基于transformer的端到端目标检测

NicolasCarion[?]，FranciscoMassa[?]，Gabriel Synnaeve, Nicolas Usunier,
Alexander Kirillov，和Sergey Zagoruyko

Facebook的人工智能

摘要本文提出一种新方法，将目标检测视为直接集预测问题。我们的方法简化了检测管道，有效地消除了对许多手工设计组件的需要，如非极大值抑制程序或明确编码我们关于任务的先验知识的锚生成。新框架的主要组成部分，称为检测变压器或DETR，是基于集合的全局损失，通过双向匹配强制进行唯一预测，以及变压器编码器-解码器架构。给定一组固定的小学习对象查询，DETR对对象和全局图像上下文的关系进行推理，以直接并行输出最终的预测集。与许多其他现代检测器不同，新模型在概念上很简单，不需要专门的库。在具有挑战性的COCO目标检测数据集上，DETR展示了与成熟且高度优化的Faster R-CNN基线相当的准确性和运行时性能。此外，DETR可以很容易地推广到统一的全视分割。实验表明，它明显优于有竞争力的基线。训练代码和预训练模型可在https://github.com/facebookresearch/detr上获得。

Introduction

对象检测的目标是为每个感兴趣的对象预测一组边界框和类别标签。现代检测器通过在大量建议集[37,5]、锚点[23]或窗口中心[53,46]上定义代理回归和分类问题，以间接的方式解决这一集合预测任务。它们的性能受到以下因素的显著影响:瓦解近似重复预测的后处理步骤、锚点集的设计以及将目标框分配给锚点[52]的启发式方法。为了简化这些管道，本文提出了一种直接集预测方法来绕过代理任务。这种端到端的哲学导致了复杂结构化预测任务(如机器翻译或语音识别)的显著进步，但在目标检测方面还没有:之前的尝试[43,16,4,39]要么添加了其他形式的先验知识，要么没有被证明在具有挑战性的基准上具有竞争力。本文旨在弥合这一差距。

[?] Equal contribution

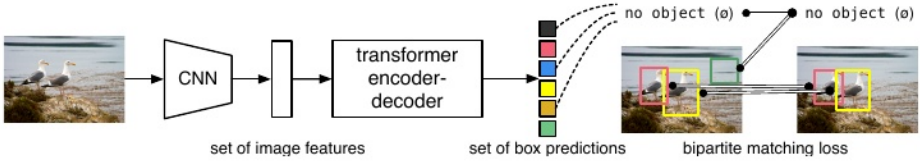


图1:DETR通过将通用CNN与变压器架构相结合，直接预测(并行)最终检测集。在训练过程中，二部图匹配将预测唯一地分配给地面真值框。无匹配的预测应该产生一个“无对象”(∅)类预测。

我们通过将对象检测视为直接集预测问题来简化训练管道。我们采用了一种基于transformer[47]的编码器-解码器架构，这是一种用于序列预测的流行架构。transformer的自注意力机制显式地对序列中元素之间的所有成对相互作用进行建模，使这些架构特别适合于集合预测的特定约束，如删除重复预测。

我们的检测变压器(DETR，见图1)一次预测所有对象，并使用一组损失函数进行端到端训练，该函数在预测对象和真实对象之间执行双部分匹配。DETR通过删除多个手动设计的组件来简化检测管道，这些组件编码先验知识，如空间锚定或非最大抑制。与大多数现有的检测方法不同，DETR不需要任何自定义层，因此可以很容易地在包含标准CNN和变压器类的任何框架中复制。

与之前大多数直接集预测工作相比，DETR的主要特征是二部匹配损失和变压器与(非自回归)并行解码的结合[29,12,10,8]。相比之下，之前的工作侧重于用rnn进行自回归解码[43,41,30,36,42]。我们的匹配损失函数将一个预测唯一地分配给一个地面真值对象，并且对预测对象的排列是不变的，因此我们可以并行发射它们。

我们在最流行的目标检测数据集之一COCO上评估DETR[24]，对比非常有竞争力的Faster R-CNN基线[37]。Faster R-CNN经过多次设计迭代，其性能自最初发布以来有了很大的提高。我们的实验表明，我们的新模型取得了相当的性能。更准确地说，DETR在大型对象上表现出明显更好的性能，这可能是由变压器的非局部计算实现的。然而，它在小物体上获得了较低的性能。我们期望未来的工作将像FPN[22]的发展对Faster R-CNN所做的那样，在这方面进行改进。

DETR的训练设置在多个方面与标准目标检测器不同。新模型需要超长的训练时间表和好处

[1] In our work we use standard implementations of Transformers [47] and ResNet [15] backbones from standard deep learning libraries.

来自transformer中的辅助解码损失。彻底探索了哪些组件对演示的性能至关重要。

DETR的设计精神很容易扩展到更复杂的任务。在我们的实验中，我们证明了在预训练的DETR上训练的简单分割头在Panoptic分割上的表现优于竞争基线[19]，这是一项具有挑战性的像素级识别任务，最近得到了普及。

2 相关工作

我们的工作建立在几个领域的之前工作的基础上:集合预测的二分匹配损失，基于transformer的编码器-解码器架构，并行解码和目标检测方法。

2.1 集预测

目前还没有一个典型的深度学习模型可以直接预测集合。基本的集合预测任务是多标签分类(参见例如，[40,33]以获取计算机视觉背景下的参考)，对于这些问题，基准方法，即一对其余，不适用于元素之间存在底层结构(即，几乎相同的盒子)的检测等问题。这些任务中的第一个困难是避免近似重复。目前大多数检测器使用后处理(如非极大值抑制)来解决这个问题，但直接集合预测是后处理免费的。它们需要全局推理方案，对所有预测元素之间的交互进行建模，以避免冗余。对于固定大小的集合预测，密集的全连接网络[9]是足够的，但成本很高。一种通用的方法是使用自回归序列模型，如循环神经网络[48]。在所有情况下，损失函数应该通过预测的排列保持不变。通常的解决方案是基于匈牙利算法[20]设计一个损失函数，在ground truth和prediction之间找到一个二分匹配。这强制了置换不变性，并保证每个目标元素都有唯一的匹配。我们遵循二分图匹配损失方法。然而，与之前的大多数工作相比，我们放弃了自回归模型，并使用具有并行解码的transformer，我们在下面描述。

2.2 transformer和并行解码

transformer是由Vaswani等人[47]引入的，作为一种新的基于注意力的机器翻译构建模块。注意力机制[2]是聚合整个输入序列信息的神经网络层。transformer引入了自注意力层，与非局部神经网络[49]类似，它扫描序列的每个元素，并通过从整个序列中聚合信息来更新它。基于注意力的模型的一个主要优点是它的全局计算和完美的记忆，这使得它比rnn更适合于长序列。transformer现在是

在自然语言处理、语音处理和计算机视觉的许多问题上取代rnn[8,27,45,34,31]。

在早期的序列到序列模型[44]之后，transformer首先被用于自回归模型，逐个生成输出token。然而，抑制型推理成本(与输出长度成正比，且难以批量处理)导致了并行序列生成的发展，在音频[29]、机器翻译[12,10]、单词表示学习[8]，以及最近的语音识别[6]等领域。我们还将transformer和并行解码结合起来，以获得它们在计算成本和执行集合预测所需的全局计算能力之间的合适权衡。

2.3 对象检测

大多数现代目标检测方法相对于一些初始猜测进行预测。两阶段检测器[37,5]预测框w.r.t.提案，而单阶段方法预测w.r.t.锚点[23]或可能的对象中心网格[53,46]。最近的工作[52]表明，这些系统的最终性能在很大程度上取决于这些初始猜测的确切设置方式。在我们的模型中，我们能够删除这个手工制作的过程，并通过对输入图像而不是锚点的绝对框预测直接预测检测集来简化检测过程。

基于集合的损失。几个目标检测器[9,25,35]使用了二部图匹配损失。然而，在这些早期的深度学习模型中，不同预测之间的关系仅用卷积或全连接层来建模，手工设计的NMS后处理可以提高它们的性能。最近的检测器[37,23,53]在基础真值和预测值之间使用非唯一分配规则以及NMS。

可学习的NMS方法[16,4]和关系网络[17]显式地用注意力对不同预测之间的关系进行建模。使用直接设置损失，它们不需要任何后处理步骤。然而，这些方法采用了额外的手工设计的上下文特征，如建议框坐标，以有效地对检测之间的关系进行建模，同时我们寻找减少模型中编码的先验知识的解决方案。

复发性探测器。与我们的方法最接近的是对象检测[43]和实例分割的端到端集合预测[41,30,36,42]。与我们类似，他们使用基于CNN激活的编码器-解码器架构的二部匹配损失来直接产生一组边界框。然而，这些方法只在小型数据集上进行了评估，而没有针对现代基线进行评估。特别是，它们是基于自回归模型(更准确地说，是rnn)，因此它们没有利用最近的transformer进行并行解码。

3 DETR模型

两个成分对于检测中的直接集合预测是必不可少的:(1)一个集合预测损失，它强制在预测和真实值之间进行唯一匹配

盒子;(2)一个架构，可以预测(在一次传递中)一组对象，并对它们的关系进行建模。我们在图2中详细描述了我们的架构。

3.1 对象检测集预测损失

在通过解码器的单次传递中，DETR推断出固定大小的N个预测集，其中N被设置为明显大于图像中典型对象的数量。训练的主要困难之一是相对于真实值对预测对象(类别、位置、大小)进行评分。我们的损失在预测和地面真值对象之间产生最优的二部图匹配，然后优化特定对象(边界框)损失。

让我们用y表示对象的基准真值集，y⊠= {y⊠$_i$} $_{Ni=1}$表示N个预测的集合。假设N大于图像中物体的数量，我们也认为y是一个大小为N的集合，用∅(无物体)填充。为了找到这两个集合之间的二分匹配，我们搜索N个元素的排列σ∈S_N，其代价最低:

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}), \tag{1}$$

其中L_{match}(y_i，y⊠$_{σ(i)}$)是ground truthy_i和索引σ(i)的预测之间的成对匹配成本。这种最优分配是在之前的工作(例如[43])之后，用匈牙利算法高效地计算出来的。

匹配成本同时考虑了类预测和预测框与地面真值框的相似性。地面真值集的每个元素i都可以看作是一个y_i=(c_i，b_i)其中c_i是目标类标签(可能是∅)，b_i∈[0,1]-^{4}是一个定义地面真值框中心坐标及其相对于图像大小的高度和宽度的向量。对于索引σ(i)的预测，我们定义类c_i的概率为p⊠$_{σ(i)}$(c_i)，预测框为⊠bσ(i)。使用这些符号，我们定义Lmatch(yi, y⊠σ(i))为−1 $_{ci6=∅}$p⊠σ(i) (ci) + 1{ci6=∅}Lbox(bi，⊠bσ(i))。

这个寻找匹配的过程扮演着与现代检测器中用于将候选框[37]或锚点[22]匹配到真值对象的启发式分配规则相同的角色。主要的区别在于，对于没有重复的直接集预测，我们需要找到一对一的匹配。

第二步是计算损失函数，即上一步匹配的所有对的匈牙利损失。我们定义的损失类似于常见目标检测器的损失，即用于类别预测的负对数似然和稍后定义的框损失的线性组合:

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right], \tag{2}$$

其中σ⊠是第一步(1)中计算的最优分配。在实践中，当c_i=∅by a factor 10时，我们对对数概率项进行降权以说明

类不平衡。这类似于Faster R-CNN训练程序如何通过对[37]进行下采样来平衡正/负建议。注意，对象和∅之间的匹配成本不依赖于预测，这意味着在这种情况下，成本是一个常数。在匹配代价中，我们使用概率$p\boxtimes_{\sigma\boxtimes(i)}(c_i)$而不是对数概率。这使得类预测术语与$L_{box}(\cdot, \cdot)$可公度(如下所述)，并且我们观察到更好的经验表现。

边界框损失。匹配成本和匈牙利损失的第二部分是对边界框进行评分的$L_{box}(\cdot)$。与许多将框预测作为Δw.r.t的检测器不同。一些最初的猜测，我们直接进行框预测。虽然这种方法简化了实现，但它提出了一个与损失的相对规模有关的问题。最常用的`1loss对于小盒子和大盒子将有不同的尺度，即使它们的相对误差是相似的。为了缓解这个问题，我们使用`1损失和广义IoU损失[38]$L_{iou}(\cdot, \cdot)$的线性组合，这是尺度不变的。总的来说，我们的盒损失是$L_{box}(b_i, \boxtimes b_{\sigma(i)})$定义为λiouLiou(bi, ⊠bσ(i)) + λL1||bi−⊠bσ(i) ||1其中λiou, λL1∈R是超参数。这两种损失通过批内对象的数量进行归一化。

3.2 DETR架构

整个DETR体系结构非常简单，如图2所示。它包含三个主要组件，我们在下面描述:提取紧凑特征表示的CNN骨干，编码器-解码器变压器，以及进行最终检测预测的简单前馈网络(FFN)。

与许多现代检测器不同，DETR可以在任何深度学习框架中实现，该框架提供了一个通用的CNN主干和一个只有几百行代码的变压器架构实现。在PyTorch中，DETR的推理代码可以用不到50行来实现[32]。我们希望我们方法的简洁性能够吸引新的研究人员加入检测社区。

骨干。从初始图像$x_{img}\in R^{3\times H0\times W_0}$(具有3个颜色通道s2)开始，传统的CNN主干生成低分辨率激活图$\in R^{C\times H\times W}$。我们使用的典型值是C = 2048和H, W = $_{H32^0}$, $_{W32^{0\circ}}$

变压器编码器。首先，1x1卷积将高层激活图f的通道维度从C降低到更小的维度d。创建一个新的特征图$z_0\in R^{d\times H\times W}$。编码器期望一个序列作为输入，因此我们将$z_0$的空间维度折叠为一维，产生d×HW特征图。每个编码器层都有一个标准的架构，由一个多头自关注模块和一个前馈网络(FFN)组成。由于transformer架构是置换不变性的，我们补充了固定位置编码[31,3]，这些编码被添加到每个注意力层的输入中。我们遵从补充材料对架构的详细定义，该定义遵循[47]中描述的定义。

[2] The input images are batched together, applying 0-padding adequately to ensure they all have the same dimensions (H_0, W_0) as the largest image of the batch.

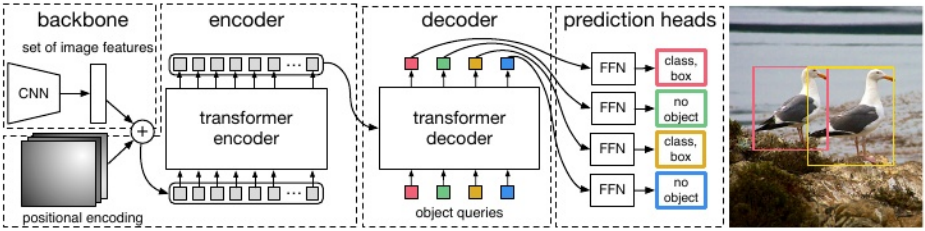


图2:DETR使用传统的CNN主干来学习输入图像的二维表示。模型将其展平，并在将其传递给transformer编码器之前用位置编码对其进行补充。然后，transformer解码器将少量固定数量的学习到的位置嵌入(我们称之为对象查询)作为输入，并额外关注编码器输出。我们将解码器的每个输出嵌入传递给一个共享前馈网络(FFN)，该网络预测检测(类和边界框)或“无对象”类。

变压器解码器。解码器遵循transformer的标准架构，使用多头自-编码器-解码器注意力机制转换大小为d的N个嵌入。与原始transformer的不同之处在于，我们的模型在每个解码器层并行解码N个对象，而Vaswani et al.[47]使用了一个自回归模型，每次预测一个元素的输出序列。我们建议不熟悉这些概念的读者参考补充材料。由于解码器也是置换不变性的，因此N个输入嵌入必须不同才能产生不同的结果。这些输入嵌入是学习到的位置编码，我们称之为对象查询，与编码器类似，我们将它们添加到每个注意力层的输入中。这N个对象查询被解码器转换成一个输出嵌入。然后，它们被前馈网络(在下一小节中描述)独立解码为框坐标和类标签，得到N个最终预测。在这些嵌入上使用self- and encoder-decoder attention，模型使用它们之间的成对关系全局地推理所有对象，同时能够使用整个图像作为上下文。

预测前馈网络(ffn)最终的预测由一个具有ReLU激活函数和隐藏维度d的3层感知器和一个线性投影层计算。FFN预测输入图像周围框的归一化中心坐标、高度和宽度，lin-ear层使用softmax函数预测类标签。由于我们预测的是N个固定大小的边界框集合，其中N通常比图像中实际感兴趣的对象数量大得多，因此使用了额外的特殊类标签∅来表示在一个槽内没有检测到对象。这个类与标准对象检测方法中的“背景”类扮演着类似的角色。

辅助解码损失。我们发现在训练过程中使用辅助损失[1]在解码器中是有帮助的，特别是帮助模型输出正确的数字

每个类别的对象。我们在每个解码器层后添加预测ffn和匈牙利损失。所有预测ffn都共享它们的参数。我们使用一个额外的共享层范数来规范化来自不同解码器层的预测ffn的输入。

4 实验

我们表明，与Faster R-CNN相比，DETR在COCO的定量评估中取得了有竞争力的结果。对架构和损失进行了详细的消融研究，并给出了见解和定性结果。最后，为了证明DETR是一个通用的、可扩展的模型，我们给出了在全光学分割上的结果，在固定的DETR模型上只训练一个小的扩展。我们在https://github.com/facebookresearch/detr上提供了代码和预训练模型来重现我们的实验。

数据集。我们在COCO 2017检测和全视分割数据集上进行实验[24,18]，其中包含118k张训练图像和5k张验证图像。每张图像都标注了边界框和全景分割。平均每张图像有7个实例，训练集中单张图像最多63个实例，在相同的图像上从小到大不等。如果没有指定，我们将AP报告为bbox AP，即多个阈值上的积分度量。为了与Faster R-CNN进行比较，我们报告了最后一个训练epoch的验证AP，对于消融，我们报告了过去10个epoch的验证结果中位数。

技术细节。我们使用AdamW[26]训练DETR，将初始变压器的学习率设置为10^{-4}，主干的学习率设置为10^{-5}，权重衰减为10^{-4}。所有的transformer权重都用Xavier init[11]初始化，而backbone是用来自torchvision的imagenet预训练的ResNet模型[15]与冻结的batchnorm层。我们报告了两个不同的骨干的结果:ResNet-50和ResNet-101。相应的模型分别称为DETR和DETR- r101。在[21]之后，我们还通过对主干的最后一阶段添加一个扩张并从这一阶段的第一个卷积中删除一个步幅来增加特征分辨率。相应的模型分别称为DETR-DC5和DETR-DC5- r101(扩张的C5期)。这种修改将分辨率提高了2倍，从而提高了小物体的性能，代价是编码器的自注意力成本提高了16倍，导致计算成本整体增加了2倍。表1给出了这些模型与Faster R-CNN FLOPs的完整比较。

我们使用尺度增强，调整输入图像的大小，使最短的边至少为480，最多为800像素，最长的边最多为1333[50]。为了通过编码器的自关注来帮助学习全局关系，我们还在训练期间应用随机裁剪增强，将性能提高了大约1 AP。具体而言，以0.5的概率将训练图像裁剪为随机矩形块，然后将其重新调整大小为800-1333。transformer以默认的dropout为0.1进行训练。在推理

表1:ResNet-50和ResNet-101骨干网在COCO验证集上与Faster R-CNN的比较。顶部部分显示Detectron2[50]中更快R-CNN模型的结果，中间部分显示具有GIoU[38]、随机作物训练时间增强和长9倍训练计划的更快R-CNN模型的结果。DETR模型获得了与经过大量调整的更快R-CNN基线相当的结果，具有较低的AP_S但大大改善了AP_L。我们使用torchscript 更快R-CNN和DETR模型来测量FLOPS和FPS。名称中没有R101的结果对应于ResNet-50。

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**

时间，一些槽预测空类。为了优化AP，我们使用相应的置信度，用得分第二高的类别覆盖这些槽的预测。与过滤空槽相比，这将使AP提高2点。其他训练超参数可以在A.4节中找到。对于我们的消融实验，我们使用300 epoch的训练时间表，在200 epoch后学习率下降10倍，其中单个epoch是对所有训练图像的一次遍历。在16个V100 GPU上训练300次epoch的基线模型需要3天，每个GPU有4张图像(因此总批大小为64)。与更快R-CNN相比，我们训练了500次，400次后学习率下降。与较短的时间段相比，该时间段增加了1.5个AP数。

4.1 与更快R-CNN的比较

变形金刚通常使用亚当或Adagrad优化器进行训练，训练计划很长，而且会辍学，DETR也是如此。然而，更快R-CNN是用最小数据增强的SGD训练的，我们不知道亚当或dropout的成功应用。尽管存在这些差异，但我们试图使更快R-CNN基线更强。为了使其与DETR保持一致，我们将广义IoU[38]添加到盒损失中，相同的随机作物增加和已知的可以改善结果的长时间训练[13]。结果如表1所示。在顶部部分，我们展示了Detectron2模型动物园[50]对使用3x时间表训练的模型的更快R-CNN结果。在中间部分，我们显示了相同模型但经过训练的结果(带有“+”)

表2:编码器大小的影响。每一行对应一个编码器层数可变、解码器层数固定的模型。随着编码器层数的增加，性能逐渐提高。

#layers	GFLOPS/FPS	#params	AP	AP_{50}	AP_S	AP_M	AP_L
0	76/28	33.4M	36.7	57.4	16.8	39.6	54.2
3	81/25	37.4M	40.1	60.6	18.5	43.8	58.6
6	86/23	41.3M	40.6	61.6	19.9	44.3	60.2
12	95/20	49.2M	41.6	62.1	19.8	44.9	61.9

使用9x时间表(109个epoch)和所描述的增强，总共增加了1-2个AP。在表1的最后一部分中，我们展示了多个DETR模型的结果。为了在参数数量上具有可比性，我们选择了一个具有6个transformer和6个宽度为256的解码器层和8个注意力头的模型。与带FPN的更快R-CNN一样，该模型有41.3M参数，其中23.5M在ResNet-50中，17.8M在变压器中。尽管随着训练时间的延长，更快R-CNN和DETR仍有可能进一步提高，但我们可以得出结论，在参数数量相同的情况下，DETR可以与更快R-CNN竞争，在COCO值子集上达到42 AP。DETR实现这一目标的方式是通过改进AP_L(+7.8)，但请注意该模型仍然落后于AP_S(-5.5)。具有相同参数数量和相似的FLOP计数的DETR-DC5具有更高的AP，但在AP_S方面仍然明显落后。更快R-CNN和ResNet-101骨干网的DETR也显示出类似的结果。

4.2 消融

transformer解码器中的注意力机制是对不同检测的特征表示之间的关系进行建模的关键组件。在消融分析中，我们探索了架构和损耗的其他组件如何影响最终性能。我们选择基于resnet -50的DETR模型，该模型具有6层编码器，6层解码器，宽度为256。该模型参数为41.3M，在短、长时间段AP分别达到40.6和42.0,FPS运行速度为28，类似于具有相同主干的Faster R-CNN-FPN。

编码器层数。我们通过改变编码器层数来评估全局图像级自我注意的重要性(表2)。没有编码器层，整体AP下降了3.9分，大型物体的AP下降了6.0分。我们假设，通过使用全局场景推理，编码器对于解缠物体很重要。在图3中，我们将经过训练的模型的最后一层编码器层的注意力图可视化，重点关注图像中的几个点。编码器似乎已经分离了实例，这很可能简化了解码器的对象提取和定位。

解码器层数。我们在每个解码层之后应用辅助损失(见第3.2节)，因此，预测ffn被设计训练为pre-

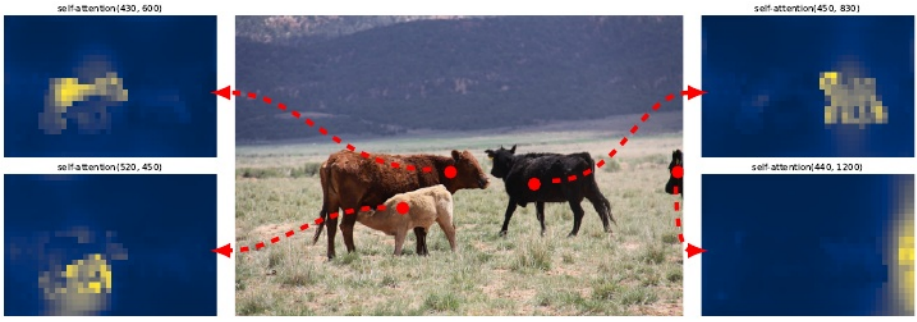


图3:一组参考点的编码器自注意力。编码器能够分离单个实例。利用基线DETR模型对验证集图像进行预测。

从每个解码器层的输出中提取Dict对象。我们通过评估解码每个阶段预测的对象来分析每个解码器层的重要性(图4)。AP和AP_{50}在每一层之后都有所提高，在第一层和最后一层之间的AP提高了+8.2/9.5，这是非常显著的。由于DETR的损失是基于集合的，因此在设计上不需要NMS。为了验证这一点，我们为每个解码器后的输出运行一个带有默认参数[50]的标准NMS过程。NMS提高了来自第一个解码器的预测的性能。这可以用transformer的单个解码层无法计算输出元素之间的任何互相关来解释，因此它容易对同一个对象进行多次预测。在第二层和后续层中，激活上的自注意力机制允许模型抑制重复预测。我们观察到NMS带来的改善随着深度的增加而减弱。在最后一层，我们观察到AP中的小损失，因为NMS错误地删除了真正预测。

与可视化编码器注意力类似，我们在图6中可视化解码器注意力，将每个预测对象的注意力图着色为不同的颜色。我们观察到，解码器的注意力是相当局部的，这意味着它主要关注物体的四肢，如头或腿。我们假设，在编码器通过全局注意力分离实例后，解码器只需要关注端点以提取类和对象边界。

FFN的重要性。变压器内部的FFN可以看作是1个×1卷积层，使得编码器类似于注意力增强卷积网络[3]。我们尝试将其完全移除，只在transformer层中保留注意力。通过将网络参数数量从41.3M减少到28.7M，仅留下108m在变压器中，性能下降了2.3 AP，因此我们得出FFN对于取得良好效果很重要。

位置编码的重要性。在我们的模型中有两种位置编码:空间位置编码和输出位置编码-

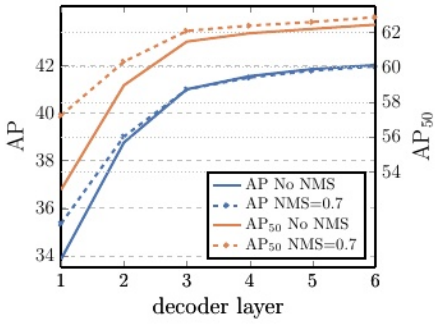


图4:各解码器层后的AP和AP_{50}性能。评估单个长调度基线模型。DETR不需要设计上的NMS，这个图验证了这一点。NMS降低了最后一层的AP，去掉了TP预测，但提高了第一层解码器层的AP，去掉了双重预测，因为第一层没有通信，并略微提高了AP50。

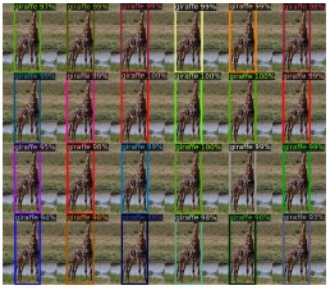


图5:稀有类的分布外泛化。即使训练集中没有超过13只长颈鹿的图像，DETR也不难推广到同一类的24个或更多实例。

Ings(对象查询)。我们对固定编码和学习编码的各种组合进行了实验，结果可以在表3中找到。输出位置编码是必需的，不能删除，因此我们实验要么在解码器输入时传递它们一次，要么在每个解码器注意力层添加到查询中。在第一个实验中，我们完全删除了空间位置编码，并在输入处传递了输出位置编码，有趣的是，该模型仍然实现了超过32 AP，比基线损失了7.8 AP。然后，我们将固定正弦空间位置编码和输出编码在输入处传递一次，如原始变压器[47]，发现与直接传递位置编码相比，这导致1.4 AP下降。传递到注意力的学习到的空间编码给出了类似的结果。令人惊讶的是，我们发现在编码器中不传递任何空间编码只会导致1.3 AP的轻微下降。当我们将编码传递给注意时，它们在所有层之间共享，并且输出编码(对象查询)总是被学习。

考虑到这些损耗，我们得出结论，变压器组件:编码器中的全局自关注、FFN、多个解码器层和位置编码，都对最终的目标检测性能有重要贡献。

损失的替代品。为了评估匹配成本和损失的不同组成部分的重要性，我们训练了几个模型，分别打开和关闭它们。损失有三个组成部分:分类损失、$_1$边界盒距离损失和GIoU[38]损失。分类损失对训练是必不可少的，不能关闭，所以我们训练了一个没有边界框距离损失的模型，和一个没有GIoU损失的模型，并与基线进行比较，这三种损失都训练过。结果如表4所示。GIoU自身的损失

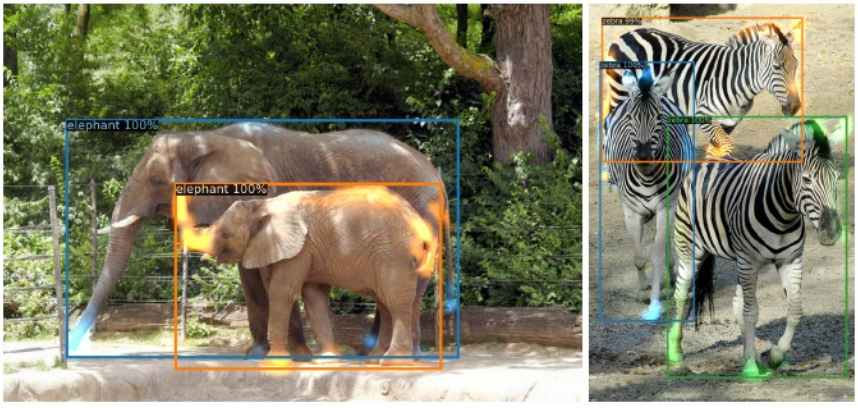


图6:每个预测对象的解码器注意力可视化(来自COCO值集的图像)。采用DETR-DC5模型进行预测。注意力分数用不同的颜色对不同的物体进行编码。Decoder通常会关注物体的四肢，比如腿和头。彩色图像观看效果最好。

表3:与基线(最后一行)相比，不同位置编码的结果，基线在编码器和解码器的每个注意力层都传递了固定的正弦pos编码。学习到的嵌入在所有层之间共享。不使用空间位置编码会导致AP显著下降。有趣的是，在解码器中传递它们只会导致较小的AP下降。所有这些模型都使用学习到的输出位置编码。

spatial pos. enc. encoder	spatial pos. enc. decoder	output pos. enc. decoder	AP	Δ	AP_{50}	Δ
none	none	learned at input	32.8	-7.8	55.2	-6.5
sine at input	sine at input	learned at input	39.2	-1.4	60.0	-1.6
learned at attn.	learned at attn.	learned at attn.	39.6	-1.0	60.7	-0.9
none	sine at attn.	learned at attn.	39.3	-1.3	60.3	-1.4
sine at attn.	sine at attn.	learned at attn.	**40.6**	-	**61.6**	-

表4:损失分量对AP的影响。我们训练了两个关闭',损失和GIoU损失的模型，并观察到',单独给出的结果很差，但当与GIoU结合使用时，AP_M和AP_L得到了改善。我们的基线(最后一行)结合了这两种损失。

class	ℓ_1	GIoU	AP	Δ	AP_{50}	Δ	AP_S	AP_M	AP_L
✓	✓		35.8	-4.8	57.3	-4.4	13.7	39.8	57.9
✓		✓	39.9	-0.7	**61.6**	0	**19.9**	43.2	57.9
✓	✓	✓	**40.6**	-	**61.6**	-	**19.9**	**44.3**	**60.2**

对于大多数模型性能，仅损失0.7 AP与综合损失的基线。使用',而不使用GIoU的结果很差。我们只研究了

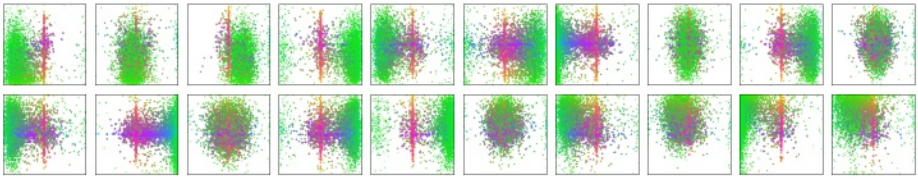


图7:DETR解码器中总共N = 100个预测槽中的20个，对来自COCO 2017 val集的所有图像的所有框预测的可视化。每个盒子预测被表示为一个点，其中心的坐标在每个图像大小归一化的1乘1正方形中。这些点被颜色编码，因此绿色对应小框，红色对应大的水平框，蓝色对应大的垂直框。我们观察到，每个插槽通过几种操作模式学习专攻某些领域和盒子大小。我们注意到，几乎所有的槽都有一个预测COCO数据集中常见的大图像范围框的模式。

不同损失的简单消融(每次使用相同的权重)，但其他组合它们的方法可能会获得不同的结果。

4.3 分析

在图7中，我们可视化了COCO 2017 val集中所有图像的不同槽预测框。DETR为每个查询槽学习不同的专门化。我们观察到每个槽都有几种专注于不同区域和盒子大小的操作模式。特别是，所有插槽都有预测图像范围内盒子的模式(可见在图的中间对齐的红点)。我们假设这与COCO中物体的分布有关。

对未见过的实例数量的泛化。COCO中的一些类不能很好地表示为同一图像中同一类的许多实例。例如，训练集中没有超过13只长颈鹿的图像。我们创建了一个合成图像3来验证DETR的泛化能力(见图5)。我们的模型能够在明显不分布的图像上找到所有24只长颈鹿。这个实验证实了在每个对象查询中并没有很强的类特化。

4.4 DETR用于全视分割

全景分割[19]最近引起了计算机视觉界的广泛关注。类似于将Faster R-CNN[37]扩展为Mask R-CNN [14]， DETR可以通过在解码器输出顶部添加掩码头来自然扩展。在本节中，我们演示了这样的头部可以通过处理stuff和thing类来产生全景分割[19]

[3] Base picture credit: https://www.piqsels.com/en/public-domain-photo-jzlwu

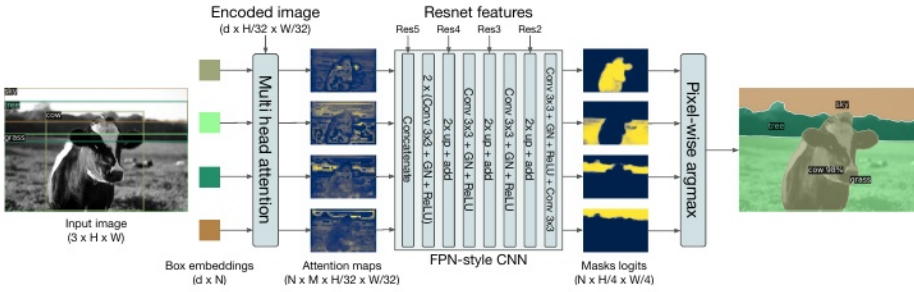


图8:全景头的插图。为每个检测到的对象并行生成二进制掩模，然后使用逐像素argmax合并掩模。

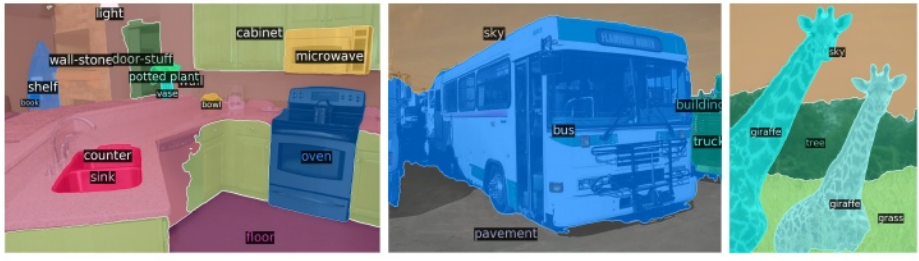


图9:DETR-R101生成的全视分割定性结果。DETR以统一的方式对事物和东西产生对齐的掩码预测。

以统一的方式。我们在COCO数据集的全景注释上进行了实验，该数据集有53个材料类别和80个事物类别。

我们训练DETR来预测COCO上的东西和东西类的盒子，使用相同的配方。为了使训练成为可能，预测盒子是必需的，因为匈牙利匹配是使用盒子之间的距离计算的。我们还添加了一个掩码头，用于预测每个预测框的二进制掩码，参见图8。它将每个对象的transformer解码器的输出作为输入，并在编码器的输出上计算此嵌入的多头(具有M个头)注意力分数，在小分辨率中为每个对象生成M个注意力热图。为了做出最终的预测并增加分辨率，使用了类似fpn的架构。我们在补充中对该架构进行了更详细的描述。掩模的最终分辨率为stride 4，每个掩模都是使用DICE/F-1损失[28]和焦点损失[23]独立监督的。

面具头可以联合训练，也可以分两步训练，我们只训练盒子的DETR，然后冻结所有的权重，只训练面具头25次。实验表明，这两种方法给出了类似的结果，我们报告了使用后一种方法的结果，因为它导致了较短的总时钟训练时间。

表5:与COCO val数据集上最先进的方法UPSNet[51]和PanopticFPN[18]的比较。我们使用与DETR相同的数据增强量，以18倍的时间表重新训练PanopticFPN，以进行公平比较。UPSNet使用1x时间表，UPSNet- m是具有多尺度测试时间增强的版本。

Model	Backbone	PQ	SQ	RQ	PQ^{th}	SQ^{th}	RQ^{th}	PQ^{st}	SQ^{st}	RQ^{st}	AP
PanopticFPN++	R50	42.4	79.3	51.6	49.2	82.4	58.8	32.3	74.8	40.6	37.7
UPSnet	R50	42.5	78.0	52.5	48.6	79.4	59.6	33.4	75.9	41.7	34.3
UPSnet-M	R50	43.0	79.1	52.8	48.9	79.7	59.7	34.1	78.2	42.3	34.3
PanopticFPN++	R101	44.1	79.5	53.3	**51.0**	**83.2**	60.6	33.6	74.0	42.1	**39.7**
DETR	R50	43.4	79.3	53.8	48.2	79.8	59.5	36.3	78.5	45.3	31.1
DETR-DC5	R50	44.6	79.8	55.0	49.4	80.5	60.6	**37.3**	**78.7**	**46.5**	31.9
DETR-R101	R101	**45.1**	**79.9**	**55.5**	50.5	80.9	**61.7**	37.0	78.5	46.0	33.0

为了预测最终的全景分割，我们简单地在每个像素的掩码分数上使用argmax，并将相应的类别分配给结果掩码。这个过程保证了最终的蒙版没有重叠，因此，DETR不需要启发式[19]，通常用于对齐不同的蒙版。

培训的细节。我们按照边界盒检测的配方训练DETR、DETR- dc5和DETR-r101模型来预测COCO数据集中的东西和事物类周围的盒子。新的mask头部进行了25个epoch的训练(详细信息请参见补充)。在推理过程中，我们首先过滤置信度低于85%的检测，然后计算每个像素的argmax以确定每个像素属于哪个掩码。然后，我们将相同内容类别的不同掩膜预测折叠为一个，并过滤空的(小于4像素)。

主要的结果。定性结果见图9。在表5中，我们将我们统一的全景分割方法与几种不同对待事物和东西的既定方法进行了比较。我们报告的Panoptic质量(PQ)和分解的东西(PQ^{th})和东西(PQ^{st})。我们还报告遮罩AP(在things类上计算)，在任何panoptic后处理之前(在我们的例子中，在采取逐像素argmax之前)。我们表明，DETR优于COCO-val 2017上发表的结果，以及我们强大的PanopticFPN基线(为了公平比较，使用与DETR相同的数据增强训练)。结果分解表明，DETR在材料类上尤其占主导地位，我们假设编码器注意允许的全局推理是这一结果的关键因素。对于事物类，尽管与掩模AP计算的基线相比，存在高达8 mAP的严重缺陷，但DETR仍然具有竞争力PQ^{th}。我们还在COCO数据集的测试集上评估了我们的方法，并获得了46个PQ。我们希望我们的方法能在未来的工作中激发对全景分割的完全统一模型的探索。

5 结论

提出了一种新的基于变换和二部匹配损失的目标检测系统DETR，用于直接集预测。该方法在具有挑战性的COCO数据集上获得了与优化的Faster R-CNN基线相当的结果。DETR易于实现，具有灵活的架构，易于扩展到全光分割，具有竞争力的结果。此外，它在大型对象上的性能明显优于Faster R-CNN，这可能要归功于自关注对全局信息的处理。

这种新的检测器设计也带来了新的挑战，特别是在训练、优化和小物体上的性能方面。目前的探测器需要几年的改进才能解决类似的问题，我们希望未来的工作能够成功地解决DETR的问题。

6 确认

我们感谢Sainbayar Sukhbaatar、Piotr Bojanowski、Natalia Neverova、David Lopez-Paz、Guillaume Lample、Danielle Rothermel、Kaiming He、Ross Girshick、Xinlei Chen和整个Facebook AI Research Paris团队的讨论和建议，没有这些讨论和建议，这项工作就不可能完成。

参考文献

1. Al-Rfou, R., Choe, D., Constant, N., Guo, M., Jones, L.: Character-level language modeling with deeper self-attention. In: AAAI Conference on Artificial Intelligence (2019)
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: ICLR (2015)
3. Bello, I., Zoph, B., Vaswani, A., Shlens, J., Le, Q.V.: Attention augmented convo-lutional networks. In: ICCV (2019)
4. Bodla, N., Singh, B., Chellappa, R., Davis, L.S.: Soft-NMS improving object detection with one line of code. In: ICCV (2017)
5. Cai, Z., Vasconcelos, N.: Cascade R-CNN: High quality object detection and in-stance segmentation. PAMI (2019)
6. Chan, W., Saharia, C., Hinton, G., Norouzi, M., Jaitly, N.: Imputer: Sequence modelling via imputation and dynamic programming. arXiv:2002.08926 (2020)
7. Cordonnier, J.B., Loukas, A., Jaggi, M.: On the relationship between self-attention and convolutional layers. In: ICLR (2020)
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (2019)
9. Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: Scalable object detection using deep neural networks. In: CVPR (2014)
10. Ghazvininejad, M., Levy, O., Liu, Y., Zettlemoyer, L.: Mask-predict: Parallel de-coding of conditional masked language models. arXiv:1904.09324 (2019)
11. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS (2010)

12. Gu, J., Bradbury, J., Xiong, C., Li, V.O., Socher, R.: Non-autoregressive neural machine translation. In: ICLR (2018)

13. He, K., Girshick, R., Doll´ ar, P.: Rethinking imagenet pre-training. In: ICCV (2019)

14. He, K., Gkioxari, G., Doll´ ar, P., Girshick, R.B.: Mask R-CNN. In: ICCV (2017)

15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)

16. Hosang, J.H., Benenson, R., Schiele, B.: Learning non-maximum suppression. In: CVPR (2017)

17. Hu, H., Gu, J., Zhang, Z., Dai, J., Wei, Y.: Relation networks for object detection. In: CVPR (2018)

18. Kirillov, A., Girshick, R., He, K., Doll´ ar, P.: Panoptic feature pyramid networks. In: CVPR (2019)

19. Kirillov, A., He, K., Girshick, R., Rother, C., Dollar, P.: Panoptic segmentation. In: CVPR (2019)

20. Kuhn, H.W.: The hungarian method for the assignment problem (1955)

21. Li, Y., Qi, H., Dai, J., Ji, X., Wei, Y.: Fully convolutional instance-aware semantic segmentation. In: CVPR (2017)

22. Lin, T.Y., Doll´ ar, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: CVPR (2017)

23. Lin, T.Y., Goyal, P., Girshick, R.B., He, K., Doll´ ar, P.: Focal loss for dense object detection. In: ICCV (2017)

24. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Doll´ ar, P., Zitnick, C. L.: Microsoft COCO: Common objects in context. In: ECCV (2014)

25. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: ECCV (2016)

26. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: ICLR (2017)

27. L¨ uscher, C., Beck, E., Irie, K., Kitza, M., Michel, W., Zeyer, A., Schluter, ¨ R., Ney, H.: Rwth asr systems for librispeech: Hybrid vs attention - w/o data augmentation. arXiv:1905. 03072 (2019)

28. Milletari, F., Navab, N., Ahmadi, S.A.: V-net: Fully convolutional neural networks for volumetric medical image segmentation. In: 3DV (2016)

29. Oord, A.v.d., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G.v.d., Lockhart, E., Cobo, L.C., Stimberg, F., et al.: Parallel wavenet: Fast high-fidelity speech synthesis. arXiv:1711.10433 (2017)

30. Park, E., Berg, A.C.: Learning to decompose for object detection and instance segmentation. arXiv:1511.06449 (2015)

31. Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., Tran, D.: Image transformer. In: ICML (2018)

32. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019)

33. Pineda, L., Salvador, A., Drozdzal, M., Romero, A.: Elucidating image-to-set pre-diction: An analysis of models, losses and datasets. arXiv:1904.05709 (2019)

34. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019)

35. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: CVPR (2016)

36. Ren, M., Zemel, R.S.: End-to-end instance segmentation with recurrent attention. In: CVPR (2017)

37. Ren, S., He, K., Girshick, R.B., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. PAMI (2015)

38. Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., Savarese, S.: General-ized intersection over union. In: CVPR (2019)

39. Rezatofighi, S.H., Kaskman, R., Motlagh, F.T., Shi, Q., Cremers, D., Leal-Taix´e, L., Reid, I.: Deep perm-set net: Learn to predict sets with unknown permutation and cardinality using deep neural networks. arXiv:1805.00613 (2018)

40. Rezatofighi, S.H., Milan, A., Abbasnejad, E., Dick, A., Reid, I., Kaskman, R., Cremers, D., Leal-Taix, l.: Deepsetnet: Predicting sets with deep neural networks. In: ICCV (2017)

41. Romera-Paredes, B., Torr, P.H.S.: Recurrent instance segmentation. In: ECCV (2015)

42. Salvador, A., Bellver, M., Baradad, M., Marqu´es, F., Torres, J., Gir´ o, X.: Recurrent neural networks for semantic instance segmentation. arXiv:1712.00617 (2017)

43. Stewart, R.J., Andriluka, M., Ng, A.Y.: End-to-end people detection in crowded scenes. In: CVPR (2015)

44. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: NeurIPS (2014)

45. Synnaeve, G., Xu, Q., Kahn, J., Grave, E., Likhomanenko, T., Pratap, V., Sri-ram, A., Liptchinsky, V., Collobert, R.: End-to-end ASR: from supervised to semi-supervised learning with modern architectures. arXiv:1911.08460 (2019)

46. Tian, Z., Shen, C., Chen, H., He, T.: FCOS: Fully convolutional one-stage object detection. In: ICCV (2019)

47. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NeurIPS (2017)

48. Vinyals, O., Bengio, S., Kudlur, M.: Order matters: Sequence to sequence for sets. In: ICLR (2016)

49. Wang, X., Girshick, R.B., Gupta, A., He, K.: Non-local neural networks. In: CVPR (2018)

50. Wu, Y., Kirillov, A., Massa, F., Lo, W.Y., Girshick, R.: Detectron2. https:// github.com/facebookresearch/detectron2 (2019)

51. Xiong, Y., Liao, R., Zhao, H., Hu, R., Bai, M., Yumer, E., Urtasun, R.: Upsnet: A unified panoptic segmentation network. In: CVPR (2019)

52. Zhang, S., Chi, C., Yao, Y., Lei, Z., Li, S.Z.: Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. arXiv:1912.02424 (2019)

53. Zhou, X., Wang, D., Kr¨ ahenb¨ uhl, P.: Objects as points. arXiv:1904.07850 (2019)

一个附录

A.1 初赛:多头注意层

由于我们的模型是基于Transformer架构的，我们在这里提醒一下我们用于耗尽的注意力机制的一般形式。注意力机制遵循[47]，除了遵循[7]的位置编码细节(见公式8)。

多头具有M个d维的多头注意力的一般形式是具有以下签名的函数(使用d0 = Md，并在括号中给出矩阵/张量的大小)

$$\text{mh-attn}: \underbrace{X_\text{q}}_{d\times N_\text{q}}, \underbrace{X_\text{kv}}_{d\times N_\text{kv}}, \underbrace{T}_{M\times 3\times d'\times d}, \underbrace{L}_{d\times d} \mapsto \underbrace{\tilde{X}_\text{q}}_{d\times N_\text{q}} \tag{3}$$

其中X_q是长度为N_q的查询序列，X_{kv}是长度为N_{kv}的键值序列(为简化阐述，通道数d相同)，T是计算所谓的查询、键和值嵌入的权重张量，L是投影矩阵。输出的大小与查询序列相同。在给出细节之前修正词汇，多头自注意力(mh-s-attn)是特殊情况$X_q=X_{kv}$，即。

$$\text{mh-s-attn}(X, T, L) = \text{mh-attn}(X, X, T, L)\,. \tag{4}$$

多头注意力简单来说就是M个单注意力头的拼接，然后加上l的投影。常见的做法是[47]使用残差连接、dropout和层归一化。换句话说，表示X⊠$_q$=mh-attn(X_q，X_{kv}，T, L)和X¯(q)的注意力头的拼接，我们有

$$X'_\text{q} = [\text{attn}(X_\text{q}, X_\text{kv}, T_1); ...; \text{attn}(X_\text{q}, X_\text{kv}, T_M)] \tag{5}$$

$$\tilde{X}_\text{q} = \text{layernorm}\big(X_\text{q} + \text{dropout}(LX'_\text{q})\big)\,, \tag{6}$$

其中[;]表示通道轴上的拼接。

单头一个具有权重张量$T^0\in R^{3\times d_0\times d}$，表示为attn($X_q$，$X_{kv}$，$T^0$)的注意力头，依赖于附加的位置编码$P_q\in R^{d\times N_q}$和$P_{kv}\in R^{d\times N_{kv}}$。它在添加query和key位置编码[7]后，首先计算所谓的query, key和value嵌入:

$$[Q; K; V] = [T'_1(X_\text{q} + P_\text{q}); T'_2(X_\text{kv} + P_\text{kv}); T'_3 X_\text{kv}] \tag{7}$$

其中T^0是$T_{1-}{}^0$，$T_{2-}{}^0$，$T_{3-}{}^0$的拼接。然后根据查询和键之间的点积的softmax计算注意力权重α，这样查询序列的每个元素都照顾到键值序列的所有元素(i是查询索引，j是键值索引):

$$\alpha_{i,j} = \frac{e^{\frac{1}{\sqrt{d'}} Q_i^T K_j}}{Z_i} \quad \text{where } Z_i = \sum_{j=1}^{N_\text{kv}} e^{\frac{1}{\sqrt{d'}} Q_i^T K_j}\,. \tag{8}$$

在我们的例子中，位置编码可能是学习或固定的，但对于给定的query/key-value序列，在所有的注意力层之间共享，因此我们没有显式地将它们写为注意力的参数。在描述编码器和解码器时，我们给出了关于它们确切值的更多细节。最终输出是通过注意力权重对valuespn加权的聚合:第i行由attni(Xq, Xkv, t0) = kv j=1 αi,jVj给出。

原始变压器交替使用多头注意力和所谓的FFN层[47]，它们实际上是多层1x1卷积，在我们的例子中有Md个输入和输出通道。我们考虑的FFN由两层1x1卷积和ReLU激活组成。在两层之后还有一个残差连接/dropout/layernorm，类似于方程6。

A.2 损失

为了完整性，我们详细介绍了我们的方法中使用的损失。所有损失都通过批内对象的数量进行归一化。对于分布式训练必须格外小心:由于每个GPU接收一个子批，因此仅按本地批中的对象数量进行归一化是不够的，因为一般情况下子批在GPU之间是不均衡的。相反，重要的是要按所有子批中的对象总数进行归一化。

与[41,36]类似，我们在损失中使用了交联的软版本，并在@ b上使用了'1损失:

$$\mathcal{L}_{\text{box}}(b_{\sigma(i)}, \hat{b}_i) = \lambda_{\text{iou}}\mathcal{L}_{\text{iou}}(b_{\sigma(i)}, \hat{b}_i) + \lambda_{\text{L1}}||b_{\sigma(i)} - \hat{b}_i||_1 \,, \tag{9}$$

其中λiou， λL1∈R是超参数，Liou(·)是广义的IoU [38]:

$$\mathcal{L}_{\text{iou}}(b_{\sigma(i)}, \hat{b}_i) = 1 - \left(\frac{|b_{\sigma(i)} \cap \hat{b}_i|}{|b_{\sigma(i)} \cup \hat{b}_i|} - \frac{|B(b_{\sigma(i)}, \hat{b}_i) \setminus b_{\sigma(i)} \cup \hat{b}_i|}{|B(b_{\sigma(i)}, \hat{b}_i)|} \right). \tag{10}$$

|。|的意思是“面积”，盒子坐标的并和交作为盒子本身的简写。并集或交点的面积由bσ(i)和⊠bi的线性函数的min / max计算，这使得损失对于随机梯度来说表现得足够好。B(bσ(i)， ⊠bi)表示包含bσ(i)， ⊠bi的最大框(涉及B的面积也基于框坐标的线性函数的min / max计算)。

DICE/F-1损耗[28]DICE系数与交联密切相关。如果我们用m⊠表示模型的原始掩码logits预测，m表示二进制目标掩码，则损失定义为:

$$\mathcal{L}_{\text{DICE}}(m, \hat{m}) = 1 - \frac{2m\sigma(\hat{m}) + 1}{\sigma(\hat{m}) + m + 1} \tag{11}$$

其中σ为sigmoid函数。这个损失通过对象的数量进行归一化。

A.3 详细体系结构

图10给出了DETR中使用的变压器的详细描述，在每个注意层都传递了位置编码。来自CNN主干的图像特征通过transformer编码器传递，连同空间位置编码，这些编码被添加到每个多头自注意力层的查询和键中。然后，解码器接收查询(最初设置为0)，输出位置编码(对象查询)和编码器记忆，并通过多个多头自注意力和解码器-编码器注意力产生最终的预测类标签和边界框集合。可以跳过第一个解码器层中的第一个自注意力层。

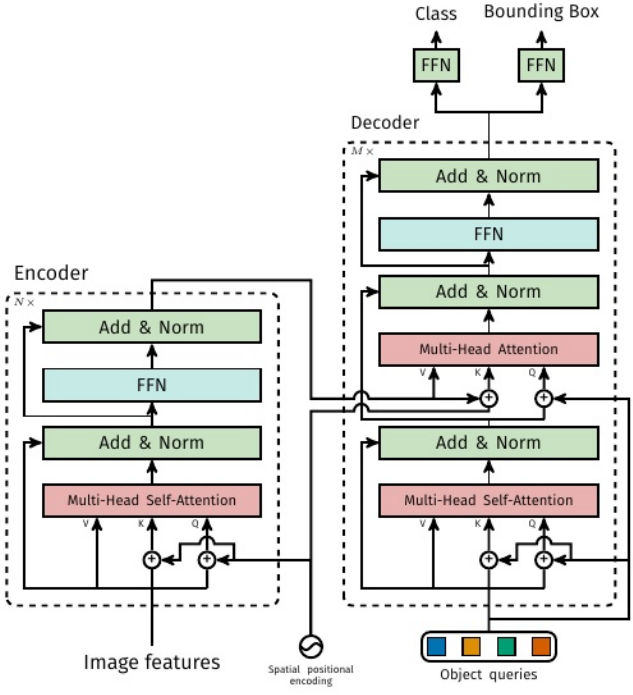


图10:DETR变压器的结构。详情见A.3节。

计算复杂度编码器中每个自关注的复杂度为O(d2HW+d(HW)2): O(d0d)是计算单个查询/键/值嵌入的代价(Md0 = d)， O(d0(HW)2)是计算一个头部的注意权值的代价。其他的计算可以忽略不计。在解码器中，每个自注意为0 (d2N + dn2)，编码器和解码器之间的交叉注意为0 (d2(N + HW) + dNHW)，这远远低于实际中NHW以来的编码器。

考虑到Faster R-CNN的FLOPS取决于图像中提案的数量，我们报告COCO 2017验证集中前100张图像的平均FLOPS数。我们使用Detectron2[50]中的工具触发器计数运算符来计算FLOPS。我们不加修改地将其用于Detectron2模型，并将其扩展到考虑DETR模型的批处理矩阵乘法(bmm)。

A.4 培训hyperparameters

我们使用AdamW[26]训练DETR，改进了权重衰减处理，设置为10^{-4}。我们还应用了梯度裁剪，最大梯度范数为0.1。backbone和transformer的处理略有不同，我们现在讨论一下两者的细节。

Backbone ImageNet预训练的Backbone ResNet-50是从torchvision导入的，舍弃了最后一个分类层。Backbone批量归一化权重和统计在训练过程中被冻结，这是遵循在目标检测中广泛采用的做法。我们使用10^{-5}的学习率对backbone进行微调。我们观察到，让骨干学习率比其他网络大致小一个数量级对稳定训练很重要，特别是在前几个epoch。

Transformer以10−4的学习率训练Transformer。在每次多头注意力之后应用0.1的加性dropout，在层归一化之前应用FFN。权重用Xavier初始化随机初始化。

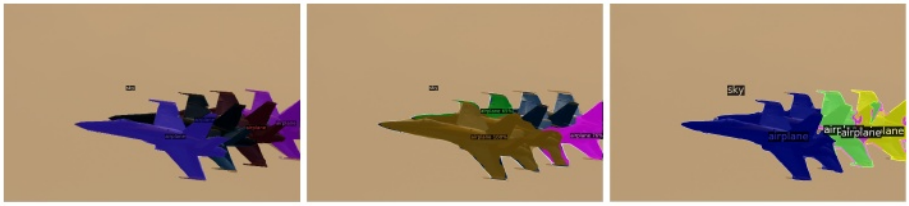
我们使用'$_1$和GIoU损失的线性组合分别以λ_{L1}= 5和λ_{iou}= 2权值进行边界盒回归。所有模型都是用N = 100个解码器查询槽进行训练的。

我们增强的Faster-RCNN+基线使用GIoU[38]损失和标准的$_1$损失进行边界框回归。我们进行了网格搜索以找到损失的最佳权重，最终模型仅使用权重为20和1的GIoU损失分别用于box和proposal回归任务。对于基线，我们采用与DETR中使用的相同的数据增强，并以9×时间表(大约109个epoch)进行训练。所有其他设置都与Detectron2模型zoo[50]中的相同模型相同。

空间位置编码编码器激活与图像特征的相应空间位置相关联。在我们的模型中，我们使用固定的绝对编码来表示这些空间位置。我们采用了原始Transformer[47]编码到2D情况[31]的泛化。具体来说，对于每个嵌入的空间坐标，我们独立使用不同频率的$_{d2}$正弦和余弦函数。然后我们将它们连接起来以获得最终的d通道位置编码。

A.5 额外的结果

DETR-R101模型全光学预测的一些额外定性结果如图11所示。



(a) Failure case with overlapping objects. PanopticFPN misses one plane entirely, while DETR fails to accurately segment 3 of them.

(b)以全分辨率预测事物掩模，这允许比PanopticFPN更清晰的边界

图11:全景预测的比较。从左到右:Ground truth, Panop-ticFPN with ResNet 101, DETR with ResNet 101

通过设计，DETR预测的对象不能超过查询槽数，即在我们的实验中为100个。在本节中，我们将分析接近该极限时DETR的行为。我们选择给定类的标准正方形图像，在10×10网格上重复它，并计算模型遗漏实例的百分比。为了测试少于100个实例的模型，我们随机屏蔽了一些单元格。这确保了无论可见的物体有多少，物体的绝对尺寸都是相同的。为了解释掩模中的随机性，我们使用不同的掩模重复实验100次。结果如图12所示。跨类的行为是相似的，虽然当最多50个可见的实例时，模型检测到所有实例，但随后它开始饱和并错过越来越多的实例。值得注意的是，当图像包含全部100个实例时，模型平均只检测到30个实例，这比图像只包含全部检测到的50个实例要少。模型的反直觉行为很可能是因为图像和检测结果离训练分布很远。

请注意，这个测试是一个泛化分布外的测试设计，因为只有很少的示例图像具有单个类的大量实例。很难从实验中分离出两种类型的域外泛化:图像本身vs每个类的对象数量。但由于很少到没有COCO图像只包含大量同类对象，这种类型的实验代表了我们对了解查询对象是否过度拟合数据集的标签和位置分布所做的最大努力。总的来说，实验表明，该模型在这些分布上没有过拟合，因为它产生了多达50个对象的近乎完美的检测。

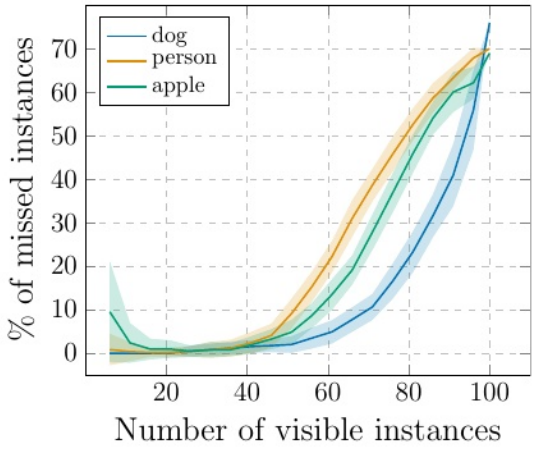


图12:根据图像中存在的数量，DETR错过的各种类的实例数量分析。我们报告了均值和标准差。当实例数量接近100时，DETR开始饱和，并错过越来越多的对象

A.6 PyTorch推理代码

为了演示该方法的简单性，我们在清单1中包含了使用PyTorch和Torchvision库的推理代码。该代码在Python 3.6+， PyTorch 1.4和Torchvision 0.5下运行。请注意，它不支持批处理，因此它只适合使用DistributedDataParallel进行推理或训练，每个GPU一个图像。另外请注意，为了清晰起见，此代码在编码器中使用学习到的位置编码而不是固定的，并且位置编码仅添加到输入中，而不是在每个transformer层。进行这些更改需要超越PyTorch实现的变压器，这会影响可读性。重现实验的整个代码将在会议之前提供。

```
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)
```

清单1:DETR PyTorch推理代码。为清晰起见，它在编码器中使用学习到的位置编码而不是固定的，并且位置编码仅添加到输入中，而不是在每个transformer层。进行这些更改需要超越PyTorch实现的变压器，这会影响可读性。重现实验的整个代码将在会议之前提供。