DETR: End-to-End Object DEtection with TRansformer

Nicolas Carion, Francisco Massa, et al.

Presented by Yiru Xiong

Abstract

Abstract. We present a new method that views object detection as a direct set prediction problem. Our approach streamlines the detection pipeline, effectively removing the need for many hand-designed components like a non-maximum suppression procedure or anchor generation that explicitly encode our prior knowledge about the task. The main ingredients of the new framework, called DEtection TRansformer or DETR, are a set-based global loss that forces unique predictions via bipartite matching, and a transformer encoder-decoder architecture. Given a fixed small set of learned object queries, DETR reasons about the relations of the objects and the global image context to directly output the final set of predictions in parallel. The new model is conceptually simple and does not require a specialized library, unlike many other modern detectors. DETR demonstrates accuracy and run-time performance on par with the well-established and highly-optimized Faster R-CNN baseline on the challenging COCO object detection dataset. Moreover, DETR can be easily generalized to produce panoptic segmentation in a unified manner. We show that it significantly outperforms competitive baselines. Training code and pretrained models are available at https://github.com/facebookresearch/detr.

Object Detection

Detection Transformer

Self-attention

Bipartite matching Loss

Parallel Decoding

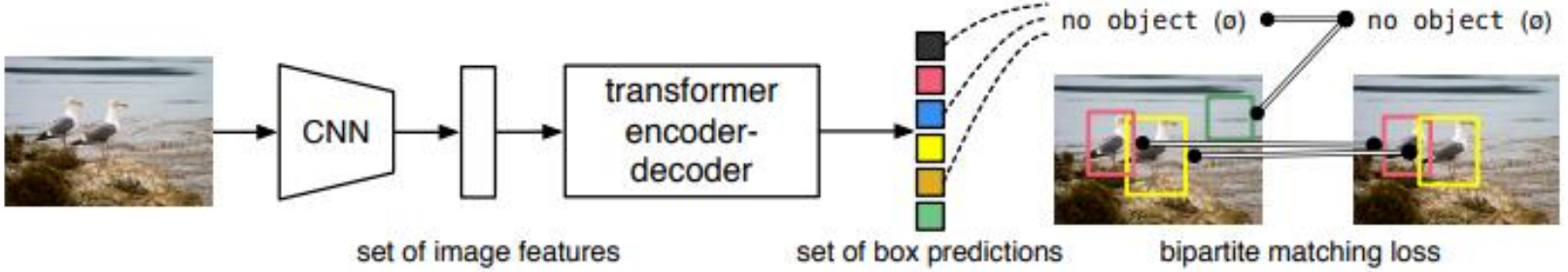
Set Predictions

Image Segmentation

Summary

- Introduce a new end-to-end object detection system based on CNN, transformers and bipartite matching loss for direct set predictions.
- Offer an end-to-end set prediction for object detection, eliminating the need for prior traditions like separate region proposals and post-processing steps
- Pioneer in its simplicity and directness in the architecture

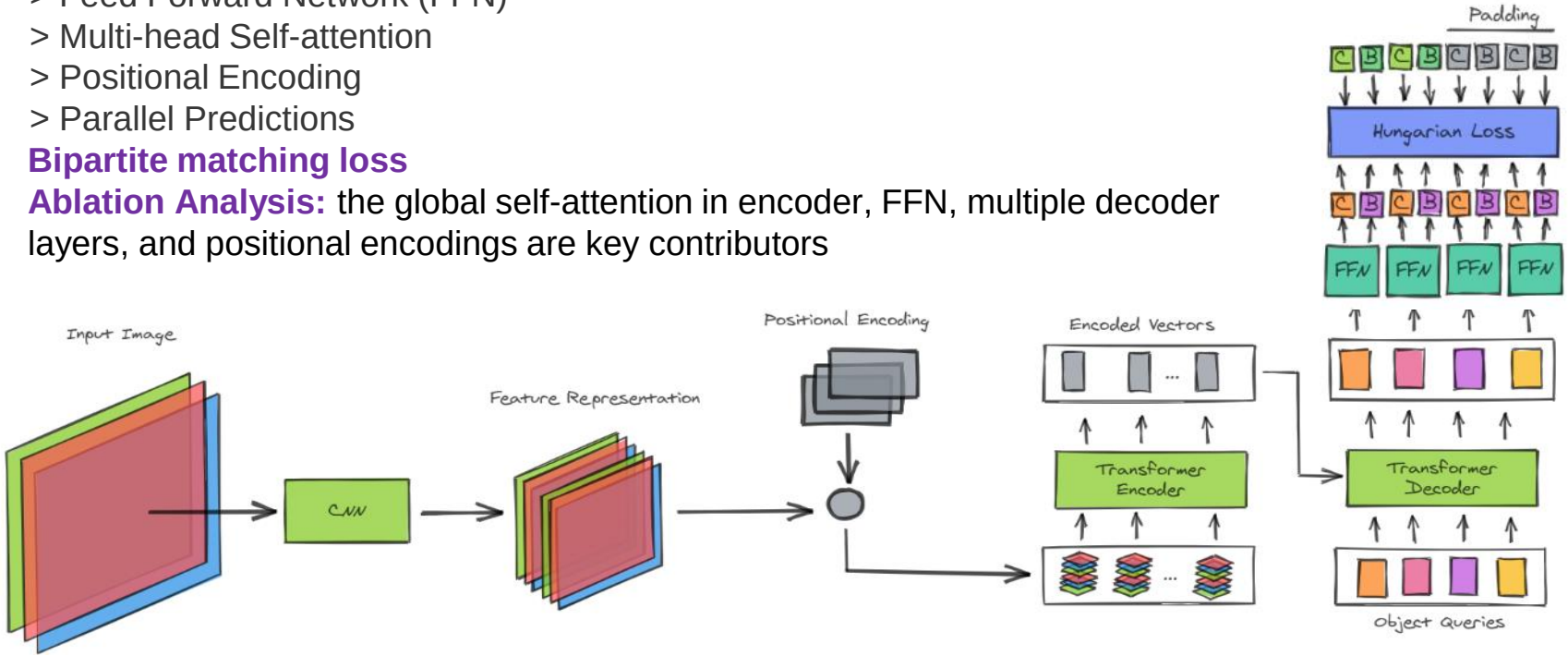
DERT – High Level



- Three-Channeled RGB Input Image (multiple objects)
- CNN
- Transformer
- Predictions: Tuple(class label, bounding box), no-object class
- Manually annotated Image: Tuple(ground truth class label, bounding box), paddings for non-object class

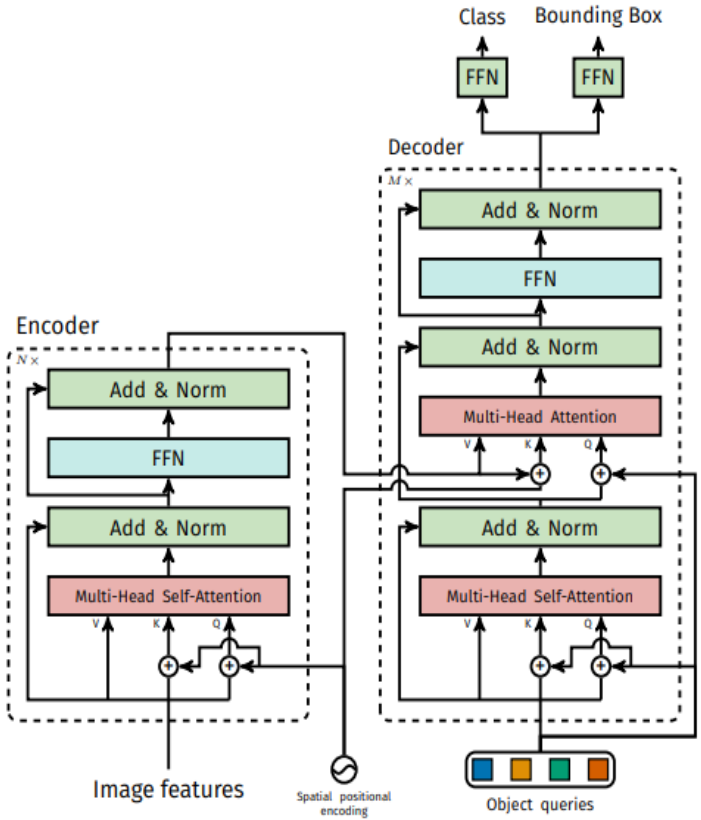
DERT - Major Components

- **CNN backbone:** Lower resolution feature map $f \in \mathbb{R}^{C \times H \times W}$
  Typical values used $C = 2048$ and $H, W = \frac{H_0}{32}, \frac{W_0}{32}$
- **Transformers**
  > Feed Forward Network (FFN)
  > Multi-head Self-attention
  > Positional Encoding
  > Parallel Predictions
- **Bipartite matching loss**
- **Ablation Analysis:** the global self-attention in encoder, FFN, multiple decoder layers, and positional encodings are key contributors



Transformer

- **Encoder:**
  - 1x1 convolution reduce activation map dimension from C to d feature map $z_0 \in \mathbb{R}^{d \times H \times W}$
  - sequence as input -> d x HW feature map
  - standard architecture with multi-head self-attention module and a feed forward network (FFN).
  - Positional encoding added to the input of each attention layer
- **Decoder:** N object queries transformed into output embedding, independently decoded into class labels and box coordinates tuples by a FFN in parallel
- Train the transformer with a learning rate of 10^−4 . Additive dropout of 0.1 is applied after every multi-head attention and FFN before layer normalization. The weights are randomly initialized with Xavier initialization



Bipartite Matching Loss

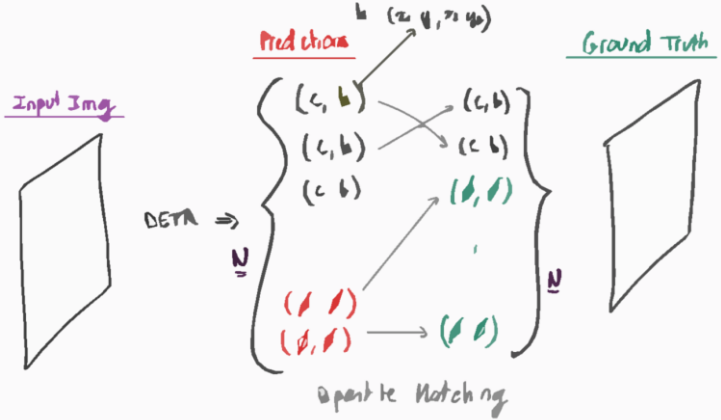
- Minimize the pair-wise N elements matching cost
- All losses are normalized by the number of objects inside the batch

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

- Optimal matching is calculated using the Hungarian algorithm

$$\mathrm{L_{Hungarian}}(\mathrm{y}, \hat{\mathrm{y}}) = \sum_{\mathrm{i}=1}^{\mathrm{N}} [-\log \hat{\mathrm{p}}_{\hat{\sigma}(\mathrm{i})}(\mathrm{c_i}) + \mathbb{I}_{\{\mathrm{c_i} \neq \phi\}} \mathrm{L_{box}}(\mathrm{b_i}, \hat{\mathrm{b}}_{\mathrm{i}})]$$

- L1 loss have different scales for small and large boxes, use a linear combination of the L1 loss and the generalized IoU loss to mitigate this box loss issue



y - ground truth set of objects

$\hat{y}$ - set of predictions from 1 to N

$y_i = (c_i, b_i)$

c_i - ground truth class

$\hat{p}_i$ - predicted class probability

$b_i \in [0, 1]^4$ - vector defining $[center_x, center_y, height, width]$

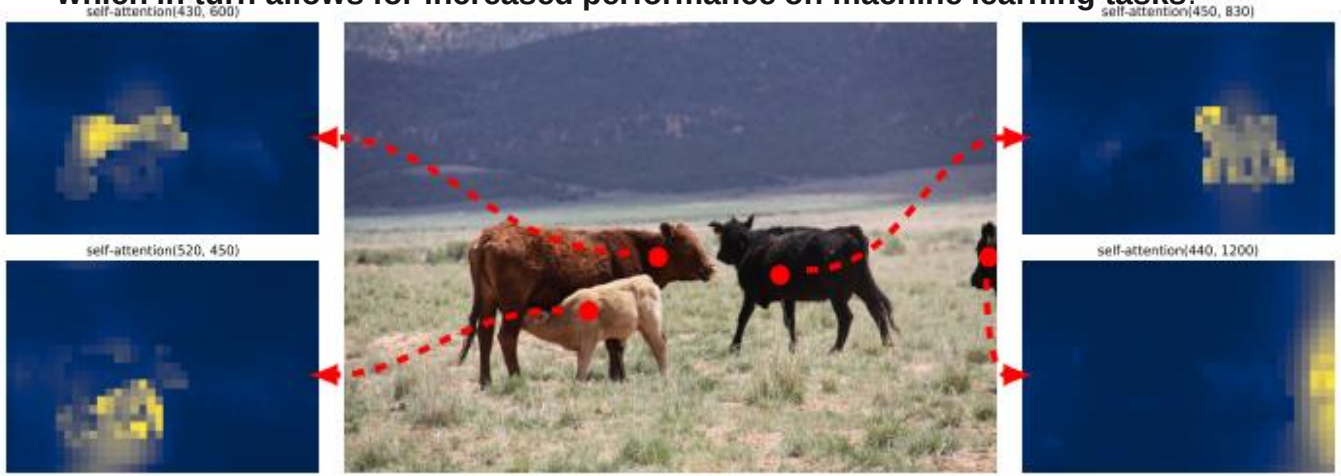
$$\hat{\sigma} = \arg\min_{\sigma \in \mathrm{N}} \sum_{\mathrm{i}}^{\mathrm{N}} -\mathbb{I}_{\mathrm{c_i} \neq \phi} \hat{\mathrm{p}}_{\hat{\sigma}(\mathrm{i})}(\mathrm{c_i}) + \mathbb{I}_{\mathrm{c_i} \neq \phi} \mathrm{L_{box}}(\mathrm{b_i}, \hat{\mathrm{b}}_{\mathrm{i}})$$

$\mathbb{I}$ - identity function equals to 1 when $c_i \neq \phi$ else 0

$$\mathrm{L_{box}} = \lambda_{\mathrm{iou}} \mathrm{L_{iou}}(\mathrm{b_i}, \hat{\mathrm{b}}_{\mathrm{i}}) + \lambda_{\mathrm{L1}} ||\mathrm{b_i} - \hat{\mathrm{b}}_{\hat{\sigma}(\mathrm{i})}||$$

Attention Mechanism

- An attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence
- Multi-head attention allows for the neural network to control the mixing of information between pieces of an input sequence, leading to the creation of richer representations, which in turn allows for increased performance on machine learning tasks.



Terminologies

- **Direct Set Prediction**

  Instead of using the conventional two-stage process involving region proposal networks (RPNs) and subsequent object classification, SETR aims to predict classes of all objects in a image as a set

- **Object Queries**

  - Number is fixed (ensuring a N-> N predictions)
  - N input embeddings with learnt positional encoding

- **Feed-forward Networks**

  The final prediction is computed by a 3-layer perceptron with ReLU activation function and hidden dimension d, and a linear projection layer. The FFN predicts the normalized center coordinates, height and width of the box w.r.t. the input image, and the linear layer predicts the class label using a SoftMax function

- **Parallel Prediction**

  Using the information gathered from the self-attention mechanism, DETR simultaneously predicts the class and location (bounding box) for each object query. This parallel prediction is a departure from traditional object detectors, which often rely on sequential processing

Applications – Panoptic Segmentation

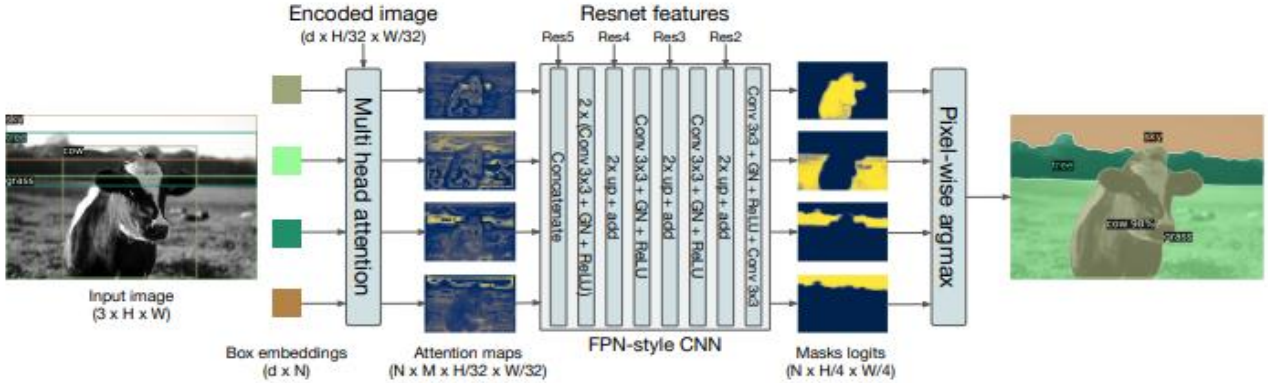


Fig. 8: Illustration of the panoptic head. A binary mask is generated in parallel for each detected object, then the masks are merged using pixel-wise argmax.

Input through the detector

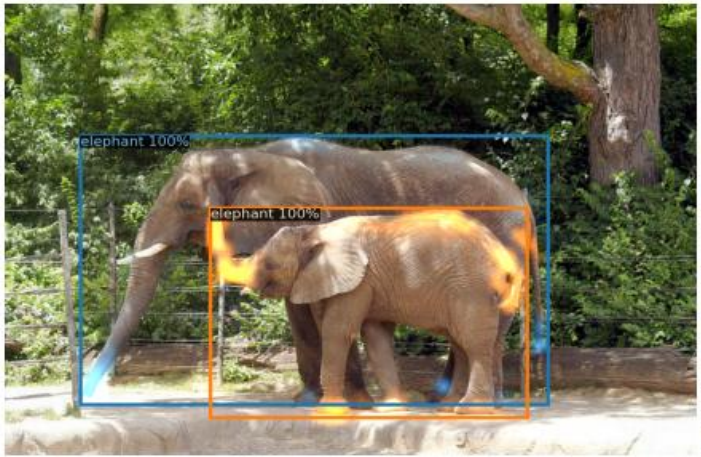
Takes the attention maps of the instances

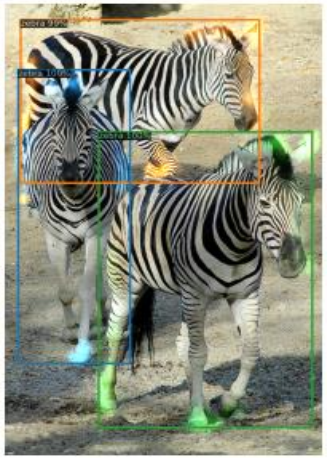
CNN (reversed): scaled up

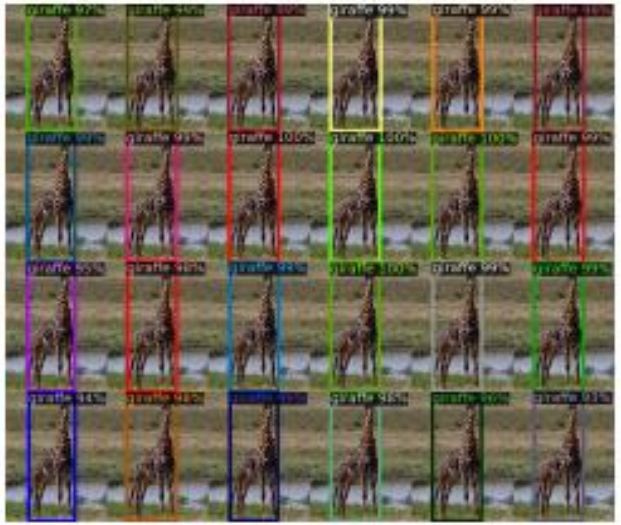
Classify each pixels

Merge all predictions

Innovations & Advancement







- Simplicity
- Obviating the necessity for traditional region proposal mechanisms
- Extensible to other image detection relevant tasks
- Generalize to unseen numbers of instances
- Attention Mechanism – overlapping objects

Limitations & Further Studies

- Relatively longer training time
- High computing cost

  DETR was trained on 8 GPUs for 6 days
- DETR requires specifying the number of object queries in advance, which can be a limitation when dealing with scenes containing varying numbers of objects
- DETR demonstrates significantly better performance on large objects, a result likely enabled by the non-local computations of the transformer. It obtains, however, lower performances on small objects.

DETR Inference Code

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

class DETR(nn.Module):

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

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

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

- Less than 50 lines!!!
- Use learned positional encoding instead of fixed one in the encoder
- Positional encoding added to input layer only
- Github-DETR
- Simple Demo

Reference

Carion, Nicolas & Massa, Francisco & Synnaeve, Gabriel & Usunier, Nicolas & Kirillov, Alexander & Zagoruyko, Sergey. (2020). End-to-End Object Detection with Transformers. 10.1007/978-3-030-58452-8_13.

Prajapati, K. (2022, December 22). Detr: End to end object detection with Transformer. Medium. https://kiran-prajapati.medium.com/detr-end-to-end-object-detection-with-transformer-6d1deb67113d

YouTube. (2020). YouTube. Retrieved October 30, 2023, from https://www.youtube.com/watch?v=T35ba_VXkMY

Paper notes: End-to-end object detection with Transformers. Shivam Shakti. (2020, July 20). https://shakti.dev/attention/computer%20vision/2020/07/20/detr.html

Discussion