ViT 架构在图像检测与分割的应用

Vision Transformer

code:https://github.com/google-research/vision_transformer

paper:https://arxiv.org/abs/2010.11929

ViT 模型的输入是图像块序列，而不是像传统卷积神经网络那样直接输入整个图像。输入的图像块序列首先通过一个嵌入层进行编码，然后通过多个 Transformer 编码器层进行处理，最后通过一个全局平均池化层得到图像的表示。

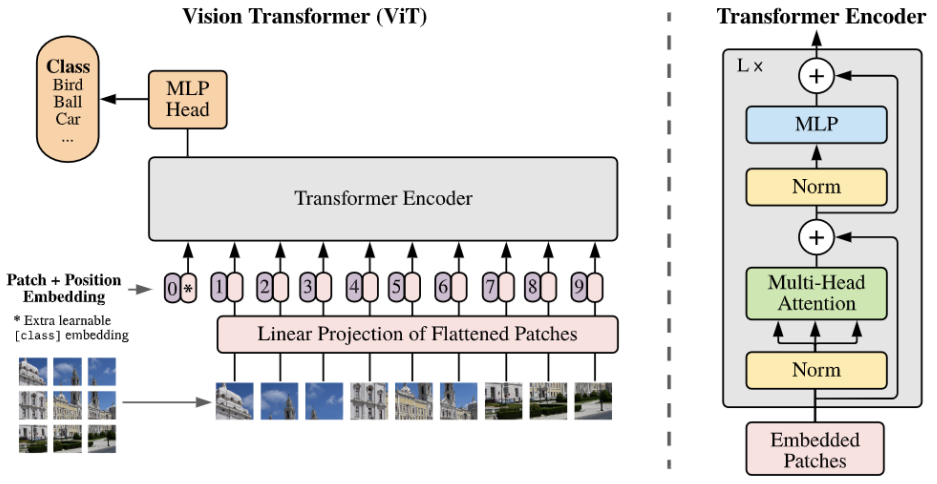


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

partition

$$x \in R^{H \times W \times C} -> x_p \in R^{N \times (P^2 \cdot C)}$$

$$N = HW/P^2$$

x 是输入图片，H、W、C 分别是高、宽、通道数

N 是图像块数(patch),每个图像块的分辨率是(P,P)

```python
def window_partition(x: torch.Tensor, window_size: int) -> Tuple[t
    """
    Partition into non-overlapping windows with padding if needed.
    Args:
        x (tensor): input tokens with [B, H, W, C].
        window_size (int): window size.

    Returns:
        windows: windows after partition with [B * num_windows, wi
        (Hp, Wp): padded height and width before partition
    """
    B, H, W, C = x.shape

    pad_h = (window_size - H % window_size) % window_size
    pad_w = (window_size - W % window_size) % window_size
    if pad_h > 0 or pad_w > 0:
        x = F.pad(x, (0, 0, 0, pad_w, 0, pad_h))
    Hp, Wp = H + pad_h, W + pad_w

    x = x.view(B, Hp // window_size, window_size, Wp // window_siz
    windows = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(-1, wi
    return windows, (Hp, Wp)
```

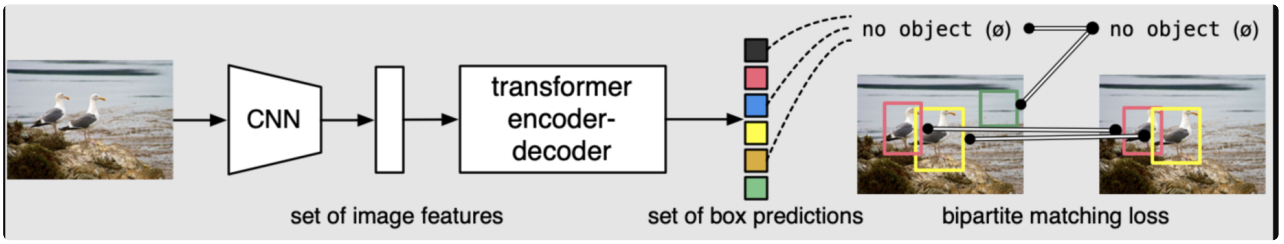
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

DETR (DEtection TRansformer)

code:https://github.com/facebookresearch/detr

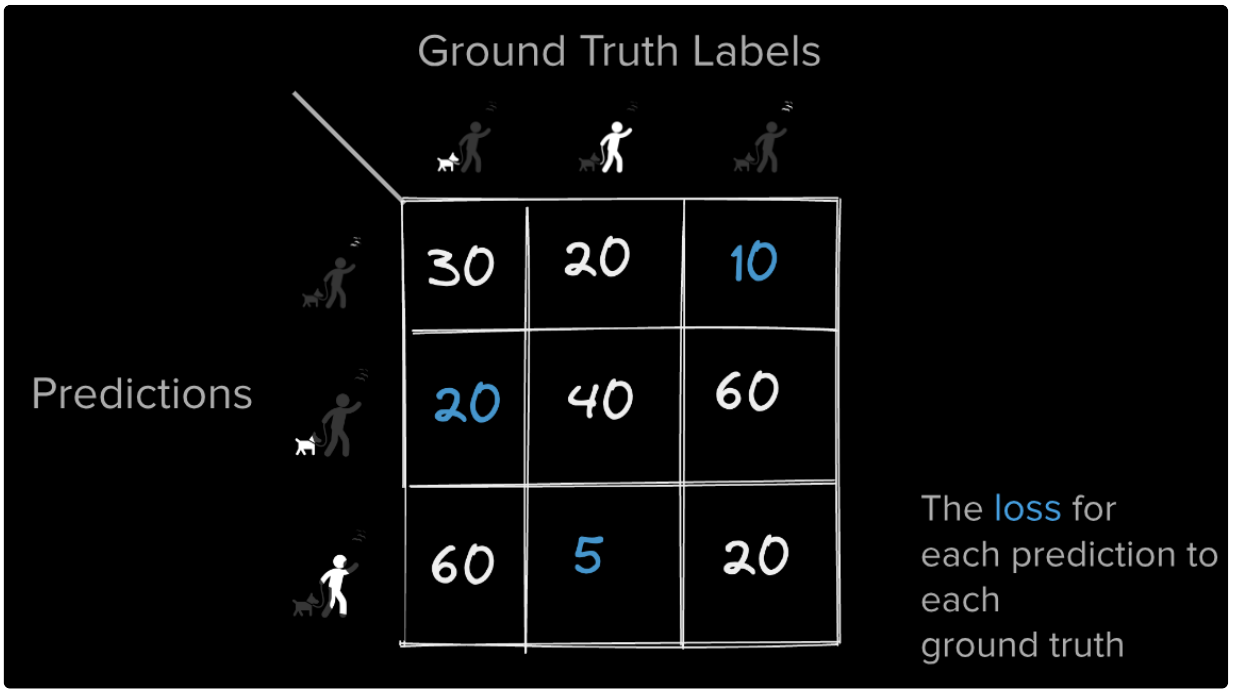
paper:链接

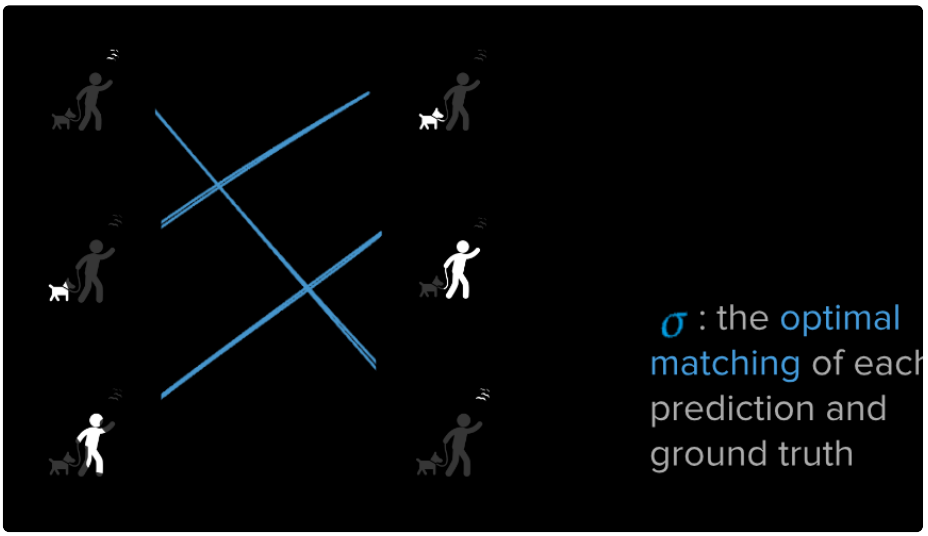


DETR 框架的主要思想是将目标检测视为**直接集合预测问题**，通过全局损失和 Transformer 编码器–解码器架构实现端到端的目标检测。相比于其他现代检测器，DETR 框架的优点在于简化了检测流程，不需要手动设计的组件，如非最大值抑制过程或锚点生成，也不需要专门的库。

bipartite matching loss

二分匹配是图论中的一个数学概念，它是指在二部图中找到边的子集，使得图中的每个顶点最多与子集中的一条边相关联，并且使子集的大小最大化的过程。该边缘子集称为最大基数匹配或简称为最大匹配。





Let us denote by y the ground truth set of objects, and $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$ the set of N predictions. Assuming N is larger than the number of objects in the image, we consider y also as a set of size N padded with $\varnothing$ (no object). To find a bipartite matching between these two sets we search for a permutation of N elements $\sigma \in \mathfrak{S}_N$ with the lowest cost:

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

where $\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$ is a pair-wise *matching cost* between ground truth y_i and a prediction with index $\sigma(i)$. This optimal assignment is computed efficiently with the Hungarian algorithm, following prior work (*e.g.* [43]).

The second step is to compute the loss function, the *Hungarian loss* for all pairs matched in the previous step. We define the loss similarly to the losses of common object detectors, *i.e.* a linear combination of a negative log-likelihood for class prediction and a box loss defined later:

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

指示函数，满足条件取 1，否则为 0

Grounded DINO

paper:https://arxiv.org/abs/2303.05499

code:https://github.com/IDEA-Research/GroundingDINO

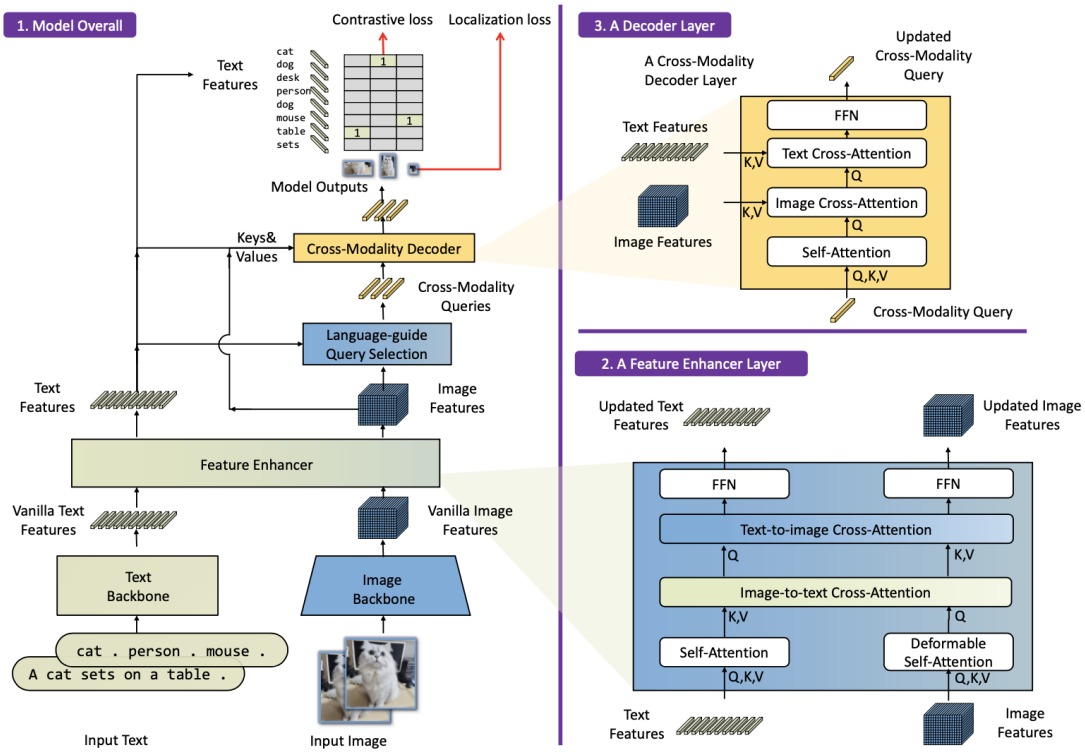


Figure 3. The framework of Grounding DINO. We present the overall framework, a feature enhancer layer, and a decoder layer in block 1, block 2, and block 3, respectively.

Grounding DINO is a dual-encoder-single-decoder architecture.

Grounding DINO 包含一个图像骨干网络和一个文本骨干网络，用于提取图像和文本特征。

这些特征然后被送入一个特征增强器中进行跨模态特征融合。最后，**我们使用一个解码器来生成多个对象框和相应的名词短语，以实现图像文本对齐任务。**

Given an (Image, Text) pair, we extract multi-scale image features with an image backbone like **Swin Transformer**, and text features with a text backbone like **BERT**.

在 Grounding DINO 模型中，我们使用了两种损失函数来训练模型：对比损失（Contrastive loss）和定位损失（Localization loss）。

对比损失用于学习图像和文本之间的语义相似性。具体来说，我们将每个图像和文本对编码为一个向量，并计算它们之间的余弦相似度。然后，我们使用对比损失来最小化正样本之间的距离，并最大化负样本之间的距离。这使得模型能够更好地区分不同的图像和文本，并将它们映射到一个共同的嵌入空间中。

定位损失用于学习对象框的位置和大小。具体来说，我们使用 L1 loss 和 GIOU loss 来计算预测框与真实框之间的距离。然后，我们使用定位损失来最小化预测框与真实

框之间的距离，从而使得模型能够更准确地预测对象框。

具体来说，在训练初期，对比损失起着主导作用，帮助模型学习图像和文本之间的语义相似性。而在训练后期，定位损失起着主导作用，帮助模型更准确地预测对象框。

GIOU的计算很简单，对于两个bounding box A，B。我们可以算出其最小凸集（包围A、B的最小包围框）C。有了最小凸集，就可以计算GIOU：

$$GIOU = IOU - \frac{C - (A \cup B)}{C}$$

SAM

github:https://github.com/facebookresearch/segment-anything

paper:https://ai.facebook.com/research/publications/segment-anything/

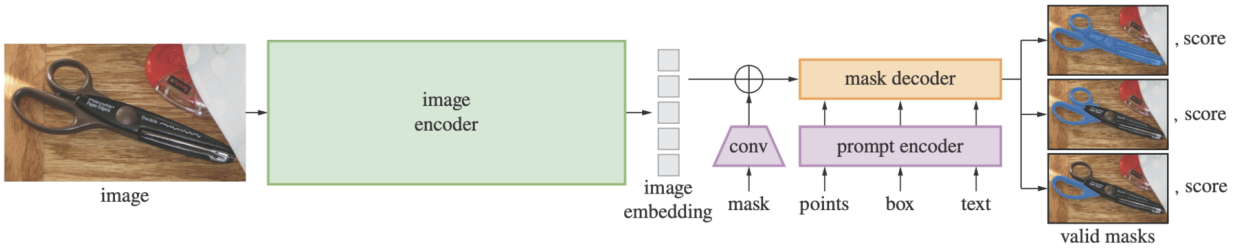


Figure 4: Segment Anything Model (SAM) overview. A heavyweight image encoder outputs an image embedding that can then be efficiently queried by a variety of input prompts to produce object masks at amortized real-time speed. For ambiguous prompts corresponding to more than one object, SAM can output multiple valid masks and associated confidence scores.

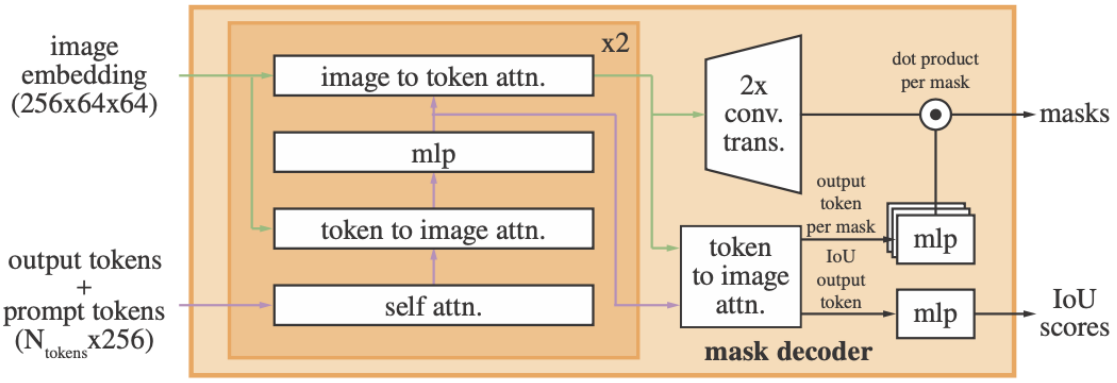


Figure 14: Details of the lightweight mask decoder. A two-layer decoder updates both the image embedding and prompt tokens via cross-attention. Then the image embedding is upscaled, from which the updated output tokens are used to dynamically predict masks. (Not illustrated for figure clarity: At every attention layer, positional encodings are added to the image embedding, and the entire original prompt token (including position encoding) is re-added to the token queries and keys.)

Python

```python
class MaskDecoder(nn.Module):
    def predict_masks(
        self,
        image_embeddings: torch.Tensor,
        image_pe: torch.Tensor,
        sparse_prompt_embeddings: torch.Tensor,
        dense_prompt_embeddings: torch.Tensor,
    ) -> Tuple[torch.Tensor, torch.Tensor]:
        """Predicts masks. See 'forward' for more details."""
        # Concatenate output tokens
        output_tokens = torch.cat([self.iou_token.weight, self.mas
        output_tokens = output_tokens.unsqueeze(0).expand(sparse_p
        tokens = torch.cat((output_tokens, sparse_prompt_embedding

        # Expand per-image data in batch direction to be per-mask
        src = torch.repeat_interleave(image_embeddings, tokens.sha
        src = src + dense_prompt_embeddings
        pos_src = torch.repeat_interleave(image_pe, tokens.shape[0
        b, c, h, w = src.shape

```

```
21
22          # Run the transformer
23          hs, src = self.transformer(src, pos_src, tokens)
24          iou_token_out = hs[:, 0, :]
25          mask_tokens_out = hs[:, 1 : (1 + self.num_mask_tokens), :]
26
27          # Upscale mask embeddings and predict masks using the mask
28          src = src.transpose(1, 2).view(b, c, h, w)
29          upscaled_embedding = self.output_upscaling(src)
30          hyper_in_list: List[torch.Tensor] = []
31          for i in range(self.num_mask_tokens):
32              hyper_in_list.append(self.output_hypernetworks_mlps[i]
33          hyper_in = torch.stack(hyper_in_list, dim=1)
34          b, c, h, w = upscaled_embedding.shape
35          masks = (hyper_in @ upscaled_embedding.view(b, c, h * w)).
36
37          # Generate mask quality predictions
38          iou_pred = self.iou_prediction_head(iou_token_out)
39
            return masks, iou_pred
```

predict API

Python

```python
@torch.no_grad()
def predict_torch(
    self,
    point_coords: Optional[torch.Tensor],
    point_labels: Optional[torch.Tensor],
    boxes: Optional[torch.Tensor] = None,
    mask_input: Optional[torch.Tensor] = None,
    multimask_output: bool = True,
    return_logits: bool = False,
) -> Tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
    if not self.is_image_set:
        raise RuntimeError("An image must be set with .set_image(.

    if point_coords is not None:
        points = (point_coords, point_labels)
    else:
        points = None
```

```
18
19      # Embed prompts
20      sparse_embeddings, dense_embeddings = self.model.prompt_encode
21          points=points,
22          boxes=boxes,
23          masks=mask_input,
24      )
25
26      # Predict masks
27      low_res_masks, iou_predictions = self.model.mask_decoder(
28          image_embeddings=self.features,
29          image_pe=self.model.prompt_encoder.get_dense_pe(),
30          sparse_prompt_embeddings=sparse_embeddings,
31          dense_prompt_embeddings=dense_embeddings,
32          multimask_output=multimask_output,
33      )
34
35      # Upscale the masks to the original image resolution
36      masks = self.model.postprocess_masks(low_res_masks, self.input
37
38      if not return_logits:
39          masks = masks > self.model.mask_threshold
40
41      return masks, iou_predictions, low_res_masks
```

参考链接

https://pyimagesearch.com/2023/06/12/detr-breakdown-part-2-methodologies-and-algorithms/