## 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 .



### 注意力机制的变种

可变形注意力：

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）

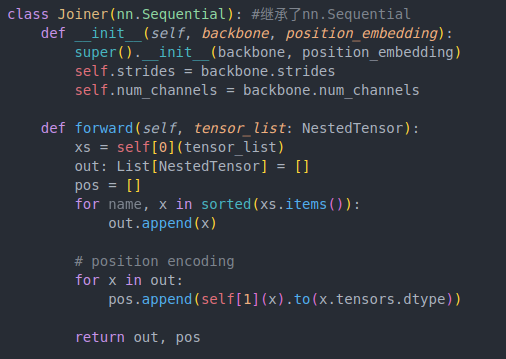


应为代码配置不好，现在BEV Former先放放

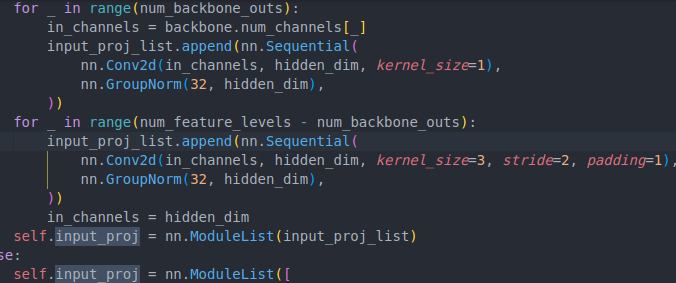
Bev Former原论文在A100上配置，4090配置挺麻烦的

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]) |

说明：特征图尺寸为4

### 1.3 BevFusion



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

## 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：

## 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

## 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。