

End-to-End Object Detection with Transformers

尼古拉斯·卡里翁*、弗朗西斯科·马萨*、加布里埃尔·辛纳夫、尼古拉·乌松尼尔、亚历山大·基里洛夫和谢尔盖·扎戈鲁伊科

Facebook 人工智能

Abstract. 我们提出了一种新方法，将目标检测视为直接的集合预测问题。该方法简化了检测流程，有效消除了对许多手工设计组件的需求，如非极大值抑制过程或锚框生成，这些组件显式编码了我们对任务的先验知识。新框架名为DEtection TRansformer (DETR)，其核心要素包括：通过二分匹配强制实现唯一预测的基于集合的全局损失，以及Transformer编码器-解码器架构。给定一组固定的小型学习对象查询，DETR通过分析对象间关系及全局图像上下文，直接并行输出最终的预测集合。新模型概念简洁，且与许多现代检测器不同，无需依赖专用库。在极具挑战性的COCO目标检测数据集上，DETR的准确率和运行时性能与经过高度优化的Faster R-CNN基准相当。此外，DETR能够以统一方式轻松泛化以实现全景分割。实验表明，其性能显著优于竞争基准。训练代码与预训练模型详见

<https://github.com/facebookresearch/detr>。

1 Introduction

目标检测的目标是为每个感兴趣物体预测一组边界框和类别标签。现代检测器通过在大规模候选框[37,5]、锚点[23]或窗口中心[53,46]上定义替代回归与分类问题，以间接方式解决这一集合预测任务。其性能显著受后处理步骤（用于消除近似重复预测）、锚点集设计以及将目标框分配给锚点的启发式规则[52]所影响。为简化流程，我们提出直接集合预测方法以绕过替代任务。这种端到端理念已在机器翻译、语音识别等复杂结构化预测任务中取得重大突破，但在目标检测领域尚未实现：先前尝试[43,16,4,39]要么引入其他形式的先验知识，要么未能在挑战性基准测试中证明其优于强基线。本文旨在弥合这一差距。

* Equal contribution

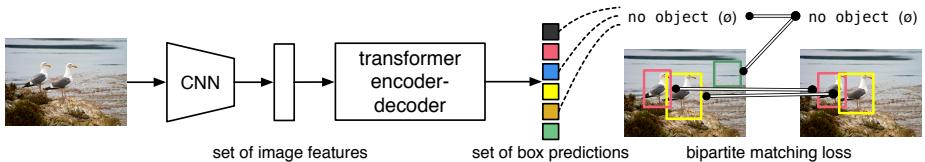


图1：DETR通过将常规CNN与transformer架构相结合，直接（并行）预测最终的检测集合。在训练过程中，二分匹配唯一地将预测框与真实标注框进行配对。未匹配的预测应输出“no object”（ \emptyset ）类别预测。

我们将目标检测视为一个直接的集合预测问题，从而简化了训练流程。我们采用了基于transformer[47]的编码器-解码器架构，这是一种在序列预测中广泛使用的结构。transformer的自注意力机制能显式建模序列元素间的所有两两交互，使得该架构特别适合集合预测的特定约束条件，例如消除重复预测。

我们的DEtection TRansformer (DETR, 见图1) 一次性预测所有对象，并通过一个执行预测对象与真实对象间二分匹配的集合损失函数进行端到端训练。DETR通过摒弃多个手工设计的编码先验知识的组件（如空间锚点或非极大值抑制），简化了检测流程。与大多数现有检测方法不同，DETR不需要任何定制层，因此可以在包含标准CNN和transformer类的任何框架中轻松复现¹。

与以往大多数直接集合预测的研究相比，DETR的核心特征在于结合了二分图匹配损失与基于（非自回归）并行解码的transformer架构[29,12,10,8]。而先前工作主要采用RNN进行自回归解码[43,41,30,36,42]。我们的匹配损失函数能够唯一地将预测结果分配给真实目标对象，并且对预测对象的排列顺序具有不变性，因此可以实现并行输出。

我们在最流行的目标检测数据集之一COCO[24]上评估DETR，并与极具竞争力的Faster R-CNN基线模型[37]进行对比。Faster R-CNN历经多次设计迭代，自原始论文发表以来性能已大幅提升。实验表明，我们的新模型取得了与之相当的性能表现。具体而言，DETR在大尺寸目标上展现出显著更优的性能，这一结果很可能得益于Transformer的非局部计算特性。然而，该模型在小尺寸目标上的表现稍逊。我们预期未来工作会在这方面实现改进，正如FPN[22]的发展对Faster R-CNN所起到的推动作用。

DETR的训练设置与标准目标检测器在多个方面存在差异。新模型需要超长的训练周期，并从中获益。

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

从Transformer中的辅助解码损失出发。我们深入探究了哪些组件对展现出的性能至关重要。

DETR的设计理念能轻松扩展到更复杂的任务中。在我们的实验中，一个基于预训练DETR构建的简单分割头在Panoptic Segmentation[19]上超越了竞争性基线，这一具有挑战性的像素级识别任务近来备受关注。

2 Related work

我们的工作建立在多个领域的先前研究基础之上：用于集合预测的双边匹配损失、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 Set Prediction

目前尚无标准的深度学习模型能直接预测集合。基本的集合预测任务是多标签分类（例如，在计算机视觉领域可参考[40,33]），其基线方法“一对多”并不适用于检测等存在元素间内在结构（如近乎相同的边界框）的问题。这类任务的首要难点在于避免近重复项。当前大多数检测器采用非极大值抑制等后处理手段解决该问题，而直接集合预测则无需后处理。它们需要建模所有预测元素间交互的全局推理方案来消除冗余。对于固定大小的集合预测，密集全连接网络[9]虽足够但成本高昂。一种通用方法是采用自回归序列模型，如循环神经网络[48]。无论哪种情况，损失函数都应对预测排列顺序保持不变。通常的解决方案是基于匈牙利算法[20]设计损失函数，以在真实标注与预测间寻找二分匹配。这确保了排列不变性，并保证每个目标元素都有唯一匹配。我们遵循二分匹配损失方法。但与多数先前工作不同，我们摒弃了自回归模型，转而使用下文所述的并行解码Transformer架构。

2.2 Transformers and Parallel Decoding

Transformer由Vaswani *et al.* [47]提出，作为一种基于注意力机制的全新机器翻译构建模块。注意力机制[2]是一种神经网络层，能够从整个输入序列中聚合信息。Transformer引入了自注意力层，其原理类似于非局部神经网络[49]，通过遍历序列中的每个元素，并整合整个序列的信息来更新该元素。基于注意力的模型主要优势之一在于其全局计算能力和完美记忆性，这使得它们相比RNN更擅长处理长序列。Transformer如今

在许多自然语言处理、语音处理和计算机视觉问题中取代了RNN[8,27,45,34,31]

◦ Transformer最初被用于自回归模型，遵循早期的序列到序列模型[44]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量处理），推动了并行序列生成技术的发展，这一技术在音频[29]、机器翻译[12,10]、词表示学习[8]以及最近的语音识别[6]等领域得到了应用。我们同样结合了Transformer与并行解码技术，以在计算成本与执行集合预测所需的全局计算能力之间取得适宜的平衡。

2.3 Object detection

大多数现代目标检测方法都是基于某些初始猜测进行预测的。两阶段检测器[37, 5]相对于候选框预测边界框，而单阶段方法则相对于锚点[23]或可能的目标中心网格[53,46]进行预测。近期研究[52]表明，这些系统的最终性能很大程度上取决于初始猜测的具体设定方式。在我们的模型中，我们能够摒弃这一人工设计的过程，通过直接预测相对于输入图像而非锚点的绝对框坐标集合，从而简化检测流程。

Set-based loss. 多个目标检测器[9,25,35]采用了二分图匹配损失。然而在这些早期深度学习模型中，不同预测间的关系仅通过卷积层或全连接层建模，而手工设计的非极大值抑制(NMS)后处理能提升其性能。较新的检测器[37,23,53]则采用真实标注与预测间的非唯一分配规则，并配合NMS使用。

可学习的NMS方法[16,4]和关系网络[17]通过注意力机制显式建模不同预测之间的关系。它们采用直接集合损失，无需任何后处理步骤。然而，这些方法使用了手工设计的上下文特征（如提议框坐标）来高效建模检测间关系，而我们寻求的是减少模型编码先验知识的解决方案。

Recurrent detectors. 与我们的方法最为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，它们利用基于CNN激活的编码器-解码器架构，结合二分图匹配损失，直接生成一组边界框。然而，这些方法仅在小规模数据集上进行了评估，并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），因此未能利用近期支持并行解码的transformer技术。

3 The DETR model

直接集合预测在检测中不可或缺的两个要素是：(1) 一种集合预测损失，它强制预测结果与真实值之间实现唯一匹配

框；(2)一种架构，能够(单次)预测一组对象并建模它们之间的关系。我们在图2中详细描述了这一架构。

3.1 Object detection set prediction loss

DETR通过解码器的一次前向传播推断出一个固定大小的 N 预测集合，其中 N 被设定为远大于图像中常见物体数量。训练过程中的主要难点之一是如何根据真实值对预测物体(类别、位置、大小)进行评分。我们的损失函数在预测物体与真实物体之间建立了最优二分匹配，并进一步优化针对物体(边界框)的特定损失。

我们用 y 表示真实目标对象的集合， $\hat{y} = \{\hat{y}_i\}_{i=1}^N$ 表示 N 预测结果的集合。假设 N 大于图像中目标的数量，我们将 y 也视为一个大小为 N 的集合，并用 \emptyset (无目标)进行填充。为了在这两个集合之间找到一个二分匹配，我们搜索 N 个元素 $\sigma \in \mathfrak{S}_N$ 的一个排列，使得匹配成本最低：

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

其中 $\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$ 是真实值 y_i 与索引为 $\sigma(i)$ 的预测值之间的成对 matching cost。这一最优分配通过匈牙利算法高效计算得出，遵循先前工作(*e.g.* [43])。

匹配成本同时考虑了类别预测以及预测框与真实框之间的相似度。真实集合中的每个元素 i 可视为 $y_i = (c_i, b_i)$ ，其中 c_i 表示目标类别标签(可能为 \emptyset)，而 $b_i \in [0,1]^4$ 是一个向量，定义了真实框的中心坐标及其相对于图像尺寸的高度和宽度。对于索引为 $\sigma(i)$ 的预测，我们将类别 c_i 的概率定义为 $\hat{p}_{\sigma(i)}(c_i)$ ，预测框定义为 $\hat{b}_{\sigma(i)}$ 。基于这些符号表示，我们将 $\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$ 定义为 $-1_{\{c_i \neq \emptyset\}} \hat{p}_{\sigma(i)}(c_i) + 1_{\{c_i \neq \emptyset\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})$ 。

这一寻找匹配的过程，其作用与现代检测器中用于将提案[37]或锚点[22]与真实对象配对的启发式分配规则相同。主要区别在于，我们需要为直接集合预测找到一对匹配，避免重复。

第二步是计算损失函数，即对上一步匹配的所有配对计算 Hungarian loss。我们以类似于常见物体检测器损失的方式定义该损失，*i.e.*，即类别预测的负对数似然与后续定义的边界框损失的线性组合：

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

其中 $\hat{\sigma}$ 是第一步(1)中计算得到的最优分配。实际操作中，当 $c_i = \emptyset$ 时，我们会将对数概率项按10倍系数进行降权处理，以考虑

类别不平衡。这与Faster R-CNN训练过程中通过子采样平衡正负提案的做法类似[37]。需要注意的是，物体与 \emptyset 之间的匹配成本并不依赖于预测结果，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 $\hat{p}_{\hat{\sigma}(i)}(c_i)$ 而非对数概率。这使得类别预测项与下文描述的 $\mathcal{L}_{\text{box}}(\cdot, \cdot)$ 具有可比性，并且我们观察到了更好的实证性能。

Bounding box loss. 匹配代价的第二部分与匈牙利损失中的 $\mathcal{L}_{\text{box}}(\cdot)$ 用于对边界框进行评分。与许多检测器不同，它们基于某些初始猜测以 Δ 形式进行框预测，我们则直接进行框预测。虽然这种方法简化了实现，却带来了损失相对缩放的问题。最常用的 ℓ_1 损失对于小框和大框会有不同的尺度，即使它们的相对误差相似。为了缓解这一问题，我们采用了 ℓ_1 损失与广义IoU损失[38] $\mathcal{L}_{\text{iou}}(\cdot, \cdot)$ 的线性组合，后者具有尺度不变性。总体而言，我们的框损失 $\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})$ 定义为 $\lambda_{\text{iou}}\mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{\text{L1}}||b_i - \hat{b}_{\sigma(i)}||_1$ ，其中 $\lambda_{\text{iou}}, \lambda_{\text{L1}} \in \mathbb{R}$ 为超参数。这两种损失通过批次内的对象数量进行了归一化。

3.2 DETR architecture

DETR的整体架构出奇地简单，如图2所示。它包含三个主要组件，我们将在下面进行描述：用于提取紧凑特征表示的CNN主干网络、一个编码器-解码器结构的transformer，以及一个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可以在任何提供通用CNN主干网络和Transformer架构实现的深度学习框架中实现，仅需数百行代码。在PyTorch中，DETR的推理代码实现甚至不超过50行[32]。我们希望这一方法的简洁性能够吸引新研究人员加入检测领域。

Backbone. 从初始图像 $x_{\text{img}} \in \mathbb{R}^{3 \times H_0 \times W_0}$ （（包含3个颜色通道²））出发，传统的CNN骨干网络会生成一个较低分辨率的激活图 $f \in \mathbb{R}^{C \times H \times W}$ 。我们采用的典型值为 $C = 2048$ 和 $H, W = \frac{H_0}{32}, \frac{W_0}{32}$ 。

Transformer encoder. 首先，一个1x1卷积将高层激活图 f 的通道维度从 C 降至较小维度 d ，生成新特征图 $z_0 \in \mathbb{R}^{d \times H \times W}$ 。编码器要求输入为序列，因此我们将 z_0 的空间维度压缩为一维，得到 $d \times HW$ 特征图。每个编码层采用标准架构，包含多头自注意力模块和前馈网络（FFN）。由于Transformer架构具有排列不变性，我们为其添加了固定位置编码[31,3]，这些编码被叠加到每个注意力层的输入上。架构的详细定义遵循文献[47]所述，具体内容可参阅补充材料。

² 输入图像被批量处理，适当应用0填充以确保它们都具有与批次中最大图像相同的尺寸 (H_0, W_0) 。



图2：DETR采用传统CNN骨干网络学习输入图像的二维表示。模型将其展平并添加位置编码后，输入至Transformer编码器。随后，Transformer解码器以少量固定的可学习位置嵌入（我们称之为`object queries`）作为输入，并额外关注编码器的输出。我们将解码器的每个输出嵌入传递至共享前馈网络（FFN），该网络预测检测结果（类别与边界框）或“`no object`”类别。

Transformer decoder. 解码器遵循标准的Transformer架构，通过多头自注意力及编码器-解码器注意力机制，将尺寸为 d 的 N 嵌入向量进行转换。与原始Transformer的不同之处在于，我们的模型在每一解码层并行解码 N 对象，而Vaswani等人[47]采用自回归模型逐个元素预测输出序列。不熟悉相关概念的读者可参阅补充材料。由于解码器同样具有置换不变性， N 输入嵌入必须有所区分以产生不同结果。这些输入嵌入是我们称为`object queries`的可学习位置编码，与编码器类似，我们将其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，随后通过前馈网络（下小节详述）解码为边界框坐标和类别标签，最终得到 N 预测结果。模型通过对这些嵌入向量施加自注意力及编码器-解码器注意力，利用对象间的成对关系全局推理所有对象，同时能将整幅图像作为上下文信息加以利用。

Prediction feed-forward networks (FFNs). 最终预测通过一个3层感知机计算得出，该感知机采用ReLU激活函数，隐藏维度为 d ，并包含一个线性投影层。前馈网络(FFN)预测的是相对于输入图像的归一化边界框中心坐标、高度和宽度，而线性层则通过softmax函数预测类别标签。由于我们预测的是固定大小的 N 个边界框集合，其中 N 通常远大于图像中实际感兴趣对象的数量，因此额外引入了一个特殊类别标签 \emptyset ，用于表示该位置未检测到任何对象。这一类别在标准目标检测方法中扮演着类似于“背景”类别的角色。

Auxiliary decoding losses. 我们发现，在训练过程中于解码器使用辅助损失[1]颇为有益，尤其有助于模型输出正确的 $\{v^*\}$ 数值

每个类别的对象数量。我们在每个解码器层后添加预测前馈网络（FFNs）和匈牙利损失。所有预测FFNs共享它们的参数。我们使用一个额外的共享层归一化来标准化来自不同解码器层的预测FFNs输入。

4 Experiments

我们展示了DETR在COCO数据集上的定量评估中与Faster R-CNN相比取得了具有竞争力的结果。随后，我们对架构和损失函数进行了详细的消融研究，并提供了深入的见解和定性分析结果。最后，为了证明DETR是一个多功能且可扩展的模型，我们展示了全景分割的实验结果，仅需在固定DETR模型上进行小规模扩展训练即可实现。我们提供了代码和预训练模型以便复现实验，详情见 $\{v^*\}$ 。

Dataset. 我们在COCO 2017检测与全景分割数据集[24,18]上开展实验，该数据集包含11.8万张训练图像和5千张验证图像。每张图像均标注了边界框和全景分割信息。训练集中平均每张图像包含7个实例，单张图像最多可达63个实例，且同一图像中的实例尺寸从小型到大型不等。若无特别说明，我们采用边界框平均精度（bbox AP）作为评估指标，即多阈值下的综合度量。与Faster R-CNN对比时，我们报告最终训练周期后的验证集AP值；进行消融实验时，则取最后10个周期验证结果的中位数。

Technical details. 我们使用AdamW[26]训练DETR，将初始transformer的学习率设为 10^{-4} ，主干网络的学习率设为 10^{-5} ，权重衰减设为 10^{-4} 。所有transformer权重采用Xavier初始化[11]，主干网络则采用TORCHVISION提供的ImageNet预训练ResNet模型[15]，并冻结批归一化层。我们报告了两种不同主干网络的结果：ResNet-50和ResNet-101，对应模型分别称为DETR和DETR-R101。遵循[21]的方法，我们通过在主干网络最后阶段添加膨胀卷积并移除该阶段首个卷积的步长来提升特征分辨率，对应模型分别命名为DETR-DC5和DETR-DC5-R101（膨胀C5阶段）。这一修改使分辨率提升两倍，从而改善小物体检测性能，但会导致编码器自注意力计算成本增加16倍，总体计算量上升2倍。表1完整列出了这些模型与Faster R-CNN的FLOPs对比。

我们采用尺度增强技术，将输入图像调整至最短边至少480像素、最长边不超过1333像素（上限800像素）[50]。为了通过编码器的自注意力机制促进全局关系的学习，训练过程中还应用了随机裁剪增强，使性能提升约1 AP。具体而言，训练图像有50%的概率被随机裁剪为矩形区域，随后再次调整至800-1333像素范围。Transformer模型训练时采用默认0.1的dropout率。在推理阶段

表1：与基于ResNet-50和ResNet-101骨干网络的Faster R-CNN在COCO验证集上的性能对比。顶部区块展示了Detectron2[50]中Faster R-CNN模型的性能结果，中部区块则展示了采用GIoU[38]、随机裁剪训练时数据增强及长周期 $9\times$ 训练计划的Faster R-CNN模型结果。DETR模型与经过深度调优的Faster R-CNN基线取得了相当的结果，虽AP_S略低，但AP_L显著提升。我们采用torchscript封装的Faster R-CNN和DETR模型来测算FLOPs与FPS。未标注R101的名称对应ResNet-50架构。

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

在训练过程中，部分预测槽会输出空类别。为了优化平均精度（AP），我们采用置信度第二高的类别覆盖这些槽的预测结果。相比直接过滤空槽，该方法可使AP提升2个百分点。其余训练超参数详见章节A.4。消融实验采用300轮训练计划，前200轮结束后学习率下降为十分之一，每轮训练指完整遍历所有训练图像一次。基线模型在16块V100 GPU上训练300轮（每GPU处理4张图像，总批次大小为64）耗时3天。与Faster R-CNN对比时采用的延长训练计划为500轮，学习率在400轮后下降，该方案较短期训练可额外提升1.5 AP。

4.1 Comparison with Faster R-CNN

Transformer模型通常采用Adam或Adagrad优化器进行训练，配合极长的训练周期和dropout技术，DETR也不例外。而Faster R-CNN则使用SGD优化器进行训练，仅采用最小限度的数据增强，目前尚未见Adam优化器或dropout技术在其上的成功应用案例。尽管存在这些差异，我们仍尝试强化Faster R-CNN的基线性能。为使其与DETR对齐，我们在边界框损失函数中引入广义交并比[38]，采用相同的随机裁剪增强策略，并延长训练周期——这些改进措施已被证实能提升性能[13]。实验结果如表1所示：顶部区块展示的是Detectron2模型库[50]中采用 $3\times$ 训练方案的Faster R-CNN结果；中间区块（标记为+）则呈现相同模型在改进训练策略后的性能表现。

表2：编码器大小的影响。每行对应一个编码器层数不同但解码器层数固定的模型。随着编码器层数的增加，性能逐渐提升。

#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

采用9x调度（109个训练周期）及前述增强措施后，整体性能提升了1-2个AP。表1最后部分展示了多种DETR模型的结果。为保持参数量可比性，我们选择了具有6层宽度为256的Transformer编码器和6层解码器（含8个注意力头）的模型。与带FPN的Faster R-CNN类似，该模型总参数量为4130万，其中2350万来自ResNet-50，1780万来自Transformer模块。尽管Faster R-CNN和DETR通过更长时间训练仍有提升空间，但可以得出结论：在参数量相同的情况下，DETR在COCO验证集子集上达到42 AP，已具备与Faster R-CNN竞争的实力。DETR通过显著提升AP_L (+7.8)实现这一目标，但需注意该模型在AP_S (-5.5)指标上仍存在差距。具有相同参数量和相近FLOPs的DETR-DC5虽然获得了更高AP，但在AP_S指标上也明显落后。采用ResNet-101骨干网络时，Faster R-CNN与DETR同样表现出可比性能。

4.2 Ablations

Transformer解码器中的注意力机制是建模不同检测特征表示间关系的核心组件。在我们的消融分析中，我们探究了架构其他组成部分及损失函数对最终性能的影响。本研究选用基于ResNet-50的DETR模型，配置为6层编码器、6层解码器及256宽度。该模型参数量达4130万，在短周期和长周期训练方案下分别取得40.6和42.0的平均精度（AP），并以28帧/秒的速度运行，与同骨干网络的Faster R-CNN-FPN性能相当。

Number of encoder layers. 我们通过改变编码器层数（表2）来评估全局图像级自注意力机制的重要性。在没有编码器层的情况下，整体AP下降了3.9个点，其中大尺寸物体的AP下降更为显著，达到6.0。我们假设，通过利用全局场景推理，编码器在解耦物体方面起着关键作用。在图3中，我们可视化了一个训练好的模型最后一层编码器的注意力图，聚焦于图像中的若干点。编码器似乎已经能够分离实例，这可能简化了解码器对物体的提取和定位过程。

Number of decoder layers. 我们在每个解码层之后应用辅助损失（见第3.2节），因此，预测前馈网络（FFNs）在设计上就被训练用于预

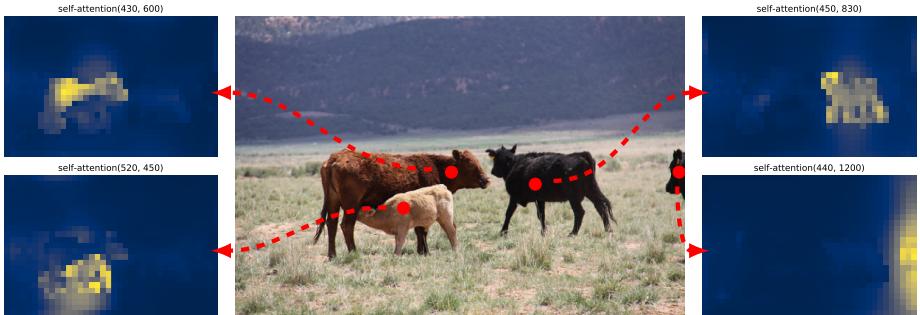


图3：针对一组参考点的编码器自注意力机制。编码器能够区分各个独立实例。预测结果基于基线DETR模型在验证集图像上得出。

从每个解码器层的输出中提取字典对象。我们通过评估解码各阶段预测出的对象，来分析每个解码器层的重要性（图4）。AP和 AP_{50} 在每一层后都有提升，从第一层到最后一层总共实现了非常显著的+8.2/9.5 AP提升。得益于其基于集合的损失函数，DETR在设计上无需NMS处理。为验证这一点，我们在每个解码器输出后使用默认参数[50]运行标准NMS流程。NMS对首个解码器的预测结果有性能提升，这是因为Transformer的单个解码层无法计算输出元素间的互相关性，容易对同一目标产生重复预测。在第二层及后续层中，通过激活值的自注意力机制，模型能够抑制重复预测。我们观察到NMS带来的改进随网络深度增加而减弱。在最后几层中，由于NMS错误移除了真实正例预测，AP会出现小幅下降。

与可视化编码器注意力类似，我们在图6中对解码器注意力进行了可视化，用不同颜色为每个预测对象的注意力图上色。我们观察到解码器注意力相当局部化，这意味着它主要关注对象的 extremities（如头部或腿部）。我们假设，在编码器通过全局注意力分离实例后，解码器只需关注这些 extremities 即可提取类别和对象边界。

Importance of FFN. Transformer中的FFN可视为 1×1 卷积层，这使得编码器类似于注意力增强的卷积网络[3]。我们尝试完全移除它，仅保留transformer层中的注意力机制。通过将网络参数从4130万减少至2870万，transformer部分仅剩1080万参数时，性能下降了2.3个AP点，因此我们得出结论：FFN对取得良好结果至关重要。

Importance of positional encodings. 我们的模型中包含两种位置编码：空间位置编码和输出位置编码

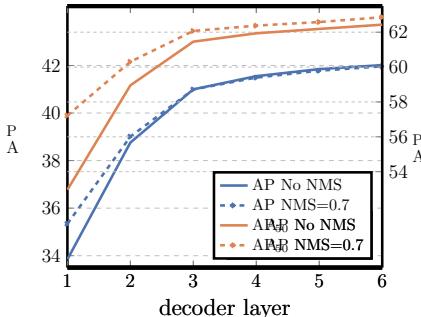


图4：各解码器层后的AP与AP₅₀性能表现。评估的是一个单一长周期基线模型。DETR通过设计无需NMS处理，该图验证了这一点。NMS在最终层会降低AP，因为移除了真实正例预测，但在初始解码器层能提升AP，通过消除重复预测（由于首层缺乏信息交互），并轻微改善AP₅₀。



图5：稀有类别的分布外泛化能力。尽管训练集中没有任何一张图像包含超过13只长颈鹿，DETR却能毫无困难地泛化至24只甚至更多同类别实例。

我们尝试了固定编码与学习编码的各种组合方式，实验结果见表3。输出位置编码是必需的且不可移除，因此我们尝试了两种方案：要么在解码器输入端一次性传入位置编码，要么在每个解码器注意力层中将其添加到查询向量中。在首个实验中，我们完全移除了空间位置编码，仅在输入端传入输出位置编码。有趣的是，模型仍能保持32以上的AP值，仅比基线低7.8 AP。随后，我们按照原始Transformer[47]的做法，在输入端一次性传入固定的正弦空间位置编码和输出编码，发现这种方式相比直接在注意力机制中传入位置编码会导致1.4 AP的下降。而传入学习得到的空间位置编码至注意力机制则获得相近结果。令人惊讶的是，当仅在编码器中完全不使用任何空间编码时，AP值仅出现1.3的小幅下降。当我们把编码传入注意力机制时，这些编码会在所有层间共享，而输出编码（对对象查询）始终采用学习获得的方式。

通过这些消融实验，我们得出结论：transformer组件——编码器中的全局自注意力机制、FFN、多层解码器以及位置编码，均对最终的目标检测性能有显著贡献。

Loss ablations. 为了评估匹配成本和损失函数中不同组件的重要性，我们训练了多个模型，逐一开启和关闭这些组件。损失函数包含三个组成部分：分类损失、 ℓ_1 边界框距离损失以及GIoU[38]损失。分类损失是训练过程中不可或缺的，无法关闭，因此我们分别训练了不含边界框距离损失的模型和不含GIoU损失的模型，并与同时使用三种损失的基线模型进行对比。结果如表4所示。单独使用GIoU损失时

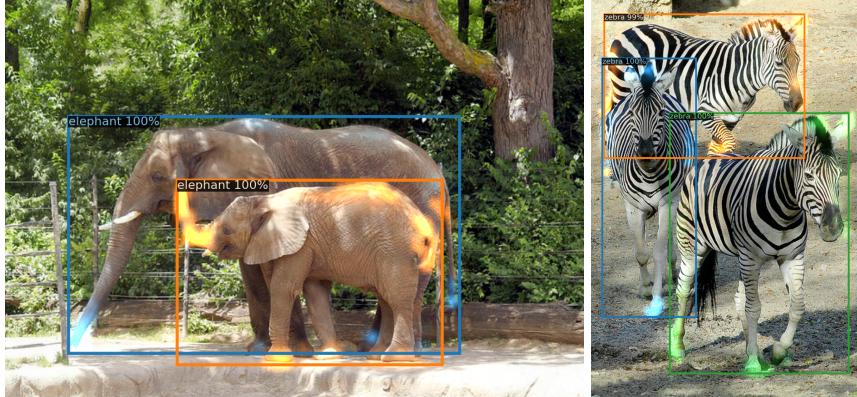


图6：可视化每个预测对象的解码器注意力（图像来自COCO val数据集）。预测由DET R-DC5模型生成。不同对象的注意力分数以不同颜色编码。解码器通常关注对象的 extremities，如腿部和头部。建议彩色查看以获得最佳效果。

表3：不同位置编码与基线（最后一行）的对比结果，基线在编码器和解码器的每个注意力层均采用固定的正弦位置编码。所有层共享学习得到的嵌入向量。不使用空间位置编码会导致AP显著下降。有趣的是，仅在解码器中传递这些编码仅引起AP轻微下降。所有这些模型均采用学习得到的输出位置编码。

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

表4：损失组件对AP的影响。我们训练了两个模型，分别关闭 ℓ_1 损失和GIoU损失，观察到 ℓ_1 单独使用时效果不佳，但与GIoU结合后能提升AP_M和AP_L。我们的基线模型（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

在大多数模型性能指标上，仅比采用组合损失的基准低0.7 AP。使用不含GIoU的 ℓ_1 方案则表现不佳。我们仅研究了

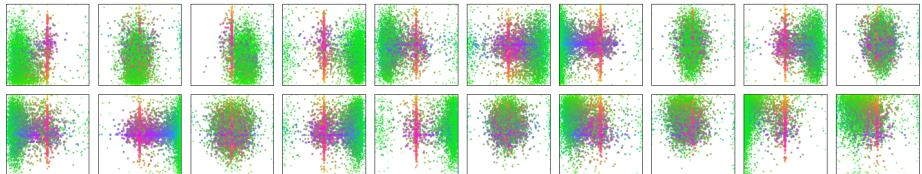


图7：在DETR解码器的 $N=8$ 个预测槽位中，展示了20个槽位对COCO 2017验证集所有图像的边界框预测可视化结果。每个预测框以其中心坐标表示，归一化至 1×1 正方形范围内（根据各图像尺寸调整）。点按颜色编码：绿色对应小尺寸框，红色代表大尺寸横向框，蓝色表示大尺寸纵向框。观察发现，各槽位会针对特定区域和框尺寸发展出多种预测模式。值得注意的是，几乎所有槽位都具备预测图像全局大尺寸框的模式，这在COCO数据集中较为常见。

对不同的损失进行简单的消融实验（每次使用相同的权重），但采用其他组合方式可能会得到不同的结果。

4.3 Analysis

Decoder output slot analysis 在图7中，我们可视化了不同查询槽针对COCO 2017验证集所有图像预测的边界框。DETR模型为每个查询槽习得了不同的专长。我们观察到，各槽位存在多种运作模式，分别关注不同区域和框体尺寸。值得注意的是，所有槽位均具备预测图像全局边界框的模式（在图中体现为中间对齐的红点）。我们推测这一现象与COCO数据集中物体的分布特性有关。

Generalization to unseen numbers of instances. COCO中的某些类别在同一图像中缺乏大量同类实例的充分体现。例如，训练集中没有一张图像包含超过13只长颈鹿。为此，我们合成了一张³图像以验证DETR的泛化能力（见图5）。我们的模型成功检测出图中全部24只长颈鹿，这显然超出了数据分布范围。该实验证实了每个对象查询并不存在强烈的类别专一性。

4.4 DETR for panoptic segmentation

全景分割[19]最近引起了计算机视觉社区的广泛关注。类似于将Faster R-CNN[37]扩展为Mask R-CNN[14]的过程，DETR只需在解码器输出之上添加一个掩码头即可自然延伸。本节我们将展示，通过统一处理背景类 $\{v^*\}$ 与目标类，该头部可用于生成全景分割[19]结果。

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

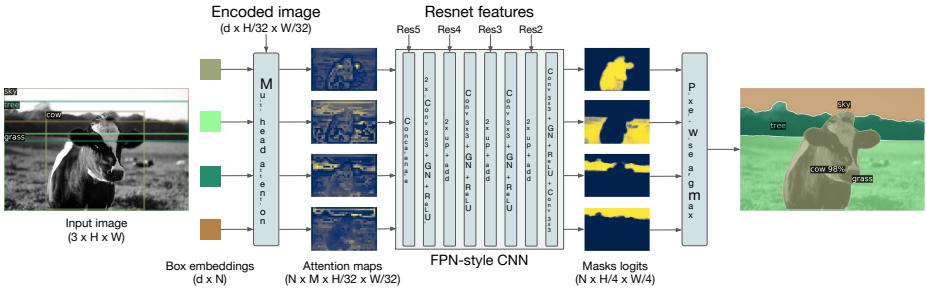


图8：全景分割头的示意图。为每个检测到的对象并行生成一个二值掩码，然后使用像素级的argmax操作合并这些掩码。



图9：DETR-R101生成的全景分割定性结果。DETR以统一的方式为物体（things）和背景（stuff）生成对齐的掩模预测，公式标记 $\{v^*\}$ 保持不变。

以统一的方式。我们在COCO数据集的全景标注上进行实验，该数据集除了包含80个物品类别外，还有53个材料类别。

我们采用相同的配方训练DETR模型，使其在COCO数据集上预测围绕`stuff`和`things`类别的边界框。由于匈牙利匹配是通过计算边界框之间的距离来实现的，预测边界框是训练得以进行的必要条件。我们还增加了一个掩码头，用于为每个预测框生成二值掩码，如图8所示。该模块以Transformer解码器输出的每个对象特征作为输入，通过多头部（含 M 个头）注意力机制计算这些特征嵌入与编码器输出之间的注意力分数，从而为每个对象生成低分辨率的 M 个注意力热图。为生成最终预测并提升分辨率，我们采用了类FPN架构。更多架构细节将在补充材料中详述。最终输出的掩码分辨率为步长4，每个掩码通过DICE/F-1损失[28]和Focal损失[23]进行独立监督。

掩码头部可以采用联合训练的方式，也可以分两步进行：首先仅针对边界框训练DETR模型，随后冻结所有权重，单独训练掩码头部25个周期。实验表明，这两种方法效果相近，我们采用后一种方法报告结果，因其所需总训练时间更短。

表5：与当前最先进方法UPSNNet[51]和Panoptic FPN[18]在COCO val数据集上的比较 为了公平对比，我们采用与DETR相同的数据增强策略重新训练PanopticFPN，训练周期为18倍。UPSNNet采用1x周期方案，UPSNNet-M则是包含多尺度测试时增强的版本。

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
UPSNNet	R50	42.5	78.0	52.5	48.6	79.4	59.6	33.4	75.9	41.7	34.3
UPSNNet-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结果，我们只需在每个像素点上对掩码分数进行argmax操作，并将对应的类别分配给生成的掩码。这一过程确保了最终掩码之间不存在重叠，因此DETR无需依赖常用于对齐不同掩码的启发式方法[19]。

Training details. 我们按照边界框检测的配方训练了DETR、DETR-DC5和DETR-R101模型，以预测COCO数据集中物品与物体类别周围的框。新的掩码头训练了25个周期（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，然后计算逐像素的argmax以确定每个像素属于哪个掩码。接着，我们将同一物品类别的不同掩码预测合并为一个，并过滤掉空掩码（少于4个像素的情况）。

Main results. 定性结果如图9所示。在表5中，我们将统一全景分割方法与几种对物体和背景采用不同处理的现有方法进行对比。我们报告了全景质量(PQ)及其在物体(PQth)和背景(PQst)上的细分指标。同时汇报了掩码AP值（基于物体类别计算），该指标未经过任何全景后处理（本研究中指未进行像素级argmax操作前的结果）。实验表明，DETR在COCO-val 2017数据集上超越了已发表成果，也优于我们采用相同数据增强策略训练的PanopticFPN基线模型（确保公平对比）。细分数据显示DETR在背景类别上优势尤为显著，我们推测编码器注意力机制实现的全局推理能力是取得该结果的关键因素。在物体类别方面，尽管掩码AP计算中较基线模型存在高达8 mAP的显著差距，DETR仍获得了具有竞争力的PQth指标。我们在COCO测试集上进一步评估本方法，取得了46 PQ的成绩。期待本研究能为未来探索完全统一的全景分割模型提供启发。

5 Conclusion

我们提出了DETR，这是一种基于Transformer和二分图匹配损失的全新物体检测系统设计，用于直接集合预测。该方法在具有挑战性的COCO数据集上取得了与优化后的Faster R-CNN基线相当的结果。DETR实现简单，架构灵活，可轻松扩展至全景分割任务，并取得有竞争力的成绩。此外，得益于自注意力机制对全局信息的处理能力，DETR在大物体检测上显著优于Faster R-CNN。

这一新型检测器设计也带来了新的挑战，尤其是在小目标训练、优化与性能方面。现有检测器耗数年改进才得以解决类似问题，我们期待未来研究能成功为DETR应对这些挑战。

6 Acknowledgements

我们感谢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巴黎团队的讨论和建议，没有这些，这项工作将无法完成。

References

1. Al-Rfou, R., Choe, D., Constant, N., Guo, M., Jones, L.: 基于深层自注意力的字符级语言建模。见：AAAI人工智能会议（2019）
2. Bahdanau, D., Cho, K., Bengio, Y.: 通过联合学习对齐与翻译的神经机器翻译。见：ICLR（2015）
3. Bello, I., Zoph, B., Vaswani, A., Shlens, J., Le, Q.V.: 注意力增强的卷积网络。见：ICCV（2019）
4. Bodla, N., Singh, B., Chellappa, R., Davis, L.S.: Soft-NMS：一行代码提升目标检测性能。见：ICCV（2017）
5. Cai, Z., Vasconcelos, N.: Cascade R-CNN：高质量目标检测与实例分割。PAMI（2019）
6. Chan, W., Saharia, C., Hinton, G., Norouzi, M., Jaitly, N.: Imputer：基于插值与动态规划的序列建模。arXiv:2002.08926（2020）
7. Cordonnier, J.B., Loukas, A., Jaggi, M.: 论自注意力与卷积层的关系。见：ICLR（2020）
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT：面向语言理解的深度双向Transformer预训练。见：NAACL-HLT（2019）
9. Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: 基于深度神经网络的可扩展目标检测。见：CVPR（2014）
10. Ghazvininejad, M., Levy, O., Liu, Y., Zettlemoyer, L.: Mask-predict：条件掩码语言模型的并行解码。arXiv:1904.09324（2019）
11. Glorot, X., Bengio, Y.: 深度前馈神经网络训练难度的理解。见：AISTATS（2010）

12. 顾坚、Bradbury、熊超、李伟强、Socher: 《非自回归神经机器翻译》, ICLR (2018)
13. 何恺明、Girshick、Dollár: 《重新思考ImageNet预训练》, ICCV (2019)
14. 何恺明、Gkioxari、Dollár、Girshick: 《Mask R-CNN》, ICCV (2017)
15. 何恺明、张翔、任少卿、孙剑: 《深度残差学习在图像识别中的应用》, CVPR (2016)
16. Hosang、Benenson、Schiele: 《学习非极大值抑制》, CVPR (2017)
17. 胡瀚、顾坚、张兆、代季峰、魏云超: 《用于目标检测的关系网络》, CVPR (2018)
18. Kirillov、Girshick、何恺明、Dollár: 《全景特征金字塔网络》, CVPR (2019)
19. Kirillov、何恺明、Girshick、Rother、Dollar: 《全景分割》, CVPR (2019)
20. Kuhn: 《匈牙利方法解决分配问题》(1955)
21. 李益、齐浩、代季峰、纪煦、魏云超: 《全卷积实例感知语义分割》, CVPR (2017)
22. 林腾宇、Dollár、Girshick、何恺明、Hariharan、Belongie: 《用于目标检测的特征金字塔网络》, CVPR (2017)
23. 林腾宇、Goyal、Girshick、何恺明、Dollár: 《密集目标检测的焦点损失》, ICCV (2017)
24. 林腾宇、Maire、Belongie、Hays、Perona、Ramanan、Dollár、Zitnick: 《Microsoft COCO: 上下文中的常见物体》, ECCV (2014)
25. 刘伟、Anguelov、Erhan、Szegedy、Reed、傅城燕、Berg: 《SS D: 单次多框检测器》, ECCV (2016)
26. Loshchilov、Hutter: 《解耦权重衰减正则化》, ICLR (2017)
27. Lüscher、Beck、Irie、Kitza、Michel、Zeyer、Schlüter、Ney: 《RWTH ASR系统在LibriSpeech上的表现: 混合模型与注意力机制——无数据增强》, arXiv:1905.03072 (2019)
28. Milletari、Navab、Ahmadi: 《V-Net: 全卷积神经网络用于体积医学图像分割》, 3DV (2016)
29. Oord、李益、Babuschkin、Simonyan、Vinyals、Kavukcuoglu、Driessche、Lockhart、Cobo、Stimberg等: 《并行WaveNet: 快速高保真语音合成》, arXiv:1711.10433 (2017)
30. Park、Berg: 《学习分解以实现目标检测和实例分割》, arXiv:1511.06449 (2015)
31. Parmar、Vaswani、Uszkoreit、Kaiser、Shazeer、Ku、Tran: 《图像变换器》, ICML (2018)
32. Paszke、Gross、Massa、Lerer、Bradbury、Chanan、Killeen、林志、Gimelshein、Antiga等: 《PyTorch: 一种命令式风格的高性能深度学习库》, NeurIPS (2019)
33. Pineda、Salvador、Drozdzal、Romero: 《阐明图像到集合预测: 模型、损失和数据集分析》, arXiv:1904.05709 (2019)
34. Radford、吴俊、Child、Luan、Amodei、Sutskever: 《语言模型是无监督多任务学习者》 (2019)
35. Redmon、Divvala、Girshick、Farhadi: 《你只需看一次: 统一实时目标检测》, CVPR (2016)
36. 任勉、Zemel: 《端到端实例分割与循环注意力》, CVPR (2017)

37. 任少卿, 何恺明, Girshick R.B., 孙剑: Faster R-CNN: 基于区域提议网络的实时目标检测。PAMI (2015) 38. Rezatofighi H., Tsoi N., Gwak J., Sadeghian A., Reid I., Savarese S.: 广义交并比。载于: CVPR (2019) 39. Rezatofighi S.H., Kaskman R., Motlagh F.T., Shi Q., Cremers D., Leal-Taixé L., Reid I.: 深度排列集网络: 利用深度神经网络预测未知排列与基数的集合。arXiv:1805.00613 (2018) 40. Rezatofighi S.H., Milan A., Abbasnejad E., Dick A., Reid I., Kaskman R., Cremers D., Leal-Taixé L.: 深度集合网络: 用深度神经网络预测集合。载于: ICCV (2017) 41. Romera-Paredes B., Torr P.H.S.: 循环实例分割。载于: ECCV (2015) 42. Salvador A., Bellver M., Baradad M., Marqués F., Torres J., Giró X.: 用于语义实例分割的循环神经网络。arXiv:1712.00617 (2017) 43. Stewart R.J., Andriluka M., Ng A.Y.: 拥挤场景中的端到端行人检测。载于: CVPR (2015) 44. Sutskever I., Vinyals O., Le Q.V.: 基于神经网络的序列到序列学习。载于: NeurIPS (2014) 45. Sønnaeve G., Xu Q., Kahn J., Grave E., Likhomolenko T., Pratap V., Sriram A., Liptchinsky V., Collobert R.: 端到端语音识别: 从监督学习到现代架构的半监督学习。arXiv:1911.08460 (2019) 46. 田子德, 沈春华, 陈海强, 何涛: FCOS: 全卷积单阶段目标检测。载于: IC CV (2019) 47. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I.: 注意力机制就是你所需要的一切。载于: NeurIPS (2017) 48. Vinyals O., Bengio S., Kudlur M.: 顺序至关重要: 集合的序列到序列学习。载于: ICLR (2016) 49. 王翔, Girshick R.B., Gupta A., 何恺明: 非局部神经网络。载于: 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: 统一全景分割网络。载于: CVPR (2019) 52. 张帅, 迟聪聪, 姚远, 雷震, 李世鹏: 通过自适应训练样本选择弥合基于锚点与无锚点检测的差距。arXiv:1912.02424 (2019) 53. Zhou X., Wang D., Krähenbühl P.: 以点为对象。arXiv:1904.07850 (2019)

A Appendix

A.1 Preliminaries: Multi-head attention layers

由于我们的模型基于Transformer架构，为求全面性，在此重述我们所采用注意力机制的一般形式。该注意力机制遵循文献[47]，仅位置编码细节（见公式8）沿用了文献[7]的方案。

Multi-head multi-head attention 的通用形式，具有 M 个头，每个头的维度为 d' ，是一个具有以下签名的函数（使用 $d' = \frac{d}{M}$ ，并在下括号中给出矩阵/张量的大小）

$$\text{mh-attn} : \underbrace{X_q}_{d \times N_q}, \underbrace{X_{kv}}_{d \times N_{kv}}, \underbrace{T}_{M \times 3 \times d' \times d}, \underbrace{L}_{d \times d} \mapsto \underbrace{\tilde{X}_q}_{d \times N_q} \quad (3)$$

其中 X_q 是长度为 *query sequence* 的 N_q ， X_{kv} 是长度为 N_{kv} （且通道数相同（为简化说明记为 d ）的 *key-value sequence*， T 是用于计算所谓查询、键和值嵌入的权重张量， L 为投影矩阵。输出与查询序列大小相同。在深入细节前明确术语，多头self-注意力（mh-s-attn）是 $X_q = X_{kv}$ 的特例，即

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

多头注意力机制简单来说，就是将 M 个独立的注意力头进行拼接后，通过 L 进行投影变换。通常的做法[47]是结合残差连接、dropout以及层归一化技术。换言之，用 $\tilde{X}_q = \text{layernorm}(X_q + \text{dropout}(LX'_q))$ 表示 $\text{mh-attn}(X_q, X_{kv}, T, L)$ ， $\tilde{X}^{(q)}$ 代表各注意力头的拼接结果，我们得到

$$X'_q = [\text{attn}(X_q, X_{kv}, T_1); \dots; \text{attn}(X_q, X_{kv}, T_M)] \quad (5)$$

$$\tilde{X}_q = \text{layernorm}(X_q + \text{dropout}(LX'_q)), \quad (6)$$

其中 $[;]$ 表示在通道轴上的拼接。

Single head 一个带有权重张量 $T' \in \mathbb{R}^{3 \times d' \times d}$ 的注意力头，记作 $\text{attn}(X_q, X_{kv}, T')$ ，依赖于额外的位置编码 $P_q \in \mathbb{R}^{d \times N_q}$ 和 $P_{kv} \in \mathbb{R}^{d \times N_{kv}}$ 。它首先在添加查询和键的位置编码后，计算所谓的查询、键和值嵌入[7]：

$$[Q; K; V] = [T'_1(X_q + P_q); T'_2(X_{kv} + P_{kv}); T'_3 X_{kv}] \quad (7)$$

其中 T' 是 T'_1, T'_2, T'_3 的拼接。随后，*attention weights* α 基于查询与键的点积的 softmax 计算得出，使得查询序列的每个元素都能关注到键值序列的所有元素（ i 为查询索引， j 为键值索引）：

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

在我们的案例中，位置编码可以是学习得到或固定的，但对于给定的查询/键值序列，它们会在所有注意力层间共享，因此我们并未将其显式地写作注意力机制的参数。在描述编码器与解码器时，我们会详述其具体取值。最终输出是由注意力权重加权的值聚合而成：第 i 行由 $\text{attn}_i(X_q, X_{kv}, T') = \sum_{j=1}^{N_{kv}} \alpha_{i,j} V_j$ 给出。

Feed-forward network (FFN) layers 原始Transformer模型交替使用多头注意力层和所谓的FFN层[47]，后者实质上是多层1x1卷积，在我们的设定中具有 Md 个输入和输出通道。我们所考虑的FFN由两层带ReLU激活的1x1卷积构成，与方程6类似，这两层之后同样包含残差连接/丢弃/层归一化操作。

A.2 Losses

为了完整性，我们详细介绍了本方法中采用的损失函数。所有损失均按批次内的对象数量进行归一化处理。在分布式训练中需格外注意：由于每个GPU接收的是子批次，仅对本地批次中的对象数量进行归一化是不够的，因为子批次在各GPU间通常并不均衡。关键在于依据所有子批次中对象的总数进行归一化。

Box loss 与[41,36]类似，我们在损失函数中采用了软交并比 (IoU) 的变体，并结合了针对 \hat{b} 的 ℓ_1 损失：

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

其中 $\lambda_{\text{iou}}, \lambda_{\text{L1}} \in \mathbb{R}$ 为超参数， $\mathcal{L}_{\text{iou}}(\cdot)$ 是广义IoU[38]：

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

|.| “面积”指的是区域，而边界框坐标的并集和交集被用作边界框本身的简写。并集或交集的面积通过计算 $b_{\sigma(i)}$ 和 \hat{b}_i 线性函数的min / max得出，这使得损失函数足够适用于随机梯度下降。 $B(b_{\sigma(i)}, \hat{b}_i)$ 表示包含 $b_{\sigma(i)}, \hat{b}_i$ (的最大边界框，涉及 B 的面积同样基于边界框坐标线性函数的min / max计算得出)。

DICE/F-1 loss [28] DICE系数与交并比 (Intersection over Union) 密切相关。若以 \hat{m} 表示模型的原始掩膜逻辑预测值， m 代表二值目标掩膜，则损失函数定义为：

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

其中 σ 为sigmoid函数。该损失通过物体数量进行了归一化处理。

A.3 Detailed architecture

DETR中采用的Transformer详细描述如图10所示，其中每个注意力层均传递了位置编码。来自CNN骨干网络的图像特征与空间位置编码一同输入Transformer编码器，这些位置编码会在每个多头自注意力层被添加到查询和键中。随后，解码器接收初始设为零的查询、输出位置编码（即对象查询）以及编码器记忆，并通过多个多头自注意力机制和解码器-编码器注意力层，最终生成预测的类别标签和边界框集合。第一层解码器中的首个自注意力层可被跳过。

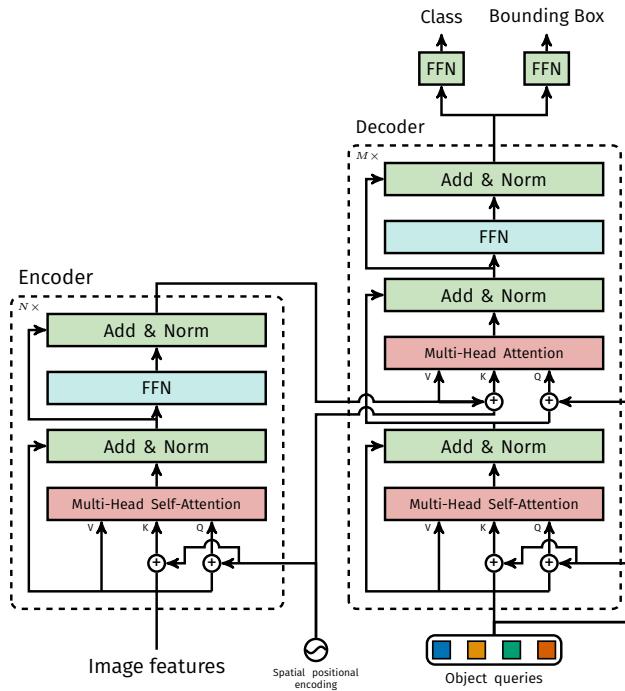


图10：DETR的transformer架构。详情请参阅A.3节。

Computational complexity 编码器中的每个自注意力机制的复杂度为 $\mathcal{O}(d^2 HW + d(HW)^2)$ ： $\mathcal{O}(d'd)$ 是计算单个查询/键/值嵌入（及 $Md' = d$ ）的成本，而 $\mathcal{O}(d'(HW)^2)$ 则是计算一个注意力头的注意力权重所需的开销。其余计算可忽略不计。在解码器中，每个自注意力机制处于 $\mathcal{O}(d^2 N + dN^2)$ 复杂度，而编码器与解码器之间的交叉注意力则处于 $\mathcal{O}(d^2(N + HW) + dNHW)$ ，由于实际中 $N \ll HW$ 的存在，这一开销远低于编码器。

FLOPS computation 鉴于Faster R-CNN的浮点运算次数(FLOPS)取决于图像中的提议数量，我们报告了COCO 2017验证集前100张图像的平均FLOPS值。我们使用Detectron2[50]中的工具flop_count_operators来计算FLOPS。对于Detectron2模型，我们直接使用该工具不作修改；而对于DETR模型，我们则扩展了该工具以考虑批量矩阵乘法(bmm)的影响。

A.4 Training hyperparameters

我们使用AdamW[26]训练DETR，改进了权重衰减处理，设置为 10^{-4} 。同时应用梯度裁剪，最大梯度范数为0.1。我们对主干网络和变换器的处理略有不同，接下来将详细讨论两者的具体细节。

Backbone ImageNet预训练的主干网络ResNet-50从Torchvision导入，并移除了最后的分类层。遵循目标检测领域的广泛实践，训练过程中主干网络的批归一化权重与统计量保持冻结。我们采用 10^{-5} 的学习率对主干网络进行微调。实验发现，将主干网络的学习率设置为比网络其余部分大约低一个数量级，对稳定训练过程至关重要，尤其是在最初的几个训练周期中。

Transformer 我们以 10^{-4} 的学习率训练该transformer模型。在每层多头注意力机制和前馈网络之后、层归一化之前，应用0.1的加性dropout。权重采用Xavier初始化方法进行随机初始化。

Losses 我们采用 ℓ_1 与GIoU损失的线性组合进行边界框回归，权重分别为 $\lambda_{L1} = 5$ 和 $\lambda_{iou} = 2$ 。所有模型均使用 $N = 100$ 个解码器查询槽进行训练。

Baseline 我们增强版的Faster-RCNN+基线模型采用GIoU[38]损失函数与标准 ℓ_1 损失函数相结合的方式进行边界框回归。通过网格搜索确定了各损失函数的最佳权重，最终模型仅使用GIoU损失函数，其中边界框回归和提议回归任务的权重分别为20和1。基线模型采用与DETR相同的数据增强策略，并按照 $9 \times$ 训练计划（约109个周期）进行训练。其余所有设置均与Detectron2模型库[50]中的对应模型保持一致。

Spatial positional encoding 编码器激活与图像特征对应的空间位置相关联。在我们的模型中，我们采用固定的绝对编码来表示这些空间位置。我们将原始Transformer[47]的编码方法推广至二维情况[31]。具体而言，对于每个嵌入的两个空间坐标，我们独立使用 $\frac{d}{2}$ 不同频率的正弦和余弦函数，随后将它们拼接起来，得到最终的 d 通道位置编码。

A.5 Additional results

DETR-R101模型在全景预测方面的一些额外定性结果如图11所示。



(a) 重叠物体导致的失败案例。PanopticFPN完全漏检了一架飞机，而DETR则未能准确分割其中的3架。



(b) Things 掩码以全分辨率进行预测，相比PanopticFPN能够实现更清晰的边界

图11：全景预测结果对比。从左至右分别为：真实标注、基于ResNet 101的PanopticFPN、基于ResNet 101的DETR

Increasing the number of instances 根据设计，DETR无法预测超出其查询槽数量的物体，即我们实验中设定的100个。本节将分析DETR在接近这一极限时的表现。我们选取给定类别的标准方形图像，将其以 10×10 网格形式重复排列，并计算模型漏检的实例比例。为测试模型在少于100个实例时的表现，我们随机遮蔽部分单元格。这确保了无论可见物体数量如何，其绝对尺寸保持不变。考虑到遮蔽操作的随机性，我们使用不同掩码重复实验100次。结果如图12所示。各类别的表现相似：当可见实例不超过50个时，模型能检测到全部实例；但随着数量增加，模型开始饱和并漏检越来越多的实例。值得注意的是，当图像包含全部100个实例时，模型平均仅检测到30个——这甚至低于仅含50个实例（全部被检出）时的表现。这种反直觉现象很可能是因为图像及其检测结果严重偏离了训练数据分布。

需要注意的是，该测试本质上是对分布外泛化能力的检验，因为示例图像中单类别实例大量出现的情况极为罕见。实验中难以区分两种分布外泛化类型：图像本身的分布差异与每类别物体数量的分布差异。但由于COCO数据集中几乎不存在仅包含大量同类物体的图像，此类实验仍是我们理解查询物体是否过度拟合数据集标签及位置分布的最佳尝试。总体而言，实验结果表明模型并未对这些分布产生过拟合，因为在物体数量达到50个时仍能保持近乎完美的检测效果。

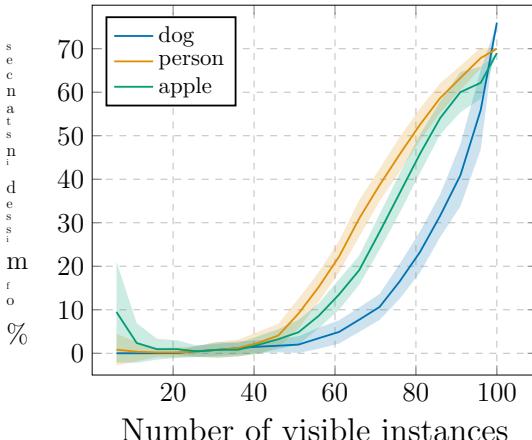


图12：分析DETR漏检各类别实例的数量与其在图像中出现数量的关系。我们报告了平均值和标准差。当实例数量接近100时，DETR开始趋于饱和，漏检对象逐渐增多

A.6 PyTorch inference code

为了展示该方法的简洁性，我们在代码清单1中提供了基于PyTorch和Torchvision库的推理代码。该代码运行环境为Python 3.6+、PyTorch 1.4及Torchvision 0.5。需注意，此代码不支持批处理，因此仅适用于单GPU单图像的推理或配合DistributedDataParallel进行训练。此外，为保持代码清晰，示例中编码器采用学习式位置编码而非固定式，且位置编码仅添加至输入层而非每个transformer层。若需调整这些设计，需深入修改PyTorch原生的transformer实现，这将影响代码可读性。完整实验复现代码将于会议前公开。

```

1 import torch
2 from torch import nn
3 from torchvision.models import resnet50
4
5 class DETR(nn.Module):
6
7     def __init__(self, num_classes, hidden_dim, nheads,
8                  num_encoder_layers, num_decoder_layers):
9         super().__init__()
10        # We take only convolutional layers from ResNet-50 model
11        self.backbone = nn.Sequential(*list(resnet50(pretrained=True).children())[:-2])
12        self.conv = nn.Conv2d(2048, hidden_dim, 1)
13        self.transformer = nn.Transformer(hidden_dim, nheads,
14                                         num_encoder_layers, num_decoder_layers)
15        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
16        self.linear_bbox = nn.Linear(hidden_dim, 4)
17        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
18        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
19        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
20
21    def forward(self, inputs):
22        x = self.backbone(inputs)
23        h = self.conv(x)
24        H, W = h.shape[-2:]
25        pos = torch.cat([
26            self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
27            self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
28        ], dim=-1).flatten(0, 1).unsqueeze(1)
29        h = self.transformer(pos + h.flatten(2).permute(2, 0, 1),
30                            self.query_pos.unsqueeze(1))
31        return self.linear_class(h), self.linear_bbox(h).sigmoid()
32
33 detr = DETR(num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6)
34 detr.eval()
35 inputs = torch.randn(1, 3, 800, 1200)
36 logits, bboxes = detr(inputs)

```

列表1：DETR PyTorch推理代码。为清晰起见，该代码在编码器中使用学习得到的位置编码而非固定编码，且位置编码仅添加至输入而非每个transformer层。这些改动需超越PyTorch原生的transformer实现，影响了代码可读性。完整实验复现代码将于会议前公开。