使用 Transformer 进行端到端对象检测

尼古拉斯·卡里恩、弗朗西斯科·马萨、加布里埃尔·辛纳夫、尼古拉斯·乌苏尼耶、
亚历山大·基里洛夫和谢尔盖·扎戈鲁伊科

脸书人工智能

摘要。我们提出了一种将目标检测视为直接集合预测问题的新方法。我们的方法简化了检测流程，有效地消除了对许多手工设计组件的需求，如非极大值抑制程序或显式编码我们对任务先验知识的锚点生成。这个被称为检测转换器（DETR）的新框架的主要组成部分是：通过二分图匹配强制进行唯一预测的基于集合的全局损失函数，以及transformer编码器-解码器架构。给定一组固定的少量学习型目标查询，DETR分析物体之间的关系和全局图像上下文，直接并行输出最终的预测集合。这个新模型在概念上很简单，不像许多其他现代检测器那样需要专门的库。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基准相当的准确性和运行时性能。此外，DETR可以轻松地以统一的方式生成全景分割。我们展示了它显著优于竞争基准。训练代码和预训练模型可在https://github.com/facebookresearch/detr获取。

1 介绍

目标检测的目标是为每个感兴趣的目标预测一组边界框和类别标签。现代检测器通过在大量候选框[37,5]、锚框[23]或窗口中心[53,46]上定义代理回归和分类问题，以间接方式解决这个集合预测任务。它们的性能显著受到后处理步骤的影响，这些步骤包括合并近似重复的预测、锚框集合的设计以及将目标框分配给锚框的启发式方法[52]。为了简化这些流程，我们提出了一种直接的集合预测方法来绕过代理任务。这种端到端的理念已经在机器翻译或语音识别等复杂的结构化预测任务中取得了重大进展，但在目标检测中尚未实现：先前的尝试[43,16,4,39]要么增加了其他形式的先验知识，要么未能在具有挑战性的基准测试中与强基线模型展现出竞争力。本文旨在弥合这一差距。

?平等贡献

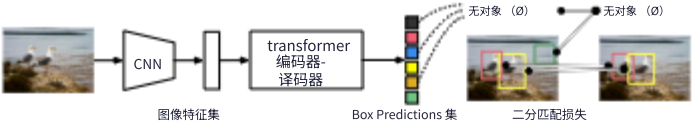

图1: DETR通过将常用的CNN与transformer架构相结合，直接(并行)预测最终的检测结果集。在训练过程中，二分图匹配将预测结果与真实标注框进行唯一配对。未匹配的预测

我们通过将目标检测视为一个直接的集合预测问题来简化训练流程。我们采用基于Transformer的编码器-解码器架构，这是一种常用于序列预测的架构。Transformer的自注意力机制可以显式地模拟序列中所有元素之间的两两交互，这使得这种架构特别适合集合预测的特定约束，比如消除重复预测。

我们的检测 Transformer（DETR，见图 1）可以立即预测所有对象，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二分法匹配。DETR 通过删除多个手动设计的组件来简化检测管道，这些组件对先验知识进行编码，例如空间定位点或非极大值抑制。与大多数现有的检测方法不同，DETR 不需要任何自定义层，因此可以在任何包含标准 CNN 和 transformer 类的框架中轻松复制。

与大多数之前关于直接集合预测的研究相比，DETR的主要特点在于结合了二分图匹配损失函数和具有（非自回归）并行解码能力的transformer架构。相比之下，之前的研究主要集中在使用RNN进行自回归解码。我们的匹配损失函数将预测结果唯一地分配给真实目标，且对预测对象的排列具有不变性，因此我们可以并行输出预测结果。

我们在最受欢迎的目标检测数据集之一COCO[24]上评估了DETR，并与一个非常有竞争力的Faster R-CNN基准[37]进行了对比。Faster R-CNN经历了多次设计迭代，其性能自最初发布以来得到了极大提升。我们的实验表明，新模型达到了可比的性能。更具体地说，DETR在大目标物体上表现出明显更好的性能，这很可能得益于transformer的非局部计算特性。然而，它在小目标物体上的表现较差。我们预期未来的工作将会改善这一方面，就像FPN[22]对Faster R-CNN所做的改进一样。

DETR 的训练设置与标准对象检测器有多种不同。新模式需要超长的训练计划和优势

1 在我们的工作中，我们使用 Transformers [47] 和 ResNet [15] 的标准实现来自标准深度学习库的 backbones。

来自变压器中的辅助解码损耗。我们彻底探讨了哪些组件对于所展示的性能至关重要。

DETR的设计理念可以轻松扩展到更复杂的任务。在我们的实验中，我们表明，在预训练的DETR基础上训练的简单分割头在全景分割[19]（一项最近备受关注的具有挑战性的像素级识别任务）上的表现优于具有竞争力的基准模型。

2 相关工作

我们的工作建立在多个领域的前期研究基础之上：用于集合预测的二分图匹配损失、基于transformer的编码器-解码器架构、并行解码以及目标检测方法。

2.1设置预测

目前没有直接预测集合的标准深度学习模型。基本的集合预测任务是多标签分类（参见计算机视觉领域的[40,33]等参考文献），其中基准方法one-vs-rest并不适用于检测等存在元素间内在结构（即近似相同的框）的问题。这类任务的首要难点是避免近似重复。大多数现有检测器使用非极大值抑制等后处理方法来解决这个问题，但直接集合预测则无需后处理。它们需要建模所有预测元素之间交互的全局推理方案来避免冗余。对于固定大小的集合预测，密集全连接网络[9]虽然可行但成本较高。一种通用方法是使用循环神经网络[48]等自回归序列模型。在所有情况下，损失函数都应该对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]设计损失函数，在真实值和预测值之间找到二分图匹配。这确保了排列不变性，并保证每个目标元素都有唯一的匹配。我们采用二分图匹配损失方法。然而，与大多数先前工作不同，我们摒弃了自回归模型，转而使用具有并行解码能力的transformer，下面我们将对此进行描述。

2.2 Transformer模型与并行解码

Transformer由Vaswani等人[47]提出，作为机器翻译的一种新型注意力机制基础模块。注意力机制[2]是一种从整个输入序列中聚合信息的神经网络层。Transformer引入了自注意力层，类似于非局部神经网络[49]，通过扫描序列中的每个元素并通过聚合整个序列的信息来更新它。基于注意力的模型的主要优势之一是其全局计算能力和完美的记忆力，这使得它们在处理长序列时比RNN更适用。Transformer现在

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

Transformer最初用于自回归模型，继承了早期的序列到序列模型[44]的特点，逐个生成输出标记。然而，由于推理成本过高（与输出长度成正比，且难以批处理），促使了并行序列生成技术的发展，应用于音频[29]、机器翻译[12,10]、词表示学习[8]以及最近的语音识别[6]等领域。我们也将transformer和并行解码相结合，以在计算成本和执行集合预测所需的全局计算能力之间取得适当的平衡。

2.3 目标检测

大多数现代目标检测方法都是相对于某些初始猜测进行预测。两阶段检测器[37,5]相对于候选框进行预测，而单阶段方法则相对于锚框[23]或可能的目标中心网格[53,46]进行预测。最近的研究[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具体设置方式。在我们的模型中，我们能够去除这种人工设计的过程，通过直接预测相对于输入图像的绝对框位置而不是锚框来简化检测过程。

基于集合的损失函数。一些目标检测器[9,25,35]使用二分图匹配损失。然而，在这些早期深度学习模型中，不同预测之间的关系仅通过卷积层或全连接层建模，且使用手工设计的NMS后处理可以提高其性能。更近期的检测器[37,23,53]在真实值和预测值之间使用非唯一分配规则，并结合NMS。

可学习的NMS方法[16,4]和关系网络[17]使用注意力机制显式地建模不同预测之间的关系。使用直接的集合损失，它们不需要任何后处理步骤。然而，这些方法使用额外的手工设计的上下文特征（如候选框坐标）来有效地建模检测之间的关系，而我们寻求的是减少模型中编码的先验知识的解决方案。

循环检测器。与我们方法最接近的是用于目标检测[43]和实例分割[41,30,36,42]的端到端集合预测。与我们类似，它们使用基于CNN激活的编码器-解码器架构，结合二分图匹配损失直接生成边界框集合。然而，这些方法仅在小型数据集上进行评估，且未与现代基准进行比较。特别是，它们基于自回归模型（更准确地说是RNN），因此没有利用具有并行解码能力的最新transformer架构。

3 DETR 模型

检测中进行直接集合预测需要两个关键要素：(1) 一个用于强制预测结果与真实标注进行唯一匹配的集合预测损失函数

框；(2) 一种能够预测一组对象及其关系的架构（仅需一次处理）。我们在图2中详细描述了我们的架构。

3.1目标检测集预测损失

DETR在单次解码器传递中推断出固定大小的N个预测集合，其中N设置为显著大于图像中典型物体数量。训练的主要难点之一是对预测的物体（类别、位置、大小）相对于真实标注进行评分。我们的损失函数在预测对象和真实标注对象之间产生最优的二分匹配，然后优化特定物体的（边界框）损失。

让我们用 y 表示物体的真实集合，用 ˆy = {ˆyi}Ni=1 表示 N 个预测的集合。假设 N 大于图像中物体的数量，我们也将 y 视为一个大小为 N 的集合，并用 ∅（无物体）进行填充。为了在这两个集合之间找到二分图匹配，我们寻找具有最低成本的 N 个元素的排列 σ ∈ SN：

$$\hat{\sigma} = \underset{s \in SN}{\text{最小\ arg}} \sum_{\text{我}}^{N} \text{Lmatch}(yi, \hat{y}\sigma(i)) \text{匹配损失函数} \qquad (1)$$

其中 Lmatch（yi，ˆyσ（i））是索引为 σ（i） 的地面真实 yi anda 预测之间的成对匹配成本。这个最优赋值是按照先前的工作（例如 [43]）使用 Hungarian 算法有效地计算的。

匹配成本同时考虑类别预测和预测框与真实框之间的相似度。真实值集合中的每个元素i可以视为yi = (ci, bi)，其中ci是目标类别标签（可能为∅），bi ∈ [0, 1]4是一个向量，定义了相对于图像尺寸的真实框中心坐标及其高度和宽度。对于索引为σ(i)的预测，我们将类别ci的预测概率定义为ˆpσ(i)(ci)，预测框定义为ˆbσ(i)。基于这些符号，我们将Lmatch(yi, ˆyσ(i))定义为−1{ci6=∅} ˆpσ(i)(ci) + 1{ci6=∅}Lbox(bi, ˆbσ(i))。

这种查找匹配的过程与现代检测器中用于将建议 [37] 或锚点 [22] 匹配到真实对象的启发式分配规则起着相同的作用。主要区别在于我们需要找到一对一的匹配，用于没有重复项的直接集合预测。

第二步是计算损失函数，即上一步中匹配的所有对的匈牙利损失。我们定义损失的方式与常见对象检测器的损失类似，即用于类预测的负对数似然和稍后定义的盒损失的线性组合：

$$\text{LHungarian（y，ˆy）} = \sum_{i=1}^{N} [-\log \hat{p}\hat{\sigma}\text{（i）（ci）} + 1\{ci6=\varnothing\}\text{Lbox（bi，}\hat{b}\hat{\sigma}\text{（i））}], \qquad (2)$$

其中 ˆσ 是在第一步 （1） 中计算的最优赋值。在实践中，当 ci = ∅ 时，我们将对数概率项进行减重 10 倍，以考虑

阶级不平衡。这类似于 Faster R-CNN 训练程序如何通过子采样来平衡正/负建议 [37]。请注意，对象和∅之间的 matchingcost 不取决于预测，这意味着在这种情况下，成本是一个常数。在匹配成本中，我们使用概率 ˆpˆσ（i）（ci）而不是对数概率。这使得类预测术语与 Lbox（・，・） 相当（如下所述），我们观察到更好的实证表现。边界框丢失。匹配成本和 Hungarianloss 的第二部分是对边界框进行评分的 Lbox（・）。与许多将 boxpredictions 作为 Δ 进行一些初始猜测的检测器不同，我们直接进行 box 预测。虽然这种方法简化了实现，但它对损失的相对缩放提出了问题。最常用的 '1 损失对于小箱子和大箱子将具有不同的刻度，即使它们的相对误差相似。为了缓解这个问题，我们使用了 '1 损失和广义 IoU 损失 [38]Liou（・， ・） 的线性组合，它是尺度不变的。总的来说，我们的 box 损失是 Lbox（bi，ˆbσ（i）） 定义为 λiouLiou（bi，ˆbσ（i））+λL1||bi − ˆbσ（i）||1 中，λiou、λL1 ∈ R 是超参数。这两个损失按批处理中的对象数量进行标准化。

3.2 DETR 架构

整个 DETR 架构非常简单，如图 2 所示。它包含三个主要组件，我们将在下面描述：一个用于提取紧凑特征表示的 CNN 主干、一个编码器-解码器转换器，以及一个用于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 可以在任何提供通用 CNN 主干和变压器架构实现的深度学习框架中实现，只需几百行代码。DETR的推理代码可以在 PyTorch 中用不到 50 行的时间内实现 [32]。我们希望我们方法的简单性能够吸引新的研究人员加入检测社区。骨干。从初始图像 ximg ∈ R3×H0×W0（具有 3 个颜色通道2）开始，传统的 CNN 主干生成∈ RC×H×W 的较低分辨率激活映射。我们使用的典型值是 C = 2048 和 H， W = H032 ， W032 。变压器编码器。首先，1x1 卷积将高级激活映射 f 的通道维度从 C 减少到更小的维度 d。创建新的特征映射 z0 ∈ Rd×H×W。编码器需要一个序列作为输入，因此我们将 z0 的空间维度折叠成一个维度，从而得到一个 d×HWfeature 图。每个编码器层都有一个标准架构，由一个多头自注意力模块和一个前馈网络 （FFN） 组成。由于 transformer 架构是排列不变的，我们用固定位置编码 [31\u20123] 来补充它，这些编码被添加到每个注意力层的输入中。我们将架构的详细定义留给补充材料，该定义遵循 [47] 中描述的定义。

2 将输入图像一起批处理，并充分应用 0 填充以确保
它们都具有与批次中最大图像相同的维度 （H0， W0）。

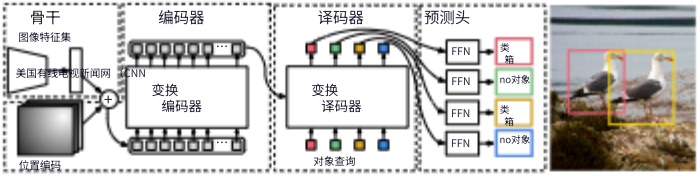

图 2： DETR 使用传统的 CNN 主干来学习输入图像的 2D 表示。该模型将其扁平化，并在将其传递到 transformer 编码器之前用位置编码对其进行补充。然后，transformer 解码器将少量固定数量的学习位置嵌入作为输入，我们称之为对象查询，并额外关注编码器输出。我们将解码器的每个输出嵌入传递给共享的前馈网络 （FFN），该网络预测检测（类和边界框）或 “无对象” 类。

Transformer 解码器。解码器遵循 transformer 的标准架构，使用多头自和编码器-解码器注意力机制转换大小为 d 的 N 个嵌入。与原始 trans-former 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 等人 [47] 使用自回归模型，一次预测一个元素的 outputsequence。我们建议不熟悉这些概念的读者阅读补充材料。由于解码器也是排列不变的，因此 N 个输入嵌入必须不同才能产生不同的结果。这些 in-put 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。解码器将 N 对象查询转换为输出嵌入。然后，它们通过前馈网络（在下一小节中描述）独立解码为框坐标和类标签，从而产生 N 个最终预测。利用对这些嵌入的自和编码器-解码器注意力，该模型使用它们之间的成对关系对所有对象进行全局推理，同时能够将整个图像用作上下文。

预测前馈网络 （FFN）。最终预测由一个具有 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组成。FFN 预测输入图像的框的归一化中心坐标、高度和宽度，而 lin-ear 层使用 softmax 函数预测类标签。由于我们预测了一组固定大小的 N 个边界框，其中 N 通常比图像中实际感兴趣的对象数量大得多，因此使用了一个额外的特殊类 la-bel ∅ 来表示在插槽内没有检测到对象。该类与标准对象检测方法中的 “background” 类起着类似的作用。

辅助解码损失。我们发现在训练过程中使用 auxiliary losses [1] indecoder 很有帮助，特别是可以帮助模型输出正确的数字

每个类的对象。我们在每个解码器层之后添加预测 FFN 和匈牙利损失。所有预测 FFN 共享其参数。我们使用 additionalshared layer-norm 来规范来自不同解码器层的预测 FFN 的输入。

4 实验

我们表明，与 COCO 上的 Faster R-CNNin 定量评估相比，DETR 取得了有竞争力的结果。然后，我们提供了结构和损失的详细消融研究，包括见解和定性结果。最后，为了证明 DETR 是一种多功能且可扩展的模型，我们展示了全景分割的结果，在固定的 DETRmodel 上仅训练了一个小的扩展。我们提供代码和预训练模型来 athttps://github.com/facebookresearch/detr.Dataset 重现我们的实验。我们在 COCO 2017 检测和全景分离数据集 [24,18] 上进行了实验，其中包含 118k 训练图像和 5k 验证图像。每张图像都用边界框和全景分割进行注释。每张图像平均有 7 个实例，在训练集中的单个图像中最多 63 个实例，在同一图像上从小到大不等。如果未指定，我们将 AP 报告为 bbox AP，即多个阈值的整数量度。为了与 Faster R-CNN 进行比较，我们报告了最后一个训练时期的验证 AP，对于消融，我们报告了最近 10 个时期的中位数超过验证结果。技术细节。我们使用 AdamW [26] 训练 DETR，将初始 trans-former 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 transformer 权重都使用 Xavier init [11] 初始化，而 backboneis 使用 ImageNet 预训练的 ResNet 模型 [15] 进行初始化，该模型来自 torchvision 和 frozenbatchnorm 层。我们使用两个不同的主干网络报告结果：ResNet-50 和 ResNet-101。相应的型号分别称为 DETR 和 DETR-R101。在 [21] 之后，我们还通过在主干的最后阶段添加一个膨胀并从该阶段的第一个卷积中删除一个步幅来提高特征分辨率。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两倍，从而提高了小对象的性能，代价是编码器的自注意力成本增加了 16 倍，导致总体计算成本增加了 2 倍。表 1 给出了这些模型与 Faster R-CNN 的 FLOP 的完整比较。

我们使用缩放增强，调整输入图像的大小，使最短的边至少为 480 像素，最多 800 像素，而最长的边最多为 1333 [50]。为了帮助通过编码器的自我注意来学习全局关系，我们还在训练过程中应用了随机裁剪增强，将性能提高了大约 1 AP。具体来说，将火车图像以 0.5 的概率裁剪为随机矩形块，然后再次调整大小为 800-1333。转换器使用默认 dropout 0.1 进行训练。在推理时

表 1：与具有 ResNet-50 和 ResNet-101 主干的 Faster R-CNN 的比较 COCO 验证集。上半部分显示了 Detectron2 [50] 中更快 R-CNN 模型的结果，中间部分显示了使用 GIoU [38] 的快速 R-CNN 模型的结果，随机裁剪训练时间增强和漫长的 9 倍训练计划。DETR 模型获得的结果与高度调整的 Faster R-CNN 基线相当，具有较低的 APS 但大大提高了 APL。我们使用 torchscript Faster R-CNN 和 DETR 模型来测量 FLOPS 和 FPS。名称中没有 R101 的结果与 ResNet-50 对应。

型	GFLOPS/FPS	#params	AP	AP50	AP75	APS	APM	APL
更快的 RCNN-DC5	320/16	166 分钟	39.0	60.5	42.3	21.4	43.5	52.5
更快的 RCNN-FPN	180/26	42 分钟	40.2	61.0	43.8	24.2	43.5	52.0
更快的 RCNN-R101-FPN	246/20	60 分钟	42.0	62.5	45.9	25.2	45.6	54.6
更快的 RCNN-DC5+	320/16	166 分钟	41.1	61.4	44.3	22.9	45.9	55.0
更快的 RCNN-FPN+	180/26	42 分钟	42.0	62.1	45.5	26.6	45.4	53.4
更快的 RCNN-R101-FPN+	246/20	60 分钟	44.0	63.9	47.8	27.2	48.1	56.0
DETR 公司	86/28	41 分钟	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5型	187/12	41 分钟	43.3	63.1	45.9	22.5	47.3	61.1
产品型号 DETR-R101	152/20	60 分钟	43.5	63.8	46.4	21.9	48.0	61.8
产品型号：DETR-DC5-R101	253/10	60 分钟	44.9	64.7	47.7	23.7	49.5	62.3

时间，一些 slot 预测空类。为了优化 AP，我们使用相应的 confidence，用第二高分的类覆盖这些槽的预测。与过滤掉空槽相比，这将 AP 提高了 2 个百分点。其他训练超参数可在 A.4 节中找到。对于我们的消融实验，我们使用 300 个 epoch 的训练计划，学习率在 200 个 epoch 后下降 10 倍，其中单个 epoch 是所有训练图像的一次传递。在 16 个 V100 GPU 上训练 300 个时期的基线模型，每个 GPU 有 4 张图像（因此总批量大小为 64）。对于用于与更快的 R-CNN 进行比较的较长计划，我们训练 500 个 epoch，400 个 epoch 后学习率下降。与较短的计划相比，此计划增加了 1.5 个 AP。

4.1 与 Faster R-CNN 的比较

Transformer 通常由 Adam 或 Adagrad 优化器进行训练，训练计划很长，并且会退出，DETR 也是如此。然而，FasterR-CNN 是使用 SGD 训练的，数据增强最小，我们不知道 Adam 或 dropout 的成功应用。尽管存在这些差异，我们仍尝试使 Faster R-CNN 基线更强大。为了使其与 DETR 保持一致，我们将广义 IoU [38] 添加到盒子损失中，相同的随机作物增强和已知可以改善结果的长期训练 [13]。结果如表 1 所示。在顶部，我们显示了使用 3 倍计划训练的模型的 Faster R-CNN 结果来自 Detectron2 Model Zoo [50]。在中间部分，我们显示了相同模型的结果（带有“+”），但经过训练

表 2：编码器尺寸的影响。每行对应于一个具有不同数量的编码器层和固定数量的解码器层的模型。随着编码器层的增加，性能逐渐提高。

#layers	GFLOPS/FPS 浮点运算	#params	美联社	AP50 系列	APS	APM	APL
0	76/28	33.4 分钟	36.7	57.4	16.8	39.6	54.2
3	81/25	37.4 分钟	40.1	60.6	18.5	43.8	58.6
6	86/23	41.3 分钟	40.6	61.6	19.9	44.3	60.2
12	95/20	49.2 米	41.6	62.1	19.8	44.9	61.9

使用 9x 计划（109 个纪元）和所描述的增强功能，总共增加 1-2 个 AP。在表 1 的最后一部分，我们显示了 multipleDETR 模型的结果。为了在参数数量上具有可比性，我们选择具有 6 个 transformer 和 6 个宽度为 256 的解码器层和 8 个 attentionheads 的模型。与带有 FPN 的 Faster R-CNN 一样，该模型具有 41.3M 参数，其中 23.5M 在 ResNet-50 中，17.8M 在变压器中。尽管 Faster R-CNN 和 DETR 仍有可能随着更长时间的训练而进一步提高，但我们可以得出结论，DETR 可以与具有相同参数数量的 Faster R-CNN 竞争，在 COCO val 子集上实现 42 AP。DETR 实现这一目标的方法是提高 APL （+7.8），但请注意，该模型在 APS （-5.5） 方面仍然落后。具有相同参数数量和相似 FLOP 计数的 DETR-DC5 具有更高的 AP，但在 APS 中也明显落后。更快的 R-CNN 和 DETR 与 ResNet-101 主干网也显示出可比的结果。

4.2 消融术

transformer 解码器中的注意力机制是模拟不同检测的特征表示之间关系的关键组件。在消融分析中，我们探讨了架构和损耗的其他组件如何影响最终性能。在这项研究中，我们选择了基于 ResNet-50 的 DETRmodel，具有 6 个编码器、6 个解码器层和宽度 256。该模型具有 41.3M 参数，在短期和长期计划中分别达到 40.6 和 42.0 AP，并以 28 FPS 的速度运行，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的数量来评估全局图像级自我注意的重要性（表 2）。在没有编码器层的情况下，整体 AP 下降了 3.9 个百分点，其中更显著的下降幅度为

6.0 AP 在大型对象上。我们假设，通过使用全局场景推理，编码器对于解开对象很重要。在图 3 中，我们可视化了经过训练的模型最后一个编码器层的注意力图，重点关注图像中的几个点。编码器似乎已经分离了实例，这可能会简化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（参见第 3.2 节），因此，预测 FFN 被设计为预先

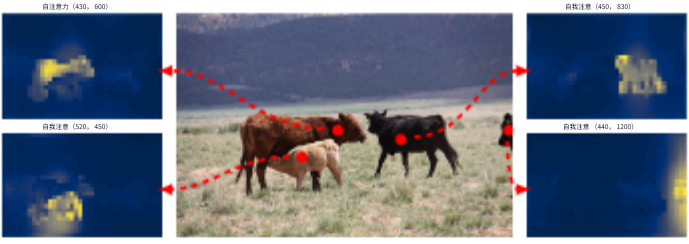

图 3：一组参考点的编码器自注意力。编码器能够分离单个实例。使用基线 DETR 模型对价值集图像进行预测。

dict 对象。我们通过评估在解码的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 AP50 在每一层后都有所提升，在第一层和最后一层之间总共有非常显著的 +8.2/9.5 AP 提升。由于其基于集合的损失，DETR 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标准NMS 程序，其中包含默认参数[50]。NMS 提高了来自第一个解码器的预测性能。这可以通过以下事实来解释：transformer 的单个解码层无法计算输出元素之间的任何互相关，因此它很容易对同一对象进行多次预测。在第二层和后续层中，激活上的自我注意机制允许模型抑制重复预测。我们观察到 NMS 带来的改进会随着深度的增加而减弱。在最后几层，我们观察到 AP 中的一个小损失，因为 NMS 错误地删除了真阳性预测。

与可视化编码器注意力类似，我们在 Fig.6、为每个预测对象着色不同颜色的注意力图。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，如头部或腿部。我们假设，在 encoder 通过全局注意力分离实例后，解码器只需要关注四肢以提取类和对象边界。FFN 的重要性。转换器内部的 FFN 可以看作是 1×1 个对话层，使编码器类似于注意力增强卷积网络 [3]。我们试图完全删除它，只在 transformer 层中留下注意力。通过将网络参数的数量从 41.3M减少到 28.7M，变压器中只留下 10.8M，性能下降了 2.3 AP，我们得出结论，FFN 对于获得良好的结果很重要。位置编码的重要性。在我们的模型中，有两种位置 en-codings：空间位置编码和输出位置 encod-

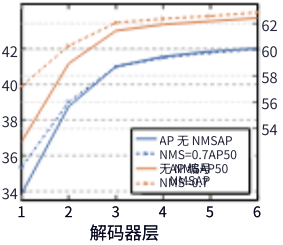

图 4：每个解码器层之后的 AP 和 AP50 性能。评估单个长计划基线模型。DETR 不需要 NMS by de-sign，此图对此进行了验证。NMS 在最后一层降低了 AP，去除了 TP 预测，但提高了第一解码器层的 AP，重新移动了双重预测，因为第一层没有通信，并略微提高了 AP50。

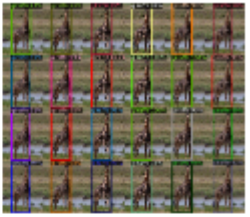

图 5： 稀有类的 Out of distribution gen-eralization。即使训练集中没有图像超过 13 只长颈鹿，DETR 也没有困难推广到同一类的 24 个或更多实例。

ings （对象查询） 。我们试验了固定编码和学习编码的各种组合，结果可以在表 3 中找到。输出位置编码是必需的，并且不能删除，因此我们尝试在解码器输入时传递一次它们，或者在每个解码器注意力层添加到查询中。在第一个实验中，我们完全删除了输入处的空间位置编码和传递输出位置编码，有趣的是，该模型仍然实现了超过 32 个 AP，比基线损失了 7.8 个 AP。然后，我们在输入处传递固定的正弦空间位置编码和输出编码一次，就像在originaltransformer [47] 中一样，发现与直接在 attention 中传递位置编码相比，这导致了 1.4 AP 的下降。传递给 attention 的学习空间编码会得到类似的结果。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 AP 下降 1.3 AP。当我们将编码传递给 attention 时，它们会在所有层之间共享，并且输出编码（对象查询）总是被学习的。

鉴于这些削弱，我们得出结论，transformer 组件：编码器中的全局自我注意、FFN、多解码器层和位置编码，都对最终的目标检测性能有显着贡献。

损失消融。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来打开和关闭它们。损失有三个组成部分：分类损失、'1 bounding boxdistance 损失和 GIoU [38] 损失。分类损失对于训练是必不可少的，不能关闭，因此我们训练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，训练了所有三个损失。结果见表 4。GIoU 自身账户亏损

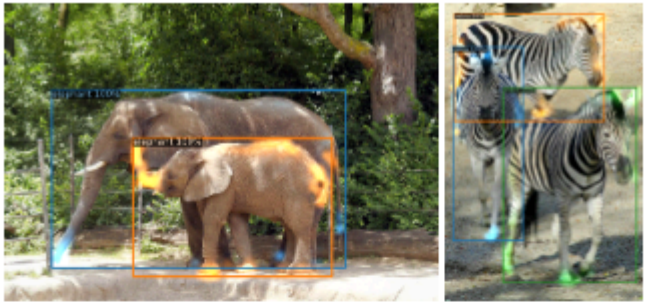

图 6： 可视化每个预测对象的解码器注意力 （来自 COCOval 集的图像）。使用 DETR-DC5 模型进行预测。注意力分数针对不同的对象使用不同的颜色编码。解码器通常关注物体的四肢，例如腿和头。最好用彩色观看。

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有固定的正弦正位编码。学习的嵌入在所有层之间共享。不使用空间位置编码会导致 AP 显着下降。有趣的是，传递它们 indecoder 只会导致 AP 的小幅下降。所有这些模型都使用学习到的输出 positionalencodings。

空间位置 enc.输出位置 enc.编码器解码器		美联社	Δ	AP50 系列	Δ
壬烯酮	在输入时学习	32.8	-7.8	55.2	-6.5
Movie 和 Inputsine 和 Input	在输入时学习	39.2	-1.4	60.0	-1.6
在 attn 学习。	在 ATTN 学习。	39.6	-1.0	60.7	-0.9
没有 正弦在 attn.	在 ATTN 学习。	39.3	-1.3	60.3	-1.4
sine at attn.sine at attn.	在 ATTN 学习。	40.6	-	61.6	-

表 4：损耗分量对 AP 的影响。我们训练两个模型关闭 '1 损失和 GIoU 损失，并观察到 '1 本身给出的结果很差，但当与 GIoU 结合使用时，APM 和 APL 会得到改善。我们的基线（最后一行）结合了两种损失。

类	'1	GIoU	美联社	Δ	AP50 系列	Δ	APS	APM	APL
二十			35.8	-4.8	57.3	-4.4	13.7	39.8	57.9
二十			39.9	-0.7	61.6	0	19.9	43.2	57.9
二十		X	40.6	-	61.6	-	19.9	44.3	60.2

对于大多数模型性能，与合并损失相比，基线仅损失 0.7 AP。使用不带 GIoU 的 '1 显示结果很差。我们只学习

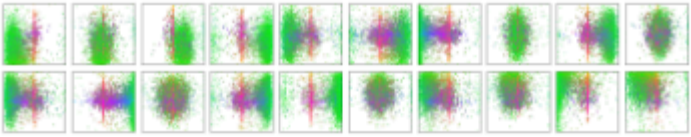

图 7：在 DETR 解码器中，总共 N = 100 个预测槽中的 20 个来自 COCO 2017 val set的所有图像上的所有框预测的可视化。每个框预测都表示为一个点，其中心坐标在 1 x 1 正方形中，按每个图像大小进行归一化。这些点采用颜色编码，因此绿色对应于小框，红色对应于大的水平框，蓝色对应于大的垂直框。我们观察到，每个插槽都通过多种作模式学习专注于某些区域和盒子大小。我们注意到，几乎所有的插槽都有一种预测大图像宽框的模式，这在 COCO 数据集中很常见。

不同损失的简单消融（每次都使用相同的权重），但其他组合它们的方法可能会得到不同的结果。

4.3 分析

解码器输出槽分析 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的框。DETR 学习每个查询槽的 differentspecialization。我们观察到每个插槽都有几种作模式，分别针对不同的区域和盒子大小。特别是，所有插槽都具有预测图像宽框的模式（可见为在图中间对齐的红点）。我们假设这与 COCO 中对象的分布有关。泛化到看不见的实例数。COCO 中的某些类不能很好地表示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）。我们的模型能够在图像上找到所有 24 只长颈鹿，这显然是分布式的。此实验证实，每个对象查询中没有 strongclass-specialization。

4.4 用于全景分割的 DETR

全景分割 [19] 最近引起了计算机视觉界的广泛关注。与 Faster R-CNN [37] toMask R-CNN [14] 的扩展类似，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，我们证明了这样的头可以通过处理 stuff 和 thing 类来产生全景分割 [19]

3 基本图片来源：https://www.piqsels.com/en/public-domain-photo-jzlwu

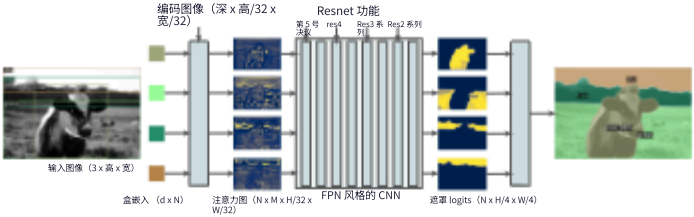

图 8：全视头图示。为每个检测到的对象并行生成一个二进制掩码，然后使用像素级 argmax 合并掩码。

图 9：DETR-R101 生成的全景分割定性结果。DETR 以统一的方式为事物生成对齐的掩码预测。

以统一的方式。我们对 COCO 数据集的全景注释进行了实验，该数据集除了 80 个事物类别外，还有 53 个内容类别。

我们训练 DETR 使用相同的配方来预测 onCOCO 上的 stuff 和 things 类周围的盒子。预测框是训练可能的必要条件，因为匈牙利匹配是使用框之间的距离计算的。我们还添加了一个掩码头，它为每个预测的框预测一个二进制掩码，参见图 8。它以每个对象的 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，从而以小分辨率生成 M 个注意力热图对象。为了进行最终预测并增加解决方案，使用了类似 FPN 的架构。我们将在补充中更详细地描述架构。掩模的最终分辨率为 stride 4，并且每个掩模都使用 DICE/F-1 损失 [28] 和Focal loss [23] 独立监督。

掩码头可以联合训练，也可以分两步训练，我们只训练盒子的 DETR，然后冻结所有权重，只训练掩码头 25 个时期。从实验上讲，这两种方法给出了相似的结果，我们使用后一种方法报告结果，因为它导致更短的总挂钟训练时间。

表 5：在 COCO val 数据集上与最先进的方法 UPSNet [51] 和 PanopticFPN [18] 的比较我们使用与 DETR 相同的数据增强重新训练了 PanopticFPN，以 18 倍的时间表进行公平比较。UPSNet 使用 1xschedule，UPSNet-M 是具有多尺度测试时增强的版本。

型	主干 PQ	平方 RQ PQth SQth RQth PQst SQst RQst								美联社
全景FPN++	R50 系列	42.4 79.3 51.6			49.2	82.4	58.8	32.3 74.8	40.6	37.7
UPSnet 公司	R50 系列	42.5 78.0 52.5			48.6	79.4	59.6	33.4 75.9	41.7	34.3
UPSnet-M 系列	R50 系列	43.0 79.1 52.8			48.9	79.7	59.7	34.1 78.2	42.3	34.3
全景FPN++	R101 系	44.1 79.5 53.3 51.0 83.2 60.6						33.6 74.0	42.1	39.7
DETR 公司	R50 系列	43.4 79.3 53.8			48.2	79.8	59.5	36.3 78.5	45.3	31.1
DETR-DC5型	R50 系列	44.6 79.8 55.0			49.4	80.5	60.6	37.3 78.7	46.5	31.9
产品型号 DETR-R101	R101 系列	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，我们只需在每个像素的掩码分数上使用 argmax，并将相应的类别分配给生成的掩码。此过程保证最终的掩码没有重叠，因此，DETR 不需要通常用于对齐不同掩码的启发式 [19]。

培训详情。我们按照边界框检测的配方训练 DETR、DETR-DC5 和 DETR-R101 模型，以预测 COCO 数据集中 stuff 和 things 类周围的框。新的掩码头训练了 25 个 epoch（有关详细信息，请参阅补充）。在推理过程中，我们首先以低于 85% 的置信度过滤掉检测，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，我们将同一内容类别的不同掩码预测折叠为一个，并过滤空的预测（小于 4 像素）。

主要结果。定性结果如图 9 所示。在表 5 中，我们将我们的统一全视分离方法与几种不同的既定方法进行了比较，这些方法以不同的方式处理事物和事物。我们报告全景质量 （PQ） 和事物细分 （PQth） 和 stuff （PQst）。我们还报告了 maskAP （在 things 类上计算），在任何 panoptic 后处理之前（在我们的例子中，在采用像素级 argmax 之前）。我们表明，DETR 在 COCO-val 2017 上发表了结果，以及我们强大的 PanopticFPNbaseline（使用与 DETR 相同的数据增强进行训练，以便公平比较）。结果分解表明 DETR 在 stuff 类上特别占主导地位，我们假设编码器注意力允许的全局推理是这个结果的关键因素。对于事物类，尽管与掩码 AP 计算的基线相比，存在高达 8 mAP 的严重赤字，但 DETRus 获得了具有竞争力的 PQth。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 PQ。我们希望我们的方法能激发在未来工作中探索完全统一的全景分割模型。

5 结论

我们提出了 DETR，这是一种基于变形器和二分匹配损失的对象检测系统的新设计，用于直接集合预测。该方法在 chal-lenging COCO 数据集上实现了与优化的 Faster R-CNN 基线相当的结果。DETR 易于实施，并且具有灵活的架构，可轻松扩展到全景分割，并具有有竞争力的结果。此外，它在大型对象上的性能明显优于 Faster R-CNN，这可能要归功于对自我注意形成的全局信息的处理。

这种新的探测器设计也带来了新的挑战，特别是在小物体的训练、优化和性能方面。Currentdetectors 需要几年的改进才能应对类似问题，我们预计未来的工作将成功解决 DETR 的问题。

6 确认

我们感谢 Sainbayar Sukhbaatar、Piotr Bojanowski、Natalia Neverova、DavidLopez-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，没有他们，这项工作就不可能完成。

引用

1. Al-Rfou，R.，Choe，D.，Constant，N.，Guo，M.，Jones，L.：具有更深自我注意的字符级语言建模。收录于：AAAI 人工智能会议（2019）

2. Bahdanau，D.，Cho，K.，Bengio，Y.：通过联合学习对齐和翻译进行神经机器翻译。收录于：ICLR（2015）

3. Bello，I.，Zoph，B.，Vaswani，A.，Shlens，J.，Le，Q.V.：注意力增强对话网络。收录于：ICCV（2019）

4. Bodla，N.，Singh，B.，Chellappa，R.，Davis，LS：软 NMS 改进对象使用一行代码进行检测。收录于：ICCV（2017）

5. Cai，Z.，Vasconcelos，N.：级联 R-CNN：高质量对象检测和位置分割。PAMI（2019）

6. Chan，W.，Saharia，C.，Hinton，G.，Norouzi，M.，Jaitly，N.：Imputer：通过插补和动态规划进行序列建模。arXiv：2002.08926（2020）

7. Cordonnier，JB，Loukas，A.，Jaggi，M.：关于自我注意和卷积层之间的关系。收录于：ICLR（2020）

8. Devlin，J.，Chang，MW，Lee，K.，Toutanova，K.：BERT：用于语言理解的深度双向转换器的预训练。收录于：NAACL-HLT（2019）

9. Erhan，D.，Szegedy，C.，Toshev，A.，Anguelov，D.：使用深度神经网络进行可扩展对象检测。收录于：CVPR（2014）

10. Ghazvininejad，M.，Levy，O.，Liu，Y.，Zettlemoyer，L.：掩码预测：条件掩码语言模型的并行解码。arXiv：1904.09324（2019）

11. Glorot，X.，Bengio，Y.：了解训练深度前馈神经网络的难度。在：AISTATS（2010）

12. Gu，J.，Bradbury，J.，Xiong，C.，Li，V.O.，Socher，R.：非自回归神经机器翻译。收录于：ICLR（2018）
13. He，K.，Girshick，R.，Doll'ar，P.：重新思考 imagenet 预训练。收录于：ICCV（2019）
14. He，K.，Gkioxari，G.，Doll'ar，P.，Girshick，RB：面具 R-CNN。收录于：ICCV（2017）
15. He，K.，Zhang，X.，任，S.，Sun，J.：用于图像识别的深度残差学习.收录于：CVPR（2016）

16. Hosang，J.H.，Benenson，R.，Schiele，B.：学习非极大值抑制。收录于：CVPR（2017）

17. 胡，H.，顾，J.，张，Z.，戴，J.，魏，Y.：用于对象检测的关系网络.收录于：CVPR（2018）

18. Kirillov，A.，Girshick，R.，He，K.，Doll'ar，P.：全景特征金字塔网络。收录于：CVPR（2019）

19. Kirillov，A.，He，K.，Girshick，R.，Rother，C.，Dollar，P.：全景分割。收录于：CVPR（2019）

20. Kuhn，H.W.：赋值问题的匈牙利方法（1955）
21. Li，Y.，Qi，H.，Dai，J.，Ji，X.，Wei，Y.：完全卷积实例感知语义分割。收录于：CVPR（2017）

22. Lin，TY，Doll'ar，P.，Girshick，R.，He，K.，Hariharan，B.，Belongie，S.：用于对象检测的特征金字塔网络。收录于：CVPR（2017）
23. Lin，T.Y.，Goyal，P.，Girshick，RB，He，K.，Doll'ar，P.：密集物体检测的焦点损失。收录于：ICCV（2017）
24. Lin，TY，Maire，M.，Belongie，S.，Hays，J.，Perona，P.，Ramanan，D.，Doll'ar，P.，Zitnick，CL：Microsoft COCO：上下文中的常见对象。在：ECCV（2014）
25. Liu，W.，Anguelov，D.，Erhan，D.，Szegedy，C.，Reed，S.E.，Fu，C.Y.，Berg，A.C.：Ssd：单发多箱探测器。在：ECCV（2016）
26. Loshchilov，I.，Hutter，F.：解耦权重衰减正则化。收录于：ICLR（2017）
27. L¨uscher，C.，Beck，E.，Irie，K.，Kitza，M.，Michel，W.，Zeyer，A.，Schl¨uter，R.，Ney，H.：用于图书馆语音的 Rwth asr 系统：混合与注意力 - 无数据增强.arXiv：1905.03072（2019）

28. Milletari，F.，Navab，N.，Ahmadi，S.A.：V-net：用于体积医学图像分割的全卷积神经网络。在：3DV（2016）
29. Oord，A.v.d.，Li，Y.，Babuschkin，I.，Simonyan，K.，Vinyals，O.，Kavukcuoglu，K.，Driessche，GVD，Lockhart，E.，Cobo，L.C.，Stimberg，F.，et al.：并行波网：快速高保真语音合成。arXiv：1711.10433（2017 年）
30. Park，E.，Berg，A.C.：学习分解以进行对象检测和实例分割。arXiv：1511.06449（2015 年）

31. Parmar，N.，Vaswani，A.，Uszkoreit，J.，Kaiser，L.，Shazeer，N.，Ku，A.，Tran，D.：图像转换器。收录于：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.:P ytorch：命令式、高性能深度学习库。在：神经IPS（2019）

33. Pineda，L.，Salvador，A.，Drozdzal，M.，Romero，A.：阐明图像到设定的预测：模型、损失和数据集的分析。arXiv：1904.05709（2019 年）
34. Radford，A.，Wu，J.，Child，R.，Luan，D.，Amodei，D.，Sutskever，I.：语言模型是无监督的多任务学习者（2019 年）
35. Redmon，J.，Divvala，S.，Girshick，R.，Farhadi，A.：您只需看一次：统一的实时对象检测。收录于：CVPR（2016）
36. 任，M.，Zemel，R.S.：具有经常性关注的端到端实例分割。收录于：CVPR（2017）

37. 任，S.，He，K.，Girshick，R.B.，Sun，J.：更快的 R-CNN：使用区域建议网络实现实时对象检测。PAMI（2015）

38. Rezatofighi，H.，Tsoi，N.，Gwak，J.，Sadeghian，A.，Reid，I.，Savarese，S.：联合上的广义交集。收录于：CVPR（2019）

39. Rezatofighi，S.H.，Kaskman，R.，Motlagh，F.T.，Shi，Q.，Cremers，D.，Leal-Taix'e，L.，Reid，I.：深度烫集网：学习使用深度神经网络预测具有未知排列和基数的集合。arXiv：1805.00613（2018）

40. Rezatofighi，SH，Milan，A.，Abbasnejad，E.，Dick，A.，Reid，I.，Kaskman，R.，Cremers，D.，Leal-Taix，I.：Deepsetnet：使用深度神经网络预测集合。收录于：ICCV（2017）

41. Romera-Paredes，B.，Torr，PHS：循环实例分割。收录于：ECCV（2015）

42. Salvador，A.，Bellver，M.，Baradad，M.，Marqu'es，F.，Torres，J.，Gir'o，X.：用于语义实例分割的递归神经网络。arXiv：1712.00617（2017 年）

43. Stewart，RJ，Andriluka，M.，Ng，AY：拥挤场景中的端到端人员检测。收录于：CVPR（2015）

44. Sutskever，I.，Vinyals，O.，Le，Q.V.：使用神经网络进行序列到序列学习。在：NeurIPS（2014）

45. Synnaeve，G.，Xu，Q.，Kahn，J.，Grave，E.，Likhomanenko，T.，Pratap，V.，Sriram，A.，Liptchinsky，V.，Collobert，R.：端到端 ASR：从现代架构的监督学习到半监督学习。arXiv：1911.08460（2019）

46. Tian，Z.，Shen，C.，Chen，H.，He，T.：FCOS：全卷积单阶段对象检测。收录于：ICCV（2019）

47. Vaswani，A.，Shazeer，N.，Parmar，N.，Uszkoreit，J.，Jones，L.，Gomez，A.N.，Kaiser，L.，Polosukhin，I.：注意力就是你所需要的。在：NeurIPS（2017）

48. Vinyals，O.，Bengio，S.，Kudlur，M.：顺序很重要：集合的序列到序列。收录于：ICLR（2016）

49. Wang，X.，Girshick，RB，Gupta，A.，He，K.：非局部神经网络。在：CVPR（2018）

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

51. 熊妍，廖，R.，赵 H.，胡，R.，白，M.，玉默，E.，乌尔塔桑，R.：Upsnet：统一全景分割网络。收录于：CVPR（2019）

52. Zhang，S.，Chi，C.，Yao，Y.，Lei，Z.，Li，S.Z.：通过自适应训练样本选择弥合基于锚点和无锚点检测之间的差距。arXiv：1912.02424（2019）

53. 周，X.，王，D.，Kr¨ahenb¨uhl，P.：作为点的对象。arXiv：1904.07850（2019）

一个 附录

答 1 Preleginaries： 多头注意力层

由于我们的模型基于 Transformer 架构，因此我们在这里提醒我们用于穷举的注意力机制的一般形式。attention机制遵循[47]，除了后面的位置编码的细节（见等式8）[7]。多头 具有 M 个维度 d 的多头注意力的一般形式是具有以下签名的函数（使用 d′ =dM ，并在下括号中给出矩阵/张量大小）

$$\text{mh-attn}: \underset{\{\}_{d\times Nq}}{xq}, \underset{\{\}_{d\times Nkv}}{Xkv}, \underbrace{T}_{\{\}_{M\times 3\times d'\times d}}, \underbrace{L}_{\{\}_{d\times d}} \to \underbrace{\tilde{}Xq}_{\{\}_{d\times Nq}} \tag{3}$$

其中 Xq 是长度为 Nq 的查询序列，Xkv 是长度为 Nkv 的键值序列（为简单阐述，通道数 d 相同），Tis 是计算所谓查询、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的大小相同。为了在给出细节之前修复词汇，多头自我注意 （mh-s-attn） 是特殊情况 Xq = Xkv，即

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

多头注意力只是 M 个单个注意力头的串联，然后是 L 的投影。常见的做法 [47] 是使用 residualconnections、dropout 和 layer normalization。换句话说，表示˜Xq =mh-attn（Xq，Xkv，T，L） 和¯X（q） 注意力头的串联，我们有

$$X'q = [\text{attn}（Xq，Xkv，T1）; ...; \text{attn}（Xq，Xkv，TM）] \tag{5}$$

$$\tilde{}Xq = \text{layernorm}（Xq + \text{dropout}（LX'q））， \tag{6}$$

其中 [;] 表示通道轴上的串联。单头 具有权重张量 T ′ ∈ R3×d′×d 的注意力头，用 attn（Xq，Xkv，T ′） 表示，取决于额外的位置编码 Pq ∈ Rd×Nq 和 Pkv ∈ Rd×Nkv。它首先计算所谓的查询、键和值嵌入在添加查询和键位置编码 [7] 之后：

$$[问;K;V] = [T '1（Xq + Pq）;T'2（Xkv + Pkv）;T'3Xkv] \tag{7}$$

其中 T ′ 是 T '1、T '2、T '3 的串联。然后，根据查询和键之间的点积的 softmax 计算注意力权重 α，以便查询序列的每个元素都关注 key-valuesequence 的所有元素（i 是查询索引，j 是键值索引）：

$$\alpha i，j = \frac{e^{\frac{1}{\sqrt{d'}} QTi\,Kj}}{子} \quad 其中\ Zi = \sum_{j=1}^{NKV} 和 \frac{1}{\sqrt{d'}} QTi\,Kj . \tag{8}$$

在我们的例子中，位置编码可能是学习的或固定的，但在给定的查询/键值序列的所有注意力层之间共享，因此我们不会明确地将它们写为注意力的参数。在描述 encoder 和 decoder 时，我们提供了有关它们的确切值的更多详细信息。最终输出是由注意力权重加权的值的聚合： 第 i 行由 attni（Xq， Xkv， T ′） = Σ Nkvj=1 αi，jVj.前馈网络 （FFN） 层给出 原始转换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，在我们的例子中具有 M d 输入和输出通道。我们考虑的 FFN 由两层 1x1 卷积和 ReLUactivations 组成。在两层之后还有一个残差 connection/dropout/layernorm，类似于公式 6。

答 2　损失

为了完整起见，我们详细介绍了我们方法中使用的损失。所有损失都按批处理中的对象数量进行规范化。分布式训练必须格外小心：由于每个 GPU 都接收一个子批次，因此仅按本地批次中的对象数量进行标准化是不够的，因为通常子批次在 GPU 之间不平衡。相反，重要的是按所有子批处理中的对象总数进行归一化。Box loss 与 [41,36] 类似，我们在 loss 中使用 Unions 的软版本 Intersection，并在 ˆb 上使用 '1 loss：

$$\mathrm{Lbox}\left(b\sigma(i),\ \hat{}bi\right) = \lambda\mathrm{iou}\mathrm{Liou}\left(b\sigma(i),\ \hat{}bi\right) + \lambda L1||b\sigma(i) - \hat{}bi||1, \tag{9}$$

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

$$\mathrm{Liou}\left(b\sigma(i),\ \hat{}bi\right) = 1 - \left(\frac{|b\sigma(i) \cap \hat{}bi||}{|B\left(b\sigma(i),\ \hat{}bi\right) \setminus b\sigma(i) \cup \hat{}bi||B\left(b\sigma(i),\ \hat{}bi\right)|}\right). \tag{10}$$

|.|表示 “面积”，框坐标的并集和交集用作框本身的简写。并集或交集的面积由 bσ（i） 和 ˆbi 的线性函数的最小值/最大值计算，这使得损失对于随机梯度来说足够好。B（bσ（i），ˆbi） 表示包含 bσ（i），ˆbi 的最大框（涉及 B 的面积也是根据框坐标的线性函数的最小值/最大值计算的）。DICE/F-1 损失 [28]DICE 系数与交集并集密切相关。如果我们用 ˆm 表示模型的原始掩码 logits 预测，用 m 表示二进制目标掩码，则损失定义为：

$$\mathrm{LDICE}\left(m,\ \hat{}m\right) = 1 - \frac{2m\sigma\left(\hat{}m\right) + 1}{\sigma\left(\hat{}m\right) + m + 1} \tag{11}$$

其中 σ 是 sigmoid 函数。此损失按对象数量标准化。

答 3 详细架构

图 10 给出了 DETR 中使用的变压器的详细描述，并在每个注意力层传递位置 en-codings。来自 CNN 主干网的图像特征通过 transformer 编码器传递，以及添加到每个多头自注意力层的查询和键的空间位置编码。然后，解码器接收查询（最初设置为零）、输出位置编码（对象查询）和编码器内存，并通过多个多头自注意力和解码器-编码器注意力生成最终的预测类标签和边界框。可以跳过第一个解码器层中的第一个自注意力层。

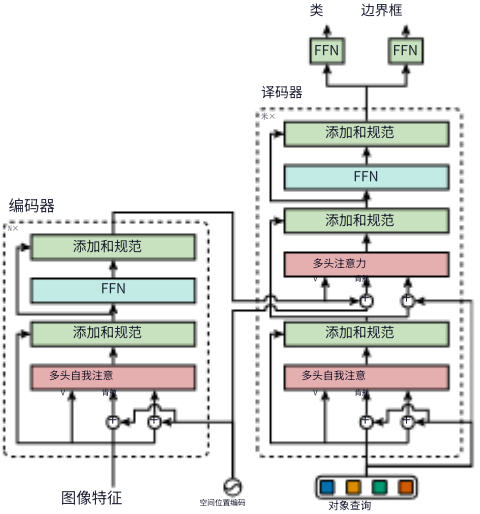

图 10： DETR 变压器的架构。有关详细信息，请参阅第 A.3 节。

计算复杂度 编码器中的每个自注意力都有复杂度 O（d2HW +d（HW ）2）：O（d′d） 是计算单个查询/键/值嵌入（和 M d′ = d）的成本，而 O（d′（HW ）2） 是计算一个头的目标权重的成本。其他计算可以忽略不计。在解码器中，每个自注意力都在 O（d2N +dN 2） 中，编码器和解码器之间的交叉注意力在 O（d2（N + HW ） + dN HW ） 中，在实践中比编码器的 N HW 低得多。

FLOPS 计算 鉴于 Faster R-CNN 的 FLOPS 取决于图像中的提议数量，我们报告 COCO 2017 验证集中前 100 张图像的平均 FLOPS 数量。我们使用 Detectron2 [50] 的工具 flop count 运算符计算 FLOPS。我们无需修改 Detectron2 模型即可使用它，并将其扩展为将批量矩阵乘法 （bmm） 考虑在 DETR 模型中。

A.4训练超参数我们使用 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，最大梯度模数为 0.1。主干和变压器的处理方式略有不同，我们现在讨论两者的细节。Backbone ImageNet 预训练的主干 ResNet-50 是从 Torchvi-sion 导入的，丢弃了最后一个分类层。主干批量归一化权重和统计数据在训练期间被冻结，遵循对象检测中广泛采用的做法。我们使用 10-5 的学习率对主干进行微调。我们观察到，让骨干学习率比网络的其余部分小大约一个数量级对于稳定训练很重要，尤其是在最初的几个 epoch 中。Transformer 我们以 10-4 的学习率训练 Transformer。在图层归一化之前，在每次多头关注和 FFN 之后应用 0.1 的 Additivedropout。权重使用 Xavier 初始化随机初始化。损失我们使用 '1 和 GIoU 损失的线性组合进行边界框回归，分别具有 λL1 = 5 和 λiou = 2 权重。所有模型都使用 N = 100 个解码器查询槽进行训练。基线：我们增强的 Faster-RCNN+ 基线使用 GIoU [38] 损失以及标准的 '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 losss 进行框和提案回归任务。对于基线，我们采用与 DETR 相同的数据增强，并使用 9× 的时间表（大约 109 个时期）对其进行训练。所有其他设置与 Detectron2 模型库 [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 absoluteencoding 来表示这些空间位置。我们采用原始 Transformer [47] 编码的泛化到 2D 情况 [31]。具体来说，对于每个嵌入的两个空间坐标，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，得到最终的通道位置编码。

A.5其他结果DETR-R101模型的全景预测的一些额外定性结果如图11所示。

（a） 对象重叠的故障情况。PanopticFPN 完全错过了一架飞机，而 DETR 未能准确分割其中的 3 架飞机。

（b） 事物掩码以全分辨率预测，这允许比 PanopticFPN 更清晰的边界

图 11： 全景预测的比较。从左到右：真值、带 ResNet 101 的 Panop-ticFPN、带 ResNet 101 的 DETR

增加实例数量 根据设计，DETR 无法预测超过其查询槽的对象，即在我们的实验中为 100 个。在本节中，我们将分析 DETR 在接近此限制时的行为。我们选择给定类的规范方形图像，在 10 × 10 网格上重复它，并计算模型丢失的实例的百分比。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。这可确保无论可见多少个对象，对象的绝对大小都是相同的。为了解释掩码的随机性，我们使用不同的掩码重复实验 100 次。结果如图 12 所示。各个类的行为是相似的，虽然模型在最多 50 个可见实例时检测到所有实例，但随后它开始饱和并错过越来越多的实例。值得注意的是，当图像包含所有 100 个实例时，模型平均只检测到 30 个，这比图像仅包含 50 个实例而所有实例都检测到的情况要少。模型的反直觉行为可能是因为图像和检测与训练分布相去甚远。

请注意，此测试是设计外泛化测试，因为具有单个类的大量实例的示例图像非常少。从实验中很难分辨出两种类型的域外泛化：图像本身与每个类的对象数量。但是，由于很少或没有 COCO 图像只包含大量同类对象，这种类型的实验代表了我们尽最大努力了解查询对象是否过度拟合数据集的标签和位置分布。总体而言，实验表明该模型不会在这些分布上过度拟合，因为它可以产生多达 50 个对象的近乎完美的检测。

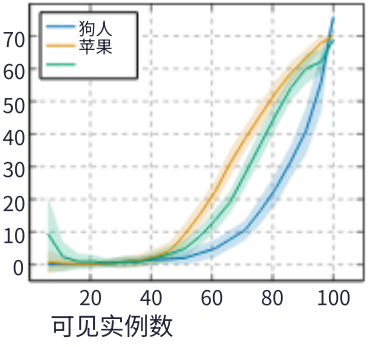

图 12： DETR 删除所遗漏的各种类别的实例数分析，取决于图像中存在的数量。我们报告平均值和标准差。当实例数接近 100 时，DETR 开始饱和并错过越来越多的对象

答 6　PyTorch 推理代码

为了演示该方法的简单性，我们在清单 1 中包含了带有 PyTorch 和 Torchvision 库的推理代码。代码使用 Python 3.6+、PyTorch 1.4 和 Torchvision 0.5 运行。请注意，它不支持批处理，因此它仅适用于使用 DistributedDataParallel 进行推理或训练，每个 GPU 一张图像。另请注意，为清楚起见，此代码在 encoder 中使用 learnt positionalencodings 而不是 fixed，并且位置编码仅添加到 input 中，而不是在每个 transformer 层。进行这些更改需要超越 PyTorch 的 transformer 实现，这会影响可读性。重现实验的完整代码将在会议之前提供。

```
Import Torch （导入火炬）
从 Torch import nn
从 torchvision.models 导入 resnet50

类 DETR（nn.模块）：

    def __init__（self， num_classes， hidden_dim， nheads，
                 num_encoder_layers、num_decoder_layers）：
        super（）.__init__（）
        # 我们只从 ResNet-50 模型中获取卷积层
        self.backbone = nn.顺序（*list（resnet50（pretrained=True）.children（）））[：-2]）
        self.conv = nos.Conv2d（2048， hidden_dim， 1）
        self.transformer = nn 的变压器（hidden_dim， nheads，
                                          num_encoder_layers、num_decoder_layers）
        self.linear_class = 否。线性 （hidden_dim， num_classes + 1）
        self.linear_bbox = 否。线性 （hidden_dim， 4）
        self.query_pos = nn.参数（torch.rand（100， hidden_dim））
        self.row_embed = nn.参数（torch.rand（50， hidden_dim // 2））
        self.col_embed = nn.参数（torch.rand（50， hidden_dim // 2））

    def forward（self， inputs）：
        x = self.backbone（输入）
        h = self.conv（x）
        H， W = h.形状[-2：]
        位置 = 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.解压（1））
        返回 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（）
输入 = torch.randn（1， 3， 800， 1200）
对数，bboxes = detr（inputs）
```

清单 1：DETR PyTorch 推理代码。为清楚起见，它在编码器中使用了学习到的位置编码而不是固定的，并且位置编码仅添加到 input，而不是在每个 transformer 层。进行这些更改需要超越 transformer 的 PyTorch 实现，这会影响可读性。重现实验的完整代码将在会议之前提供。