# CLRNet论文详解

**是2022年非常有价值的一篇论文，在各种CULane, Tusimple, Curvelane各种知名数据集上都一直排第一且居高不下，即便在2023年也依然如此。因此萌生兴趣，不管如何也必须深入的探究一下它的做法。**

**1 文章有2大动机：**

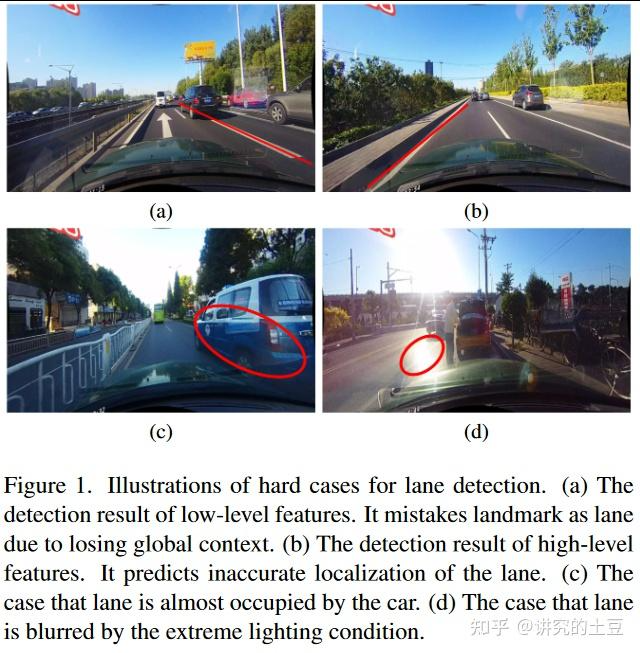
1）高层特征与底层特征都重要！下图的(a)(b)分别就是仅仅利用底层特征得到错误结果、仅仅利用高层特征得到不准确结果。

2）无视觉线索问题怎么解决？下图的(c)(d)就是车辆遮挡以及逆光造成的视觉线索不明显。一半来讲，这个问题需要全局context信息来解决。

**针对这两个问题，本文是这样解决的：**

1）针对第一个问题，本文提出一种coarse-to-fine的机制，利用高层特征检出车道线，利用低层特征调节车道线位置。

2）针对第二个问题，本文提出RoIGather模块获取全局语境信息。 提出Line IoU loss，把车道线作为一个整体来优化。

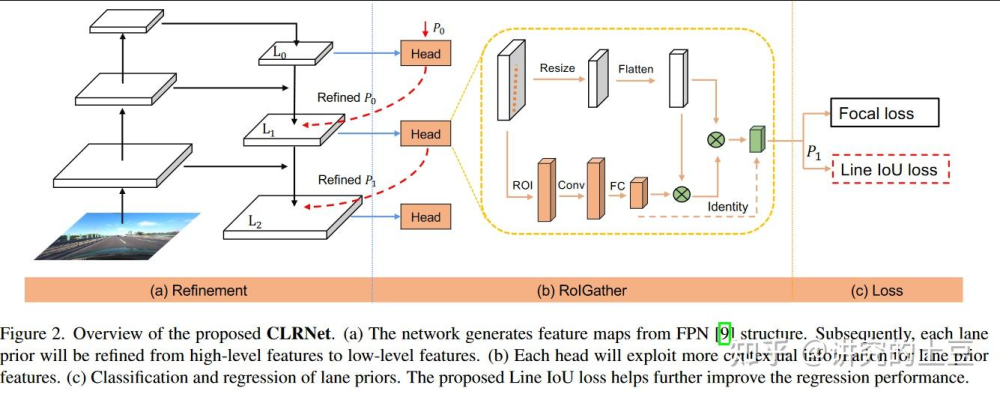


**2 论文整体网络结构与猜想**

1）但只从结构图来看，并不复杂，标准的BACKBONE+FPN金字塔结构，加入了RoiGather（我们后续详细说），一共有3个Head，Loss分成了Focal Loss可能用于车道线正负anchor的判断，Line IoU Loss大概率用来做车道线的定位和回归。

2）从图中看，推理和训练的时候，先从最小特征图开始输出P0 Head，得到车道线的信息后，加入L1特征，再次输出P1 Head，再得到车道线信息后，加入L2特征，再次输出P2 Head，那么这个P2 Head就是最终的输出了。环环相扣，给人感觉就是一种从全局信息一步一步往下推理得到局部和更精确的车道线信息的一个过程。

3）RoiGather部分，里面有乘法算子，应该使用了attention注意力机制来加强。



**3 关于算法的疑问**

根据上面整体的结构来看我们需要清晰的知道几个点：

1）3个Head应该原理一致，那是如何建模的？

2）如何从P0 Head依次到P2 Head？做到环环相扣，从上到下？Refine的是什么信息？

3）RoiGather注意力机制是怎么玩的？有什么意义？

4）损失函数如何设计的？

5）训练的过程，车道线的正样本包含哪些信息，如何筛选的？

当我们掌握了以上5点后，就完全了解了这篇论文

**4 Head建模**

首先这篇论文是基于anchor base的车道线算法，所以我们得先来说说它的anchor设计，我们只要知道anchor的设计，那么anchor+head输出就是最终车道线信息了。

**1）anchor前言：**

1）anchor是再动态变化的，在训练中不断学习的，并非只是一个初始值

2）anchor会从P0 Head到P2 Head逐层Refine的，所以anchor在第3个Head上和第1、2个Head上是不同的。这也算是一种动态吧，加上1）变成了双动态anchor，首先就觉得很牛逼不是？但是在工程部署上真的好用么？

**2）anchor设计：**

文章讲一个lane prior（我就理解成anchor）由4组信息组成：

1）前景-背景概率。这个容易理解，就是每个anchor你都要预测是前景的概率。

2）lane prior的长度。这个就是y上的72个采样点，其中车道线范围的长度。

3）x, y, theta，起始点坐标以及与x轴的夹角。

4）N个offsets值。N=72。

这篇文章比较难理解的地方就是作者把anchor 和车道线混为一谈，都称之为lane prior，不看代码的话不知道具体实现是啥样。

作者是怎么实现可学习的anchor呢？其实就是三个参数可学习：X, Y, theta（代码中y在前x在后，我这里按照习惯叙事）。只要anchor的起始点、倾角确定，anchor就确定了。初始化的anchor总数为192个，左右边各1/8，中间3/4。左右每个起始点设置两种夹角；中间每个起始点设置4种夹角。

然后根据这192个3维参数向量，生成192个78维度的anchor向量。

* 0、1两位为模型预测的正负样本概率，作为anchor时设为0；

* 2、3、4三维就是anchor参数的三个量，X, Y, Theta；

* 5是模型预测的车道线y范围长度，作为anchor时，恒为0；

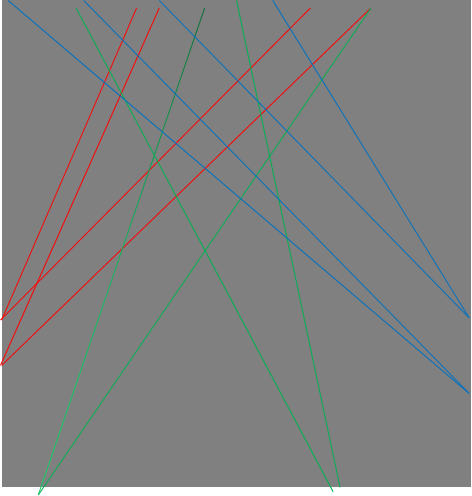
* 6~78就是72个采样点的x位置。作为anchor，它是由anchor 3参数按照直线方程得到的。anchor的72个值加上模型预测的72个offset，那就是车道线的x位置。

所以，对于anchor来说，最核心的就是2、3、4三个值，以及由他们衍生出来的72个x坐标。根据anchor以及模型预测解出车道线时，才会关注其它量。

**如下图anchor线的生成大概是这样：**

分层3类，红色是左边界的车道线anchor，底部中间是绿色车道线anchor，右边是蓝色的车道线anchor，分别数量分成：1/8， 3/4， 1/8，一共生成192个车道线anchor，anchor的数学表达为起始坐标X,Y+角度Theta，车道线的anchor是会贯穿全图的（模型输入的全图800X320，非原始全图）。

（注：生成代码里X, Y都是归一化0-1的值，Theta是角度值）



**3）基于anchors生成priors和priors\_on\_featmap**

prios的维度是（192，78）

其中192是192个anchor，78中：0,1表示正负样本

2-4表示起始点X, Y, 和Theta（）

6-78表示anchor直线方程的72个采样点对应的x轴坐标值（是绝对值0-800范围内）

priors\_on\_featmap的维度是（192, 36）

从priors中的72个采样点中，等间隔拿36个点出来组成坐标矩阵

那么这个priors其实就是真正的anchor全部信息了，后续在训练过程中priors里的2-4会在训练中变得越来越优。

**4）anchors的逐层REFINED**

前面算法疑问里，有说要了解REFINE到到底做了什么，我们来详细说说

算法输出会有三个特征L0, L1, L2。

那么拿L0举例，L0经过Head部分后会输出Prediction0（B x 192, 76）

其中B就是Batch，192表示192个anchors，76就是模型推理输出的信息，包含：

0-2：表示相对于priors里的X,Y, Theta的3个变量的偏移量，注意模型输出的是偏移量，所以最后结果要加上priors本身的值。

3：表示当前车道线的长度，长度值为Y采样的长度，0-72，归一化到0-1，这里输出的又是绝对量

4-76：表示相对于anchors里的72个采样点的偏移量，所以此结果也要加上priors本身的值

**Refined的细节和步骤：（Refined的只是priors里的X, Y, Theta这3个值）**

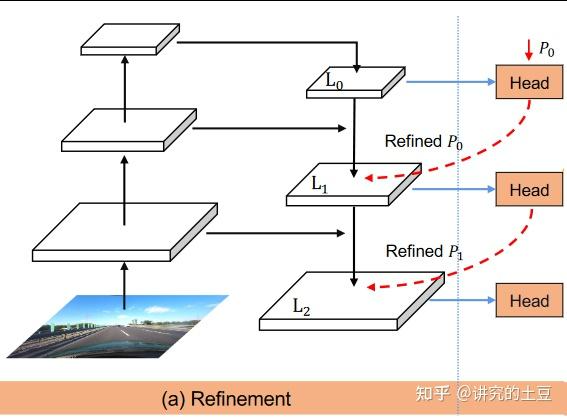
文章提出了一个从粗到细的refine结构，事实上refine的是anchor，也就是X, Y, Theta三个参数。P0从初始化的三参数开始，用梯度不断更新，因此称之为可学习anchor。每次迭代中，从P0开始，不断地refine anchor的三参数。

* P0 anchor+L0特征，首先将预测得到的X, Y, Theta重新根据直线方程计算72个点。新的72个点（注意，不是原来的）+预测的offset得到完整的72个点。获得192个78维的车道线预测，过loss模块提供监督信号。

* 上面的X, Y, Theta以及新的72个点组成新的anchor，称为P1（就是图中的refined P0），P1 anchor+L1特征更新anchor参数，并获得车道线预测，过loss模块提供监督信号。

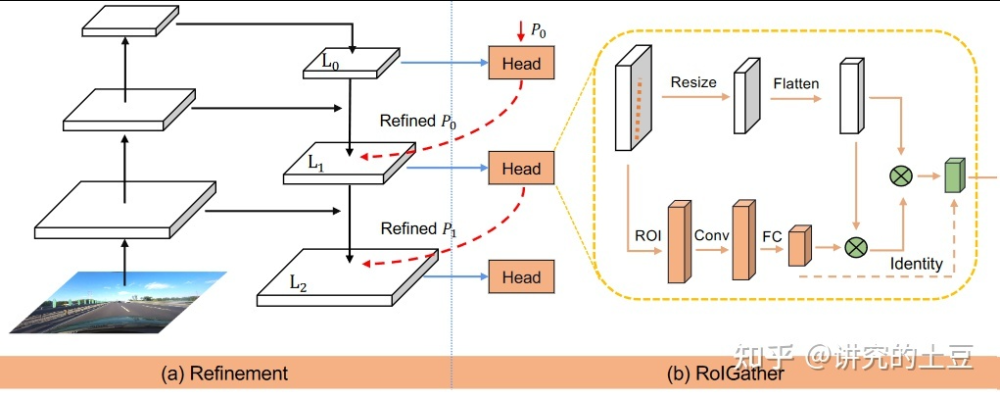
* 上面更新的anchor 我记为P2（就是图中的refined P1），P2 anchor+L2特征 同样获得车道线预测，过loss提供监督信号。推理时，这个结果就是最终的预测。

所以，从anchor的refine角度来讲，概率预测、车道线y范围长度、offset预测，都没有意义。只有三参数是有意义的。



**5 ROIGather**

**从图中可以看出从L0到经过Head这个过程其实就是ROIGather的全部了，L1,L2依次类推，那么我们下面重点看下这个过程。**



1)上面已经说了priors\_on\_featmap的含义，其实就是192个anchor下每个anchor的36个采样坐标。（192, 36）

2）模型的输出的feature有3层分别为：（B, 64, 10, 25）,（B, 64, 20, 50）,

（B, 64, 40, 100）

3）那么ROIGather的输入就定了：priors\_on\_featmap+3层feature

4）下图是基于L0层的示意图,L0的维度（64,10,25），以下流程暂时不考虑batch维度

5）grid是priors\_on\_featmap与priors\_y组合在一起的，因为priors\_on\_featmap里只有x的坐标，组合后形成x,y坐标，维度（192,36,2）

6）L0+grid通过grid\_sample算子采样特征，得到ROI0，维度（192,64,36），可以理解为：共192个anchor，每个anchor特征分辨率36，且通道数为64

7）ROI0再经过卷积+FC最后可以得到Out0(192,64)，这个先放着会用

8）ROI0再经过一个conv1d得到Q（192,64），这个Q表示query，查询的意思，是出自于transformer里attentation来的，备用。

9）L0特征通过conv+resize+flatten，兵分两路，分别得到K和V，维度都是（64,250），K，V也是出自transformer里的attentation，表示Key, Value

10）那么接下来就要开始计算注意力attention的关系矩阵了：

**Map=softmax((Q x K) / 8)=(192, 250)，注意8和通道数64是有关的。等于于开根号**

公式得到了一个关系矩阵，可以理解为L0一共有10\*25=250个特征点，192表示192个anchor，那么就是192个anchor和250个特征点的关系矩阵，里面数值越大，表示当前anchor和这个特征点的关系就大。

11）最终head输出计算公式：

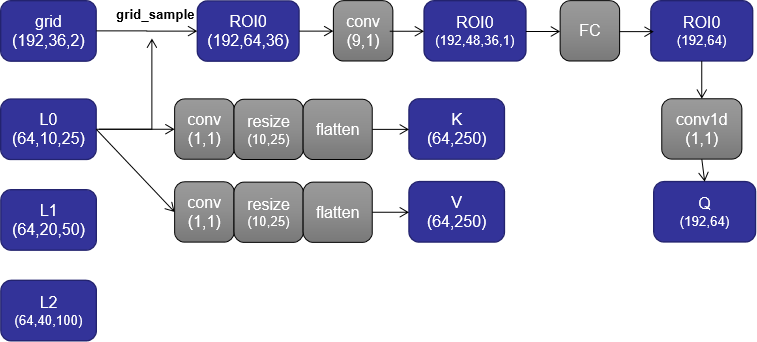
**ROI0 = ROI0 + Map x transpose(V) = (192, 64)**，就是把关系矩阵融合到ROI0中得到一个最终的加强版本特征。

12）最终的ROI0再通过2个FC得到分类输出（192，2），回归输出（192,76），这个输出我们叫做prediction，再结合anchor信息，就得到了最终车道线输出。

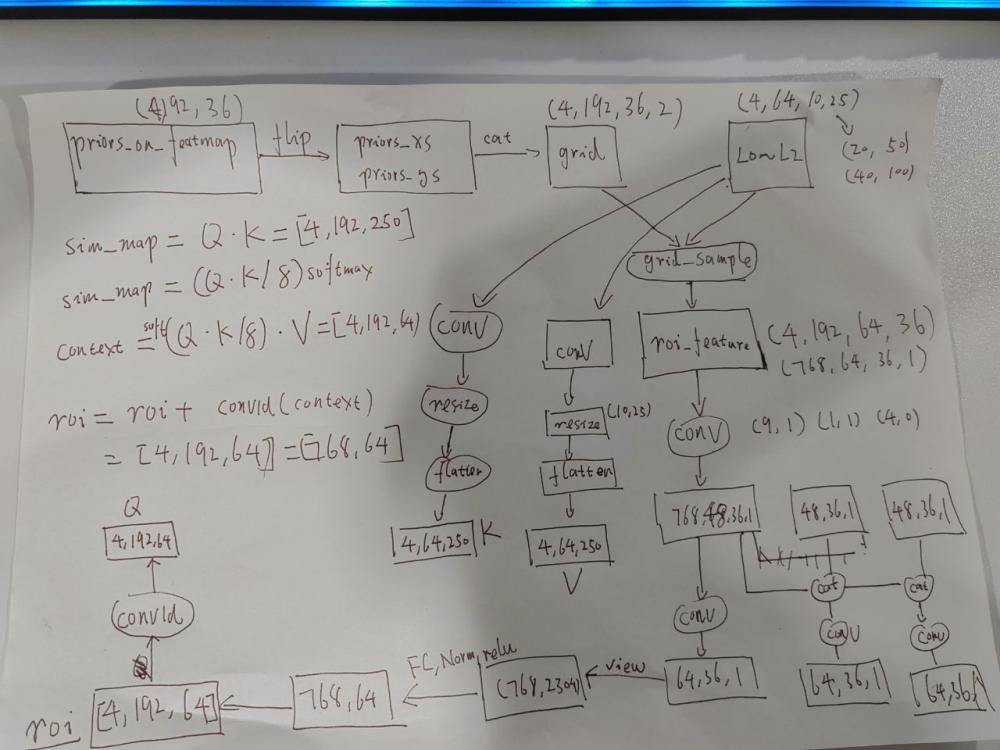
13）以上是L0层特征的处理，对于L1有点区别，L1和grid先做grid\_sample操作，然后再和ROI0做concat，进行一个卷积，最后得到ROI1，维度不变（192,64,36）

14）那么L2依次类推，要用到ROI0,ROI1一起concat，再卷积，最后还是的到ROI2，维度是不变的（192,64,36）

15）另需要注意最终的车道线输出结果应该在L2这一层上，前面的L0,L1虽然会输出车道线但是不适用，只有X,Y,Theta会往下逐层Refined



**最后附上我自己手工画的ROIGather流程图，方便以后回忆：**



**6 训练过程中的车道线正样本筛选**

正样本筛选常见于anchor base的算法中，也是训练非常重要一环，比如物体检测YOLO，有基于anchor的IOU来做正样本筛选，后续又演进了ATSS基于统计学+IOU的正样本采样，但是anchor的选择是静态的，一开始可以通过K-MEAN这种聚类方法，找到训练样本中最好的anchor尺寸，再到SimOTA这种在线动态的anchor来进行正样本筛选。

那么对于这篇车道线论文，它也是一个典型的在线动态的anchor，anchor在训练的过程中，会不断地修正变得最优，然后基于一些距离+lineIOU完成正样本匹配筛选。

我们正式来看看CLRNet的正样本筛选策略assign函数

1）已知模型的输出为prediction（192, 78），结合anchors我们就可以得到车道线最终输出prediction（192,78）

2）假设当前图像真值有3根车道线，那么target为target(3, 78)

3）计算prediction和target里72个采样点的距离成为distances，它的维度为distances(192, 3)，它表示192个基于anchor的预测值和3个target真值的距离矩阵，这个距离为72个采样点的平均距离。

4）计算distances\_score=1-distances/max(distances)，分值越高，表示当前anchor的预测和当前target的真值越匹配

5）计算prediction和target里类别的交叉熵cost，维度也是（192,3），里面的值越高，越不匹配

6）计算prediction和target里theta值得距离，维度也是（192,3）

theta\_score = 1- theta/max(theta), 分值越高，表示当前anchor的预测和当前target越匹配

7）那么总的cost公式如下：

cost = -(distances\_score \* start\_xys\_score \* theta\_score  
 )\*\*2 \* distance\_cost\_weight + cls\_score \* cls\_cost\_weight

8）这个cost的维度依然是（192, 3），表示prediction和target之间的const损失值，cost损失越小表示当前anchor的prediction和当前的target越匹配。

9）计算prediction和target的线的IOU，我们叫lineIOU，其实就是基于72个采样点来计算，每个prediction样本和target样本的72个点往左，往右扩宽15个像素，在计算交并比即可。iou的维度也是（192,3）

def line\_iou(pred, target, img\_w, length=15, aligned=True):  
 px1 = pred - length  
 px2 = pred + length  
 tx1 = target - length  
 tx2 = target + length  
  
 num\_pred = pred.shape[0]  
 invalid\_mask = target.repeat(num\_pred, 1, 1)  
 ovr = (torch.min(px2[:, None, :], tx2[None, ...]) -  
 torch.max(px1[:, None, :], tx1[None, ...]))  
 union = (torch.max(px2[:, None, :], tx2[None, ...]) -  
 torch.min(px1[:, None, :], tx1[None, ...]))  
  
 invalid\_masks = (invalid\_mask < 0) | (invalid\_mask >= img\_w)  
 ovr[invalid\_masks] = 0.  
 union[invalid\_masks] = 0.  
 iou = ovr.sum(dim=-1) / (union.sum(dim=-1) + 1e-9)  
 return iou

10）我们拿到了iou和cost后，就可以开始做动态匹配了，这里取topK=4，表示每个target的真值，最多取4个prediction里的样本作为正样本与之匹配，参与后续损失函数计算

11）那么取的原则就是先基于iou来取，得到topk\_ious，维度（4,3），表示4表示最大匹配的4个prediction正样本和target对应的iou值，3则表示3个target真值。

12）再基于cost从4个样本里选择出cost最小的1-4个和target对应上，就是我们最后选出的最好prediction正样本

13）最后输出2个列表prior\_idx和gt\_idx，第一列表是选择出来的最匹配的prediction正样本，第二列就是和第一列对应的target真值。如下代码：比较复杂

14）这里需要注意的，每个gt最少要有一个prediction样本与之对应，可以一对一，也可以一个gt对多个prediction样本，不可以存在一个prediction样本对应多个gt情况

def dynamic\_k\_assign(cost, pair\_wise\_ious):  
 """  
 Assign grouth truths with priors dynamically.  
  
 Args:  
 cost: the assign cost.  
 pair\_wise\_ious: iou of grouth truth and priors.  
  
 Returns:  
 prior\_idx: the index of assigned prior.  
 gt\_idx: the corresponding ground truth index.  
 """  
 matching\_matrix = torch.zeros\_like(cost)  
 ious\_matrix = pair\_wise\_ious  
 ious\_matrix[ious\_matrix < 0] = 0.  
 n\_candidate\_k = 4  
 #topk\_ious维度为(4, 3)  
 #含义是：一共有3个target，对应这3个target，每个target都从192个anchor里选取与之iou值  
 #排序最大的前4个  
 topk\_ious, am = torch.topk(ious\_matrix, n\_candidate\_k, dim=0)  
  
 #dynamic\_ks的维度为(3,)  
 #动态K数，对于每个target，找到的排序最大的4个anchor，去4个anchor的IOU相加数，再取整，为K的个数，最小K个数为1  
 dynamic\_ks = torch.clamp(topk\_ious.sum(0).int(), min=1)  
 num\_gt = cost.shape[1]  
  
 #cost维度为(192, 3), 表示192个anchor和3个真值之间的cost距离  
 #对列处理，这3个target，每个target取与自己cost最小的前3个anchor  
 #注意这里取出的是idex索引，为0-191，那么这个索引就是pos\_idx，这样本索引  
 #然后构建一个matching\_matrix(192, 3),把对应的pos\_idx索引都置为1，参与后面损失计算  
 for gt\_idx in range(num\_gt):  
 \_, pos\_idx = torch.topk(cost[:, gt\_idx],  
 k=dynamic\_ks[gt\_idx].item(),  
 largest=False)  
 matching\_matrix[pos\_idx, gt\_idx] = 1.0  
 del topk\_ious, dynamic\_ks, pos\_idx  
  
 #处理1个anchor会对应多个gt的情况，如果有的话，就只拿cost更小的那个赋值1，其余置0  
 matched\_gt = matching\_matrix.sum(1)  
 if (matched\_gt > 1).sum() > 0:  
 \_, cost\_argmin = torch.min(cost[matched\_gt > 1, :], dim=1)  
 matching\_matrix[matched\_gt > 1, 0] \*= 0.0  
 matching\_matrix[matched\_gt > 1, cost\_argmin] = 1.0  
  
 prior\_idx = matching\_matrix.sum(1).nonzero()  
 gt\_idx = matching\_matrix[prior\_idx].argmax(-1)  
 #每个gt最少要有一个anchor与之对应，可以一对一，也可以一个gt对多个anchor  
 #不可以存在一个anchor对应多个gt情况  
 return prior\_idx.flatten(), gt\_idx.flatten()

**7 损失函数**

这里还比较简单

1）78中的0-1是类别，用Focalloss交叉熵分类损失

2）2-5表示起始点X, Y, Theta角度，长度，用L1Loss即可。

3）6-78比较特殊使用的是lineIOU损失，和上面匹配用到的lineIOU是同一个东西，损失就是1-lineIOU即可

4）另还有一个语义分割Head需要计算损失，用的也是交叉熵损失

5）最后把上面4个损失进行加权就可以了。

loss = cls\_loss \* cls\_loss\_weight + reg\_xytl\_loss \* xyt\_loss\_weight \  
 + seg\_loss \* seg\_loss\_weight + iou\_loss \* iou\_loss\_weight

**8 推理过程**

每个anchor都会预测正负样本的概率0-1，anchor参数2-4，车道线终点到预测的起始点在y上的差值5，72个相对于预测anchor的offset（再强调一遍，不是相对于原anchor，而是本次预测的anchor三参数衍生的anchor）。L0,L1,L2都会预测这些，训练时同时贡献loss，推理时只有最后一层的结果输出来。