CenterPoint论文和代码解析 - 知乎

> 🔗 原文链接： https://zhuanlan.zhihu.com/p/584993...
>
> ⏰ 剪存时间：2023-11-26 12:26:49 (UTC+8)
>
> ✂ 本文档由 飞书剪存 一键生成

论文： **https:// arxiv.org/pdf/2006.1127 5.pdf**

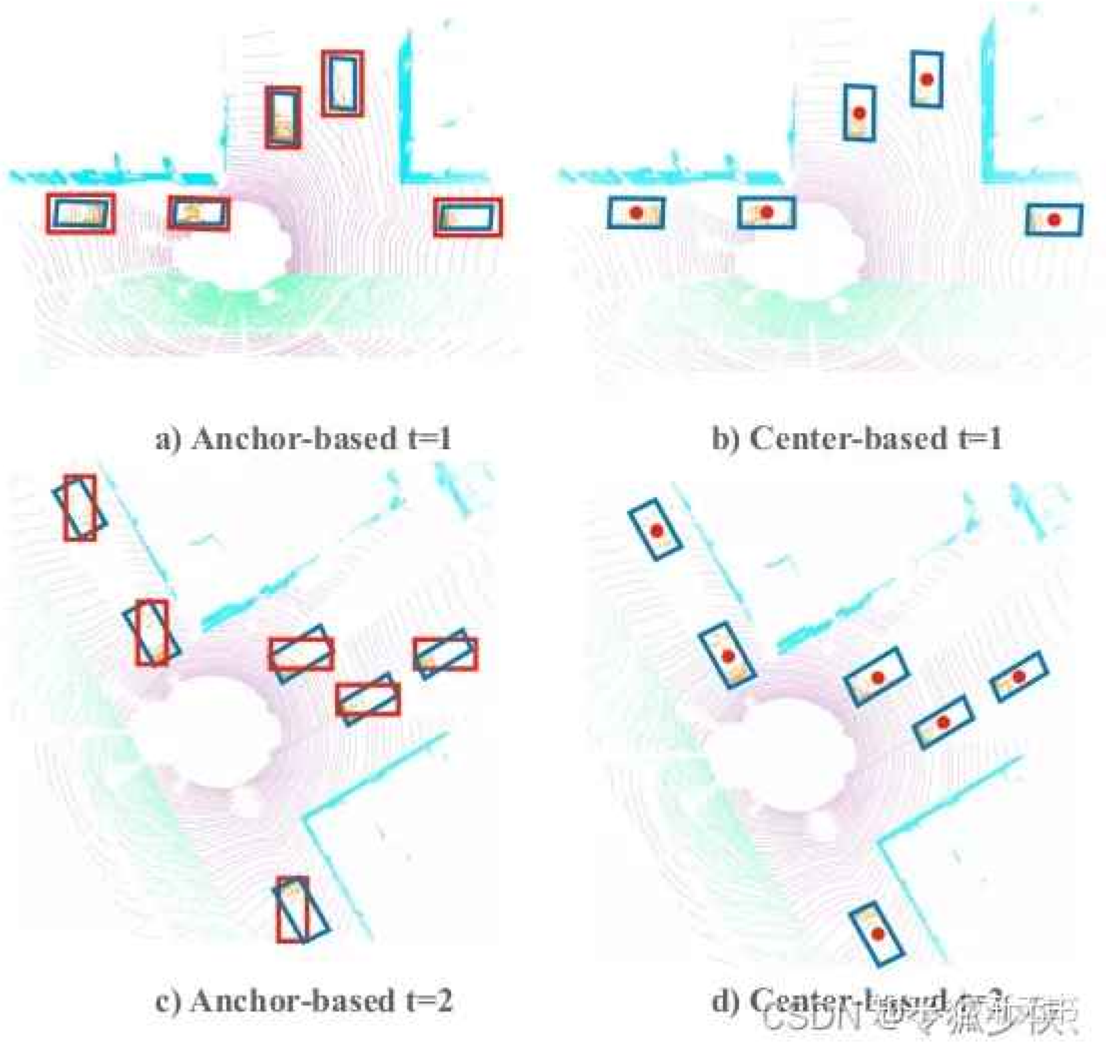
CenterPoint代码： **https:// github.com/tianweiy/Cen terPoint**

OpenPCDet代码： **https:// github.com/open-mmlab/O penPCDet/**

简介

基于 `anchor` 的检测器难以枚举所有方向或将轴对齐的边界框拟合到旋转对象， `CenterPoint` 提出了一种基于中心的基于激光雷达点云的三维目标检测与跟踪框架，首先使用关键点检测器检测对象的中心，然后回归其他属性，包括 3D 尺寸、3D 方向和速度。在第二阶段，它使用目标上的额外点特征来改进这些估计。 `CenterPoint` 简单，接近实时，在 `Waymo` 和 `nuScenes` 基准测试中实现了最先进的性能。

`CenterPoint` 使用标准的基于 `Lidar` 的骨干网，即 `VoxelNet` 或 `PointPillars` ，来构建输入点云的表示。 然后，它将这种表示平铺到一个BEV视图中，并使用基于标准图像的关键点检测器来寻找目标中心。 对于每个检测到的中心，它会从中心位置的点特征回归所有其他目标属性，如3D尺寸、方向和速度。 此外，用一个轻量级的第二阶段来改善目标位置。



a) Anchor-based t=1　b) Center-based t=1

c) Anchor-based t=2　d) Center-based t=2

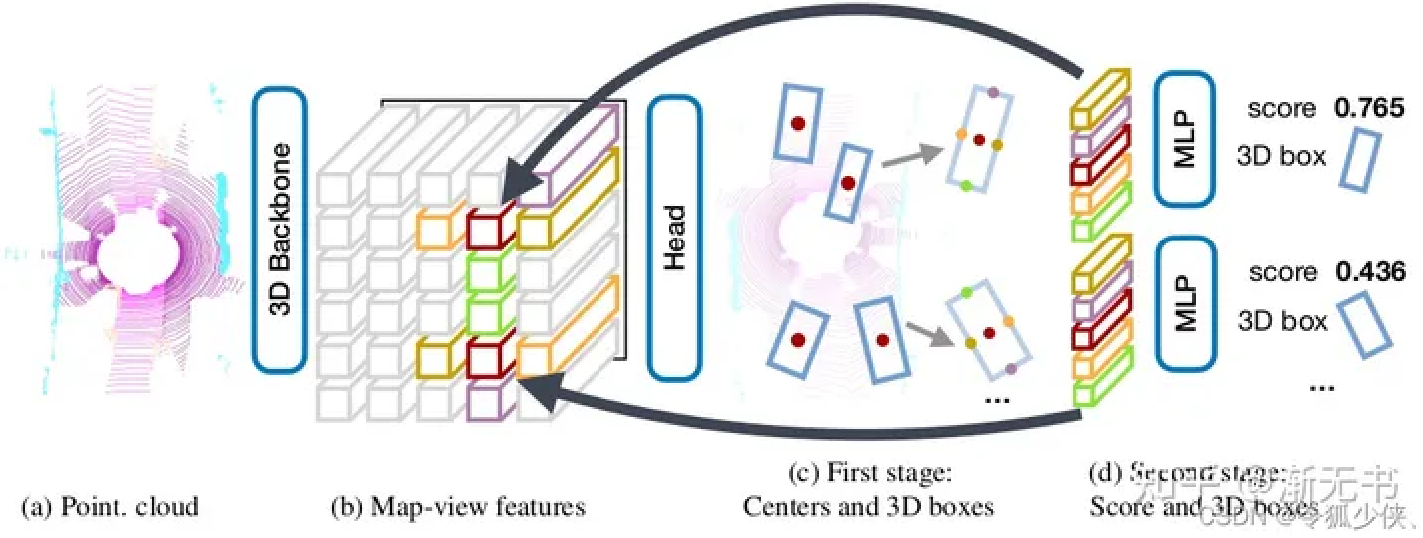
> `CenterPoint` 提出了一个基于中心的框架来表示、检测和跟踪对象。 以前的基于anchor的方法使用相对于车辆自身坐标轴对齐anchor。 当车辆在笔直的道路上行驶时，基于anchor的方法和我们的基于中心的方法都能够准确地检测到物体。 然而，在左转（底部）期间，基于anchor的方法难以将轴对齐的边界框拟合到旋转对象。 我们的基于中心的模型通过旋转不变的点准确地检测对象。

基于中心的表示法有几个关键的优点：

- 首先，与包围框不同，点没有内在的方向。 这大大减少了目标检测器的搜索空间，并允许主干学习目标的旋转不变性和等价性。
- 其次，基于中心的表示简化了下游任务，如跟踪。 如果物体是点，轨迹就是空间和时间中的路径。 中心点预测目标在连续帧和关联目标之间的相对偏移(速度)。
- 第三，基于点的特征提取使我们能够设计一个有效的两阶段细化模块，其速度远快于以往的方法。

我们在两个流行的大数据集上测试我们的模型： `Waymo Open` 和 `nuScenes` 。 我们发现，在不同的主干下，从框表示到基于中心表示的简单切换可以增加3-4个mAP。 第两阶段细化进一步带来额外的2 mAP提升，计算开销很小(< 10%)。

CenterPoint



在这里插入图片描述

图2显示了 `CenterPoint` 模型的总体框架。第一阶段首先使用 `backbone_3D` （使用voxel或pillar的形式)提取激光雷达点云的BEV特征。 然后， `backbone_2D` 检测头找到对象中心并使用中心特征回归完整的 3D 边界框（中心，长宽高，航向角，速度）。第二阶段是将第一阶段的预测框点特征传递到 `MLP` ，去细化置信度 `score` 和和 `3D box`

基于voxel

MeanVFE

利用预处理阶段计算出体素体素特征，对voxel的点特征求平均 `MeanVFE`

```
class MeanVFE(VFETemplate):
    def __init__(self, model_cfg, num_point_features, **kwargs):
        super().__init__(model_cfg=model_cfg)
        self.num_point_features = num_point_features # 5

    def get_output_feature_dim(self):
        return self.num_point_features

    def forward(self, batch_dict, **kwargs):
        """
        Args:
            batch_dict:
                voxels: (num_voxels, max_points_per_voxel, C)
                voxel_num_points: optional (num_voxels)
            **kwargs:

        Returns:
            vfe_features: (num_voxels, C)
        """
```

```
        # [num_voxels,10,5],[num_voxels]
        voxel_features, voxel_num_points = batch_dict['voxels'], batch_dict['vox
        # keepdim参数指是否对求和的结果squeeze,如果True其他维度保持不变，求和的dim维变ﾌ
        points_mean = voxel_features[:, :, :].sum(dim=1, keepdim=False) # 第二维ﾄ
        normalizer = torch.clamp_min(voxel_num_points.view(-1, 1), min=1.0).type
        points_mean = points_mean / normalizer # [num_voxels,5]
        batch_dict['voxel_features'] = points_mean.contiguous() # 深拷贝

        return batch_dict
```

VoxelResBackBone8x

`VoxelResBackBone8x`

```
def post_act_block(in_channels, out_channels, kernel_size, indice_key=None, stri
                   conv_type='subm', norm_fn=None):

    if conv_type == 'subm':
        conv = spconv.SubMConv3d(in_channels, out_channels, kernel_size, bias=Fa
    elif conv_type == 'spconv':
        conv = spconv.SparseConv3d(in_channels, out_channels, kernel_size, strid
                                   bias=False, indice_key=indice_key)
    elif conv_type == 'inverseconv':
        conv = spconv.SparseInverseConv3d(in_channels, out_channels, kernel_size
    else:
        raise NotImplementedError

    m = spconv.SparseSequential(
        conv,
        norm_fn(out_channels),
        nn.ReLU(),
    )

    return m

class VoxelResBackBone8x(nn.Module):
    def __init__(self, model_cfg, input_channels, grid_size, **kwargs):
        super().__init__()
        self.model_cfg = model_cfg
        norm_fn = partial(nn.BatchNorm1d, eps=1e-3, momentum=0.01)# 固定参数eps和
        self.sparse_shape = grid_size[::-1] + [1, 0, 0] # array([41, 1440, 1440]
        # SubMConv3d:只有当kernel的中心覆盖一个 active input site时，卷积输出才会被计
        # spatial_shape:[41, 1440, 1440] --> [41, 1440, 1440]
        self.conv_input = spconv.SparseSequential(
            spconv.SubMConv3d(input_channels, 16, 3, padding=1, bias=False, indi
            norm_fn(16),
```

```
            nn.ReLU(),
        )
        block = post_act_block
        # spatial_shape:[41, 1440, 1440] --> [41, 1440, 1440]
        self.conv1 = spconv.SparseSequential(
            SparseBasicBlock(16, 16, norm_fn=norm_fn, indice_key='res1'),
            SparseBasicBlock(16, 16, norm_fn=norm_fn, indice_key='res1'),
        )

        # SparseConv3d:就像普通的卷积一样，只要kernel 覆盖一个 active input site，就
        # spatial_shape:[41, 1440, 1440] --> [21, 720, 720]
        self.conv2 = spconv.SparseSequential(
            block(16, 32, 3, norm_fn=norm_fn, stride=2, padding=1, indice_key='s
            SparseBasicBlock(32, 32, norm_fn=norm_fn, indice_key='res2'),
            SparseBasicBlock(32, 32, norm_fn=norm_fn, indice_key='res2'),
        )
        # spatial_shape:[21, 720, 720] --> [11, 360, 360]
        self.conv3 = spconv.SparseSequential(
            block(32, 64, 3, norm_fn=norm_fn, stride=2, padding=1, indice_key='s
            SparseBasicBlock(64, 64, norm_fn=norm_fn, indice_key='res3'),
            SparseBasicBlock(64, 64, norm_fn=norm_fn, indice_key='res3'),
        )
        # spatial_shape:[11, 360, 360] --> [5, 180, 180]
        self.conv4 = spconv.SparseSequential(
            block(64, 128, 3, norm_fn=norm_fn, stride=2, padding=(0, 1, 1), indi
            SparseBasicBlock(128, 128, norm_fn=norm_fn, indice_key='res4'),
            SparseBasicBlock(128, 128, norm_fn=norm_fn, indice_key='res4'),
        )

        last_pad = 0
        last_pad = self.model_cfg.get('last_pad', last_pad)
        # spatial_shape:[5, 180, 180] --> [2, 180, 180]
        self.conv_out = spconv.SparseSequential(
            spconv.SparseConv3d(128, 128, (3, 1, 1), stride=(2, 1, 1), padding=l
            norm_fn(128),
            nn.ReLU(),
        )
        self.num_point_features = 128
        self.backbone_channels = {
            'x_conv1': 16,
            'x_conv2': 32,
            'x_conv3': 64,
            'x_conv4': 128
        }

    def forward(self, batch_dict):
        """
```

```
        Args:
            batch_dict:
                batch_size: int
                vfe_features: (num_voxels, C)
                voxel_coords: (num_voxels, 4), [batch_idx, z_idx, y_idx, x_idx]
        Returns:
            batch_dict:
                encoded_spconv_tensor: sparse tensor
        """
        # voxel_features (12000, 5) : Voxel特征均值,    voxel_coords (12000,  4) :
        # 对 voxel_features 按照 coors 进行索引，coors 在之前的处理中加入例如batch这
        voxel_features, voxel_coords = batch_dict['voxel_features'], batch_dict[
        batch_size = batch_dict['batch_size'] # 1

        # 根据voxel特征和voxel坐标以及空间形状和batch，建立稀疏tensor
        input_sp_tensor = spconv.SparseConvTensor(
            features=voxel_features, # torch.Size([12723, 5])
            indices=voxel_coords.int(), # torch.Size([12723, 4])
            spatial_shape=self.sparse_shape, # [41, 1440, 1440]
            batch_size=batch_size # 1
        )
        # 子流线稀疏卷积+BN+Relu spatial_shape:[41, 1440, 1440]-->[41, 1440, 1440]
        x = self.conv_input(input_sp_tensor)

        x_conv1 = self.conv1(x) # 经两次SparseBasicBlock spatial_shape:[41, 1440,
        x_conv2 = self.conv2(x_conv1) # 经子流线稀疏卷积、两次SparseBasicBlock spa
        x_conv3 = self.conv3(x_conv2) # 经子流线稀疏卷积、两次SparseBasicBlock spa
        x_conv4 = self.conv4(x_conv3) # 经子流线稀疏卷积、两次SparseBasicBlock spa

        # [5, 180, 180] -> [2, 180, 180] 通道128-->128
        out = self.conv_out(x_conv4) # 用的卷积形式是 SparseConv3d 而不是 SubMConv

        batch_dict.update({
            'encoded_spconv_tensor': out,
            'encoded_spconv_tensor_stride': 8
        })
        batch_dict.update({
            'multi_scale_3d_features': {
                'x_conv1': x_conv1,
                'x_conv2': x_conv2,
                'x_conv3': x_conv3,
                'x_conv4': x_conv4,
            }
        })

        batch_dict.update({
            'multi_scale_3d_strides': {
```

```
                'x_conv1': 1,
                'x_conv2': 2,
                'x_conv3': 4,
                'x_conv4': 8,
            }
        })

        return batch_dict
```

对于 `VoxelBackBone8x` 模块的前向推理 `(forward)` 部分，其输入字典中最重要的内容为 `voxel_features` 和 `voxel_coords` 。他们分别表示有效的输入特征，以及这些有效特征的空间位置。 `voxel_features` 的 `size` 为 `(N,5)`

从 `post_act_block` 中可以看出spconv有3种3D稀疏卷积： `SubMConv3d、SparseConv3d和SparseInverseConv3d`

```
conv = spconv.SubMConv3d(in_channels, out_channels, kernel_size, bias=False, ind
```

`spconv` 的 `3D` 稀疏卷积和普通卷积使用类似，唯一多了一个 `indice_key` ，这是为了在 `indice` 相同的情况下重复利用计算好的 `rulebook` 和 `hash` 表，减少计算

看下面这行代码：

```
        self.sparse_shape = grid_size[::-1] + [1, 0, 0] # array([41, 1440, 1440]
```

sparse_shape 的 Z轴为什么需要加1？

> 参考： https:// github.com/open-mmlab/m mdetection3d/issues/282

SparseEncoder将在高维度上进行下采样。加1后允许高度维度可以无误差地向下采样几次，并最终满足CenterPoint的实现。

继续看残差网络块：SparseBasicBlock

```
class SparseBasicBlock(spconv.SparseModule):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, norm_fn=None, downsample=None
        super(SparseBasicBlock, self).__init__()

```

```
        assert norm_fn is not None
        bias = norm_fn is not None
        self.conv1 = spconv.SubMConv3d(
            inplanes, planes, kernel_size=3, stride=stride, padding=1, bias=bias
        )
        self.bn1 = norm_fn(planes)
        self.relu = nn.ReLU()
        self.conv2 = spconv.SubMConv3d(
            planes, planes, kernel_size=3, stride=stride, padding=1, bias=bias,
        )
        self.bn2 = norm_fn(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        identity = x  # [41, 1440, 1440]
        # [41, 1440, 1440]
        out = self.conv1(x) # 子流线卷积 indice_key='res1'
        out = replace_feature(out, self.bn1(out.features)) # bn 调用SparseConvTe
        out = replace_feature(out, self.relu(out.features)) # relu 调用SparseCon
        # [41, 1440, 1440]
        out = self.conv2(out) # indice_key='res1'
        out = replace_feature(out, self.bn2(out.features)) # bn 调用SparseConvTe

        if self.downsample is not None:  # False
            identity = self.downsample(x)
        # 残差网络：将identity和out的feature相加后，构建新的输入SparseConvTensor
        out = replace_feature(out, out.features + identity.features) # 调用Spars
        out = replace_feature(out, self.relu(out.features)) # relu 调用SparseCon

        return out
```

重点看 `forward` 中的 `replace_feature` ，`replace_feature` 函数位于 `OpenPCDet/pcdet/utils/spconv_utils.py`

```
def replace_feature(out, new_features):
    # __dir__ 返回一个有序列表:列表包含当前对象的所有属性名及方法名
    if "replace_feature" in out.__dir__():
        # spconv 2.x behaviour
        return out.replace_feature(new_features)
    else:
        out.features = new_features
        return out
```

会调用 `spconv 2.0` 中类 `SparseConvTensor` 的 `replace_feature` 方法，代码如下：

`spconv/pytorch/core.py`

```
    def replace_feature(self, feature: torch.Tensor):
        """we need to replace x.features = F.relu(x.features) with x = x.replace
        due to limit of torch.fx
        """
        # assert feature.shape[0] == self.indices.shape[0], "replaced num of fea
        new_spt = SparseConvTensor(feature, self.indices, self.spatial_shape,
                                   self.batch_size, self.grid, self.voxel_num,
                                   self.indice_dict)
        new_spt.benchmark = self.benchmark
        new_spt.benchmark_record = self.benchmark_record
        new_spt.thrust_allocator = self.thrust_allocator
        new_spt._timer = self._timer
        new_spt.force_algo = self.force_algo

        return new_spt
```

下面总结下 `backbone3d` 稀疏卷积的具体调用过程：

```
# conv_input
# [41, 1440, 1440]-->[41, 1440, 1440]
SubMConv3d(5, 16, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], di
BatchNorm1d(16, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

# conv1
# [41, 1440, 1440]-->[41, 1440, 1440]
SubMConv3d(16, 16, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(16, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()
SubMConv3d(16, 16, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(16, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

SubMConv3d(16, 16, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(16, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()
SubMConv3d(16, 16, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(16, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

# conv2
```

```
# [41, 1440, 1440]-->[21, 720, 720]
SparseConv3d(16, 32, kernel_size=[3, 3, 3], stride=[2, 2, 2], padding=[1, 1, 1],
BatchNorm1d(32, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

SubMConv3d(32, 32, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(32, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()
SubMConv3d(32, 32, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(32, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

SubMConv3d(32, 32, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(32, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()
SubMConv3d(32, 32, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(32, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

# conv3
# [21, 720, 720]-->[11, 360, 360]
SparseConv3d(32, 64, kernel_size=[3, 3, 3], stride=[2, 2, 2], padding=[1, 1, 1],
BatchNorm1d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

SubMConv3d(64, 64, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()
SubMConv3d(64, 64, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

SubMConv3d(64, 64, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()
SubMConv3d(64, 64, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1], d
BatchNorm1d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
ReLU()

# conv4
# [11, 360, 360]-->[5, 180, 180]
SparseConv3d(64, 128, kernel_size=[3, 3, 3], stride=[2, 2, 2], padding=[0, 1, 1]
BatchNorm1d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()

SubMConv3d(128, 128, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1],
BatchNorm1d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
```

```
ReLU()
SubMConv3d(128, 128, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1],
BatchNorm1d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()

SubMConv3d(128, 128, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1],
BatchNorm1d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
SubMConv3d(128, 128, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[1, 1, 1],
BatchNorm1d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()

# conv_out
# [5, 180, 180] -> [2, 180, 180]
SparseConv3d(128, 128, kernel_size=[3, 1, 1], stride=[2, 1, 1], padding=[0, 0, 0
BatchNorm1d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
```

HeighCompression

主要目的是将提取出的点云稀疏特征 `encoded_spconv_tensor` 转换到 `BEV` 视角下。事实上这个转换过程非常简单粗暴。首先把稀疏特征转换成为体素特征的格式，然后把 `Z` 轴和通道合并，变为 `BEV` 视角上的 `2D` 特征。

```
# 在高度方向上进行压缩
class HeightCompression(nn.Module):
    def __init__(self, model_cfg, **kwargs):
        super().__init__()
        self.model_cfg = model_cfg
        self.num_bev_features = self.model_cfg.NUM_BEV_FEATURES # 256

    def forward(self, batch_dict):
        """
        Args:
            batch_dict:
                encoded_spconv_tensor: sparse tensor
        Returns:
            batch_dict:
                spatial_features:

        """
        encoded_spconv_tensor = batch_dict['encoded_spconv_tensor'] # [2, 180, 1
        spatial_features = encoded_spconv_tensor.dense() # torch.Size([1, 128, 2
        N, C, D, H, W = spatial_features.shape # 1 128 2 180 180
        spatial_features = spatial_features.view(N, C * D, H, W) # torch.Size([1
```

```
        batch_dict['spatial_features'] = spatial_features
        batch_dict['spatial_features_stride'] = batch_dict['encoded_spconv_tenso
        return batch_dict
```

`dense()` 是调用spconv中类 `SparseConvTensor` 的方法，类 `SparseConvTensor` 位于 `spconv/__init__.py` ，作用将 `backbone_3D` 经稀疏卷积的输出 `out` 转为 `(batch_size, chanels, grid_nums_z, grid_nums_y, grid_nums_x)` 形状的 `torch` 张量

```
# 根据索引indices对给定shape的零张量中的单个值或切片应用稀疏updates来创建新的张量
def scatter_nd(indices, updates, shape):
    """pytorch edition of tensorflow scatter_nd.
    this function don't contain except handle code. so use this carefully
    when indice repeats, don't support repeat add which is supported
    in tensorflow.
    """
    # indices : [N,4]
    # updates : [N,128]
    # shape: [4, 2, 180, 180, 128]
    ret = torch.zeros(*shape, dtype=updates.dtype, device=updates.device) # [4,
    ndim = indices.shape[-1] # 4
    output_shape = list(indices.shape[:-1]) + shape[indices.shape[-1]:] # [4,N]
    flatted_indices = indices.view(-1, ndim) # [N,4]
    slices = [flatted_indices[:, i] for i in range(ndim)]  # batch_index,z,y,x
    slices += [Ellipsis]
    ret[slices] = updates.view(*output_shape)
    return ret

class SparseConvTensor(object):
    def __init__(self, features, indices, spatial_shape, batch_size, grid=None):
        """
        Args:
            features: [num_points, num_features] feature tensor
            indices: [num_points, ndim + 1] indice tensor. batch index saved in
            spatial_shape: spatial shape of your sparse data
            batch_size: batch size of your sparse data
            grid: pre-allocated grid tensor. should be used when the volume of s
        """
        self.features = features # 有效的数据
        self.indices = indices # 有效的voxel空间坐标索引 (64000, 4)
        self.spatial_shape = spatial_shape # 空间大小 (41, 1600, 1408)
        self.batch_size = batch_size
        self.indice_dict = {}
        self.grid = grid

```

```
    def dense(self, channels_first=True):
        output_shape = [self.batch_size] + list(self.spatial_shape) + [self.feat
        res = scatter_nd(self.indices.to(self.features.device).long(), self.feat
        if not channels_first:
            return res
        ndim = len(self.spatial_shape) # 3
        trans_params = list(range(0, ndim + 1)) #(0,1,2,3)
        trans_params.insert(1, ndim + 1) # (0,4,1,2,3)
        return res.permute(*trans_params).contiguous() # [4, 2, 180, 180, 128] ->
```

基于pillar

DynamicPillarVFE

直接看代码和注释:

```
class PFNLayerV2(nn.Module):
    def __init__(self,
                 in_channels,
                 out_channels,
                 use_norm=True,
                 last_layer=False):
        super().__init__()

        self.last_vfe = last_layer
        self.use_norm = use_norm
        if not self.last_vfe:
            out_channels = out_channels // 2

        if self.use_norm:
            self.linear = nn.Linear(in_channels, out_channels, bias=False)
            self.norm = nn.BatchNorm1d(out_channels, eps=1e-3, momentum=0.01)
        else:
            self.linear = nn.Linear(in_channels, out_channels, bias=True)

        self.relu = nn.ReLU()

    def forward(self, inputs, unq_inv):

        x = self.linear(inputs)
        x = self.norm(x) if self.use_norm else x
        x = self.relu(x)
        # 相同索引代表同一个voxle,对相同索引的点取最大值,即取voxexl每个点的最大值
        x_max = torch_scatter.scatter_max(x, unq_inv, dim=0)[0]

        if self.last_vfe:
```

```python
            return x_max
        else:
            # 给每个voxel内的点拼接全局voxel信息
            x_concatenated = torch.cat([x, x_max[unq_inv, :]], dim=1)
            return x_concatenated


class DynamicPillarVFE(VFETemplate):
    def __init__(self, model_cfg, num_point_features, voxel_size, grid_size, poi
        super().__init__(model_cfg=model_cfg)

        self.use_norm = self.model_cfg.USE_NORM # True
        self.with_distance = self.model_cfg.WITH_DISTANCE # False
        self.use_absolute_xyz = self.model_cfg.USE_ABSLOTE_XYZ  # True
        num_point_features += 6 if self.use_absolute_xyz else 3
        if self.with_distance:
            num_point_features += 1

        self.num_filters = self.model_cfg.NUM_FILTERS # [64,64]
        assert len(self.num_filters) > 0
        num_filters = [num_point_features] + list(self.num_filters)

        pfn_layers = []
        for i in range(len(num_filters) - 1):
            in_filters = num_filters[i]
            out_filters = num_filters[i + 1]
            pfn_layers.append(
                PFNLayerV2(in_filters, out_filters, self.use_norm, last_layer=(i
            )
        self.pfn_layers = nn.ModuleList(pfn_layers)

        self.voxel_x = voxel_size[0] # 0.2
        self.voxel_y = voxel_size[1] # 0.2
        self.voxel_z = voxel_size[2] # 8
        self.x_offset = self.voxel_x / 2 + point_cloud_range[0] # -51.100000762S
        self.y_offset = self.voxel_y / 2 + point_cloud_range[1] # -51.100000762S
        self.z_offset = self.voxel_z / 2 + point_cloud_range[2] # -1.0

        self.scale_xy = grid_size[0] * grid_size[1] # 262144
        self.scale_y = grid_size[1] # 512

        # tensor([512, 512,   1], device='cuda:0')
        self.grid_size = torch.tensor(grid_size).cuda()
        # tensor([0.2000, 0.2000, 8.0000], device='cuda:0')
        self.voxel_size = torch.tensor(voxel_size).cuda()
        # tensor([-51.2000, -51.2000,  -5.0000,  51.2000,  51.2000,   3.0000],de
        self.point_cloud_range = torch.tensor(point_cloud_range).cuda()
```

```python

    def get_output_feature_dim(self):
        return self.num_filters[-1]

    def forward(self, batch_dict, **kwargs):
        points = batch_dict['points'] # (batch_idx, x, y, z, i, e)
        # 每个点的网格坐标
        points_coords = torch.floor((points[:, [1,2]] - self.point_cloud_range[[
        mask = ((points_coords >= 0) & (points_coords < self.grid_size[[0,1]])).
        points = points[mask]
        points_coords = points_coords[mask]
        points_xyz = points[:, [1, 2, 3]].contiguous()

        # 网格坐标一维
        merge_coords = points[:, 0].int() * self.scale_xy + \
                       points_coords[:, 0] * self.scale_y + \
                       points_coords[:, 1]
        # sorted:是否返回无重复张量按照数值进行排序，默认是升序排列，sorted并非表示降序
        # return_inverse:是否返回原始张量中每个元素在处理后的无重复张量中对应的索引
        # return_counts: 统计原始张量中每个独立元素的个数
        # dim:值沿那个维度进行unique的处理
        # torch.Size([40620])
        # 按voxel坐标值升序排列，计算voxel一维坐标，索引，voxel中点个数
        unq_coords, unq_inv, unq_cnt = torch.unique(merge_coords, return_inverse

        # 按第一维度，对unq_inv相同索引对应的src元素求均值
        points_mean = torch_scatter.scatter_mean(points_xyz, unq_inv, dim=0)
        # 每个点相对voxel质心的偏移
        f_cluster = points_xyz - points_mean[unq_inv, :]

        f_center = torch.zeros_like(points_xyz)
        # 每个点相对几何中心的偏移
        f_center[:, 0] = points_xyz[:, 0] - (points_coords[:, 0].to(points_xyz.d
        f_center[:, 1] = points_xyz[:, 1] - (points_coords[:, 1].to(points_xyz.d
        f_center[:, 2] = points_xyz[:, 2] - self.z_offset

        if self.use_absolute_xyz: # True
            features = [points[:, 1:], f_cluster, f_center] # x,y,z,i,e,f_cluste
        else:
            features = [points[:, 4:], f_cluster, f_center]

        if self.with_distance:# False
            points_dist = torch.norm(points[:, 1:4], 2, dim=1, keepdim=True)
            features.append(points_dist)
        features = torch.cat(features, dim=-1)

        # 两层卷积: 11->64->64
```

```
        for pfn in self.pfn_layers:
            features = pfn(features, unq_inv)

        # generate voxel coordinates
        unq_coords = unq_coords.int()
        voxel_coords = torch.stack((unq_coords // self.scale_xy, # z
                                    (unq_coords % self.scale_xy) // self.scale_y,
                                    unq_coords % self.scale_y, # x
                                    torch.zeros(unq_coords.shape[0]).to(unq_coord
                                    ), dim=1)
        # [batch_id,z,y,x] --> [batch_id,x,y,z]
        voxel_coords = voxel_coords[:, [0, 3, 2, 1]]

        batch_dict['pillar_features'] = features
        batch_dict['voxel_coords'] = voxel_coords
        return batch_dict
```

PointPillarScatter

将点云提取的特在转到bev视角下

```
class PointPillarScatter(nn.Module):
    def __init__(self, model_cfg, grid_size, **kwargs):
        super().__init__()

        self.model_cfg = model_cfg
        self.num_bev_features = self.model_cfg.NUM_BEV_FEATURES
        self.nx, self.ny, self.nz = grid_size
        assert self.nz == 1

    def forward(self, batch_dict, **kwargs):
        pillar_features, coords = batch_dict['pillar_features'], batch_dict['vox
        batch_spatial_features = []
        batch_size = coords[:, 0].max().int().item() + 1
        for batch_idx in range(batch_size):
            spatial_feature = torch.zeros(
                self.num_bev_features,
                self.nz * self.nx * self.ny,
                dtype=pillar_features.dtype,
                device=pillar_features.device)

            batch_mask = coords[:, 0] == batch_idx
            this_coords = coords[batch_mask, :]
            indices = this_coords[:, 1] + this_coords[:, 2] * self.nx + this_coo
            indices = indices.type(torch.long)
            pillars = pillar_features[batch_mask, :]
```

```python
            pillars = pillars.t()
            spatial_feature[:, indices] = pillars
            batch_spatial_features.append(spatial_feature)

        batch_spatial_features = torch.stack(batch_spatial_features, 0)
        batch_spatial_features = batch_spatial_features.view(batch_size, self.nu
        batch_dict['spatial_features'] = batch_spatial_features
        return batch_dict
```

BaseBEVBackbone

基于 `pillar` 的配置文件:

```yaml
BACKBONE_2D:
    NAME: BaseBEVBackbone
    LAYER_NUMS: [3, 5, 5]
    LAYER_STRIDES: [2, 2, 2]
    NUM_FILTERS: [64, 128, 256]
    UPSAMPLE_STRIDES: [0.5, 1, 2]
    NUM_UPSAMPLE_FILTERS: [128, 128, 128]
```

基于 `voxel` 的配置文件:

```yaml
BACKBONE_2D:
    NAME: BaseBEVBackbone
    LAYER_NUMS: [5, 5]
    LAYER_STRIDES: [1, 2]
    NUM_FILTERS: [128, 256]
    UPSAMPLE_STRIDES: [1, 2]
    NUM_UPSAMPLE_FILTERS: [256, 256]
```

下面以 `voxel` 参数为例:

使用类似于 `(SSD)` 架构来构建 `RPN` 架构。 `RPN` 的输入包括来自`backbone3d`稀疏卷积中间提取特征经通道和高度压缩后 `spatial_features` 。 `RPN` 架构由三个阶段组成。 每个阶段都从一个下采样的卷积层开始，然后是几个卷积层。 在每个卷积层之后，应用 `BatchNorm` 和 `ReLU` 层。 然后将不同下采样的特征进行反卷积操作，变成相同大小的特征图，并拼接这些来自不同尺度的特征图，构建高分辨率特征图，用于最后的检测。

基于 `voxel` 的 `centerpoint backbone2d` 部分存在两个下采样分支结构，则对应存在两个反卷积结构：经过 `HeightCompression` 得到的BEV特征图维度为： `(batch_size, 128*2, 180, 180)`

- 下采样分支一：`(batch_size, 256, 180, 180) --> (batch_size,128, 180, 180)`，对应反卷积分支一：`(batch_size, 128, 180, 180) --> (batch_size, 256, 180, 180)`
- 下采样分支二：`(batch_size, 256, 180, 180) --> (batch_size,256, 90, 90)`，对应反卷积分支二：`(batch_size, 256, 90, 90) --> (batch_size, 256, 180, 180)`

```python
class BaseBEVBackbone(nn.Module):
    def __init__(self, model_cfg, input_channels):
        super().__init__()
        self.model_cfg = model_cfg

        if self.model_cfg.get('LAYER_NUMS', None) is not None:
            # LAYER_NUMS: [5, 5]     LAYER_STRIDES: [1, 2]       NUM_FILTERS: [128
            assert len(self.model_cfg.LAYER_NUMS) == len(self.model_cfg.LAYER_ST
            layer_nums = self.model_cfg.LAYER_NUMS # [5, 5]
            layer_strides = self.model_cfg.LAYER_STRIDES # [1, 2]
            num_filters = self.model_cfg.NUM_FILTERS # [128, 256]
        else:
            layer_nums = layer_strides = num_filters = []

        if self.model_cfg.get('UPSAMPLE_STRIDES', None) is not None:
            # UPSAMPLE_STRIDES: [1, 2]     NUM_UPSAMPLE_FILTERS: [256, 256]
            assert len(self.model_cfg.UPSAMPLE_STRIDES) == len(self.model_cfg.NU
            num_upsample_filters = self.model_cfg.NUM_UPSAMPLE_FILTERS # [256, 2
            upsample_strides = self.model_cfg.UPSAMPLE_STRIDES # [1, 2]
        else:
            upsample_strides = num_upsample_filters = []
        #import pdb;pdb.set_trace()
        num_levels = len(layer_nums) # 2
        c_in_list = [input_channels, *num_filters[:-1]] # [256, 128]
        self.blocks = nn.ModuleList()
        self.deblocks = nn.ModuleList()
        self.res_backbone = self.model_cfg.get('res_backbone',False) # False
        for idx in range(num_levels):
            cur_layers = [
                nn.ZeroPad2d(1),
                nn.Conv2d(
                    c_in_list[idx], num_filters[idx], kernel_size=3,
                    stride=layer_strides[idx], padding=0, bias=False
                ),
                nn.BatchNorm2d(num_filters[idx], eps=1e-3, momentum=0.01),
                nn.ReLU()
            ]
            for k in range(layer_nums[idx]): # LAYER_NUMS: [5, 5]
```

```
                if self.res_backbone: # False
                    cur_layers.extend([
                        nn.Conv2d(num_filters[idx], num_filters[idx], kernel_siz
                        nn.BatchNorm2d(num_filters[idx], eps=1e-3, momentum=0.01
                        nn.ReLU(),
                        nn.Conv2d(num_filters[idx], num_filters[idx], kernel_siz
                        nn.BatchNorm2d(num_filters[idx], eps=1e-3, momentum=0.01
                    ])
                else:
                    cur_layers.extend([
                        nn.Conv2d(num_filters[idx], num_filters[idx], kernel_siz
                        nn.BatchNorm2d(num_filters[idx], eps=1e-3, momentum=0.01
                        nn.ReLU()
                    ])
            self.blocks.append(nn.Sequential(*cur_layers))
            if len(upsample_strides) > 0: # True
                stride = upsample_strides[idx] # 1 , 2
                if stride >= 1:
                    self.deblocks.append(nn.Sequential(
                        nn.ConvTranspose2d(
                            num_filters[idx], num_upsample_filters[idx],
                            upsample_strides[idx],
                            stride=upsample_strides[idx], bias=False
                        ),
                        nn.BatchNorm2d(num_upsample_filters[idx], eps=1e-3, mome
                        nn.ReLU()
                    ))
                else:
                    stride = np.round(1 / stride).astype(np.int)
                    self.deblocks.append(nn.Sequential(
                        nn.Conv2d(
                            num_filters[idx], num_upsample_filters[idx],
                            stride,
                            stride=stride, bias=False
                        ),
                        nn.BatchNorm2d(num_upsample_filters[idx], eps=1e-3, mome
                        nn.ReLU()
                    ))

        c_in = sum(num_upsample_filters) # 512
        if len(upsample_strides) > num_levels: # False
            self.deblocks.append(nn.Sequential(
                nn.ConvTranspose2d(c_in, c_in, upsample_strides[-1], stride=upsa
                nn.BatchNorm2d(c_in, eps=1e-3, momentum=0.01),
                nn.ReLU(),
            ))

```

```python
        self.num_bev_features = c_in # 512

    def forward(self, data_dict):
        """
        Args:
            data_dict:
                spatial_features
        Returns:
        """
        spatial_features = data_dict['spatial_features'] # torch.Size([4, 256, 1
        ups = []
        ret_dict = {}
        x = spatial_features # torch.Size([4, 256, 180, 180])
        for i in range(len(self.blocks)):
            #import pdb;pdb.set_trace()
            if self.res_backbone: # False
                x = self.blocks[i][:4](x)
                for mm in range(self.model_cfg.LAYER_NUMS[i]):
                    identity = x
                    out = self.blocks[i][4+mm*5:4+(mm+1)*5](x)
                    x = x + out
            else:
                x = self.blocks[i](x) # torch.Size([4, 128, 180, 180]) ,torch.Si

            stride = int(spatial_features.shape[2] / x.shape[2]) # 1,2
            ret_dict['spatial_features_%dx' % stride] = x # {{spatial_features_1
            if len(self.deblocks) > 0:
                ups.append(self.deblocks[i](x)) # torch.Size([1, 256, 180, 180])
            else:
                ups.append(x)
        # 拼接不同尺度上采样后的特征
        if len(ups) > 1:
            x = torch.cat(ups, dim=1) # torch.Size([4, 512, 180, 180])
        elif len(ups) == 1:
            x = ups[0]

        if len(self.deblocks) > len(self.blocks):
            x = self.deblocks[-1](x)

        data_dict['spatial_features_2d'] = x # torch.Size([4, 512, 180, 180])

        return data_dict
```

梳理下 `backbone2d` 整体的网络结构，如下：

```
# 下采样分支一: (batch_size, 128*2, 180, 180) --> (batch_size,128, 180, 180)
ZeroPad2d((1, 1, 1, 1))
Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), bias=False)
BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
# 对应反卷积分支一: (batch_size, 128, 180, 180) --> (batch_size, 256, 180, 180)
ConvTranspose2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()

# 下采样分支二: (batch_size, 256, 180, 180) --> (batch_size,256, 90, 90)
ZeroPad2d((1, 1, 1, 1))
Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), bias=False)
BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
# 对应反卷积分支二: (batch_size, 256, 90, 90) --> (batch_size, 256, 180, 180)
ConvTranspose2d(256, 256, kernel_size=(2, 2), stride=(2, 2), bias=False)
```

```
BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True
ReLU()
```

CenterHead

在nuscenes数据集中，10类目标被分为6个大类： `[['car'], ['truck', 'construction_vehicle'], ['bus', 'trailer'], ['barrier'], ['motorcycle', 'bicycle'], ['pedestrian', 'traffic_cone']]` ，网络中每个类分配一个 `head` ，对应 `SeparateHead` 类，即对每个类分配一个MLP预测 `center,center_z,dim,rot,vel,hm`

在 `assign_targets` 函数中， `centerpoint` 用到了高斯圆来计算 `heatmap` 中标签范围，首先根据真值GT和IOU阈值确定最小的高斯半径，然后基于高斯半径生成 `heatmap`

如何确定最小高斯半径？根据预测的两个角点与 `Ground Truth` 角点的位置关系，分三种情况来考虑：

- 两角点均在真值框内
- 两角点均在真值框外
- 一角点在真值框内，一角点在真值框外

参考： https:// blog.csdn.net/x55026225 7/article/details/121289242

```
def gaussian_radius(height, width, min_overlap=0.5):
    """
    Args:
        height: (N)
        width: (N)
        min_overlap:
    Returns:
    """
    # 预测框两个角点在GT框的两个角点以r为半径的圆内，如何确定半径r，保证预测框与真值框的I(
    """
    1.一角点在真值框内,一角点在真值框外
    最小IOU在预测框两个角点分别和和半径r的圆相外切和相内切时取得(例如可以固定某一角点在x方
    因此我们只需要考虑“预测的框和GTbox两个角点以r为半径的圆一个边内切,一个边外切
    min_overlap =(h-r)*(w-r)/(2*h*w-(h-r)*(w-r)) --> r
    整理为r的一元二次方程: r^2 - (h+w)*r + (1-min_overlap)*h*w / (1+min_overlap) =(
    """
    a1 = 1
    b1 = (height + width)
    c1 = width * height * (1 - min_overlap) / (1 + min_overlap)
    sq1 = (b1 ** 2 - 4 * a1 * c1).sqrt()
    r1 = (b1 + sq1) / 2

```

```
    """
    2.两角点均在真值框内
    最小IOU在预测框和半径r圆相切获取
    min_overlap =(h-2*r)*(w-2*r)/(h*w) --> r
    整理为r的一元二次方程: 4*r^2 - 2*(h+w)*r + (1-min_overlap)*h*w =0
    """
    a2 = 4
    b2 = 2 * (height + width)
    c2 = (1 - min_overlap) * width * height
    sq2 = (b2 ** 2 - 4 * a2 * c2).sqrt()
    r2 = (b2 + sq2) / 2

    """
    3.两角点均在真值框外
    最小IOU在预测框和半径r相外切时取得,只需要考虑 预测的框和GTbox两个角点以r为半径的圆外切
    min_overlap =(h*w)*(w+2*r)/(h+2*r) --> r
    整理为r的一元二次方程: 4*min_overlap*r^2 + 2*min_overlap*(h+w)*r + (min_overlap
    """
    a3 = 4 * min_overlap
    b3 = -2 * min_overlap * (height + width)
    c3 = (min_overlap - 1) * width * height
    sq3 = (b3 ** 2 - 4 * a3 * c3).sqrt()
    r3 = (b3 + sq3) / 2
    ret = torch.min(torch.min(r1, r2), r3)
    return ret
```

loss

获取每个head的推理结果，就结合真值计算分类，回归loss：

- FocalLoss
- RegLoss

`pcdet/models/detectors/centerpoint.py`

```
def forward(self, batch_dict):
    for cur_module in self.module_list:
        batch_dict = cur_module(batch_dict)

    if self.training:
     # loss :  多个head的总损失
     # tb_dict :  每个head的hm_loss,loc_loss损失,多个head的总损失rpn_loss,loss_
     # disp_dict : {}
        loss, tb_dict, disp_dict = self.get_training_loss()
```

```python

            ret_dict = {
                'loss': loss
            }
            return ret_dict, tb_dict, disp_dict
        else:
            pred_dicts, recall_dicts = self.post_processing(batch_dict)
            return pred_dicts, recall_dicts

    def get_training_loss(self):
        disp_dict = {}

        loss_rpn, tb_dict = self.dense_head.get_loss()
        tb_dict = {
            'loss_rpn': loss_rpn.item(),
            **tb_dict
        }

        loss = loss_rpn
        return loss, tb_dict, disp_dict
```

`pcdet/models/dense_heads/center_head.py`

```python
    def build_losses(self):
     # 在自定义网络，由于自定义变量不是Module类型，pytorch不会自动注册
     # add_module函数用来为网络添加自定义模块，也可以使用ModuleList来封装自定义模块，py
        self.add_module('hm_loss_func', loss_utils.FocalLossCenterNet())
        self.add_module('reg_loss_func', loss_utils.RegLossCenterNet())

    def get_loss(self):
        pred_dicts = self.forward_ret_dict['pred_dicts']
        target_dicts = self.forward_ret_dict['target_dicts']

        tb_dict = {}
        loss = 0

        for idx, pred_dict in enumerate(pred_dicts):
            pred_dict['hm'] = self.sigmoid(pred_dict['hm'])
            hm_loss = self.hm_loss_func(pred_dict['hm'], target_dicts['heatmaps'
            # 'cls_weight': 1.0
            hm_loss *= self.model_cfg.LOSS_CONFIG.LOSS_WEIGHTS['cls_weight']

            target_boxes = target_dicts['target_boxes'][idx]
            pred_boxes = torch.cat([pred_dict[head_name] for head_name in self.s

```

```
                reg_loss = self.reg_loss_func(
                    pred_boxes, target_dicts['masks'][idx], target_dicts['inds'][idx
                )
                # 'code_weights': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.2, 0.2, 1.0, 1.0]
                loc_loss = (reg_loss * reg_loss.new_tensor(self.model_cfg.LOSS_CONFI
                # 'loc_weight': 0.25
                loc_loss = loc_loss * self.model_cfg.LOSS_CONFIG.LOSS_WEIGHTS['loc_w

                loss += hm_loss + loc_loss
                tb_dict['hm_loss_head_%d' % idx] = hm_loss.item()
                tb_dict['loc_loss_head_%d' % idx] = loc_loss.item()

            tb_dict['rpn_loss'] = loss.item()
            return loss, tb_dict
```

`pcdet/utils/loss_utils.py`

FocalLoss

focal loss核心思想：对于容易分辨的样本，降低他们loss的权重，而对于难分辨的样本，相对来说提高了它们的权重；这样模型在bp时，更偏向于学习那些难分辨的样本，从而整体学习效率更高，且学习不会偏向于正样本或负样本；

$$L_{focal} = - \frac{1}{N} \begin{cases} (1-\hat y)^\alpha \log(\hat y) & \text{ if } y= 1 \\ (1-\hat y)^\beta (\hat y)^\alpha \log(1-\hat y) & \text{ otherwise } \end{cases}\\$$

其中：α 和 β 是超参数，N是 `gt` 中正样本个数

在 $\hat y =1$ 时候：

- 对于易分样本，预测值 $\hat y$ 接近1， $(1-\hat y)^\alpha$ 就是很小值，这样loss很小
- 对于难分样本，预测值 $\hat y$ 接近0， $(1-\hat y)^\alpha$ 就接近1，损失不受影响
- 权重因子 $(\hat y)^\alpha$ ，用控制正负样本对总 `loss` 的共享权重。 α 越大， $(\hat y)^\alpha$ 越小，可以降低负样本（多的那类样本）的权重，相对提高正样本的权重

代码中 $\alpha = 2$ 、 $\beta = 4$

```
def neg_loss_cornernet(pred, gt, mask=None):
    """
    Refer to https://github.com/tianweiy/CenterPoint.
    Modified focal loss. Exactly the same as CornerNet. Runs faster and costs a
    Args:
        pred: (batch x c x h x w)
        gt: (batch x c x h x w)
        mask: (batch x h x w)
```

```
    Returns:
    """
    # eq函数是遍历gt这个tensor每个element，和1比较，如果等于1，则返回1，否则返回0
    pos_inds = gt.eq(1).float()
    # 遍历gt这个tensor每个element，和1比较，如果小于1，则返回1，否则返回0
    neg_inds = gt.lt(1).float()

    neg_weights = torch.pow(1 - gt, 4)

    loss = 0

    pos_loss = torch.log(pred) * torch.pow(1 - pred, 2) * pos_inds
    neg_loss = torch.log(1 - pred) * torch.pow(pred, 2) * neg_weights * neg_inds

    if mask is not None:
        mask = mask[:, None, :, :].float()
        pos_loss = pos_loss * mask
        neg_loss = neg_loss * mask
        num_pos = (pos_inds.float() * mask).sum()
    else:
        num_pos = pos_inds.float().sum()

    pos_loss = pos_loss.sum()
    neg_loss = neg_loss.sum()

    if num_pos == 0:
        loss = loss - neg_loss
    else:
        loss = loss - (pos_loss + neg_loss) / num_pos
    return loss

class FocalLossCenterNet(nn.Module):
    """
    Refer to https://github.com/tianweiy/CenterPoint
    """
    def __init__(self):
        super(FocalLossCenterNet, self).__init__()
        self.neg_loss = neg_loss_cornernet

    def forward(self, out, target, mask=None):
        return self.neg_loss(out, target, mask=mask)
```

RegLoss

```
def _reg_loss(regr, gt_regr, mask):
```

```
 """
 regr:  [4,500,10]
 gt_regr:  [4,500,10]
 mask:  [4,500]
 """
    num = mask.float().sum()
    mask = mask.unsqueeze(2).expand_as(gt_regr).float() # [4,500,10]
    # ~ 按位取反，包括符号位。正数各位取反变为负数，显示时转化为其补码，负数本身需要先转换
    isnotnan = (~ torch.isnan(gt_regr)).float()
    mask *= isnotnan
    regr = regr * mask
    gt_regr = gt_regr * mask

    loss = torch.abs(regr - gt_regr)
    loss = loss.transpose(2, 0)

    loss = torch.sum(loss, dim=2)
    loss = torch.sum(loss, dim=1)
    # else:
    #   # D x M x B
    #   loss = loss.reshape(loss.shape[0], -1)

    # loss = loss / (num + 1e-4)
    loss = loss / torch.clamp_min(num, min=1.0)
    # import pdb; pdb.set_trace()
    return loss

def _gather_feat(feat, ind, mask=None):
 """
 feat :  [4,16384,10]
 ind  :  [4,500,10]
 """
    dim  = feat.size(2) # 10
    ind  = ind.unsqueeze(2).expand(ind.size(0), ind.size(1), dim) # [4,500,10]
    # tensor.gather(dim, indexs) 在dim维度上，按照indexs所给的坐标选择元素，返回一个
    feat = feat.gather(1, ind)
    if mask is not None:
        mask = mask.unsqueeze(2).expand_as(feat)
        feat = feat[mask]
        feat = feat.view(-1, dim)
    return feat

def _transpose_and_gather_feat(feat, ind):
 """
 feat :  [4,10,128,128]
 ind  :  [4,500,10]
 """
```

```python
    feat = feat.permute(0, 2, 3, 1).contiguous() # [4,128,128,10]
    feat = feat.view(feat.size(0), -1, feat.size(3)) # [4,16384,10]
    feat = _gather_feat(feat, ind)
    return feat

class RegLossCenterNet(nn.Module):
    """
    Refer to https://github.com/tianweiy/CenterPoint
    """

    def __init__(self):
        super(RegLossCenterNet, self).__init__()

    def forward(self, output, mask, ind=None, target=None):
        """
        Args:
            output: (batch x dim x h x w) or (batch x max_objects)
            mask: (batch x max_objects)
            ind: (batch x max_objects)
            target: (batch x max_objects x dim)
        Returns:
        """
        if ind is None:
            pred = output
        else:
         # 根据ind 选择 box预测
            pred = _transpose_and_gather_feat(output, ind)
        loss = _reg_loss(pred, target, mask)
        return loss
```

推理

`generate_predicted_boxes` 位于 `pcdet/models/dense_heads/center_head.py` 下分6个 `head` 遍历，根据热力图 `heartmap` 解码输出预测的 `box, score, lable`

```python
    def generate_predicted_boxes(self, batch_size, pred_dicts):
        post_process_cfg = self.model_cfg.POST_PROCESSING
        # POST_CENTER_LIMIT_RANGE:tensor([-61.2000, -61.2000, -10.0000,  61.2000
        post_center_limit_range = torch.tensor(post_process_cfg.POST_CENTER_LIMI

        ret_dict = [{
            'pred_boxes': [],
            'pred_scores': [],
            'pred_labels': [],
        } for k in range(batch_size)]
```

```python
        # 每个head遍历
        for idx, pred_dict in enumerate(pred_dicts):
            batch_hm = pred_dict['hm'].sigmoid() # 将值映射到0-1 torch.Size([4, 2
            batch_center = pred_dict['center'] # torch.Size([4, 2, 180, 180])
            batch_center_z = pred_dict['center_z'] # torch.Size([4, 1, 180, 180]
            batch_dim = pred_dict['dim'].exp() # torch.Size([4, 3, 180, 180])
            batch_rot_cos = pred_dict['rot'][:, 0].unsqueeze(dim=1)   # 扩展维度
            batch_rot_sin = pred_dict['rot'][:, 1].unsqueeze(dim=1)   # torch.Si
            batch_vel = pred_dict['vel'] if 'vel' in self.separate_head_cfg.HEAD
            # 根据heatmap解码输出pred_boxes, pred_scores, pred_labels
            final_pred_dicts = centernet_utils.decode_bbox_from_heatmap(
                heatmap=batch_hm,
                rot_cos=batch_rot_cos,
                rot_sin=batch_rot_sin,
                center=batch_center,
                center_z=batch_center_z,
                dim=batch_dim,
                vel=batch_vel,
                point_cloud_range=self.point_cloud_range,  # # [-51.2, -51.2,  -
                voxel_size=self.voxel_size, # torch.Size([4, 1, 180, 180])
                feature_map_stride=self.feature_map_stride, # 4
                K=post_process_cfg.MAX_OBJ_PER_SAMPLE,# 500
                circle_nms=(post_process_cfg.NMS_CONFIG.NMS_TYPE == 'circle_nms'
                score_thresh=post_process_cfg.SCORE_THRESH, # 0.1
                post_center_limit_range=post_center_limit_range # [-61.2000, -61
            )

            # 一个head多个类别
            for k, final_dict in enumerate(final_pred_dicts):
                # class_id_mapping_each_head: [tensor([0], device='cuda:0'), ten
                # tensor([6, 7], device='cuda:0'), tensor([8, 9], device='cuda:0
                final_dict['pred_labels'] = self.class_id_mapping_each_head[idx]
                if post_process_cfg.NMS_CONFIG.NMS_TYPE != 'circle_nms':
                    # nms过滤
                    selected, selected_scores = model_nms_utils.class_agnostic_n
                        box_scores=final_dict['pred_scores'], box_preds=final_di
                        nms_config=post_process_cfg.NMS_CONFIG,
                        score_thresh=None
                    )
                    final_dict['pred_boxes'] = final_dict['pred_boxes'][selected
                    final_dict['pred_scores'] = selected_scores
                    final_dict['pred_labels'] = final_dict['pred_labels'][select

                ret_dict[k]['pred_boxes'].append(final_dict['pred_boxes'])
                ret_dict[k]['pred_scores'].append(final_dict['pred_scores'])
                ret_dict[k]['pred_labels'].append(final_dict['pred_labels'])
        # 多个batch
```

```
        for k in range(batch_size):
            ret_dict[k]['pred_boxes'] = torch.cat(ret_dict[k]['pred_boxes'], dim
            ret_dict[k]['pred_scores'] = torch.cat(ret_dict[k]['pred_scores'], d
            ret_dict[k]['pred_labels'] = torch.cat(ret_dict[k]['pred_labels'], d

        return ret_dict
```

`decode_bbox_from_heatmap` 位于 `pcdet/models/model_utils/centernet_utils.py` 下

```
def _topk(scores, K=40):
    # 输入heatmap 1,1,180,180
    batch, num_class, height, width = scores.size() # 1, 1, 180, 180
    a= scores.flatten(2, 3) # torch.Size([1, 1, 16384]) 第3, 4维扁平化
    # 按scores降序排列，前k个分数及其索引
    # 假如scores: torch.Size([1, 2, 16384]) -> torch.Size([1, 2, 500])
    topk_scores, topk_inds = torch.topk(scores.flatten(2, 3), K) # torch.Size([1
    # 索引转为x,y坐标
    topk_inds = topk_inds % (height * width) # torch.Size([1, 1, 500])
    topk_ys = (topk_inds // width).float() # torch.Size([1, 1, 500])
    topk_xs = (topk_inds % width).int().float() # torch.Size([1, 1, 500])
    # 降序后的前k个大小的元素值及索引
    # 当一个任务task有多类，将多类的得分合并选取前K个最大得分及其索引
    topk_score, topk_ind = torch.topk(topk_scores.view(batch, -1), K) # torch.Si
    # 获取前K个最大得分的类别
    topk_classes = (topk_ind // K).int() # torch.Size([1, 500]) 都为0
    # 获取降序后的前K个topk_xs,topk_ys及索引topk_inds
    topk_inds = _gather_feat(topk_inds.view(batch, -1, 1), topk_ind).view(batch,
    topk_ys = _gather_feat(topk_ys.view(batch, -1, 1), topk_ind).view(batch, K)
    topk_xs = _gather_feat(topk_xs.view(batch, -1, 1), topk_ind).view(batch, K)


def decode_bbox_from_heatmap(heatmap, rot_cos, rot_sin, center, center_z, dim,
                             point_cloud_range=None, voxel_size=None, feature_ma
                             circle_nms=False, score_thresh=None, post_center_li
    batch_size, num_class, _, _ = heatmap.size() # torch.Size([4, 2, 180, 180])

    if circle_nms: # False
        # TODO: not checked yet
        assert False, 'not checked yet'
        heatmap = _nms(heatmap)
    # 降序计算前K个热力图计算得分，索引，类别，x，y
    scores, inds, class_ids, ys, xs = _topk(heatmap, K=K) # torch.Size([4, 500])
    # 根据索引计算center
    center = _transpose_and_gather_feat(center, inds).view(batch_size, K, 2) # t
```

```
    # 根据索引计算rot_sin
    rot_sin = _transpose_and_gather_feat(rot_sin, inds).view(batch_size, K, 1) #
    # 根据索引计算rot_cos
    rot_cos = _transpose_and_gather_feat(rot_cos, inds).view(batch_size, K, 1) #
    # 根据索引计算center_z
    center_z = _transpose_and_gather_feat(center_z, inds).view(batch_size, K, 1)
    # 根据索引计算dim
    dim = _transpose_and_gather_feat(dim, inds).view(batch_size, K, 3) # torch.S

    angle = torch.atan2(rot_sin, rot_cos) # torch.Size([4, 500, 1])
    xs = xs.view(batch_size, K, 1) + center[:, :, 0:1] # torch.Size([4, 500, 1])
    ys = ys.view(batch_size, K, 1) + center[:, :, 1:2] # torch.Size([4, 500, 1])
    # feature_map_stride = 4
    xs = xs * feature_map_stride * voxel_size[0] + point_cloud_range[0] # torch.
    ys = ys * feature_map_stride * voxel_size[1] + point_cloud_range[1] # torch.

    box_part_list = [xs, ys, center_z, dim, angle]
    if vel is not None:
        vel = _transpose_and_gather_feat(vel, inds).view(batch_size, K, 2) # tor
        box_part_list.append(vel) # xs, ys, center_z, dim, angle, vel

    final_box_preds = torch.cat((box_part_list), dim=-1) # torch.Size([4, 500, 9
    final_scores = scores.view(batch_size, K) # torch.Size([4, 500])
    final_class_ids = class_ids.view(batch_size, K) # torch.Size([4, 500])

    assert post_center_limit_range is not None
    # 根据预测box中心x,y,z和得分score过滤
    mask = (final_box_preds[..., :3] >= post_center_limit_range[:3]).all(2) # tc
    mask &= (final_box_preds[..., :3] <= post_center_limit_range[3:]).all(2)

    if score_thresh is not None: # 0.1
        mask &= (final_scores > score_thresh)

    ret_pred_dicts = []
    for k in range(batch_size):
        cur_mask = mask[k] # torch.Size([500])
        cur_boxes = final_box_preds[k, cur_mask] # torch.Size([292, 9])
        cur_scores = final_scores[k, cur_mask] # torch.Size([292])
        cur_labels = final_class_ids[k, cur_mask] # torch.Size([292])

        if circle_nms: # False
            assert False, 'not checked yet'
            centers = cur_boxes[:, [0, 1]]
            boxes = torch.cat((centers, scores.view(-1, 1)), dim=1)
            keep = _circle_nms(boxes, min_radius=min_radius, post_max_size=nms_p

            cur_boxes = cur_boxes[keep]
```

```
            cur_scores = cur_scores[keep]
            cur_labels = cur_labels[keep]

        ret_pred_dicts.append({
            'pred_boxes': cur_boxes,
            'pred_scores': cur_scores,
            'pred_labels': cur_labels
        })
    return ret_pred_dicts
```

Two-Stage

使用 `CenterPoint` 作为第一阶段。 第二阶段从骨干网的输出中提取额外的点特征。 我们从预测边界框的每个面的三维中心提取一个点特征。 注意，边界框的中心，顶部和底部的中心都投射到地图视图中的同一个点上。 因此，我们只考虑四个向外的框面和预测的目标中心。 对于每个点，我们使用双线性插值从主映射视图输出M中提取一个特征。接下来，我们将提取的点特征连接起来，并将它们通过一个 `MLP` 传递。 第二阶段在一级 `CenterPoint` 的预测结果之上预测一个类不可知的置信度得分和框的细化。

对于与 `class-agnostic` 的置信度分数预测，我们遵循并使用由框的 `3D IoU` 引导的分数目标和相应的 `ground truth` 边界框：

$$I=\min(1,\max(0,2\times IoU_t-0.5))$$

其中 IoU_t 是第 t 个提议框和 `ground truth` 之间的 `IoU` 。 训练由二元交叉熵损失监督：

$$L_{score}=-I_t\log(\hat I_t)-(1-I_t)\log(1-\hat I_t)$$

其中 $\hat I_t$ 是预测的置信度， 在推理，我们直接使用单阶段 `CenterPoint` 类别预测，并计算最终的置信度的几何平均， $\hat Q_t$ 是最后的预测目标 t 的置信度， $\hat Y_t=\max_{0\le k\le K}\hat Y_{p,k}$ ， $\hat I_t$ 分别是第一阶段和第二阶段目标t的置信度。

对于框回归，模型预测在第一阶段提议做出改进，我们用 $L1$ 损失训练模型。 我们的两阶段 `CenterPoint` 简化并加速了之前使用昂贵的基于 `PointNet` 的特征提取器和 `RoIAlign` 操作的两阶段 `3D` 检测器。

Experiments

在 `Waymo Open Dataset` 和 `nuScenes Dataset` 上评估 `CenterPoint` 。我们使用两种 `3D` 编码器实现 `CenterPoint: VoxelNet` 和 `PointPillars` ，分别被称为 `CenterPoint-Voxel` 和 `CenterPoint-Pillar` 。

Waymo Open Dataset. `Waymo Open Dataset` 包含798个训练序列和202个验证序列，用于车辆和行人。 点云包含激光雷达64道，对应每0.1s 180k点。 官方的三维检测评估指标包括三维包围框平均精度(mAP)和mAP加权方向精度(`mAPH`)。 `mAP` 和 `mAPH` 是基于0.7 IoU的车辆和0.5的行人。 对

于三维跟踪，官方指标是多目标跟踪精度(`MOTA`)和多目标跟踪精度(`MOTP`)。 官方评估工具包还提供了两个难度等级的性能分解： `LEVEL_1` 是包含5个以上激光雷达点的框， `LEVEL_2` 是包含至少1个激光雷达点的框。

我们的 `Waymo` 模型对X轴和Y轴的检测范围为 `[-75.2m, 75.2m]` ，对Z轴的检测范围为 `[2m, 4m]` 。 `CenterPoint-Voxel` 使用 `(0.1m, 0.1m, 0.15m)` 体素大小，遵循 `PV-RCNN` ，而 `CenterPoint-Pillar` 使用网格大小 `(0.32m, 0.32m)` 。

nuScenes Dataset. `nuScenes` 包含 `1000` 个驱动序列，分别有 `700、150、150` 个序列用于训练、验证和测试。 每个序列大约 `20` 秒长，激光雷达频率为 `20` `FPS` 。 数据集为每个激光雷达帧提供校准的车辆姿态信息，但每 `10` 帧 `(0.5s)` 只提供框标注。 `nuScenes` 使用 `32` 道激光雷达，每帧产生大约 `3` 万个点。 总共有 `28k, 6k, 6k` ，用于训练，验证和测试的注释框架。 这些注释包括10个具有长尾分布的类。 官方的评估指标是类别的平均水平。 对于 `3D` 检测，主要指标是平均平均精度 `(mAP)` 和 `nuScenes` 检测评分 `(NDS)` 。

- `mAP` 使用鸟瞰中心距离 `< 0.5m, 1m, 2m, 4m` ，而不是标准的框重叠。
- `NDS` 是 `mAP` 和其他属性度量的加权平均值，包括平移、比例、方向、速度和其他框属性。

在我们的测试集提交之后， `nuScenes` 团队添加了一个新的神经规划度量 `(PKL)` 。 `PKL` 度量基于规划者路线的KL散度(使用3D检测)和 `ground-truth` 轨迹来度量 `3D` 目标检测对下行自主驾驶任务的影响。 因此，我们也报告了在测试集上评估的所有方法的PKL度量。

对于 `3D` 跟踪， `nuScenes` 使用 `AMOTA` ，它会惩罚 `ID` 开关、假阳性和假阴性，平均超过各种召回阈值。

对于 `nuScenes` 的实验，我们将 `X、Y` 轴的检测范围设置为 `[51.2m, 51.2m]` ， Z轴是 `[5m, 3m]` 。 `CenterPoint-Voxel` 使用 `(0.1m, 0.1m, 0.2m)` 体素大小， `CenterPoint-Pillars` 使用 `(0.2m, 0.2m)` 网格。

Training and Inference. 我们使用与先前工作相同的网络设计和训练计划。 详细的超参数见补充。 在两阶段 `CenterPoint` 的训练过程中，我们从第一阶段的预测中随机抽取了 `128` 个正负比为 `1:1` 的框。 如果一个提议与至少 `0.55 IoU` 的 `ground truth` 注释重叠，则该提议是正样本。 在推断过程中，我们对非最大抑制 `(NMS)` 之后的前 `500` 个预测运行第二阶段。 推断时间是在 `Intel Core i7 CPU` 和 `Titan RTX GPU` 上测量的。

Main Results

3D Detection 我们首先在 `Waymo` 和 `nuScenes` 的测试集上展示我们的 `3D` 检测结果。 这两个结果都使用了一个 `CenterPoint-Voxel` 模型。 表1和表2总结了我们的结果。 在 `Waymo` 测试集上，我们的模型实现了 `71.8 level 2 mAPH` 的车辆检测和 `66.4 level 2 mAPH` 的行人检测，车辆和行人的 `mAPH` 分别比之前的方法提高了 `7.1%` 和 `10.6%` 。 在 `nuScenes` (表2)上，我们的模型在多尺度输入和多模型集成方面比去年的冠军 `CBGS` 高出 `5.2% mAP` 和 `2.2% NDS` 。 如后面所示，我们的模型也快得多。 补充材料包含了沿着类的细分。 我们的模型在所有类别中显示了一致的性能改进，并在小类别(交通锥 `+5.6 mAP`)和极端纵横比类别(自行车 `+6.4 mAP` ，施工

车辆 `+7.0 mAP`)中显示了更显著的改善。 更重要的是，我们的模型在神经平面度量(`PKL`)下显著优于所有其他提交的模型。 在我们的排行榜提交后。 这突出了我们框架的泛化能力。

Difficulty	Method	Vehicle mAP	Vehicle mAPH	Pedestrian mAP	Pedestrian mAPH
Level 1	StarNet [36]	61.5	61.0	67.8	59.9
	PointPillars [28]	63.3	62.8	62.1	50.2
	PPBA [36]	67.5	67.0	69.7	61.7
	RCD [5]	72.0	71.6	–	–
	Ours	**80.2**	**79.7**	**78.3**	**72.1**
Level 2	StarNet [36]	54.9	54.5	61.1	54.0
	PointPillars [28]	55.6	55.1	55.9	45.1
	PPBA [36]	59.6	59.1	63.0	55.8
	RCD [5]	65.1	64.7	–	–
	Ours	**72.2**	**71.8**	**72.2**	**66.4**

在这里插入图片描述

> 表 1： `Waymo` 测试集上 `3D` 检测的最新比较。 我们展示了 1 级和 2 级基准的 `mAP` 和 `mAPH` 。

Method	mAP↑	NDS↑	PKL↓
WYSIWYG [23]	35.0	41.9	1.14
PointPillars [28]	40.1	55.0	1.00
CVCNet [7]	55.3	64.4	0.92
PointPainting [49]	46.4	58.1	0.89
PMPNet [62]	45.4	53.1	0.81
SSN [68]	46.3	56.9	0.77
CBGS [67]	52.8	63.3	0.77
Ours	**58.0**	**65.5**	**0.69**

在这里插入图片描述

> 表 2： `nuScenes` 测试集上 3D 检测的最新比较。 我们展示了 `nuScenes` 检测分数 (`NDS`) 和平均平均精度 (`mAP`)。

Difficulty	Method	MOTA↑ Vehicle	MOTA↑ Ped.	MOTP↓ Vechile	MOTP↓ Ped.
Level 1	AB3D [48,53]	42.5	38.9	18.6	34.0
	Ours	**62.6**	**58.3**	**16.3**	**31.1**
Level 2	AB3D [48,53]	40.1	37.7	18.6	34.0
	Ours	**59.4**	**56.6**	[illegible]	**31.2**

在这里插入图片描述

> 表 3: `Waymo` 测试集上 `3D` 跟踪的最新比较。我们展示了 `MOTA` 和 `MOTP` 。 ↑ \uparrow 代表越高越好， ↓ \downarrow 代表越低越好。

Method	AMOTA↑	FP↓	FN↓	IDS↓
AB3D [53]	15.1	**15088**	75730	9027
Chiu et al. [10]	55.0	17533	33216	950
Ours	**63.8**	18612	[illegible]	[illegible]

在这里插入图片描述

> 表 4：nuScenes 测试集上 3D 跟踪的最新比较。我们展示了 `AMOTA` 、 `false positives` (FP)、 `false negatives` `(FN)` 、 `id switches (IDS)` 和每个类别的 `AMOTA` 。 ↑ \uparrow 代表越高越好， ↓ \downarrow 代表越低越好。

3D Tracking 表3显示了 `CenterPoint` 在 `Waymo` 测试集上的跟踪性能。我们在第 `4` 节中描述的基于速度的最接近距离匹配显著优于 `Waymo` 论文中的官方跟踪基线，后者使用基于卡尔曼滤波的跟踪器。我们观察到车辆和行人跟踪的 `MOTA` 分别提高了 `19.4` 和 `18.9` 。在 `nuScenes` (表4)上，我们的框架比上次挑战的获胜者 `Chiu et al.` 高出 `8.8 AMOTA` 。值得注意的是，我们的跟踪不需要单独的运动模型，运行时间可以忽略不计，比检测时间长1毫秒。

Ablation studies

Encoder	Method	Vehicle	Pedestrain	mAPH
VoxelNet	Anchor-based	66.1	54.4	60.3
	Center-based	**66.5**	**62.7**	**64.6**
PointPillars	Anchor-based	64.1	50.8	57.5
	Center-based	**66.5**	**57.4**	**62.0**

在这里插入图片描述

> 表 5：在 `Waymo` 验证集中基于锚点和基于中心的 `3D` 检测方法的比较。我们展示了每类和平均 `LEVEL 2 mAPH`

Encoder	Method	mAP	NDS
VoxelNet	Anchor-based	52.6	63.0
	Center-based	**56.4**	**64.8**
PointPillars	Anchor-based	46.2	59.1
	Center-based	**50.3**	**60.2**

在这里插入图片描述

> 表 6： `nuScenes` 验证中基于锚点和基于中心的 `3D` 检测方法的比较。我们展示了平均精度 (`mAP`) 和 `nuScenes` 检测分数 `(NDS)` 。

	Vehicle			Pedestrian		
Rel. yaw	0°-15°	15°-30°	30°-45°	0°-15°	15°-30°	30°-45°
# annot.	81.4%	10.5%	8.1%	71.4%	15.8%	12.8%
Anchor-based	67.1	**47.7**	45.4	55.9	32.0	26.5
Center-based	**67.8**	46.4	**51.6**	**64.0**	**42.1**	**35.7**

在这里插入图片描述

> 表 7：基于锚点和基于中心的方法检测不同航向角目标的比较。在第二行和第三行中列出了旋转角度的范围及其对应的目标部分，在 `Waymo` 验证集展示显示了这两种方法的 `LEVEL 2` `mAPH`

Method	Vehicle			Pedestrian		
	small	medium	large	small	medium	large
Anchor-based	58.5	**72.8**	64.4	29.6	60.2	60.1
Center-based	**59.0**	72.4	**65.4**	**38.5**	**69.5**	**69.0**

在这里插入图片描述

> 表 8：目标大小对基于锚点和基于中心的方法性能的影响。我们展示了不同大小范围内对象的每类 `LEVEL 2 mAPH` ：小 `33%` 、中 `33%` 和大 `33%`

Center-based vs Anchor-based 我们首先比较了基于中心的单阶段检测器和基于锚的同类检测器。在 `Waymo` 上，我们遵循最先进的 `PV-RCNN` 来设置 `anchor` 超参数：我们在每个位置使用两个 `anchor` ，分别为 `0°` 和 `90°` ，车辆的正/负IoU阈值为 `0.55/0.4` ，行人的 `0.5/0.35` 。在 `nuScenes` 上，我们遵循上一届挑战赛冠军 `CBGS` 的 `anchor` 分配策略。所有其他参数与我们的 `CenterPoint` 模型相同

如表5所示，在 `Waymo` 数据集上，简单地从 `anchor` 转换到中心， `VoxelNet` 和 `PointPillars` 编码器分别得到 `4.3 mAPH` 和 `4.5 mAPH` 的改进。 在nuScenes上(表6)， `CenterPoint` 通过不同主干 提升 `3.8-4.1 mAP` 和 `1.1-1.8 NDS` 。 为了了解改进的来源，我们进一步展示了基于 `Waymo` 验证集上的目标大小和方向角度的不同子集的性能细分

我们首先根据它们的方向角度将 `ground truth` 实例分为三个条：0°到15°，15°到30°，和30°到45°。 该部门测试检测器检测严重旋转的箱体的性能，这对安全部署自动驾驶至关重要。 我们还将数据集分为三个部分：小、中、大，每个部分包含1/3的地面真值框。

表7和表8总结了结果。 当框旋转或偏离框的平均大小时，我们基于中心的检测器比基于锚的基线性能要好得多，这证明了模型在检测目标时捕获旋转和大小不变性的能力。 这些结果令人信服地突出了使用基于点的 `3D` 目标表示的优势。

One-stage vs. Two-stage 在表9中，我们展示了在 `Waymo` 验证中使用 `2D CNN` 特征的单级和两级CenterPoint模型之间的比较。 具有多个中心特征的两级细化为两种3D编码器提供了很大的精度提升，开销较小(6ms-7ms)。 我们还与 `RoIAlign` 进行了比较， `RoIAlign` 对RoI中的6 × 6点进行了密集采样，我们基于中心的特征聚合取得了类似的性能，但速度更快、更简单。

Encoder	Method	Vehicle	Ped.	$T_{proposal}$	T_{refine}
VoxelNet	First Stage	66.5	62.7	71ms	-
	+ Box Center	68.0	64.9	71ms	5ms
	+ Surface Center	**68.3**	65.3	71ms	6ms
	Dense Sampling	68.2	**65.4**	71ms	8ms
PointPillars	First Stage	66.5	57.4	56ms	-
	+ Box Center	67.3	57.4	56ms	6ms
	+ Surface Center	**67.5**	57.9	56ms	7ms
	Dense Sampling	67.3	**57.9**	56ms	8ms

在这里插入图片描述

> 表 9：在 `Waymo` 验证集中使用单级、具有 `3D` 中心特征的两级和具有 `3D` 中心和表面中心特征的两级比较 `VoxelNet` 和 `PointPillars` 编码器的 `3D LEVEL 2 mAPH` 。

体素量化限制了两阶段 `CenterPoint` 对 `PointPillars` 行人检测的改进，因为行人在模型输入中通常只停留在1像素内。 在我们的实验中，两阶段细化并没有带来单阶段 `CenterPoint` 模型在 `nuScenes` 上的改进。 这部分是由于 `nuScenes` 中稀疏的点云。 `nuScenes` 使用32道激光雷达，每帧产生约3万个激光雷达点，约为 `Waymo` 数据集点数的1/6。 这限制了可获得的信息和两阶段改进的潜力。 在 `PointRCNN` 和 `PV-RCNN` 两阶段方法中也观察到类似的结果。

在这里插入图片描述

> 图 3： `Waymo` 验证集中 `CenterPoint` 的示例定性结果。 我们以蓝色显示原始点云，以绿色边界框显示我们检测到的对象，以红色显示边界框内的激光雷达点。

Effects of different feature components 在我们的两阶段 `CenterPoint` 模型中，我们只使用 `2D CNN` 特征图中的特征。 然而，以前的方法也提出利用体素特征进行第二阶段的精化。 在这里，我们比较两种体素特征提取基线

- `Voxel-Set Abstraction` ： `PV-RCNN` 提出了体素集抽象(`VSA`)模块，它扩展了 `Point-Net++` 的集合抽象层，以在一个固定半径球中聚合体素特征。
- `Radial basis function (RBF) Interpolation` ： `Point-Net++` 和 `SA-SSD` 使用径向基函数从三个最近的非空 `3D` 特征体聚合网格点特征。
  对于这两个基线，我们使用官方实现将鸟瞰视图特征与体素特征结合。 表10总结了结果。 这表明鸟瞰图特征足以提供良好的性能，同时与文献中使用的体素特征相比效率更高。
  为了与之前未对 `Waymo` 测试进行评估的工作进行比较，我们还在表11中报告了 `Waymo` 验证的结果。 我们的模型在很大程度上优于所有已发布的方法，特别是对于2级数据集具有挑战性的行人类(`+18.6 mAPH`)，其中框只包含一个激光雷达点

Methods	Vehicle	Pedestrian	Runtime
BEV Feature	68.3	65.3	77ms
w/ VSA [44]	68.3	65.2	98ms
w/ RBF Interpolation [20, 41]	**68.4**	**65.7**	[illegible]

在这里插入图片描述

> 表 10：两阶段细化模块的不同特征组件的消融研究。 `VSA` 代表 `Voxel Set Abstraction` ，这是 `PV-RCNN` 中使用的特征聚合方法。 `RBF` 使用径向基函数对 3 个最近邻进行插值。 我们在 `Waymo` 验证中使用 `LEVEL 2 mAPH` 比较鸟瞰图和 `3D` 体素特征。

Difficulty	Method	Vehicle mAP	Vehicle mAPH	Pedestrian mAP	Pedestrian mAPH
Level 1	DOPS [35]	56.4	_	_	
	PointPillars [28]	56.6	_	59.3	_
	PPBA [36]	62.4	_	66.0	_
	MVF [65]	62.9	_	65.3	_
	Huang et al. [24]	63.6	_	_	
	AFDet [14]	63.7	_	_	
	CVCNet [7]	65.2	_	_	
	Pillar-OD [52]	69.8	_	72.5	_
	PV-RCNN [44]	74.4	73.8	61.4	53.4
	CenterPoint-Pillar(ours)	76.1	75.5	76.1	65.1
	CenterPoint-Voxel(ours)	**76.7**	**76.2**	**79.0**	**72.9**
Level 2	PV-RCNN [44]	65.4	64.8	53.9	46.7
	CenterPoint-Pillar(ours)	68.0	67.5	68.1	57.9
	CenterPoint-Voxel(ours)	**68.8**	**68.3**	**71.9**	[illegible]

在这里插入图片描述

> 表 11：Waymo 验证集中 3D 检测的最新比较。

3D Tracking. 表12显示了基于nuScenes验证的三维跟踪消融实验。 我们与去年的挑战赛冠军Chiu et al.进行了比较，后者使用基于马氏距离的卡尔曼滤波来关联CBGS检测结果。 我们将评估分解为检测器和跟踪器，使比较严格。 对于相同的检测目标，使用简单的基于速度的最近点距离匹配比基于卡尔曼滤波的马氏距离匹配的效果要好3.7 AMOTA(第1行vs. 3行，第2行vs. 4行)。 有两个改进的来源：

- 用学到的点速度建模物体运动，而不是用卡尔曼滤波器建模三维包围框动态；
- 通过中心点距离来匹配目标，而不是框状态的马氏距离或3D边界框IoU。
- 更重要的是，跟踪是一个简单的最近邻匹配，没有任何隐藏状态计算。 这节省了3D卡尔曼滤波器的计算开销

编辑于 2023-03-31 16:58 ・IP 属地浙江