## Bevformer

### 1.1 DETR

1. 二分图匹配确定有效的预测框：匈牙利算法（N>M如何匹配）
2. 问题：正向推理怎么办？

DETR（DEtection TRansformer）在前向推理时，会生成固定数量的预测框。通常情况下，默认是100个。这100个预测框中，可能有一部分会被分类为“无目标”（no object），而其余的预测框会根据模型的预测对应到具体的类别和位置信息。

这是因为DETR采用了一种基于查询的机制，它会对输入图像生成固定数量的查询，然后通过Transformer解码器生成对应的预测框。即使图片中物体的实际数量少于100个，模型也会输出100个预测框，其中一些可能会被标记为无效。

如果你需要调整这个数量，可以通过修改DETR模型的num\_queries参数来改变生成的预测框数量。



1. 

Transfomer训练和测试环节输入有很大不同！

Transfomer的decoder的输入输出是错位的。

输入： S I want a beer

输出： I want a beer .



### Deformable-Detr

可变形注意力：

可变形注意力机制结合了稀疏注意力的思想和动态偏移的创新。它通过生成动态偏移来调整采样位置，从而在稀疏注意力的基础上实现更加灵活的注意力分配。具体来说，可变形注意力机制首先通过稀疏选择减少计算范围，然后在这些关键位置上应用动态偏移，以获取更加准确的注意力权重。

PS：注意！

encoder中reference\_points是直接采样（函数get\_reference\_points）获得，相当于没有进行稀疏选择,但是decoder中reference\_points是学习获得，所以这里进行了稀疏选择

重点在于只和采样偏移的点进行了多头注意力运算，所以重点在于选择采样点





问题: query feature怎么获得？是一个序列还是一群序列？图像中实际上只显示了一个

Reference point每个对应一个query

这个是在单尺度（也就是一个特征图），多尺度如下







Nlevels是特征图scale的个数（原论文中4个）。Locations以下靠CUDA代码



上面的运算流程是用CUDA代码实现的，改不了

深入理解full attention的公式：



token,multi-head(其实是直接把feature拆分)

参考：[Deformable DETR ｜ 2、backbone 、MultiHeadAttention 公式讲解\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1vr4y1o73B?spm_id_from=333.788.videopod.sections&vd_source=d454dd668337bd1b6955178189b20c20)



上面的公式其实是指定了某一个token作为query时的计算结果，为6x1（实际上还要重复这个操作然后计算每个token作为query的）

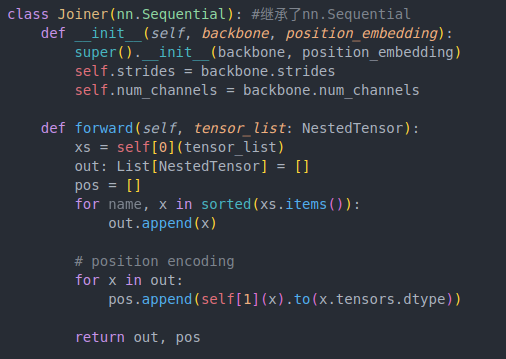


应为代码配置不好，现在BEV Former先放放

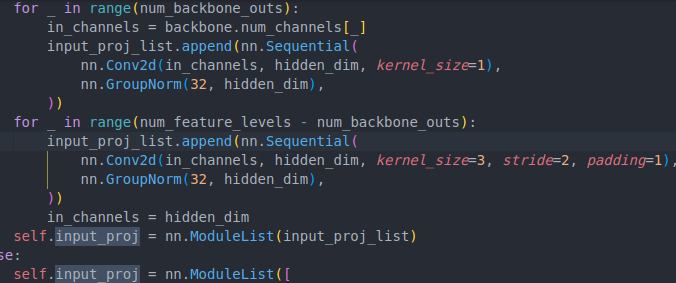
Bev Former原论文在A100上配置，4090配置挺麻烦的

deformabel-detr运行记录：

**backbone部分**

bacbone采用resnet101,position\_embbeding采用行+列编码

backbone的out采用orderedict，应为需要输出4中尺寸不同的特征图（通过out[0,1,2,3]来保存，不能放在batch中，应为特征图大小不同）

input\_proj采用索引方式：nn.ModuleList.append(nn.sequential)

encoder的输入详解:

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 格式 |
| src\_flatten | 变为token的图片 | torch.Size([2, 12320, 256]) |
| spatial\_shapes | 特征图的大小 | torch.Size([4, 2]) |
| level\_start\_index | 每个特征图开始的索引 | torch.Size([4]) |
| valid\_ratios | 每个图片有效数据（填充） | torch.Size([2, 4, 2]) |
| lvl\_pos\_embed\_flatten | 位置编码 | torch.Size([2, 12320, 256]) |
| mask\_flatten | 掩码 | torch.Size([2, 12320]) |

说明：batch =2,d\_model=256,level=4

|  |
| --- |
| **get\_reference\_points函数详解：**  在 PyTorch 中，使用 None 可以在张量的特定位置添加一个新的维度。这种操作也被称为“维度扩展”或“升维”。在你的代码中，None 被用来扩展张量的维度，以便进行广播操作。（为啥不直接用unsqueeze） |

接下来就是使用MS\_deform\_attn函数的使用方法：

|  |  |  |
| --- | --- | --- |
| 声明参数 | 含义 | 说明 |
| d\_model | token的维度 | 和输入一致，如256 |
| n\_levels | 特征图数量 | 和输入一致，如4 |
| n\_heads | 感知头数量 | 如8 |
| n\_points | 方向点数量 | 如4 |
| 输入参数 | 含义 | 说明 |
| query | 图像组成的token+pos\_em | torch.Size([2, 12320, 256]) |
| reference\_points | 图像采样的参考点 | \*torch.Size([2, 12320, 4, 2]) |
| input\_flatten | 图像组成的token | torch.Size([2, 12320, 256]) |
| input\_spatial\_shapes | 每个特征图的尺度 | torch.Size([4, 2]) |
| input\_level\_start\_index | 每个特征图起始索引 | torch.Size([4]) |
| input\_padding\_mask | 图像的掩码（部分pad） | torch.Size([2, 12320]) |
| 输出 | 含义 | 说明 |
| output | 最终输出 | torch.Size([2, 12320, 256]) |

说明：特征图尺寸为4

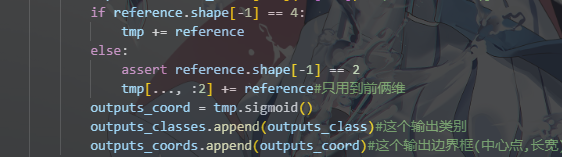
\*表示encoder和decoder是进行的refernece\_points生成策略有所不同

Encoder使用get\_reference\_points产生

下面给出decoder中cross attention调用的参数

|  |  |  |
| --- | --- | --- |
| 输入参数 | 含义 | 说明 |
| tgt | 查询向量query | torch.Size([2, 300, 256]) |
| reference\_points | 稀疏采样点 | \*torch.Size([2, 300, 4, 2]) |
| src | 图像特征的token（value） | torch.Size([2, 12320, 256]) |
| src\_spatial\_shapes | 每个特征图的尺度 | torch.Size([4, 2]) |
| level\_start\_index | 每个特征图起始索引 | torch.Size([4]) |
| src\_padding\_mask | 图像的掩码（部分pad） | torch.Size([2, 12320]) |
| 输出 | 含义 | 说明 |
| tgt2 | 最终输出 | torch.Size([2, 300, 256]) |

可以看到，最终的输出大小tgt2.shape==tgt.shape，表示查询的结果



最终边界框在outputs\_coord中，不包含置信度（不需要NMS）

最后详细学习一下如何计算loss

|  |
| --- |
| **Loss的计算方法**  这里还涉及到匈牙利损失和二分匹配算法  第一步：二分图匹配  表示ground truth集合(用控物体填充)，表示预测的集合 |
| 没有置信度分数，所以也没办法进行NMS，只能对每一个框进行遍历，取最小的损失值预测框作为真实匹配框  <https://blog.csdn.net/weixin_47936614/article/details/141398813>  计算类别损失  目标检测中会采用Focal Loss,专门用于处理类别不平衡问题的损失函数    目标检测框的损失是mxn计算的，所以最后结果都是返回mxn的矩阵  分别计算L1损失和giou损失    最后匈牙利算法进行二分图匹配（直接调用linear\_sum\_assignment函数）   * i 是预测边界框的索引。 * j 是目标边界框的索引。   第二步：计算匈牙利损失  定义的损失类似于常见目标检测器的损失    代码中实现实际上由4种loss组成  具体4种loss计算方法目前先不展开学习 |

### 1.3 BevFusion



Lidar的处理不特别，Camera的view transform部分是重点

### Lane Detection

目前车道线检测基本都是纯视觉的，因为Lidar的车道线检测数据不是很明显，最多只能和视觉融合补充

|  |
| --- |
| 创新点提取：   1. 车道线检测的静态结构约束，参考StructLane,可以捕捉车道的结构 2. 深度估计模块，参考bevdepth(需要激光雷达的数据),可以缓解地面不平的问题，也可以通过多帧数据时序融合的方法（带有一定的深度估计信息）（还可以画出深度图） 3. 时序融合的方法，这个参考PF-Track的anchor传递过程以及anchor3Dlane的方法，车道线本身是静止的，训练时应该捕捉前后两帧车道线的信息 4. 三阶段的语义分割方法，思路来自于PID-Net，全局和局部信息的提取有助于目前虚实属性的检查效果（双阶段的已经被3D-lanenet用过了），车道线既涉及到分类也涉及到分割，需要同时融合局部和全局信息，通过PID-Net想办法融合进DA   最终计划做的创新点：   1. 引入Lidar进行深度监督，参考BevDepth 2. 引入时序anchor更新策略，参考Anchor3DLane 3. 引入PV和BEV的融合策略，方法待定 4. 引入车道线后处理策略，参考PolyLaneNet   数据集选择：  比较的方法：3D-LaneNet,Gen-LaneNet,SALAD,PersFormer,Anchor3DLane, 3D-GeoNet,LaneCPP,BEV-LaneDet  Anchor3DLane:OpenLane,ONCE-3DLanes  CureFormer:OpenLane,ONCE-3DLanes  GenLaneNet:Apollo Synthetic  LaneCPP\*:OpenLane,Apollo 3D Synthetic  Persformer:OpenLane,Apollo 3D Synthetic |

多感知任务的系统：

RepVF:多感知任务，Waymo和OpenLane数据集

PETRv2:多感知任务，nuscenes，OpenLane和Waymo数据集

Bevformer:多感知任务，nuscenes, Waymo数据集

Bevfusion：多感知任务，nuscenes,Waymo数据集

Waymo数据集至少2TB，空间可能不太够

BEV相关的在nuscenes上都可以测试

单车道线检测的系统：

PersFormer:车道线检测，OpenLane数据集(尝试部署一个子集)

BEV-LaneDet:车道线检测，OpenLane和Apollo数据集

Anchor3Dlane:车道线检测，OpenLane和Apollo数据集

回去后考虑下载waymo的子集以供学习使用

Nuscenes上Lane的标注基本都是2D的，所以用的人不多

整理一下，目前在我的机器上可以运行的有：

BevFormer,PETRv2,

可以下载一个openlane的子集，然后跑一个inference

2024最新工作：

Lanecpp,Clrernet,Lane2seq 全是CVPR

还有两个数据集：CULane和TuSimple 一般是IEEE TVT使用的

|  |
| --- |
| 数据处理流程  参考MAL数据湖  从原始的标注数据进行前处理后获得GT表  数据处理的pipeline构造方法是套用的MMCV的模块构建module函数  Gt具有深度标注信息（深度gt信息在bev视角下，通过4D场景重建获得，方法类似于3DGS，所以各方面精度很高。nuscenes想获得深度信息比较困难，只能把gt数据通过IPM坐标变换到BEV视角下，数据处理上会简单很多）  很多数据集上并没有2D标注，但是可以通过3D标注产生  Momenta纯视觉方案的静态感知  FO任务：提升车道线宽窄和虚实属性的检测效果  方法：  1.洗数据，看哪些数据上模型训练效果比较差  2.写算法，通过特殊的监督检测模块判断宽窄线  Apollo数据集中的车道线只定义了4种类别：    黄白的虚实属性，但是persformer的代码支持大概11种车道线类别 |

|  |
| --- |
| **BEV-LaneDet**    仅使用前视相机，主要是Waymo5个相机基本视角不重叠  IPM:从图像坐标系投影到世界坐标系的整个过程（假设严苛，应为深度不明）   1. Virtula Camera   统一单位焦平面，便于多个数据集不同内参的相机使用（其实也没啥用？就是相机内参只需要在最开始参与运算，中间环节的视角变换矩阵可以公用）  深入了解相机内外参数，比如焦距等系数的作用    四个坐标系：世界坐标系，相机坐标系，图像物理坐标系，图像像素坐标系  世界坐标系-刚体变换-相机坐标系-透视投影-图像坐标系-仿射变换-像素坐标系  有的时候会绘制一个虚拟像平面，注意：所有世界坐标系的物体必须通过相机坐标系的光心才能投影到相平面上，因此一定可以形成上图一样的连线关系。相机生成的图像实际上是被倒置的  单位焦平面就是焦距统一为1，透视投影矩阵固定   1. Key points的方法   每太明白，或许还应该去看看代码 |

|  |
| --- |
| **PersFormer**    PersFormer还包括一个OpenLane数据集的贡献  注意这里的，指代的相机外参extrinsic是指相机光心到车辆中心的转换矩阵，车辆运行过程中保持不变（所谓的ego2cam）,不是常规意义上的车辆中心到世界坐标系的转移矩阵（ego2world pose）  目前还没有开源的多时序融合的车道线检测方法（有一个CurveFormer++）  目前提取创新点：   1. 车道线检测的静态约束，这个目前还没有具体思路 2. 深度估计模块，参考bevdepth(需要激光雷达的数据),可以缓解地面不平的问题，也可以通过多帧数据时序融合的方法 3. 时序融合的方法，这个参考PF-Track的anchor传递过程 4. 三阶段的语义分割方法，思路来自于PID-Net，全局和局部信息的提取有助于目前虚实属性的检查效果（双阶段的已经被3D-lanenet用过了）   发表在期刊上，特点是：低存储，高速率，高精度  PersFormer代码详解：  数据集详解：  Apollo数据集的label中包含：  cameraHeight,cameraPitch都是确定相机标定的数据，可以用来计算内外参  laneList当前车道中心线，laneboundaryList车道边界线  模型详解：  单应性矩阵  可以把一个平面上的点转移到另一个平面上的点（注意：要求都是平面，或者都缺少一个维度信息） 假设平面坐标Z=0，计算单应性矩阵需要两个平面中的4个点对  这里可能有三个平面：图像平面，像素平面，地面平面（近似为平面）    地面坐标系：    可以看到范围是x [-10,10],y [3,103] 非常合理  IPM（BEV）视角：  右上角的矩阵，基本上采用了左上角为00的CV2坐标系规则，便于成图  后面还会使用计算像素平面到地面平面的单应性矩阵  单应性矩阵可参考[单应性矩阵的理解及求解\_图像处理与深度学习-CSDN专栏](https://download.csdn.net/blog/column/10559636/118371131)  Apollo数据集没有预测车道线的类别（我说为什么识别度那么高）    Load\_Data.py的WIP\_getitem函数需要细读，了解Apollo数据集的获取方式  其中gt\_laneline\_img的理解尤其困难，包含了很多数据信息    以及如何使用gt\_anchor?这个也是论文中的一个重点，这里应该来自于标注  尤其注意如何产生gt\_anchor以及如何计算loss等等，后面Anchor3Dlane同理的  Gt\_anchor的思想直接来自于论文3D-LaneNet和GenLanenet    图中的anchor是x,y轴均匀分布，然后再Yref上判断GT和Anchor的匹配关系  几个矩阵和坐标系的说明：  g:ground地面坐标系，表示物体的3D坐标所在的坐标系，车辆自身坐标系  几个变量的说明：  Gt\_lane\_img:表示图像上每条线的2D像素位置以及类别（但是没有3D位置）    Gt\_anchor：设置的车道线检测锚点，最为重要的数据结构（生成方法很复杂）   1. init\_dataset\_3D   这个函数应该是抄袭3D\_lanenet，    总之特别复杂  关注一下anchor的实现函数    这里实现了如何获得  在BEV space下创建anchors，（也就是ground坐标系，y向前x向右）  进行插值获得y\_steps的代码    Gt\_anchor本身也具有物理含义，反应了物体应该的位置，逻辑上和动态物体检测的anchor bbox有很多相似之处  Gt\_anchor包含：所有属性都是位于BEV（ground坐标系）  每条线的x\_off\_values,每条线的z\_values，每条线的visibility,  每条线的数据位于数组中ass\_id的索引下面  计算细节：  Xbev范围[-10,10],共计16个采样点；同时还有7个角度，共计112个x  Y范围[3,103]，共计10个采样点  看样子已经舍弃了Yref的方法（重合的线会识别报错）    Self.anchor\_grid\_x 112x10,x\_values 10  论文中提到的两个问题：1.anchor具有额外的7个角度 2.没有使用Yref   1. anchor\_grid\_x     本来设定的steps有16个+不同角度的7个，但是这里好像和论文中的arctan对应不上.注意：这里论文中提到了3Danchor设置的是倾斜角（incline anchor），其实是2D空间中zox平面上直线和x的夹角  个人感觉创行点主要是直接在BEV视角设定anchor，从而避免了IPM矩阵的使用，相当于让网络学习IPM矩阵的方法   1. 模型部分   模型输入只需要图片和IPM变换矩阵（辅助初始估计）      与这张图基本保持一致，·self.encoder是EfficientNet-B7  Input 8,3,360,480 -> out\_featList 8,32,180,240 注意：特征图仅有一个尺度  Self.neck 是一个sequential,neck\_out 8,128,180,240  Self.laneatt应该是从laneattnet偷来的2D lane detection的代码，内容差不多就是2D目标检测的回归方法，具体如下      112 anchor数量（图片中可能出现lane的位置），2种类（Apollo只分割车道线或者背景），144车道线对应的采样x\_values（y是samples获得的）    注意：预测的结果还是有一系列pre-defined值      Value从frontview\_features获得，K在DA中会产生    \_M\_inv只用来产生ref\_3d的初始值（所以其实没有在ipm平面上进行任何操作）  但是ref\_2d,ref\_3d都是2维，原版的DA中也没有ref2d,ref3d区别  自注意力使用ref\_2d，query自己交互;交叉注意力使用ref\_3d，query和feature交互  最终的lane\_out模型：  从bev\_feat运算得到3个预测结果    这里fmap也就是bev\_feat也要使用6个角度偏移  （对应的是2D上的anchor偏移）  Lan\_out负责3D Lane Seg，是最难理解的地方  Loss计算部分：   1. 使用到了一个uncertainty weigh loss，有一篇相关论文（但是代码中没有用上） 2. 其他相关的loss     由于一个gt和一个anchor绑定在一起，所以不需要nms和匈牙利算法  问题：  1.沿着y均匀采样并不合理，实际中同一个车道线可能有同一个y的点  2. |

|  |
| --- |
| **Anchor3Dlane**     1. 将FV转换到VEV然后做Detection（但是Lane的Height预测不准会导致地面平坦假设失效） 2. 将FV添加深度然后做Detection（但是   本论文采用了Bev-free的方法，本质上是第二种方法（把anchor拉回到FV平面进行特征提取，其实还是利用2D信息，同时同步完成了深度估计）    （论文创新点和LATR基本相同） |

|  |
| --- |
|  |

## 3D Object Detection

选取几个开源的代码跑通，分别有：camera,lidar两个方案

目前先运行Mono3D

区别于语义分割的点

### 2.1 Detr3D

环境配置问题：mmcv,mmdet,mmseg，mmdet3d均可以安装在4090 CUDA11.8上

但是不能参照bevformer的md，会报错，自己去看对应的官网

问题：2.0.0以上的版本会有很多报错，移除了很多文件

Detr3d的代码是在3090上运行的，安装的库最好是老版本

4090安装的版本如下：



为了运行较老代码，mmcv版本为：

pip install mmcv-full==1.4.0 （安装不了full，但是可以安装mmcv 1.4.0）

pip install mmsegmentation==0.14.1

pip install mmdet == 2.14.0

但是mmdet3d必须从源代码安装，版本为0.17.1



[出现错误“subprocess.CalledProcessError: Command ‘[‘ninja‘, ‘-v‘]‘ returned non-zero exit status 1”解决方法\_subprocess.calledprocesserror: command '['ninja', -CSDN博客](https://blog.csdn.net/fq9200/article/details/125362088)

感觉还是pytorch版本太高，没有其他办法（CUDA 11.8只能安装2.0.0的），已经老实

### BevFusion

BevFusion可以尝试配置

注意：openMPI需要管理员权限sudo



已经老实，BevFusion需要MMCV 1.4.0，最好是3090或者A100，4090根本不行

看样子最好得自己配置一个docker镜像

尽力了，安了CUDA 11.7都没用

### CLOCS

结果级别的融合方案，非常有意思

先看NMS

这些都先放着，这里的电脑跑不了

One-stage和two-stage



2.。

### 语义分割

语义分割目前只有一篇cam+lidar的融合方法，其他基本都是基于Lidar

### SMOKE

以这篇作为mono3D的学习入门方法

中间提到了RPN （region proposal network）——这个方法首次来自于Faster-RCNN(详情请见下面)

首先梳理相机内参和外参的作用





内参反映了像素坐标系和相机坐标系的坐标变换关系

外参反映了相机坐标系和世界坐标系的坐标变换关系

另外注意kitti数据集的理解：

[KITTI 3D目标检测数据集解析（完整版）\_kitti数据集-CSDN博客](https://blog.csdn.net/qq_16137569/article/details/118873033)

3D框标注

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 字段长度 | 单位 | 含义 |
| Type | 1 | - | 目标类型 |
| Truncated | 1 | - | 目标截断程度：0~1之间的浮点数，表示目标距离图像边界的程度 |
| Occluded | 1 | - | 目标遮挡程度：0~3之间的整数，0：完全可见 1：部分遮挡 2：大部分遮挡 3：未知 |
| Alpha | 1 | 弧度 | 目标观测角： |
| Bbox | 4 | 像素 | 目标2D检测框位置：左上顶点和右下顶点的像素坐标 |
| Dimensions | 3 | 米 | 3D目标尺寸：高、宽、长 |
| Location | 3 | 米 | 目标3D框底面中心坐标：( x , y , z ) (x, y,z)(x,y,z)，相机坐标系 |
| Rotation\_y | 1 | 弧度 | 目标朝向角： |

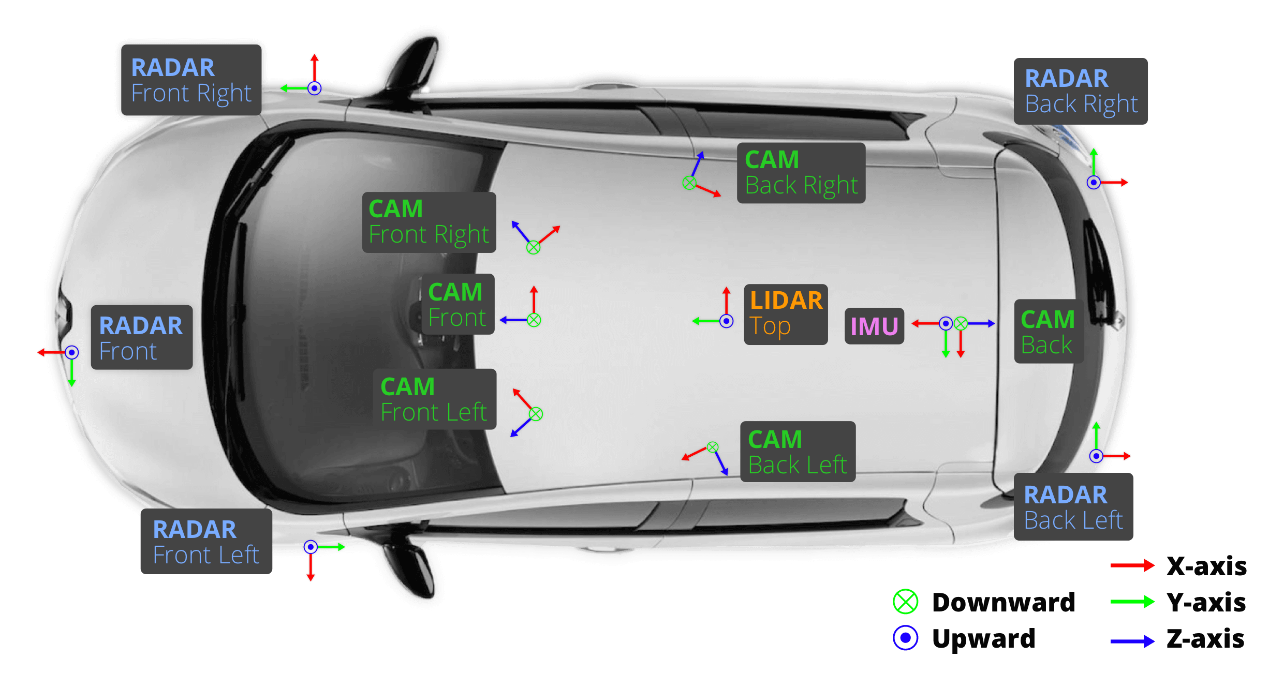
注意：Alpha和Rotation\_y可以相互转化，关系如下：（这里是相对于数据车的相机作为参考）



偏航角的计算方法：

### 2.6 Metrics

首先学习nuScenes tracking task和指标的含义



* [目标检测](https://so.csdn.net/so/search?q=目标检测&spm=1001.2101.3001.7020)(object Detection)：是指在图像或视频帧中识别并定位一个或多个感兴趣的目标对象的过程 。
* 目标跟踪(object Tracking)：是指在**视频帧序列中**连续地监测和定位一个或多个**目标对象的位置**的过程。

具体区别：

1.输入数据不同：目标检测处理单一图像，目标跟踪处理视频序列（多帧图像）

2.目标不同：目标检测关注于定位物体，目标跟踪关注于物体的变化（还有ID）

数据格式：

|  |
| --- |
| └── NUSCENES\_TRAINVAL\_DATASET\_ROOT  ├── samples <-- key frames  ├── sweeps <-- frames without annotation  ├── maps <-- unused  ├── lidarseg <-- lidar semantic labels  └── v1.0-trainval <-- metadata and annotations  └── NUSCENES\_TEST\_DATASET\_ROOT  ├── samples <-- key frames  ├── sweeps <-- frames without annotation  ├── maps <-- unused  ├── lidarseg <-- lidar semantic labels  └── v1.0-test <-- metadata  说明：  1.samples所有传感器的数据关键帧（包括cam+lidar+radar）  2.sweeps所有传感器的数据插帧（没有标注，但是会用来作为补充数据帧）  3.maps场景的地图（一般用不上）  4.v1.0-trainval所有数据标注文件      说白了，sample代表一个关键帧，sample\_data代表这个关键帧所有传感器的数据 |

学习模型：

BEVFusion,TransFusion,SImpleTrack,Tracker,Sparse4D,BEVTrack

表一 多模态

|  |  |  |
| --- | --- | --- |
| 名称 | 数据 | 结构 |
| BEVFusion | cam+lidar |  |
| TransFusion | Cam+lidar |  |
| MCTack | Cam+lidar |  |
| MV2DFusion | Cam+lidar |  |

表二 视觉

|  |  |  |
| --- | --- | --- |
| 名称 | 数据 | 结构 |
| Sparse4D | cam |  |
| \*SparseAD | cam |  |

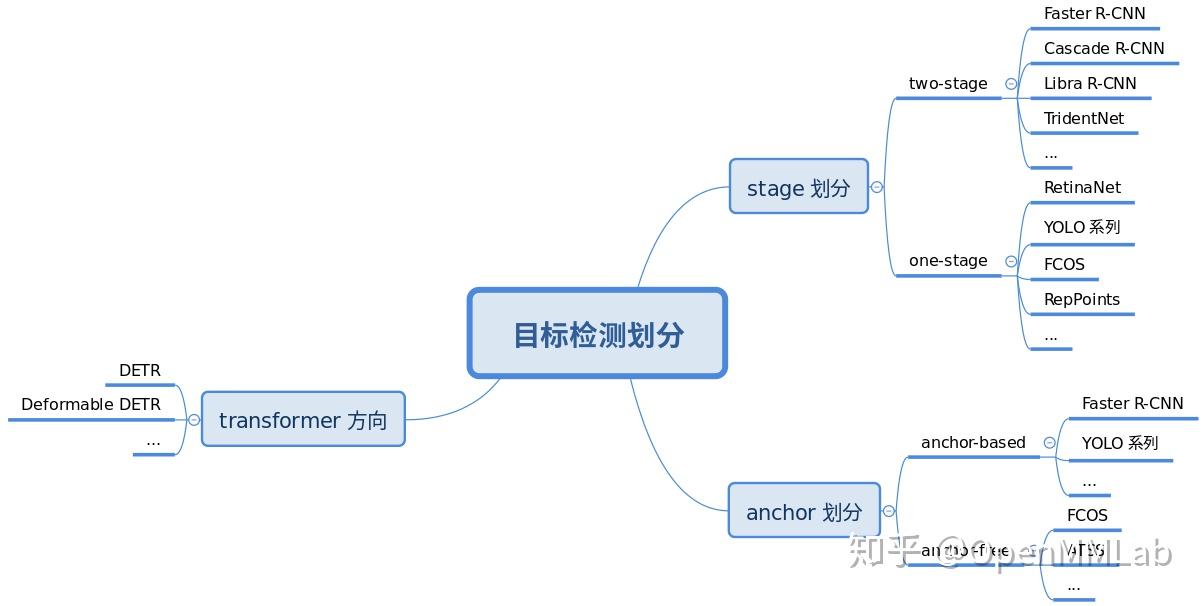
表三 激光雷达

|  |  |  |
| --- | --- | --- |
| 名称 | 数据 | 结构 |
| SimpleTrack | lidar |  |
| BEVTrack | lidar |  |
| LEGO | lidar |  |

|  |
| --- |
| **总结计划**  1.首先研究基于检测的方法，包括但不限于：sparse4d,transfusion  2.然后研究跟踪的方法，总结检测和跟踪的区别  3.最后探索多传感器融合的跟踪方法，并扩展到openPCD框架 |

### MM-openlab框架学习

|  |
| --- |
| MMCV  **配置**  学习 from mmcv import Config的用法，包括：  配置config.py，继承关系  **注册器**  学习from mmcv import Registry的用法，包括：  如何创建注册器的子注册器，例如：  所有下游代码库中所有MODELS注册器都是MMCV MODELS注册器的子注册器  替代原本pytorch注册神经网络的方法  不同Registry的实例的区别是：其实都是MMCV\_MODELS的实例  严格来说使用方法没有区别  **执行器**  学习runner类的用法  **卷积神经网络**  BaseModule 继承自 torch.nn.Module, 它们之间唯一的不同是 BaseModule 实现了 init\_weight  Sequential 继承自 BaseModule 和 torch.nn.Sequential  ModuleList 继承自 BaseModule 和 torch.nn.ModuleList  Init\_cfg是用来初始化模型的  **Hook**  钩子编程（hooking），也称作“挂钩”，是计算机程序设计术语，指通过拦截软件模块间的函数调用、消息传递、事件传递来修改或扩展操作系统、应用程序或其他软件组件的行为的各种技术。处理被拦截的函数调用、事件、消息的代码，被称为钩子（hook） |



|  |
| --- |
| **MMDetection**    训练过程：   1. 任何一个 batch 的图片先输入到 backbone 中进行特征提取，典型的骨干网络是 ResNet 2. 输出的单尺度或者多尺度特征图输入到 neck 模块中进行特征融合或者增强，典型的 neck 是 FPN 3. 上述多尺度特征最终输入到 head 部分，一般都会包括分类和回归分支输出 4. 在整个网络构建阶段都可以引入一些即插即用增强算子来增加提取提取能力，典型的例如 SPP、DCN 等等 5. 目标检测 head 输出一般是特征图，对于分类任务存在严重的正负样本不平衡，可以通过正负样本属性分配和采样控制 6. 为了方便收敛和平衡多分支，一般都会对 gt bbox 进行编码 7. 最后一步是计算分类和回归 loss，进行训练 8. 在训练过程中也包括非常多的 trick，例如优化器选择等，参数调节也非常关键   MMDetection中实现的部分  1.backbone:mmdet/models/backbones  *# 骨架的预训练权重路径*  pretrained**=**'torchvision://resnet50',  backbone**=**dict(  type**=**'ResNet', *# 骨架类名，后面的参数都是该类的初始化参数*  depth**=**50,  num\_stages**=**4,  out\_indices**=**(0, 1, 2, 3),  frozen\_stages**=**1,  norm\_cfg**=**dict(type**=**'BN', requires\_grad**=**True),  norm\_eval**=**True,  style**=**'pytorch'),  2.neck: mmdet/models/necks  neck**=**dict(  type**=**'FPN',  in\_channels**=**[256, 512, 1024, 2048], *# 骨架多尺度特征图输出通道*  out\_channels**=**256, *# 增强后通道输出*  num\_outs**=**5), *# 输出num\_outs个多尺度特征图*   1. 训练测试算法流程：     在 MMDetection 中 DataLoader 输出的内容不是 pytorch 能处理的标准格式    前面说个，训练和验证的时候实际上调用了 model 内部的 train\_step 和 val\_step 函数 |

|  |
| --- |
| **MMDetection3d** |

|  |
| --- |
| **3D Multiple Object Tracking ( 3D MOT)**  Nuscenes数据集中包含tracking相关的数据 |

### 2.8 Sparse4D v1

|  |
| --- |
| **环境配置问题：**  1.需要利用mmdet3D库处理数据集  数据集处理的命令  python tools/create\_data.py nuscenes --root-path /data/wangzhaohui/github/Sparse4D/data/nuscenes --out-dir /data/wangzhaohui/github/Sparse4D/data/nuscenes\_cam --extra-tag nuscenes  主要是为了产生几个pkl文件  2.其他一堆问题 |

|  |
| --- |
| **核心流程**      核心代码就是forward\_train，代表了模型的前向传播过程    Nuscenes每帧数据  Img\_metas:4\*4矩阵，T\_global，代表目前车辆的世界坐标系位姿  Gt\_bboxes\_3d: LiDARInstance3DBoxes,代表每个3D框在雷达坐标系的位姿  总之很麻烦  具体流程：  **1.特征提取部分**    最终提取的feature\_maps是4个不同尺度对应6张图像的特征图  **2.检测头head部分**  这部分的重点在于理解提出的Deformabel 4D Aggregation模块基本结构 |

### 2.9 Sparse4D v3

V3和v1差别还是比较大的，需要结合v2一起看

|  |
| --- |
| **核心流程**    Input:img,data  V3输入的是连续6个关键帧的数据！（不一定，这里应该是batch=6）  Gt\_boxes\_3d:每个box9个值，没用使用mmdet3d格式的BaseInstance3DBoxes  LiDARInstance3DBoxes(tensor**=**[[l**/**2, w**/**2, 0, l, w, h, 0]])  构建一个boxinstance只需要7个值      这个深度图模块输入时encder的不同尺度featuremap,输入焦距，输出不同维度的depths map（逻辑很奇怪）    输入的gt\_depth居然已经整理为这种格式了  六个相机的feat.shape torch.Size([1, 6, 256, 64, 176])合起来输入一个大的图  gt\_depth必须处理，应为lidar的数据无法覆盖整个图像  所以参与监督的深度图其实也是一个阉割版  6张周围图融合各个焦距获得一个角度的深度图（有点离谱）  **Temporal Instance Denoising**    产生dn\_anchor，需要产生正样本和负样本，总之直接把三个种类的anchor进行cat  **Decoupled Attention**    **Tracking** |
| id和bboxes有一一对应的关系  跟踪的环节    跟踪的环节其实是严重依赖于检测结果：cls,anchor的结果都会直接影响id的产生 |

### 3.0 PF-Track

说明;主体目标为3DMOT，关键在于研究它的Tracking策略

|  |
| --- |
| tracking部分也调用了匈牙利匹配 |

### 3.2 SimpleBEV

|  |
| --- |
| 稀疏卷积Spconv  通过对输入输出不为空的数据建立位置哈希表和RuleBook，只对有效数据计算卷积，从而减少计算量  两种卷积操作：SC 和 VSC  Sparse Convolution,Valid Sparse Convolution  site可以理解为image中的pixel和点云中的point，对第一层来说active site就是有数据的pixel或point  上面这个图使用广泛  运行中的神奇报错：  RuntimeError: one of the variables needed for gradient computation has been modified by an inplace operation  出现这种报错特别难弄，主要是变量很多不好找  可能原因：  1.将Pytorch中 torch.relu()通过设置inplace=True进行inplace操作；  2.对于代码中类似x += y等是操作，改成x = x + y；（我的问题最终是这个原因，需要我们非常认真的检查每个变量名）  3.把pytorch恢复到1.4之前的环境；  4.把更新梯度的步骤调后放在一起；  5.将loss.backward( )改成loss.backward(retain\_graph=True)；  6.在pytorch中， inplace operation 可以是两个激活函数串联在一起导致的，将两个串联的激活函数删去一个。  解决方法：  1.使用torch.autograd.Variable：将要修改的变量封装在torch.autograd.Variable中，这样可以跟踪操作历史。然后通过调用variable.data获取原始张量进行修改  2. |

### 3.1 Poly-MOT & MCTrack

在2024/12/16为Tracking部分rank 1，结构比较复杂，值得深入挖掘

而且直接支持Kitti和nuscenes数据集（感觉能同时做这两个的不多了，但是MMCV好像提供了这两的BaseDataset）

|  |
| --- |
| **MCTrack**  标准的TBD模型结构  正样本： 与真值对应的目标类别来说该样本为正样本。  负样本： 与真值不对应的其他所有目标类别来说该样本为负样本。  困难样本： 预测时与真值标签误差较大的样本。  简单样本： 预测时与真值标签误差较小的样本。    如：  图片分类：。  需要识别马、羊、牛三个类别。  给一张马的图片。对于预测马来说这个样本为正样本，对于预测羊和牛来说该样本为负样本。  真值one-hot标签：[1, 0, 0],而在预测出概率分布为[0.3, 0.3, 0.4]时，真值one-hot标签相差较大，则此时该样本是困难样本。而预测出[0.98, 0.01, 0.01]时，与真值one-hot标签相差较小，则此时该样本为简单样本。  语音识别：  需要识别“我 爱 中 国”四个字。  语音片段对应“我”。则对于预测“我”来说这个样本为正样本，对于预测其他字来说该样本为负样本。  2.基础定义  通常在机器学习的二分类领域中，对模型的结果评估是必不可少的，本文主要总结了各个评估指标，对每个指标的定义，作用进行阐述。  TP：True Positive  需要满足以下三个条件  （调整置信度，TP和FN的数量会发生变化）  置信度(Confidence Score)大于阈值，实际上预测出的所有的框都要满足这个条件；  预测类型与标签类型匹配；  预测的 Bounding Box 与 Ground Truth 的交并比 (Intersection over Union，IoU，后续会详细介绍) 大于阈值 (e.g. 0.5) ，当有多个满足条件的预选框，则选择置信度最大的作为TP，其余的作为 FP。  FP：False Positive  TN：True Negtive  FN：False Negtive  以上四个定义是基础，Positive表示对样本作出的是正的判断，T表示判断正确，F表示判断错误（Negtive类似）。比如TP表示样本为正，我们模型也判断为正，FP则表示模型判断为正，但是判断错误，样本为负。  3.Accaracy，Precision，Recall，F1  Accuracy = （TP+TN）/（TP+FP+TN+FN）  准确率，表示在所有样本中分对（即正样本被分为正，负样本被分为负）的样本数占总样本数的比例。  Precision = TP / （TP+ FP）  精确率，表示模型预测为正样本的样本中真正为正的比例。  Recall = TP /（TP + FN）  召回率，表示模型准确预测为正样本的数量占所有正样本数量的比例。  所以召回率往往是针对某一个特定的类来说的，可以对每个class计算对应的Recall然后取平均值  <https://mp.weixin.qq.com/s?__biz=MzU0NjgzMDIxMQ==&mid=2247615023&idx=2&sn=bdfab1128d4c927914b66f43cf5414b7&chksm=fa8b74b10cbab6ca121cb90b8eec6d4c9decbe65fff1e04e452772142adf160257858c4c72a8&scene=27>  注意：AB，A代表检测是否正确，B代表检测给出Label是正负  AP(average precision 平均精度)：虽然名为平均精度，但AP的计算方法并不是计算Precision的平均值，而是计算每个类别的PR曲线与坐标轴围成的面积，可以用积分的方法进行计算。如果一个模型的AP越大，也就是说PR曲线与坐标轴围成的面积越大，Precision与Recall在整体上也相对较高。  AP,mAP的计算方法（只是评价指标，不会作为loss来训练）  在这张图中，IOU的阈值和置信度两个值变化可以在一个类产生多个P-R曲线 |

TBD,TBA,TJD,三种Tracking的模型，目前TBD较多，而且效果更好

一点新思路都没用

tracking可以建立在detection之上，多源融合的方法往往还是先用一个MO3D然后接数据关联方法（卡尔曼滤波），纯视觉模型往往是靠TBA来维护query

TBD全是策略，没啥能做的

TBD：poly-mot学习一下

TBA: PF-Track已经看过了，考虑如何扩展

TJD: 找一篇论文看看

|  |
| --- |
| **Poly-MOT**  需要考虑车辆运动学模型（bicycle模型）  这个论文和NN没啥关系，全是动力学模型，和kalman filter有关  3D Detector是重点，这里检测精度高直接决定后续模型跟踪性能 |

### 3.1 TransFusion & FocalFormer3D

确定方向：以基于transformer的结构出发来学习多传感器数据融合方法

创新点：CUDA算子，具体怎么做后面慢慢学

|  |
| --- |
| **Transfusion**  目标：学习数据融合方法（尤其是point cloud的处理方法）  在服务器上一堆bug，放弃了 |

### 3.1 Our Work

|  |
| --- |
| **结构化数据输入**  1.首先实现记录前N时刻的bbox/feature queue  2.然后实现解码使用的Decoder  3.最后获得tracking结果，需要loss |

## 3.2D目标检测方法

### 3.1 Yolov1

Yolov1:one-stage的代表性方法

Anchor-free:



通过代码深入学习yolo的输入数据格式和loss计算方法



Yolo需要将数据集的label处理为的格式，表示每一个grid的所有bounding\_box负责的obeject(每一个grid只能负责一个obeject，也就是说两个框的中心如果落在一个grid上就无法识别了)

### 3.2 R-CNN

R-CNN:two-stage的代表性方法

|  |
| --- |
| **R-CNN**算法流程：  1.通过Selective Search算法对一张图像产生1k~2k个候选区域  2.对每个候选区域resize为统一大小的尺寸  3.对这个图片进行分类 |



### 3.3 Faster-RCNN



Faster-RCNN结构如上：重点关注RPN结构如何打造的

|  |
| --- |
| 1. **Conv layers**  无论输入图片多大，网络都会整形为MxN，（如1000x600）,然后经过conv后变为M/16xN/16(变为60x40), 60\*40\*512(特征图通道固定为512)  2. **RPN(Region Proposal Networks)**  负责计算预测框的大小，然后与先验框叠加就是最终的建议框  3. **RoiPoolingConv**  这里需要使用from torchvision.ops import RoIPool  pooled\_features = roi\_pool(features, rois)  features 是输入特征图，形状为 (N, C, H, W)，其中 N 是批次大小，C 是通道数，H 和 W 是特征图的高度和宽度。  rois 是感兴趣区域，形状为 (num\_rois, 5)，每行格式为 (batch\_index, x1, y1, x2, y2) |



总之可以直接计算ROI对应区域的特征图



在训练过程中，正样本和负样本还需要进行创建（同一个真实框会对应上多个建议框，阈值一般不会太高）

Loss部分使用smooth L1 loss函数

torch.where 用于计算平滑L1损失，是一个条件选择函数

### 3.4 PID-net (yikongzijia)

实际上是2D车轴分割，这个方案来自于PID-Net

|  |
| --- |
| **双分支网络（Two-Branch Network,TBN）**  最初概念应该来自于BiSeNet,分别提取空间信息和感知野  左边是空间路径，负责输出高分辨率特征图  右边是上下文路径，负责输出高维度特征  注意：该网络是全卷积网络，BiSeNet是2018年  **PID控制算法**  比例，积分，控制  总的来说，当得到系统的输出后，将输出经过比例，积分，微分3种运算方式，叠加到输入中，从而控制系统的行为  自动化算法，细节就不看了 |

|  |
| --- |
| **PIDnet结构**  学习mmsegmentaion框架  1.做好数据预处理  所以annotation可以使用多个文件夹  总之这个文件的作用就是产生对应的gt\_mask的jpg文件  2.分布式训练运行环节  usage: launch.py [-h] [--nnodes NNODES] [--node\_rank NODE\_RANK]  [--nproc\_per\_node NPROC\_PER\_NODE] [--master\_addr MASTER\_ADDR] [--master\_port MASTER\_PORT]  [--use\_env] [-m] [--no\_python] [--logdir LOGDIR]  training\_script …  nnodes：节点的数量，通常一个节点对应一个主机，方便记忆，直接表述为主机  node\_rank：节点的序号，从0开始  nproc\_per\_node：一个节点中显卡的数量  -master\_addr：master节点的ip地址，也就是0号主机的IP地址，该参数是为了让 其他节点 知道0号节点的位，来将自己训练的参数传送过去处理  -master\_port：master节点的port号，在不同的节点上master\_addr和master\_port的设置是一样的，用来进行通信  3.mmseg框架及扩展  mmseg中读取模型和读取数据集都是靠import某个文件夹的init.py进行的  因此数据集文件和模型文件必须  1.必须放在特定的文件夹下面  2.必须register\_module  3.必须被上层库import  总之都是将对象解包转换为输入参数  **pag运算**  x\_p是大特征图，x\_i是高纬度特征  这里的运算比较像注意力机制，先通过torch.sum和sigmoid求出权重weight  **PAPPM操作**  多尺度并行融合，全是卷积等操作，还有平均池化  **BAG操作**  需要：路面，车辙，石块，管线，风柜草  三代7个相机，  之前：车辙，石块，水坑  这个gt是通过gt\_seg\_map产生的，需要拿来计算另一个loss（我找了好久）  数据处理的重点在BaseDataset的prepare\_data函数中  最后计算Loss  S-loss l0:CrossEntropyLoss  S-loss l2:OhemCrossEntropy  B-loss l1:BoundaryLoss  神奇：torch.where可以进行梯度回传  详细看一下accuracy函数  再详细看一下sem\_bd\_label诞生逻辑  标注规则：  **覆盖标注：**  定义：在标注过程中，允许一个区域完全覆盖另一个区域。标注时的区域可能有重叠，并且重叠部分只归属于更上层的标注区域  特点：优先级高的对象会覆盖优先级低的对象，标注结果中每个像素或每个位置可能只被分配给一个标注类别  缺点：可能会丢失被覆盖区域的信息，难以处理透明物体或需要多层次表示的任务  **共边标注：**  定义：在标注时，区域之间的边界是共享的，即边界两侧的区域在边界上会同时存在，各自对应自己的类别。  特点：允许在边界部分同时标注多个类别（如边界处共享信息）  缺点：标注复杂，可能导致边界模糊或难以处理  标注规则： |

## 多传感器融合定位

### 4.1 卡尔曼滤波与多传感器定位



上图是五个最为重要的方程

参考网站：<https://zhuanlan.zhihu.com/p/433560568>

推导非常详细，核心问题其实就是：通过最小化的MSE均值来求取，



基本方程：

|  |  |  |
| --- | --- | --- |
| 名称 | 公式 | 说明 |
| 状态方程 |  |  |
| 观测方程 |  |  |
| 预测方程 |  |  |
| 校正方程 |  |  |

基本步骤：

|  |  |  |
| --- | --- | --- |
| 步骤 | 公式 | 说明 |
| 预测 |  | 使用上一刻估计值计算先验值 |
| 卡尔曼增益 |  | 计算这一刻的先验误差的协方差 |
| 卡尔曼增益 |  | 计算卡尔曼增益 |
| 矫正 |  | 根据观测值和增益对先验值矫正 |
| 递推 |  | 计算这一刻的后验误差的协方差 |



### 因子图优化与多传感器定位

（比KF抽象得多）

## 机器学习算法归纳

百度面试需要。。。

**线性回归算法**

线性回归的目标是最小化损失函数，损失函数用于度量模型预测的准确度。最常用的损失函数是均方误差（MSE），直接使用梯度下降的方法即可求得

线性回归的几种变体：

1.简单线性回归：只涉及一个自变量和一个因变量。

2.多元线性回归：涉及多个自变量和一个因变量。

3.岭回归：通过引入L2正则化来解决多元线性回归中的过拟合问题。

4.套索回归：通过引入L1正则化来解决多元线性回归中的过拟合问题，并且能够进行特征选择。

**支持向量机算法**

支持向量机（Support Vector Machine，SVM）是一种监督学习算法，可以用于分类或回归任务。SVM的基本模型是在特征空间上找到一个最优的超平面，这个超平面能够最好地将不同类别的数据分开，同时保持最大的间隔（margin）。对于分类问题，SVM寻找一个超平面，使得距离最近的两个不同类别的数据点之间的距离最大，这样的超平面称为最大间隔超平面。

支持向量机其实有点像一个线性层

**最近邻居/k-近邻算法**

k-近邻（k-Nearest Neighbors，k-NN）算法是一种基本的机器学习算法，用于分类和回归任务。它的工作原理非常简单：对于一个未知类别的数据点，k-NN算法会在训练集中找到与它最接近的k个数据点，然后根据这k个“邻居”的类别来预测未知数据点的类别。如果k-NN用于回归任务，那么预测值通常是k个邻居的平均值。

**逻辑回归算法**

**决策树算法**

决策树（Decision Tree）是一种常见的机器学习算法，用于分类和回归任务。它是一种监督学习算法，以树形结构进行决策。每个内部节点代表一个特征，每个分支代表一个特征的测试结果，每个叶节点代表一个分类或回归结果。

**k-平均算法**

k-平均（k-Means）算法是一种无监督的聚类算法，它的目的是将相似的数据点分组到一起，形成多个簇（cluster）。算法通过迭代过程来优化簇内点的均方误差，直到达到收敛条件。

重复步骤：

a.分配步骤：将每个数据点分配到最近的质心所代表的簇。

b.更新步骤：重新计算每个簇的质心

**随机森林算法**

随机森林（Random Forest）是一种基于决策树的集成学习算法。它通过构建多个决策树并进行投票来提高预测的准确性和稳定性。随机森林在处理分类和回归任务时都非常有效，尤其是对于具有大量特征的数据集。（不会）

**朴素贝叶斯算法**

朴素贝叶斯（Naive Bayes）算法是一种基于贝叶斯定理的简单概率分类器，它假设特征之间相互独立。尽管这种独立性假设在现实世界中通常不成立，但朴素贝叶斯分类器在许多实际应用中仍然非常有效。

**降维算法**

降维（Dimensionality Reduction）是机器学习中的一个重要概念，它涉及到减少数据集的维数，同时尽可能地保留数据中的有用信息。降维的目的是为了简化模型，提高计算效率，减少过拟合的风险，并可能提高模型的泛化能力。

一些常用的降维方法。

1.主成分分析（PCA）：PCA是一种流行的线性降维方法，它通过保留数据中方差最大的几个主成分来降低维数。（使用频率较高）

2.因子分析（Factor Analysis）：因子分析类似于PCA，但它试图找到数据中的隐藏因子，这些因子可以解释数据中的变异。

3.线性判别分析（LDA）：LDA是一种线性降维方法，它通过最大化不同类别之间的距离来降低维数，同时最小化类内距离。

4.t-SNE：t-SNE是一种非线性降维方法，它通过模拟数据点在低维空间中的局部结构来降低维数。

5.自编码器（Autoencoder）：自编码器是一种神经网络，它试图通过重建原始输入来学习数据的低维表示。

6.局部线性嵌入（LLE）：LLE是一种非线性降维方法，它通过保持数据点在低维空间中的局部邻域结构来降低维数。

7.等距映射（Isomap）：等距映射是一种非线性降维方法，它通过保持数据点之间的测地距离来降低维数。

## 6. Occupancy相关算法

### 6.1 OpenOccupancy相关

|  |
| --- |
| **OpenOccupancy出发** |

### 6.2 深度估计相关-BevDepth

|  |
| --- |
| **BevDepth**  学习多视角相机深度估计的方法，理解目前多个网络中深度监督模块的含义  1.bug发现  num\_worker设置可能导致进程死锁  主要看看depth netde输入和输出以及lidar如何处理为深度图数据  ---在没有电云的地方直接赋值为0，后续应该这种地方不参与梯度回传即可  sweeps使用的问题：  点云可以将sweeps数据投影到关键帧上作为数据增强，  图像可以将sweeps数据的的标签进行迁移传播  本论文采取数据增强，会把后一帧的数据也通过网络提取特征然后和关键帧进行concat  论文中对此做出了相应的说明  以下重点分析depth net处理的部分：  input:key frame的img,torch.Size([2, 1, 6, 3, 256, 704])  model:  mlp-se-conv  中间采用IDA和BDA（详情可见BevDet,数据增强方法）  用于增强：    但是后续必须逆变换回来：  depth并不是直接让网络输出深度图，而是生成了类似的通道数（着重看一下如何和由lidar产生的gt）  但是为什么要在深度通道D上做归一化呢?重点看后面如何生成depth\_label  这两个计算loss,depth\_labels是激光雷达的gt  代码中的中间深度图结果：  见下方  CAM\_FRONT-LEFT CAM\_FRONT CAM\_FRONT\_RIGHT  CAM\_BACK\_LEFT CAM\_BACK CAM\_BACK\_RIGHT    112是给出的超参数，16，44是网络降维度得到的（）  output:  loss:  采用cross\_entropy,应为使用了one-hot来计算loss,但是我不理解为什么不直接用神经网络输出一个深度图 |

## 7.前沿技术探索

### 7.1 MLA技术

|  |
| --- |
| **Multi-Head Latent Attention**  来自于deepseek模型的技术，可以降低kv cache，减少内存消耗  原本的attention:    注意的是，O计算仅使用前t位置的k,v（代码中这个通过mask实现）  MLA的方法：多个head共享使用一组KV  其他的方法：MQA，GQA，都是google提出    最右边就是MQA，中间就是GQA    上面是一个对比图    下面进入 MLA学习：    只需要缓存蓝框元素    39，43的计算实际上是一个MQA的计算过程（所有head共享一个k）    但是在运行过程中还是会出现最大的存储使用吧  推荐：[缓存与效果的极限拉扯：从MHA、MQA、GQA到MLA - 科学空间|Scientific Spaces](https://spaces.ac.cn/archives/10091)  个人感觉和deformable attention思路不一样，而且DA都没有使用多头，它使用的严格来说是多尺度，而且没有前后token的概念  创新点其实就是只需要缓存两个变量到内存中(K.cpu(),V.cpu()即可) |