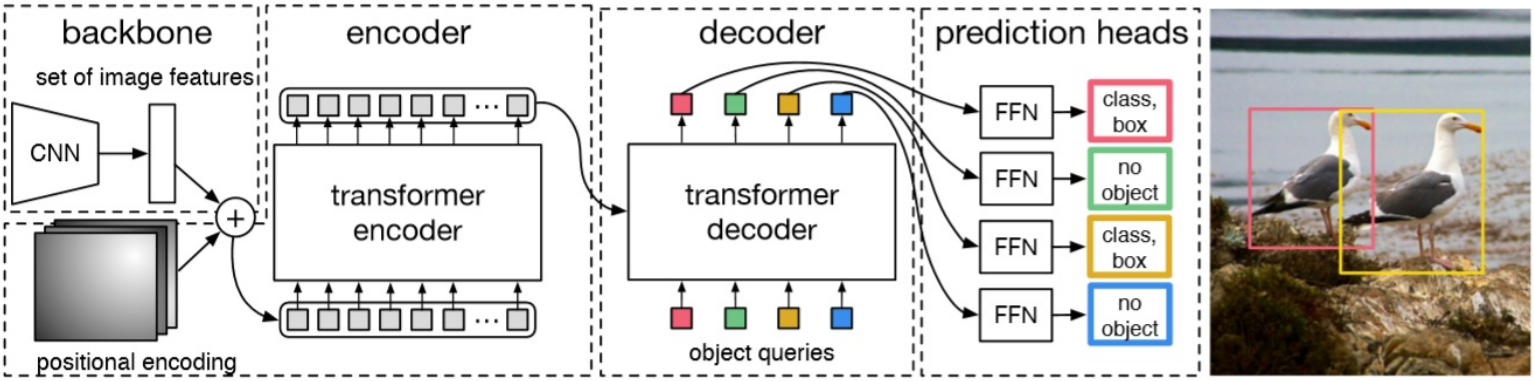


End-to-End Object Detection with Transformers-DETR

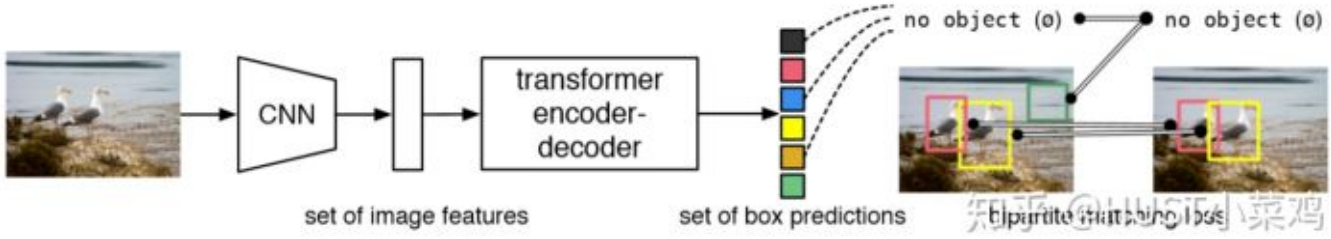
HUST小...
学生

5 人赞同了该文章

Github开源地址：facebookresearch/detr

一、创新点

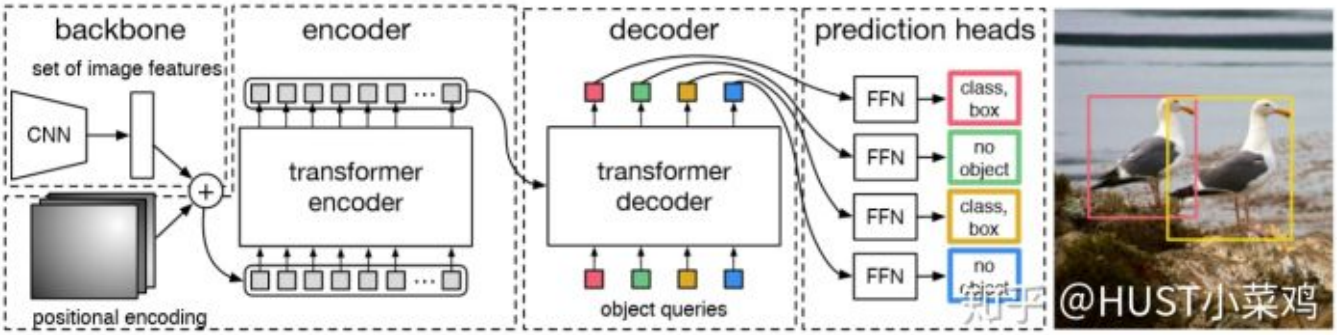
将目标检测任务转化为一个序列预测（set prediction）的任务，使用transformer编码-解码器结构和双边匹配的方法，由输入图像直接得到预测结果序列。和SOTA的检测方法不同，没有proposal（Faster R-CNN），没有anchor（YOLO），没有center(CenterNet)，也没有繁琐的NMS，直接预测检测框和类别，利用二分图匹配的匈牙利算法，将CNN和transformer巧妙的结合，实现目标检测的任务。



DETR整体结构

在本文的检测框架中，有两个至关重要的因素：①使预测框和ground truth之间一对一匹配的序列预测loss；②预测一组目标序列，并对它们之间关系进行建模的网络结构。接下来依次介绍这两个因素的设计方法。

1、模型的整体结构



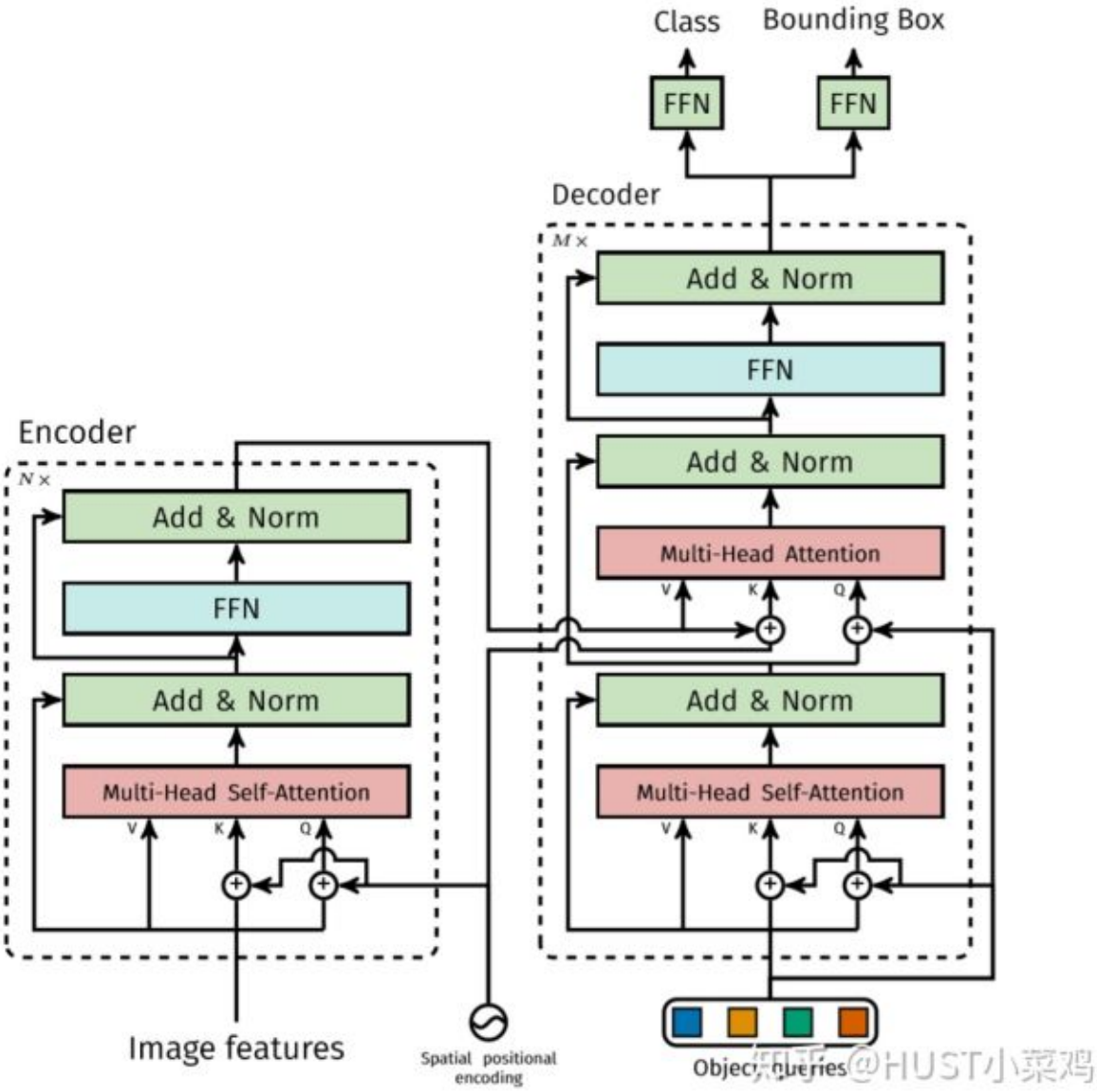
Backbone + transformer + Prediction

CNN + encoder+decoder + FFN

（1）backbone

利用传统的CNN网络，将输入的图像 $3 \times W_0 \times H_0$ 变成尺度为 $2048 \times \frac{W_0}{32} \times \frac{H_0}{32}$ 的特征图f

（2）Transformer



Transformer encoder部分首先将输入的特征图降维并flatten，然后送入下图左半部分所示的结构中，和空间位置编码一起并行经过多个自注意力分支、正则化和FFN，得到一组长度为N的预测目标序列。其中，每个自注意力分支的工作原理为可参考刘岩：详解Transformer （Attention Is All You Need），也可以参照论文：papers.nips.cc/paper/71...

接着，将Transformer encoder得到的预测目标序列经过上图右半部分所示的Transformer decoder，并行的解码得到输出序列（而不是像机器翻译那样逐个元素输出）。和传统的autogreesive机制不同，每个层可以解码N个目标，由于解码器的位置不变性，即调换输入顺序结果不变，除了每个像素本身的信息，位置信息也很重要，所以这N个输入嵌入必须不同以产生不同的结果，所以学习NLP里面的方法，加入positional encoding并且每层都加，**作者非常用力的在处理position的问题，在使用 transformer 处理图片类的输入的时候，一定要注意position的问题。**

（3）预测头部（FFN）

使用共享参数的FFNs（由一个具有ReLU激活函数和d维隐藏层的3层感知器和一个线性投影层构成）独立解码为包含类别得分和预测框坐标的最终检测结果（N个），FFN预测框的标准化中心坐标，高度和宽度w.r.t. 输入图像，然后线性层使用softmax函数预测类标签。

```
        self.class_embed = nn.Linear(hidden_dim, num_classes + 1)
        self.bbox_embed = MLP(hidden_dim, hidden_dim, 4, 3)
```

```
outputs_class = self.class_embed(hs)
outputs_coord = self.bbox_embed(hs).sigmoid()
out = {'pred_logits': outputs_class[-1], 'pred_boxes': outputs_coord[-1]}
if self.aux_loss:
    out['aux_outputs'] = [{'pred_logits': a, 'pred_boxes': b}
                          for a, b in zip(outputs_class[:-1], outputs_coord[:-1])]
return out
```


2、模型的损失函数

基于序列预测的思想，作者将网络的预测结果看作一个长度为N的固定顺序序列 $\tilde{y}$ ，$\tilde{y} = \tilde{y}_i,\ i\epsilon(1,N)$ ，（其中N值固定，且远大于图中ground truth目标的数量）$\tilde{y}_i = (\tilde{c}_i, \tilde{b}_i)$ ，同时将ground truth也看作一个序列 $y : y_i = (c_i, b_i)$ （长度一定不足N，所以用 ϕ （表示无对象）对该序列进行填充，可理解为背景类别，使其长度等于N），其中 c_i 表示该目标所属真实类别， b_i 表示为一个四元组（含目标框的中心点坐标和宽高，且均为相对图像的比例坐标）。

那么预测任务就可以看作是 y与$\tilde{y}$ 之间的二分图匹配问题，采用匈牙利算法[1]作为二分匹配算法的求解方法，定义最小匹配的策略如下：

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}),$$

求出最小损失时的匹配策略 $\tilde{\sigma}$ ，对于 L_{match} 同时考虑了类别预测损失即真实框之间的相似度预测。

对于 $\sigma(i)$ ，c_i 的预测类别置信度为 $\tilde{P}_{\sigma(i)}(c_i)$,边界框预测为 $\tilde{b}_{\sigma(i)}$,对于非空的匹配，定于 L_{match} 为：

$$-\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)}).$$

进而得出整体的损失：

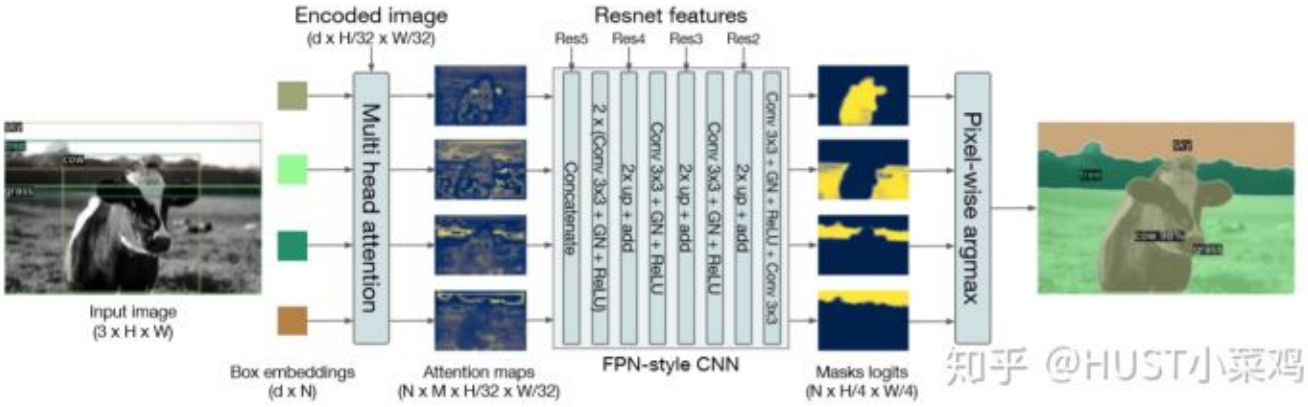
$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right],$$

考虑到尺度的问题，将L1损失和iou损失线性组合，得出 L_{box} 如下所示：

$$\lambda_{\text{iou}} \mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{\text{L1}} ||b_i - \hat{b}_{\sigma(i)}||_1$$

L_{box} 采用的是Generalized intersection over union论文提出的GIOU[2],关于GIOU后面会大致介绍。

为了展示DETR的扩展应用能力，作者还简单设计了一个基于DETR的全景分割框架，结构如下：



4、实验对比

本文中，作者主要和目标检测经典框架faster rcnn进行了对比，结果如下（其中带有后缀DC5的方法表示在主干网络的最后一个阶段加入一个dilation，并从这个阶段的第一个卷积中去除一个stride来增加特征分辨率）：

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

由上图可知，DETR框架虽然简洁，但效果与经典方法faster rcnn不相上下，其中DETR对于大目标的检测效果有所提升，但在小目标的检测中表现较差。该文提出的方法十分新颖，使用类似机器翻译的序列预测思想，打破了目标检测的传统思想，减少检测器对先验性息和后处理的依赖，使目标检测框架更加简洁的同时获得了与faster rcnn相媲美的效果。

该方法的不足表现在训练阶段，需要的时间和硬件资源需求较大，因此训练的难度还是挺大的

```python
import torch
from torch import nn
from torchvision.models import resnet50

class DETR(nn.Module):

    def __init__(self, num_classes, hidden_dim, nheads,
                 num_encoder_layers, num_decoder_layers):
        super().__init__()
        # We take only convolutional layers from ResNet-50 model
        self.backbone = nn.Sequential(*list(resnet50(pretrained=True).children())[:-2])
        self.conv = nn.Conv2d(2048, hidden_dim, 1)
        self.transformer = nn.Transformer(hidden_dim, nheads,
                                          num_encoder_layers, num_decoder_layers)
        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
        self.linear_bbox = nn.Linear(hidden_dim, 4)
        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))

    def forward(self, inputs):
        x = self.backbone(inputs)
        h = self.conv(x)
        H, W = h.shape[-2:]
        pos = torch.cat([
            self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
            self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
        ], dim=-1).flatten(0, 1).unsqueeze(1)
        h = self.transformer(pos + h.flatten(2).permute(2, 0, 1),
                             self.query_pos.unsqueeze(1))
        return self.linear_class(h), self.linear_bbox(h).sigmoid()

detr = DETR(num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6)
detr.eval()
inputs = torch.randn(1, 3, 800, 1200)
logits, bboxes = detr(inputs)
```


附录：GIOU

Algorithm 1: Generalized Intersection over Union

input : Two arbitrary convex shapes: $A, B \subseteq \mathbb{S} \in \mathbb{R}^n$

output: $GIoU$

1 For A and B, find the smallest enclosing convex object C, where $C \subseteq \mathbb{S} \in \mathbb{R}^n$

2 $IoU = \dfrac{|A \cap B|}{|A \cup B|}$

3 $GIoU = IoU - \dfrac{|C \backslash (A \cup B)|}{|C|}$

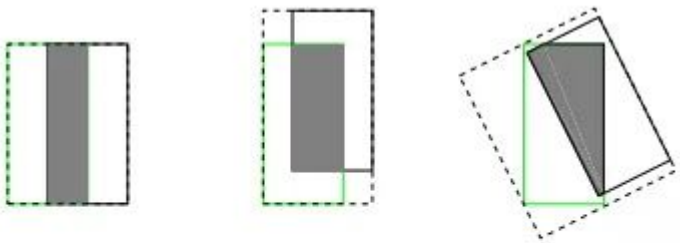


Figure 2. Three different ways of overlap between two rectangles with the exactly same IoU values, *i.e.* $IoU = 0.33$, but different $GIoU$ values, *i.e.* from the left to right $GIoU = 0.33, 0.24$ and -0.1 respectively. $GIoU$ value will be higher for the cases with better aligned orientation.

$$L_{GIoU} = 1 - GIoU$$

- 与IoU相似，GIoU也是一种距离度量，作为损失函数的话，满足损失函数的基本要求
- GIoU对scale不敏感
- GIoU是IoU的下界，在两个框无线重合的情况下，IoU=GIoU
- IoU取值[0,1]，但GIoU有对称区间，取值范围[-1,1]。在两者重合的时候取最大值1，在两者无交集且无限远的时候取最小值-1，因此GIoU是一个非常好的距离度量指标。
- 与IoU只关注重叠区域不同，GIoU不仅关注重叠区域，还关注其他的非重合区域，能更好的反映两者的重合度。

参考

1. ^ 匈牙利算法 https://baike.baidu.com/item/%E5%8C%88%E7%89%99%E5%88%A9%E7%AE%97%E6%B3%95/9089246?fr=aladdin
2. ^ Generalized intersection over union http://openaccess.thecvf.com/content_CVPR_2019/html/Rezatofighi_Generalized_Intersection_Over_Union_A_Metric_and_a_Loss_for_CVPR_2019_paper.html

发布于 06-01

目标检测　深度学习（Deep Learning）　计算机视觉

2 条评论

切换为时间排序

写下你的评论...

精选评论（1）

GYxiaOH 06-02

终于有大佬分享了。。大概看明白了，我想问下大佬，那个匈牙利算法是不是就是为了匹配预测序列和真实框啊，然后一一对应算loss，它本身自己并不是个loss，就是个二分图匹配算法？我在代码里好像也没看到这个loss

赞 查看回复

评论（2）

GYxiaOH 06-02

终于有大佬分享了。。大概看明白了，我想问下大佬，那个匈牙利算法是不是就是为了匹配预测序列和真实框啊，然后一一对应算loss，它本身自己并不是个loss，就是个二分图匹配算法？我在代码里好像也没看到这个loss

赞

HUST小菜鸡 (作者) 回复 GYxiaOH 06-02

我认为啊匈牙利算法只是一个优化策略，在代码中的损失的确不是这个，而且实施也不一样 For efficiency reasons, the targets don't include the no_object. Because of this, in general,there are more predictions than targets. In this case, we do a 1-to-1 matching of the best predictions,while the others are un-matched (and thus treated as non-objects).你看下matcher.py的class HungarianMatcher

赞