使用Transformer进行端到端的目标检测

尼古拉斯·卡里昂, 弗朗西斯科·马萨, 加布里埃尔·希内夫, 尼古拉斯·乌尚尔, 亚历山大·基里洛夫, 谢尔盖·扎戈鲁科
Facebook AI

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，有效消除了许多手工设计的组件的需要，比如非最大值抑制过程或显式编码任务对先验知识的锚点生成。新框架DEtection TRansformer或DETR的主要组成部分是通过双边匹配强制进行唯一预测的基于集合的全局损失和Transformer编码器-解码器架构。给定一组固定的学习目标查询，DETR通过推理对象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。这种新模型在概念上很简单，并且不需要专门的库，不像许多其他现代检测器。DETR在具有挑战性的COCO目标检测数据集上表现出与成熟和高度优化的Faster R-CNN基线相当的准确性和运行时性能。此外，DETR可以轻松推广到以统一的方式生成全景分割。我们展示了它明显优于竞争基准的性能。训练代码和预训练模型可在https://github.com/facebookresearch/detr获得。

1 引言

目标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的建议区域[37,5]、锚点[23]或窗口中心点[53,46]上定义替代回归和分类问题来间接解决这个集合预测任务。它们的性能在很大程度上受后处理步骤的影响，用于折叠近似重复的预测，以及锚点集合的设计，并且存在一个预测类别为“无对象”(?)的类。

我们提出了一种直接集合预测方法，绕过了替代任务。这种端到端的哲学在复杂结构预测任务（如机器翻译或语音识别）中取得了显著的进展，但在目标检测方面尚未取得进展：之前的尝试[43,16,4,39]要么添加了其他形式的先验知识，要么在具有挑战性的基准测试中与强基准方法竞争能力不足。本文旨在弥合这一差距。

图1：DETR通过将常见的CNN与变换器架构相结合，直接预测（并行地）最终的检测集。在训练过程中，双向匹配将预测与真实框唯一地匹配。没有匹配的预测应产生“无物体”（？）类别的预测。

我们将目标检测视为一种直接集合预测问题，简化了训练流程。我们采用基于变换器的编码器-解码器架构，变换器是一种用于序列预测的流行架构。变换器的自注意机制明确地对序列中的所有成对交互进行建模，使得这些架构特别适用于集合预测的特定约束，例如去除重复预测。

我们的DEtection TRansformer（DETR，见图1）一次性预测所有对象，并使用一个集合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向匹配。DETR简化了检测流程，删除了多个手动设计的组件，如空间锚点或非最大抑制。与大多数现有的检测方法不同，DETR不需要任何自定义层，因此可以在包含标准CNN和变换器类的任何框架中轻松复现。

i=1

i=1

h

(非自回归) 并行解码[29,12,10,8]。相反，先前的研究主要集中在基于RNN的自回归解码[43,41,30,36,42]。我们的匹配损失函数为每个预测分配了一个真实对象，并且对于预测对象的排列是不变的，因此我们可以并行地输出它们。我们在最流行的目标检测数据集COCO [24]上对DETR进行评估，并与非常有竞争力的Faster R-CNN基线[37]进行比较。Faster R-CNN经历了许多设计迭代，其性能在原始出版物之后得到了极大的改进。我们的实验表明，我们的新模型达到了可比的性能。更准确地说，DETR在大型物体上显示出显著更好的性能，这可能得益于变换器的非局部计算。然而，它在小物体上的表现较差。我们期望未来的工作能像FPN [22]对Faster R-CNN的发展一样改善这一方面。

DETR的训练设置与标准的目标检测器在多个方面不同。新模型需要额外长的训练计划，并在变换器中受益于辅助解码损失。我们深入探索了哪些组件对所示性能至关重要。

DETR的设计理念很容易扩展到更复杂的任务。在我们的实验中，我们表明在训练在预训练的DETR之上的简单分割头在全景分割任务[19]上优于竞争基线，这是一个具有挑战性的像素级识别任务，最近变得流行起来。

2 相关工作

我们的工作在几个领域借鉴了以前的研究：集合预测的二分匹配损失，基于变换器的编码器-解码器架构，并行解码和目标检测方法。

2.1 集合预测

[
\mathop{\mathbb{E}}{\sigma\in K{[1:n]}}\left[f(\text{x}{i,j}, \text{y}{\sigma(i)})\right]
]

在计算机视觉的背景下，针对存在元素之间基础结构（即接近相同的框）的检测等问题，传统方法（例如“一对多”）不适用。在这些任务中，首要困难是避免近似副本。大多数当前的检测器使用非极大值抑制等后处理方法来解决这个问题，但直接集合预测是无需后处理的。它们需要全局推理机制来建模所有预测元素之间的相互作用，以避免冗余。对于常量大小的集合预测，稠密全连接网络[9]就足够了，但代价很高。一种常见方法是使用自回归序列模型，例如循环神经网络[48]。在所有情况下，损失函数应该对预测的排列具有不变性。通常的解决方案是设计一个基于匈牙利算法[20]的损失函数，找到实际情况和预测之间的二分图匹配。这样可以强制实现排列不变性，并确保每个目标元素都有唯一匹配。我们采用了二分图匹配的损失函数方法。然而，与大多数先前的工作不同，我们摒弃了自回归模型，而是使用了具有并行解码的transformer模型，下面将进行描述。

2.2 Transformers和并行解码

Transformer模型是由Vaswani等人[47]提出的基于注意力机制的机器翻译的新建模块。注意力机制[2]是一种神经网络层，可以聚合整个输入序列的信息。Transformers引入了自注意力层，类似于非局部神经网络[49]，它们通过扫描序列的每个元素并从整个序列中汇总信息来更新每个元素。注意力模型的一个主要优点是其全局计算和完美的记忆能力，这使得它们比RNN在长序列上更适用。Transformer模型现在已经在自然语言处理和计算机视觉上得到广泛应用[8,27,45,34,31]。

2.3 目标检测

X0

2.1 并行序列生成

传统的序列生成模型，如循环神经网络（RNN）[44]，逐个生成输出标记。然而，这种方法计算成本高昂（与输出长度成正比，并且难以进行批处理），因此在音频处理[29]、机器翻译[12,10]、词表示学习[8]和最近的语音识别[6]等领域发展了并行序列生成方法。为了在计算成本和执行集合预测所需的全局计算之间取得恰当的平衡，我们还将Transformer和并行解码方法相结合。

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化的猜测进行预测的。两阶段检测器[37,5]根据建议（proposal）预测框，而单阶段方法根据锚点（anchor）[23]或可能的对象中心的网格[53,46]进行预测。最近的研究[52]表明，这些系统的最终性能严重依赖于初始化猜测的准确方式。在我们的模型中，我们能够通过直接预测与输入图像相关的绝对框，而不是锚点，来消除手工设计的初始化过程，简化检测过程。

集合损失。一些目标检测器[9,25,35]使用双向匹配损失（bipartite matching loss）。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通过卷积或全连接层建模，并且需要手动设计的NMS后处理来提高性能。更近期的检测器[37,23,53]使用与真实标签和预测之间的非唯一匹配规则，以及与NMS结合。可学习的NMS方法[16,4]和关系网络[17]通过注意力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上下文特征（例如建议框坐标）来高效地建模检测之间的关系，而我们寻找的解决方案旨在减少这种依赖。

对于目标检测[43]和实例分割[41,30,36,42]，利用基于CNN激活的编码器-解码器架构使用双向匹配损失来直接生成一组边界框。然而，这些方法只在小数据集上进行了评估，并未与现代基准进行对比。特别是，它们基于自回归模型（更确切地说是RNN），因此没有利用最近的并行解码的transformers。

3 DETR模型

直接进行检测集合预测需要两个要素：（1）一个集合预测损失，强制预测和实际边界框之间进行唯一匹配；（2）一种在单次传递中预测一组对象并建模它们关系的架构。如图2所示，我们详细描述我们的架构。

3.1 目标检测集合预测损失
DETR通过解码器在单次传递中推断一个固定大小的N个预测集合，其中N设置为显著大于图像中 typical 的对象数量。训练的主要困难之一是根据实际情况得分预测对象（类别、位置、大小）。我们的损失通过预测和实际对象之间的最优二分匹配来生成，并优化对象特定的（边界框）损失。

设y为实际对象集合，^y=f^yi^N
i=1为N个预测的集合。假设N大于图像中的对象数量，我们将y也视为使用?(无对象)填充的大小为N的集合。为了在这两个集合之间找到一个二分匹配，我们搜索具有最低成本的数字N个元素□2SN的排列：
^ arg min
□2SNNX
iLmatch (yi;^y□(i)); (1)
其中Lmatch (yi;^y□(i))是实际值yi和索引□(i)的预测之间的成对匹配成本。这个最优分配可以有效地使用匈牙利算法计算，这是根据先前的工作（例如[43]）实现的。

最后的损失函数包括两部分：
1fci6=?g^p□(i)(ci) + 1fci6=?gLbox(bi;^b□(i)).
i=1h
; (2)
32.
坐标转换和高度、宽度相对于图像大小的列车。对于索引为i的预测，我们定义类别c的概率为^p(i)(ci)，预测的框为^b(i)。根据这些符号，我们定义Lmatch(yi,^y(i))为1 if ci!=ci，在ci!=ci的情况下，认为^p(i)(ci)=1, ci!=ci值为1，属于类别ci的概率值，否则为0；在ci!=ci的情况下，认为^b(i) = bi, ci!=ci值为1，^b(i)不为bi的概率值，否则为0。这个寻找匹配的过程起到了与现代检测器中用于将propsal [37]或anchors [22]与ground truth对象匹配的启发式分配规则相同的作用。主要的区别在于我们需要找到直接预测集的一对一匹配，而没有重复项。

第二步是计算损失函数，对于在前一步中匹配的所有成对的匈牙利损失。我们定义损失和常见物体检测器的损失类似，即类别预测的负对数似然和定义的框损失的线性组合：
LHungarian (y,^y) =NX
i=1h
log ^p(i)(ci) +1 if ci!=ciLbox(bi,^b(i))i
; (2)
其中在第一步中计算出的^是最优分配。实际上，我们通过10倍权重减小ci!=ci的log概率项，以解决类别不平衡的问题。这类似于Faster R-CNN训练过程通过子采样平衡正负propsal [37]。注意，在一个物体和一个?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。在匹配成本中，我们使用概率^p(i)(ci)而不是对数概率。这使得类别预测项和Lbox(□;□)（下面描述）可以进行可比的处理，并且我们观察到了更好的实验性能

边界框损失。匹配成本和匈牙利损失的第二部分是Lbox(□)，评分边界框。与许多通过□的一些初始猜测进行盒子预测的检测器不同，我们直接进行盒子预测。虽然这种方法简化了实现过程，但它在损失的相对缩放方面存在问题。最常用的`1loss在32方面将具有不同的尺度;W0V
Liou（E秸，E市）是一个尺度不变的。总的来说，我们的框损失被定义为Lbox(bi; ^b(i))，其中
ōiouLiou(bi; ^b(i)) +ōL1||bi-^b(i)||1，这里ōiou和ōL1是超参数，通常需要调节。
这两个损失都根据批处理中的对象数量进行了归一化处理。
细节架构（DETR）
整体上，DETR的架构非常简单，如图2所示。它包含三个主要组件，我们逐一介绍如下：CNN骨干网络用于提取紧凑的特征表示，编码器-解码器Transformer和一个简单的前馈网络（FFN）用于进行最终的检测预测。
与许多现代检测器不同，DETR可以在任何提供常见CNN骨干网络和Transformer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 [32]中实现不到50行。我们希望我们方法的简单性能够吸引新的研究者加入检测领域。
骨干网络。从初始图像ximg2R3 × H0 × W0（具有3个颜色通道）开始，传统的CNN骨干网络生成一个较低分辨率的激活图f2R^C × H × W。我们通常使用的典型值为C = 2048和H，W = H0÷32，W32。
Transformer编码器。首先，1x1卷积将高级激活图f的通道维度从C减少到较小的维度d，创建一个新的

特征图z0^2R^d × H × W。编码器期望输入一个序列，因此我们将z0的空间维度折叠成一个维度，得到一个d × HW的特征图。每个编码器层都有一个标准架构，包括多头自注意模块和前馈网络（FFN）。由于Transformer架构是排列不变的，我们使用固定的位置编码[31,3]来补充输入到每个注意力层。关于架构的详细定义，我们推迟到补充材料中，其遵循[47]中描述的定义。
2输入图像进行批处理，适当进行0填充以确保：
X0
图2: DETR使用传统的CNN骨干网络学习输入图像的2D表示。模型将其展平，并在传递到transformer编码器之前补充了位置编码。然后，transformer解码器以输入一小固定数量已学习的位置嵌入（我们称之为object queries）并进一步与编码器输出相关联。我们将解码器的每个输出嵌入传递给共享的前馈神经网络（FFN），该网络预测检测（类别和边界框）或"无目标"类别。
Transformer解码器。解码器遵循transformer的标准架构，使用多头自注意和编码器-解码器注意机制来将大小为d的N个嵌入进行转化。与原始的transformer的区别在于，我们的模型在每个解码器层上并行解码N个对象，而Vaswani et al.[47]使用自回归模型逐个预测输出序列的每个元素。对于不熟悉这些概念的读者，我们请参考补充材料。由于解码器也具有置换不变性，为了产生不同的结果，N个输入嵌入必须是不同的。这些输入嵌入是已学习的位置编码，我们称之为object queries，并且与编码器类似，将它们添加到每个注意层的输入中。解码器将N个object queries转化为输出嵌入。然后，通过前馈神经网络（在下一小节中描述）将它们独立地解码为框坐标和类别标签，得到最终的预测。使用这些嵌入的自注意和编码器-解码器注意机制，模型在全局上推理所有对象之间的成对关系，同时能够使用整个图像作为上下文。
approaches.
epochs.
X0
坐标、高度和宽度是相对于输入图像的框，线性层使用softmax函数预测类别标签。由于我们预测了一个固定大小的N个边界框，其中N通常远大于图像中感兴趣的实际对象数量，因此使用了一个额外的特殊类标签"?"，用于表示在一个插槽内未检测到任何对象。这个类在标准目标检测方法中扮演着与"背景"类类似的角色。

辅助解码损失。我们发现在训练过程中在解码器中使用辅助损失[1]对于帮助模型输出每个类别的正确对象数量非常有帮助。我们在每个解码器层后添加预测FFNs和杭特损失。所有预测FFNs共享参数。我们使用额外的共享层范数来对来自不同解码器层的预测FFN的输入进行归一化处理。

4 实验

我们在COCO数据集上进行了定量评估，结果表明DETR与Faster R-CNN具有竞争力的结果。然后，我们通过详细的消融研究体系结构和损失，给出了有关洞察和质量结果的分析。最后，为了展示DETR是一个多才多艺且可扩展的模型，我们在固定的DETR模型上仅进行了一小部分扩展，展示了全景分割的结果。我们提供了代码和预训练模型，以重现我们的实验，网址为https://github.com/facebookresearch/detr。

数据集。我们在COCO 2017检测和全景分割数据集[24,18]上进行实验证明，该数据集包括118k张训练图像和5k张验证图像。每个图像都有边界框和全景分割的注释。平均每个图像有7个实例，在训练集中单个图像上最多有63个实例，这些实例在同一图像上大小不一。如果没有指定，我们将AP报告为bbox AP，作为多个阈值的综合度量。与Faster R-CNN进行比较，我们报告最后训练时期的验证AP。

对应于ResNet-50的AP50.
所有的transformer权重都使用Xavier初始化[11]，而backbone则使用来自torchvision的预训练的ImageNet的ResNet模型[15]，其中batchnorm层被冻结。我们报告两种不同backbone的结果：一个是ResNet-50，另一个是ResNet-101。相应的模型分别被称为DETR和DETR-R101。根据[21]的方法，我们在backbone的最后一个阶段添加了扩张(dilation)，并去掉了该阶段的第一个卷积层的stride。相应的模型分别被称为DETR-DC5和DETR-DC5-R101（扩张的C5阶段）。这种修改使分辨率提高了两倍，从而改善了小物体的性能，但编码器的self-attention成本增加了16倍，导致总体计算成本增加了2倍。这些模型和Faster R-CNN的FLOPs的完整比较如表1所示。
我们使用尺度增强(scale augmentation)，将输入图像调整为最短边至少480像素，最长边至多1333像素[50]。为了通过编码器的self-attention来帮助学习全局关系，我们还在训练过程中应用了随机裁剪增

强，这样可以提高约1个AP的性能。具体而言，训练图像有50%的概率被裁剪成一个随机的矩形区域，然后再次调整大小为800-1333。transformer使用默认的dropout为0.1进行训练。在推断阶段，我们使用具有ResNet-50和ResNet-101 backbone的Faster R-CNN模型进行比较。表1显示了在COCO验证集上与Faster R-CNN模型的比较结果。顶部部分显示了使用Detectron2[50]中的Faster R-CNN模型的结果，中间部分显示了使用GIoU[38]、训练时的随机裁剪以及长时间的9倍训练计划的Faster R-CNN模型的结果。DETR模型达到了与经过大量调整的Faster R-CNN基准模型相当的结果，具有更低的AP S但大大改善了AP L。我们使用torchscript Faster R-CNN和AP No NMS。
对应于ResNet-50的模型。
模型GFLOPS / FPS #params AP AP 50AP75APSAPMAP 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
在一些空槽预测空类时。为了优化AP，我们使用第二高得分的类覆盖这些插槽的预测，使用对应的置信度。与过滤掉空槽相比，这将AP提高了2个点。
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个epoch的训练计划，学习率在200个epoch后下降10倍，其中一个epoch是对所有训练图像的一次遍历。在16个V100 GPU上训练300个epoch的基线模型需要3天，每个GPU上4个图像（因此批量大小总共为64）。与用于与Faster R-CNN进行比较的较长计划相比，我们训练500个epoch，并在400个epoch后降低学习率。这个计划比较短的计划增加了1.5 AP。
4.1 与Faster R-CNN的比较
Transformer通常使用Adam或Adagrad优化器进行训练，具有非常长的训练计划和dropout，DETR也是如此。然而，Faster R-CNN使用SGD进行训练，最小数据增强，并且我们不知道Adam或dropout的成功应用。尽管存在这些差异，我们试图使Faster R-CNN基线更加强大。为了与DETR对齐，我们在框损失中添加了广义IoU [38]，相同的随机裁剪增强和已知改善结果的长时间训练 [13]。结果X0
从3x训练计划的Detectron2模型树中选择模型进行评估，本节中我们展示相同模型（但训练了10个Carion等人）的结果。表2展示了编码器大小的影响。每一行对应具有不同编码层数和固定解码层数的模型。随着编码层数的增加，性能逐渐提高。

\begin{table}[h]
\centering
\caption{编码器大小的影响}
\begin{tabular}{ccccccc}
\hline
#层数 & GFLOPS/FPS & #参数 & AP & AP 50 & APS & APM & APL\
\hline
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\
\hline
\end{tabular}
\end{table}

在使用9x训练计划（109个epoch）和描述的增强方法后，与之前相比总体上增加了1-2个AP。在表1的最后部分，我们展示了多个DETR模型的结果。为了在参数数量上进行比较，我们选择了一个具有6个transformer和6个解码器层，并且宽度为256的模型，具有8个attention头。和Faster R-CNN with FPN一样，该模型具有41.3M个参数，其中23.5M个参数在ResNet-50中，17.8M个参数在transformer中。尽管Faster R-CNN和DETR都有进一步改进的潜力，但我们可以得出结论，DETR在与Faster R-CNN具有相同参数数量的情况下具有竞争力，在COCO验证集上实现了42个AP。DETR之所以能够实现这一点，是通过提高AP L（+7.8），但请注意，该模型在AP S上仍然落后（-5.5）。在具有相同参数数量和类似FLOP计数的情况下，DETR-DC5具有更高的AP，但在AP Stoo方面仍然明显落后。Faster R-CNN和使用ResNet-101骨干网络的DETR也展现出相当的结果。

4.2 剔除测试中，变形器解码器中的注意机制是模型认定不同检测特征表达之间关系的关键组件。在我们的剔除分析中，我们探索了我们架构和损失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，具有6个编码器层，6个解码器层和宽度256进行研究。该模型有41.3M个自注意力($430,600$)。
并以与具有相同骨干网络的Faster R-CNN-FPN相似的速度运行（28 FPS）。
编码器层数。我们通过更改编码器层数（表2）来评估全局图像级自注意力的重要性。没有编码器层，整体AP下降了3.9个点，大型对象下降了更多的6.0 AP。我们假设编码器通过使用全局场景推理来解开对象。在图3中，我们将训练模型的最后一个编码器层的注意力图可视化，并关注图像中的一些点。编码器似乎已经将实例分开，这可能简化了解码器对对象的提取和定位。
解码器层数。我们在每个解码器层之后应用辅助损失（见第3.2节），因此，预测的FFN通过设计进行训练，以通过每个解码器层的输出来预测对象。我们通过评估每个解码器层在解码过程中将预测的对象来分析每个解码器层的重要性（图4）。每个层之后，AP和AP 50都有所改善，在第一个和最后一个层之间总共改善了非常显著的+8.2/9.5 AP。由于DETR具有基于集合的损失，因此从设计上不需要NMS。为了验证这一点，我们对每个解码器的输出运行了一个具有默认参数[50]的标准NMS过程。NMS改善了第一个解码器的预测性能。这可以解释为transformer的一个解码器层无法在输出元素之间计算任何交叉关联，因此容易对正预测进行多个预测。AP50
我们发现，随着网络层数的增加，NMS带来的改进效果逐渐减小。在最后几层中，我们观察到由于NMS错误地移除了真正的正样本预测，造成AP的小幅损失。

类似于可视化编码器的注意力，我们在图6中可视化了解码器的注意力，为每个预测的物体着色。我们观察到解码器的注意力区域较为局部化，主要关注物体的极值部位，如头部或腿部。我们假设，在编码器通过全局注意力将实例分开之后，解码器只需要关注极值部位来提取类别和目标边界的信息。

FFN的重要性。在transformer中，FFN可以被视为1×1的卷积层，使得编码器类似于注意力增强的卷积网络。我们尝试完全删除FFN，只保留transformer层中的注意力。通过将网络参数数量从41.3M降低到28.7M，其中transformer层只有10.8M参数，性能下降了2.3 AP。因此，我们得出结论，FFN对于达到良好结果至关重要。

位置编码的重要性。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。在第4节和第5节中的图表显示，位置编码对于模型的性能具有重要影响。

图4展示了每个解码器层后的AP和AP50性能。我们对单个长时间表模型进行了评估。DETR设计上不需要NMS，图中验证了这一点。NMS在最后几层中降低了AP，移除了TP的预测，但在第一批解码器层中提高了AP，去除了重复的预测，因为第一层中没有通信，并且稍微改善了AP50的性能。

图5展示了对于罕见类别的域外泛化能力。即使训练集中没有任何一张图像包含超过13个长颈鹿，DETR在24个或更多相同实例上也没有困难进行泛化。
类别。
我们尝试了固定编码和学习编码的各种组合，结果见表3。输出位置编码是必需的，不能被删除，因此我们尝试将其仅在解码器输入处传递一次，或者在每个解码器注意力层中添加到查询中。在第一次实验中，我们完全删除了空间位置编码，并将输出位置编码传递给输入，有趣的是，该模型仍然达到了超过32个AP的成绩，较基准模型少了7.8个AP。然后，我们将固定的正弦空间位置编码和输出编码在输入中传递一次，与原始的transformer [47]相同，发现与直接在注意力中传递位置编码相比，AP下降了1.4个。传递给注意力的学习的空间编码产生了类似的结果。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轻微的AP下降1.3个。当我们将编码传递给注意力时，它们在所有层之间共享，并

且输出编码（对象查询）总是被学习的。
根据这些剔除的实验，我们得出结论：transformer中的组件：编码器中的全局自注意力、FFN、多个解码器层和位置编码，都对最终的目标检测性能有显著贡献。
损失剖析。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，并将它们打开和关闭。损失有三个组成部分：分类损失、1边界框距离损失和GIoU [38]损失。分类损失对于训练至关重要，不能关闭，因此我们训练了一个没有边界框距离损失的模型，以及一个没有GIoU损失的模型，并与使用所有三种损失训练的基准模型进行了比较。结果详见表4。单独使用GIoU损失会造成较小的AP下降，而边界框距离损失对性能的贡献较小。综合这些实验结果，我们得出结论：分类损失和GIoU损失对于目标检测的性能至关重要。
使用固定的正弦位置编码通过编码器和解码器的每个注意力层。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显著下降。有趣的是，仅在解码器中传递它们会导致较小的AP下降。所有这些模型都使用了学习的输出位置编码。
空间位置编码 输出位置编码
编码器 解码器 解码器 AP AP50
无 无 输入处学习 32.8 -7.8 55.2 -6.5
输入处正弦 输入处正弦 输入处学习 39.2 -1.4 60.0 -1.6
注意力处学习 注意力处学习 注意力处学习 39.6 -1.0 60.7 -0.9
无 输入处正弦 注意力处学习 39.3 -1.3 60.3 -1.4
注意力处正弦 注意力处正弦 注意力处学习 40.6 -61.6 -
表4：损失成分对AP的影响。我们训练了两个模型，关闭 1损失和GIoU损失，并观察到 1仅在单独使用时效果差，但与GIoU结合后可以改善AP和AP L。我们的基准模型（最后一行）结合了这两个损失函数。
class 1 GIoU AP AP50 APS APM APL
X X 35.8 -4.8 57.3 -4.4 13.7 39.8 57.9
X X 39.9 -0.7 61.6 0 19.9 43.2 57.9
X X X 40.6 -61.6 - 19.9 44.3 60.2
对于大多数模型性能，与基线相比仅损失0.7 AP。使用单独的`1而没有GIoU会导致很差的结果。我们只进行了简单的删除不同损失的实验（每次使用相同的权重），但是进行了14 Carion等分析。
图7：在COCO 2017验证集上的所有图像上可视化DETR解码器中100个预测槽中的20个预测框。每个框预测以其在归一化的1x1方块中心的坐标表示，归一化使用每个图像的大小。点的颜色编码使得绿色对应于小框，红色对应于大的水平框，蓝色对应于大的垂直框。我们观察到每个槽位都会专注于特定的区域和框的尺寸，具有多种操作模式。我们注意到几乎所有槽位都有一个模式，可以预测COCO数据集中常见的大型全图框。
对于不同的损失函数进行了简单的消融实验（每次使用相同的权重），但是我们进行了4.3分析。
4.3 分析
解码器输出插槽分析 在图7中，我们可视化了在COCO 2017验证集中所有图像中由不同插槽预测的框。DETR为每个查询插槽学习了不同的专业化。我们观察到每个插槽都有几种操作模式，专注于不同的区域和框大小。特别是，所有插槽都具有预测全局框的模式（红点对齐在图的中间可见）。我们假设这与COCO中对象的分布有关。

对未见过的实例数量的泛化。COCO中的一些类别在同一图像中并没有很多相同类别的实例。例如，在训练集中没有超过13个长颈鹿的图像。我们创建一个合成图像3来验证DETR的泛化能力（见图5）。我们的模型能够找到图像上的所有24只长颈鹿，这显然是超出了分布范围。此实验证实了每个对象查询中没有强烈的类别专业化。

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很大关注。类似于将Faster R-CNN [37]扩展为Mask R-CNN [14]，DETR可以自然地通过在解码器输出之上添加一个掩膜头来进行扩展。在本节中，我们展示了这样一个头可以用于生成全景分割[19]，通过将物体和物品类别视为不同的类别进行处理。

多头注意力 输入图像
(3 x H x W)
盒嵌入
(d x N)

编码图像
(d x H/32 x W/32)
注意力图
(N x M x H/32 x W/32)
掩膜logits
(N x H/4 x W/4)
逐像素argmax
连接 2 x (Conv 3x3 + GN + ReLU) 2x 上采样 + 添加 Conv 3x3 + GN + ReLU 2x 上采样 + 添加 Conv 3x3 + GN + ReLU 2x 上采样 + 添加 Conv 3x3 + GN + ReLU + Conv 3x3
FPN风格的CNN Resnet特征
Res5 Res4 Res3 Res2
焦点损失[23]。
时间训练。
以统一的方式对物体和学科生成对齐的掩码预测。我们在COCO数据集的全景注释上进行实验，除了80个物体类别外，还有53个学科类别。我们使用相同的方法训练DETR在COCO上预测学科和物体类别周围的框。为了实现训练，需要预测框，因为匈牙利匹配是使用框之间的距离来计算的。我们还添加了一个掩码头，为每个预测的框预测一个二值掩码，详见图8。它以每个对象的transformer解码器的输出作为输入，计算此嵌入在编码器输出上的多头（具有M个头）注意力得分，生成每个对象在小分辨率下的Mattention热图。为了做出最终预测并提高分辨率，使用了类似FPN的架构。我们在补充中对架构进行了更详细的描述。掩码的最终分辨率为4倍，并且每个掩码都使用DICE/F-1损失[28]和Focal损失[23]进行独立监督。
掩码头可以同时训练，也可以通过两步过程进行训练，其中我们仅训练DETR的框，然后冻结所有权重并仅训练掩码头25个时期。实验结果表明，这两种方法的结果相似，我们选择使用后一种方法，因为它的总的实际时间较短。[16]
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相同的数据增强方法，按照公平比较的要求，在18x的框架下重新训练了PanopticFPN。UPSNet使用1x的框架，UPSNet-M是具有多尺度测试增强的版本。模型骨干PQ SQ RQ
PQthSQthRQthPQstSQstRQstAP
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
不同的掩码。
5 结论
6 致谢
(2019)
为了预测最终的全景分割结果，我们只需在每个像素上使用掩码分数的argmax，并将相应的类别分配给结果掩码。这个过程确保最终的掩码没有重叠，因此DETR不需要一个常用于对齐不同掩码的启发式方法[19]。

训练细节。我们根据边界框检测的方法训练DETR、DETR-DC5和DETR-R101模型，以预测COCO数据集中stu
和things类别周围的边界框。新的mask head在25个epochs中进行训练（详见补充材料）。在推理过程中，我们首先过滤掉置信度低于85%的检测结果，然后计算每个像素的argmax，确定它属于哪个掩码。然后，我们将同一stu
类别的不同掩码预测合并成一个掩码，并过滤掉空的掩码（少于4个像素）。

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景分割方法与几种处理things和stu
的已建立方法进行了比较。我们报告了全景质量（PQ）以及things（PQth）和stu
（PQst）的细分。我们还报告了掩码AP（在things类别上计算），在进行任何全景后处理之前（在我们的情况下，即取像素级argmax之前）。我们展示了DETR在COCO-val 2017上优于已发表的结果，以及我们的强基准模型PanopticFPN（使用与DETR相同的数据增强进行训练，以进行公平比较）。结果细分显示DETR在stu
类别上特别占优势，我们推测全局推理允许的编码器注意力是导致这一结果的关键因素。对于things类别，尽管在掩码AP计算中与基线相比存在高达8 mAP的严重缺陷，但DETR获得了有竞争力的PQth。我

们还对COCO数据集的测试集进行了评估，获得了46 PQ。我们希望我们的方法能够激发

5 结论

我们提出了DETR，这是一种基于Transformer和二分匹配损失的目标检测系统的新设计。该方法在具有挑战性的COCO数据集上实现了与优化的Faster R-CNN基线相当的结果。DETR容易实现，并且具有灵活的架构，易于扩展到全景分割并获得具有竞争力的结果。此外，DETR在大物体上的性能显著优于Faster R-CNN，可能要归功于自注意力模块对全局信息的处理。

这种新设计的检测器也带来了新的挑战，尤其是在训练、优化以及小目标上的性能方面。当前的检测器经过多年的改进才能应对类似的问题，我们期望未来的工作能够成功解决DETR上出现的问题。

6 致谢

我们要感谢Sainbayar Sukhbaatar、Piotr Bojanowski、Natalia Neverova、David Lopez-Paz、Guillaume Lample、Danielle Rothermel、Kaiming He、Ross Girshick、Xinlei Chen以及整个Facebook AI Research Paris团队的讨论和建议，没有他们，这项工作将不可能完成。

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 convolutional 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 instance segmentation. PAMI (2019)
6. Cordonnier, J.B., Loukas, A., Jaggi, M.: 关于自注意力和卷积层之间的关系。在：ICLR（2020年）
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT：用于语言理解的深度双向Transformer预训练。在：NAACL-HLT（2019年）
8. Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: 使用深度神经网络进行可伸缩的目标检测。在：CVPR（2014年）
9. Ghazvininejad, M., Levy, O., Liu, Y., Zettlemoyer, L.: Mask-predict：条件掩蔽语言模型的并行解码。arXiv：1904.09324（2019年）
10. Glorot, X., Bengio, Y.: 理解训练深度前馈神经网络的困难之处。在：AISTATS（2010年）18 Carion et al.
11. Gu, J., Bradbury, J., Xiong, C., Li, V.O., Socher, R.: 非自回归神经机器翻译。在：ICLR（2018年）
12. He, K., Girshick, R., Dollm ́ ar, P.: 重新思考ImageNet预训练。在：ICCV（2019年）
13. He, K., Gkioxari, G., Dollm ́ ar, P., Girshick, R.B.: Mask R-CNN。在：ICCV（2017年）
14. He, K., Zhang, X., Ren, S., Sun, J.: 深层残差学习用于图像识别。在：CVPR（2016年）
15. Hosang, J.H., Benenson, R., Schiele, B.: 学习非极大值抑制。在：CVPR（2017年）
16. Hu, H., Gu, J., Zhang, Z., Dai, J., Wei, Y.: 关系网络用于目标检测。在：CVPR（2018年）
17. Kirillov, A., Girshick, R., He, K., Dollm ́ ar, P.: 全景特征金字塔网络。在：CVPR（2019年）
18. Kirillov, A., He, K., Girshick, R., Rother, C., Dollar, P.: 全景分割。在：CVPR（2019年）
19. Kuhn, H.W.: 匈牙利方法解决指派问题（1955年）
20. Li, Y., Qi, H., Dai, J., Ji, X., Wei, Y.: 全卷积实例感知语义分割。在：CVPR（2017年）
21. Lin, T.Y., Dollm ́ ar, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: 特征金字塔网络用于目标检测。在：CVPR（2017年）
22. Lin, T.Y., Goyal, P., Girshick, R.B., He, K., Dollm ́ ar, P.: 密集对象检测的焦点损失。在：ICCV（2017年）
    arXiv：1905.03072（2019年）
23. Loshchilov, I., Hutter, F.: 解耦的权重衰减正则化. 在：ICLR (2017)
24. Luscher, C., Beck, E., Irie, K., Kitza, M., Michel, W., Zeyer, A., Schluter, R., Ney, H.: Rwth asr systems for librispeech: 基于混合和注意力的语音识别模型 - 不使用数据增强. arXiv:1905.03072 (2019)

25. 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)
26. 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.: 并行的Wavenet: 快速高保真语音合成. arXiv:1711.10433 (2017)
27. Park, E., Berg, A.C.: 学习分解用于目标检测和实例分割. arXiv:1511.06449 (2015)
28. Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., Tran, D.: 图像Transformer. 在：ICML (2018)
29. 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: 一种命令式风格、高性能的深度学习库. 在：NeurIPS (2019)
30. Pineda, L., Salvador, A., Drozdzal, M., Romero, A.: 阐明图像到集合的预测：模型、损失和数据集的分析. arXiv:1904.05709 (2019)
31. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: 语言模型是无监督多任务学习器 (2019)
32. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: 统一的实时目标检测. 在：CVPR (2016)
33. Ren, M., Zemel, R.S.: 基于递归注意力的端到端实例分割. 在：CVPR (2017)
34. Ren, S., He, K., Girshick, R.B., Sun, J.: Faster R-CNN: 实现实时目标检测的区域建议网络. PAMI (2015) 在：ICCV (2017)
35. Rezatoghi等(2017)提出了Deepsetnet，利用深度神经网络预测集合。
36. Romera-Paredes和Torr(2015)提出了一种循环实例分割方法。
37. Salvador等(2017)利用循环神经网络进行语义实例分割。
38. Stewart等(2015)提出了在拥挤场景中进行端到端人体检测的方法。
39. Sutskever等(2014)提出了利用神经网络进行序列到序列学习的方法。
40. Synnaeve等(2019)提出了一种从监督到半监督学习的端到端ASR方法。
41. Tian等(2019)提出了一种全卷积单阶段目标检测方法。
42. Vaswani等(2017)提出了注意力机制的Transformer模型。
43. Vinyals等(2016)提出了一种用于集合的序列到序列学习方法。
44. Wang等(2018)提出了非局部神经网络。
45. Wu等(2019)提出了Detectron2目标检测框架。
46. Xiong等(2019)提出了一种统一的全景分割网络。
47. Zhang等(2019)提出了一种通过自适应训练样本选择来桥接基于锚框和无锚框目标检测的方法。
48. Zhou等(2019)提出了一种将目标表示为点的方法。

A.1 预备知识: 多头注意力层

根据[7]中的方法，多头注意力层的计算方式如下:

[
\text{{d□Nq(3)}} = \sum_{x=0}^N q(x) \cdot \text{{softmax}}(k^T(x) \cdot q(x))
]

多头注意力机制是一种具有M个维度为d的注意力头的通用形式，其中d0为d的倍数，并且给出了以下函数的签名（使用d0 = d*M，并在括号下方给出矩阵/张量的大小）。

mh-attn :Xq|{z}
d□Nq; X kv|{z}
d□Nkv; T|{z}
M□3□d0□d; L|{z}
d□d7!~Xq|{z}
d□Nq (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（为了简单起见，具有相同的通道数d），T是权重张量，用于计算所谓的查询、键和值的嵌入，并且L是一个投影矩阵。输出与查询序列的大小相同。为了固定词汇表并给出细节，多头自注意力（mh-s-attn）是Xq=Xkv的特例，即

mh-s-attn(X;T;L) = mh-attn(X;X;T;L) (4)
多头注意力是单个注意力头的串联后由L进行投影得到的结果。常见做法[47]是使用残差连接、dropout和层标准化。换句话说，将~Xq= mh-attn(Xq;Xkv;T;L)表示为注意力头的串联，我们有
X0
q= [attn(Xq;Xkv;T1);:::; attn(Xq;Xkv;TM)] (5)
~Xq= layernorm(Xq+ dropout(LX0
q)) (6)
其中[;]表示在通道轴上进行串联。

单个注意力头是具有权重张量T02R3□d0□d的注意力头，用attn(Xq;Xkv;T0)表示，它依赖于额外的位置编码Pq2Rd□Nq和Pkv2Rd□Nkv。在添加查询和键的位置编码[7]之后，它首先计算所谓的查询、键和值的嵌入：
[Q;K;V] = [T0
1(Xq+Pq);T0
2(Xkv+Pkv);T0
3Xkv] (7)
其中T0是T0
1、T0
2和T0
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，以使查询序列的每个元素都关注键值序列的所有元素（i为查询索引，j为键值索引）：

i;j=e1p
d0QT
iKj
ZiwhereZi=NkvX
j=1e1p
d0QT
iKj: (8)

最后，通过attni(Xq;Xkv;T0) =PNkv
j=1
i;jVj将注意力头的输出权重与值进行加权求和，形成最终的输出□。
是通过注意力权重对值进行加权的聚合：第i行是由attni（Xq; Xkv; T0）= PNkv j = 1 jVi给出的。
前馈网络（FFN）层Transformer原始的交替多头注意力和所谓的FFN层[47]，后者实际上是多层1x1卷积，在我们的情况下具有Md输入和输出通道。我们考虑的FFN由两层带ReLU激活的1x1卷积组成。在这两层之后，同方程6一样还有一个残差连接/丢弃/层归一化。
A.2 损失
为了完整起见，我们详细介绍我们方法中使用的损失。所有的损失都经过了批次内对象数量的归一化处理。在分布式训练中需要特别注意：因为每个GPU接收到一个子批次，仅通过每个本地子批次中的对象数量进行归一化是不够的，因为一般来说，子批次在各个GPU上是不平衡的。相反，重要的是通过所有子批次中的总对象数量进行归一化处理。
框损失与[41,36]类似，我们在损失中使用了交并比的软版本，并加上了对^b的L1损失：
Lbox（bˆ（i）; ^ bi）= □iouLiou（bˆ（i）; ^ bi）+ □L1jjbˆ（i） - ^ bijj1;（9）
其中□iou; □L12是超参数，而Liou（·）是广义IoU[38]：
Liou（bˆ（i）; ^ bi）=1 - [(bˆ（i）∩^ bi) / (bˆ（i）∪^ bi)]²（10）
其中：j表示"面积"，并且框坐标的并集和交集用于表示框本身的简写。联合或交集的面积通过bˆ（i）和^ bi的线性函数的min = max进行计算，这使得损失在随机梯度下具有良好的行为。B（bˆ（i）; ^ bi）表示包含bˆ（i）; ^ bi的最大框（涉及B的区域也基于框坐标的线性函数的min = max计算）。
DICE/F-1损失[28]DICE系数与交并比密切相关。如果我们用^ m表示模型的原始蒙版逻辑预测，那么LDICE（m; ^ m）=1-（2m∩^ m）/（m + ^ m）是DICE系数的损失函数。
Add & Norm
(^m) +m+ 1

其中 是 sigmoid 函数。这个损失是通过物体数量进行归一化的。Carion 等人。
A.3 详细结构
DETR 中使用的 transformer 的详细描述，包括在每个注意力层中传递的位置编码，如图10所示。图像特征通过 transformer 编码器传递，同时还传递空间位置编码，空间位置编码在每个多头自注意力层中添加到查询和键中。然后，解码器接收查询（最初设为零）、输出位置编码（目标查询）和编码器记忆，并通过多个多头自注意力和解码器-编码器注意力生成最终的预测类别标签和边界框集合。可以跳过第一个解码器层中的第一个自注意力层。
Add & Norm
FFN
Add & Norm
多头自注意力
/plus.osf /plus.osfK Q VN×
图像特征编码器
多头自注意力Add & Norm多头注意力Add & NormFFNAdd & Norm
/plus.osf /plus.osfK Q V/plus.osf /plus.osfK QM×解码器
V
空间位置编码目标查询FFN FFN类别 边界框
图10：DETR transformer 的结构。请参见第A.3 节获取详细信息。
计算复杂度
编码器中的每个自注意力的复杂度为 O(d2HW + d(HW)2)：O(d0d) 是计算单个查询/键/值嵌入的成本（并且 M = d0），而O(d0(HW)2) 是计算一个头部的注意力权重的成本。其他计算可以忽略。在解码器中，每个自注意力的复杂度为 O(d2N + dN2)，而编码器和解码器之间的交叉注意力则为O(d2(N+HW) + dNHW)，这比编码器低得多，因为在实践中N□HW。使用 Transformer 进行端到端的物体检测24
FLOPS 计算
由于 Faster R-CNN 的 FLOPS 取决于图像中的 proposals 数量，我们计算了 COCO 2017 验证集中前100个图像的平均 FLOPS 数。我们将 DETR 模型的 multiply (bmm) 考虑在内。
将bmm（批矩阵乘法）考虑在DETR模型中。
A.4 训练超参数
我们使用改进的AdamW [26]训练DETR模型，设置为10^-4的权重衰减。我们还应用梯度裁剪，最大梯度范数为0.1。主干网络和transformer的处理略有不同，我们现在讨论两者的细节。
主干网络导入自Torchvision的ImageNet预训练的ResNet-50，去除最后的分类层。在训练过程中冻结主干网络的batch normalization权重和统计量，这是目标检测中广泛采用的做法。我们使用学习率为10^-5对主干网络进行微调。我们观察到，在训练中，主干网络的学习率大约比网络的其他部分小一个数量级是很重要的，特别是在最初的几个epoch中，可以稳定训练过程。
transformer模型使用学习率为10^-4进行训练。在每个多头注意力和FFN之后，应用0.1的加性dropout，并在layer normalization之前。权重使用Xavier初始化进行随机初始化。
损失函数使用`l loss和GIoU loss的线性组合来进行边界框回归，其中`1的权重为5，GIoU的权重为2。所有模型都使用N=100的解码器查询槽进行训练。
基准模型我们改进的Faster-RCNN+基准模型使用GIoU [38]损失和标准的`1损失进行边界框回归。我们进行了一个网格搜索来找到最佳的损失权重，最终模型只使用GIoU损失，其中盒子回归任务的权重为20，提议回归任务的权重为1。对于基准模型，我们采用了DETR中使用的相同的数据增强方法，并使用9倍的训练计划（大约109个epochs）进行训练。其他设置与Detectron2模型仓库中的相同模型相同[50]。
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，我们使用每个嵌入的固定绝对空间坐标独立地使用了PanopticFPN。

2. 正弦和余弦函数具有不同频率。然后，我们将它们串联起来以获得最终的d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）有重叠物体的失败案例。PanopticFPN完全错过了一个平面，而DETR无法准确分割其中的3个。
   （b）物体的掩模以完整分辨率预测，这比PanopticFPN产生了更锐利的边界。

图11：全景预测的比较。从左到右：真实情况，采用ResNet 101的PanopticFPN，采用ResNet 101的DETR。

增加实例数目 按设计，DETR不能预测比其查询槽更多的对象，即在我们的实验中为100个。在本节中，我们分析将DETR接近此限制时的行为。我们选择给定类别的规范的方形图像，在10×10的网格上重复它，并计算被模型漏掉的实例的百分比。为了用少于100个实例测试模型，我们随机屏蔽一些单元格。这样无论有多少个可见，对象的绝对大小是相同的。为了考虑屏蔽中的随机性，我们重复100次实验，使用不同的屏蔽。结果如图12所示。行为在不同类别之间是相似的，当可见的实例少于50个时，模型可以检测到所有实例，然后开始饱和并漏掉越来越多的实例。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，这比只包含50个实例并且所有实例都被检测到的图像要少。模型的这种反直觉行为可能是因为图像和检测结果与训练分布差距很大。请注意，该测试是设计成在分布之外进行的一项泛化测试，因为很少有示例图像包含大量单个类别的实例。

20 40 60 80 100010203040506070
人类
苹果
4
6
20
32
图12：根据图像中存在的实例数量，分析DETR错过的各类实例数量的变化。我们报告均值和标准差。当实例数量接近100时，DETR开始饱和，并错过越来越多的物体。

A.6 PyTorch推断代码

为了展示这种方法的简单性，我们在第1行代码中包括了使用PyTorch和Torchvision库的推断代码。该代码可在Python 3.6+、PyTorch 1.4和Torchvision 0.5上运行。请注意，该代码不支持批处理，因此仅适用于每个GPU进行推断或使用DistributedDataParallel进行训练的情况下，每个GPU仅处理一张图像。此外，为了清晰起见，该代码在编码器中使用了学习到的位置编码而不是固定编码，并且位置编码仅在输入中添加，而不是在每个Transformer层中都添加。进行这些更改需要超出PyTorch实现的transformers范围，这会影响可读性。在会议之前，将提供完整的复现实验的代码（引用号26 Carion et al.）。

```
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):
```

22 x = self.backbone(inputs)
输入图像经过骨干网络，得到特征图 x。

23 h = self.conv(x)
特征图 x 经过卷积层，得到特征张量 h。

24 H, W = h.shape[-2:]
获取特征张量 h 的高度 H 和宽度 W。

25 pos = torch.cat([
上述代码使用了 PyTorch 的 torch.cat 函数，将以下内容进行拼接。

26 self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
切片操作 self.col_embed[:W]，将其在维度 0 上进行扩展为 H 行，1 列，1 通道的张量。

27 self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
切片操作 self.row_embed[:H]，将其在维度 1 上进行扩展为 1 行，W 列，1 通道的张量。

28], dim=-1).flatten(0, 1).unsqueeze(1)
使用 torch.cat 函数将上述两个张量在维度 -1 上进行拼接。然后使用扁平化函数 torch.flatten 进行扁平化操作，将结果变为一个二维张量。最后对结果在维度 0 上添加一维，得到形状为 (H*W, 1, 2) 的张量。

29 h = self.transformer(pos + h.flatten(2).permute(2, 0, 1),
使用矩阵运算，“+” 表示张量对应位置进行加法运算。pos 是一个形状为 (H*W, 1, 2) 的张量，h.flatten(2).permute(2, 0, 1) 是将 h 在维度 2 上进行扁平化操作，并进行维度转置，使其形状为 (2, HW, 1)。两个张量进行相加后得到一个形状为 (H*W, 1, 2) 的张量。

30 self.query_pos.unsqueeze(1))
对 self.query_pos 张量在维度 1 上添加一维，得到形状为 (100, 1, 256) 的张量。

31 return self.linear_class(h), self.linear_bbox(h).sigmoid()
将特征张量 h 分别输入到 self.linear_class 和 self.linear_bbox 线性层中，并对 self.linear_bbox 的输出进行 sigmoid 函数操作后返回。返回结果是两个形状为 (HW, 1, num_classes) 和 (HW, 1, 4) 的张量。

32
创建一个 DETR 类的实例 detr，其中包含参数 num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6。

34 detr.eval()
将 detr 设置为评估模式。

35 inputs = torch.randn(1, 3, 800, 1200)
创建一个形状为 (1, 3, 800, 1200) 的张量 inputs，用于输入到 detr。

36 logits, bboxes = detr(inputs)
调用 detr 的前向传播函数，得到输出 logits 和 bboxes。

清单 1：DETR 的 PyTorch 推理代码。为了清晰起见，它在编码器中使用了学习到的位置编码，而不是固定的，并且位置编码只添加到输入中，而不是在每个 transformer 层中。进行这些更改需要超越 PyTorch 中 transformer 的实现，这会影响可读性。在会议前，将提供完整的代码以重现实验。