):
    """
    InputDropout
    """

    def __init__(self, ts = 1000):
        self.ts_ = ts

    def __call__(self, sample):
        cloud, _class, _id = sample['cloud'], sample['class'], sample['id']
        cloud = cloud[np.random.choice(cloud.shape[0], self.ts_,
replace=False), :]
        return {'cloud': cloud, 'class': _class, 'id': _id}
```

然后是做normalize，这一步是常规操作：

```
class Normalize(object):
    """
    normalize to [-0.5, 0.5]
    """

    def __init__(self):
        pass

    def __call__(self, sample):
        cloud, _class, _id = sample['cloud'], sample['class'], sample['id']
        cloud = np.reshape(cloud, (-1,3))
        lower = np.min(cloud, axis=0)
        upper = np.max(cloud, axis=0)
        center = (lower+upper)/2.0
        # move to (0,0,0)
        cloud = cloud - center
        # resize to (-0.5, 0.5)
        ratio = 1.0/(upper - lower).max()
        cloud = cloud * ratio
        return {'cloud': cloud, 'class': _class, 'id': _id}
```

再来是数据增强，包括沿着z轴旋转任意角度，以及添加高斯噪声。以下是让点云沿着z轴旋转任意角度的代码：

```
class RandomRotateOverZ(object):
    def __init__(self):
        theta = np.random.uniform(-np.pi,np.pi)
        ux, uy, uz = 0, 0, 1
        cost = np.cos(theta)
        sint = np.sin(theta)

 #https://en.wikipedia.org/wiki/Rotation_matrix#Rotation_matrix_from_axis_and_angle
        self.rot_mat = np.matrix([
          [cost+ux*ux*(1-cost),  ux*uy*(1-cost)-uz*sint, ux*uz*(1-cost)+uy*sint],
          [uy*ux*(1-cost)+uz*sint, cost+uy*uy*(1-cost),  uy*uz*(1-cost)-ux*sint],
          [uz*ux*(1-cost)-uy*sint, uz*uy*(1-cost)+ux*sint, cost+uz*uz*(1-cost)]])

    def __call__(self, sample):
        cloud, _class, _id = sample['cloud'], sample['class'], sample['id']
        cloud = np.matmul(self.rot_mat, cloud.T)
        cloud = cloud.T
        return {'cloud': cloud, 'class': _class, 'id': _id}
```

还有添加高斯噪声的代码：

```
class AddGaussianNoise(object):
    def __init__(self, amp = 0.1):
        self.amp_ = amp

    def __call__(self, sample):
        cloud, _class, _id = sample['cloud'], sample['class'], sample['id']
        cloud = cloud +
np.random.normal(loc=0,scale=self.amp_,size=cloud.shape)
        return {'cloud': cloud, 'class': _class, 'id': _id}
```

在准备数据集的时候应该有注意到，车的数量远多于其他类别，约为3000多，最少的是行人，只有三十几个样本。为了处理这种类别不均衡的问题，有一种方法是在训练时对各类别使用不同的权重做采样，代码如下：

```
def make_weights_for_balanced_classes(classes, nclasses):
    count = [0] * nclasses
    for _class in classes:
        count[_class] += 1
    weight_per_class = [0.] * nclasses
    N = float(sum(count))
    for i in range(nclasses):
        weight_per_class[i] = N/float(count[i])
    weight = [0] * len(classes)
    for idx, _class in enumerate(classes):
        weight[idx] = weight_per_class[_class]
    return weight

weights = make_weights_for_balanced_classes(train_dataset.classes,
                                                    len(train_dataset.classnames))
weights = torch.FloatTensor(weights)
sampler = torch.utils.data.sampler.WeightedRandomSampler(weights,
len(weights))

train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=batch_size, num_workers=nworkers,
    sampler = sampler, pin_memory=True)
```

这里的代码是参考https://discuss.pytorch.org/t/balanced-sampling-between-classes-with-torchvision-dataloader/2703/3，核心思想是让样本数较多的类别有较低的权重，使得模型在训练时能均匀地看到各类别的样本。

以上的准备工作做完后，就可以开始训练模型了。

模型训练完成后，开始接下来测试的流程。测试的前两步是移除地面及聚类，由于这是前几次作业的内容，这边就不再赘述。得到聚类出来的点云后，可以先设定一些规则，排除不可能是车，人或骑单车的人其中之一的点云。

我用的规则是排除点数少于某一个阈值的点云，还有排除长，宽或高大于3米的点云：

```
def check_valid(cloud):
    if cloud.shape[0] < cloud_size_thres:
        return False
    extent = np.max(cloud,axis=0)-np.min(cloud,axis=0)
    if np.max(extent) > 3:
        return False
    return True
```

我们的训练数据是已经转为rect坐标系的点云，所以在测试时，也要将从bin文件里读取的点云转为rect坐标系才能送进神经网络。

```
pts_rect = calib.lidar_to_rect(pts_lidar[:, 0:3])
```

另外，在训练时，我们是将每个样本的点数都下采样到1000；在测试时，我们可以将batch size设为1，然后针对每个样本，将PointNet的pool层的kernel size设为该样本的点数。如此一来，就能把整个点云送进神经网络进行推论而不需要进行下采样，理论上应该会有较好的效果。下面是用于推论的代码：

```
pn.pool = nn.MaxPool1d(pts_rect.shape[0])
classid, score = evaluate_one(pn, pts_rect)
```

当中调用到的`evaluate_one`函数：

```
def evaluate_one(pn, cloud, device=torch.device('cpu')):
    sample = {'cloud': cloud, 'class': -1, 'id': -1}
    transform = torchvision.transforms.Compose([
        # InputDropout(),
        Normalize(),
        RotateFromY(),
        # ToTensor()
        ])
    sample = transform(sample)
    points = sample["cloud"][np.newaxis,...]
    points = torch.from_numpy(points)
    points = points.float().to(device)
    with torch.no_grad():
        pred = pn(points)
    pred = pred.cpu().detach().numpy()[0] #batch size is 1
    pred_choice = np.argmax(pred)
    # because we are using np.log_softmax in PointNet
    # so here we need to use np.exp to convert it back to score
    score = np.exp(pred[pred_choice])
    # class, score
    return pred_choice, score
```

最后一步是将神经网络的输出转成KITTI的格式，这里我是参考https://github.com/sshaoshuai/PointRCNN/blob/master/tools/eval_rcnn.py里的写法，先准备`save_kitti_format`函数所需要的参数，然后再调用`save_kitti_format`函数。

以下是我改写过的`save_kitti_format`函数：

```
def save_kitti_format(classes, sample_id, calib, bbox3d, kitti_output_dir, scores,
img_shape):
    corners3d = boxes3d_to_corners3d(bbox3d)
    img_boxes, _ = calib.corners3d_to_img_boxes(corners3d)
    img_boxes[:, 0] = np.clip(img_boxes[:, 0], 0, img_shape[1] - 1)
    img_boxes[:, 1] = np.clip(img_boxes[:, 1], 0, img_shape[0] - 1)
    img_boxes[:, 2] = np.clip(img_boxes[:, 2], 0, img_shape[1] - 1)
    img_boxes[:, 3] = np.clip(img_boxes[:, 3], 0, img_shape[0] - 1)
    img_boxes_w = img_boxes[:, 2] - img_boxes[:, 0]
    img_boxes_h = img_boxes[:, 3] - img_boxes[:, 1]
    box_valid_mask = np.logical_and(img_boxes_w < img_shape[1] * 0.8, img_boxes_h <
img_shape[0] * 0.8)
    kitti_output_file = os.path.join(kitti_output_dir, '%06d.txt' % sample_id)
    with open(kitti_output_file, 'w') as f:
        for k in range(bbox3d.shape[0]):
            if box_valid_mask[k] == 0:
                continue
            x, z, ry = bbox3d[k, 0], bbox3d[k, 2], bbox3d[k, 6]
            beta = np.arctan2(z, x)
            alpha = -np.sign(beta) * np.pi / 2 + beta + ry
            print('%s -1 -1 %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f
%.4f' %
                  (classes[k], alpha, img_boxes[k, 0], img_boxes[k, 1], img_boxes[k, 2],
img_boxes[k, 3],
                   bbox3d[k, 3], bbox3d[k, 4], bbox3d[k, 5], bbox3d[k, 0], bbox3d[k, 1],
bbox3d[k, 2],
                   bbox3d[k, 6], scores[k]), file=f)
```

它的参数有`classes`, `sample_id`, `calib`, `boxes3d`,`kitti_output_dir`, `scores`,`image_shape`。其中`sample_id`,`calib`,`kitti_output_dir`,`image_shape`比较简单。

`classes`是将各cluster的类别收集到一个list里，`scores`亦然。

```
#classid, score = evaluate_one(pn, pts_lidar)
classes.append(CLASSES[classid])
scores.append(score)
```

接下来是准备`boxes3d`，它是由`location`,`dimensions`及`rotation_y`所组成。`location`表示bounding box的中心点。`dimensions`表示bounding box的height, width及length，这里同样要注意：代表高度方向的是y方向。`rotation_y`则设为0，表示预测出来的bounding box没有沿y轴旋转。

```
location = (np.max(pts_rect, axis=0)+np.min(pts_rect, axis=0))/2
dimensions = np.max(pts_rect, axis=0)-np.min(pts_rect, axis=0)
# (h,w,l) <-> (y,z,x)
dimensions = dimensions[[1,2,0]]
rotation_y = 0.0
boxes3d.append(location.tolist()+dimensions.tolist()+[rotation_y])
```

准备好所需的参数后，就可以调用`save_kitti_format`来储存目标检测结果了：

```
save_kitti_format(classes, sample_id, calib, boxes3d, result_dir, scores,
                  image_shape)
```