Transformer

Vit, DETR

Visual Transformer

Visual Transformer (ViT)

intro

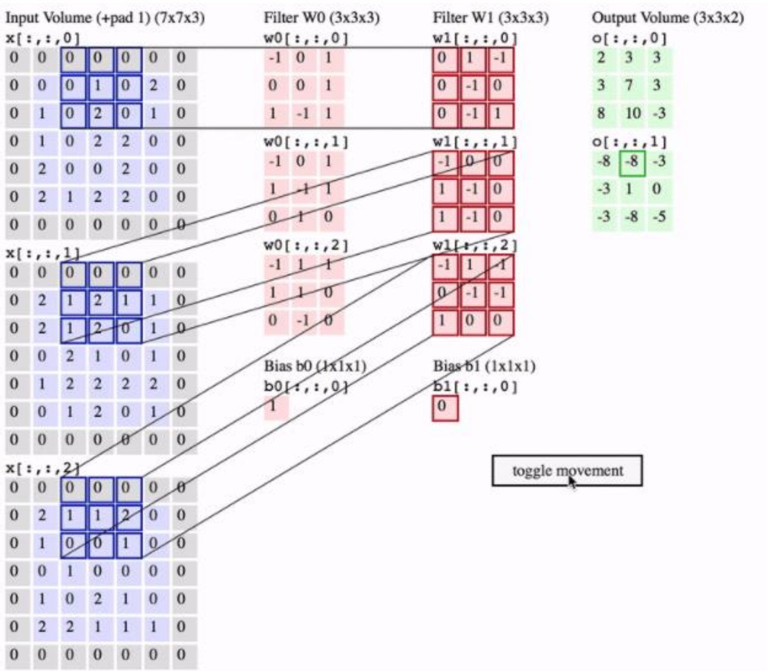
AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE, ICLR 2021

- Transformer 는 NLP 에서 대중적으로 사용하는 구조가 되었음
- Vit는 Transformer를 사용하여 이미지 인식에 적용한 연구
- 반면 컴퓨터 비전 분야에서는 Convolution 연산을 이용한 방법이 아직까지 대중적
- Convolution 연산?

Visual Transformer (ViT)

Inductive bias

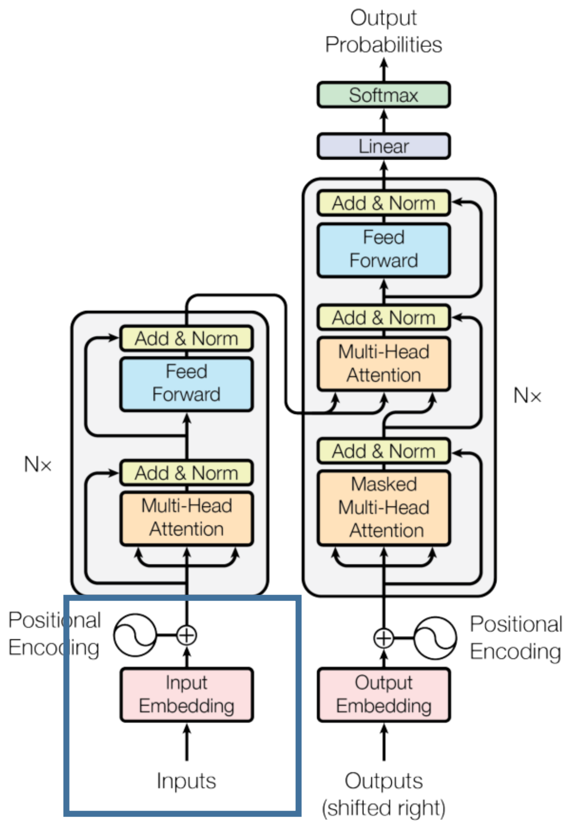
- Inductive Bias : 귀납 편향



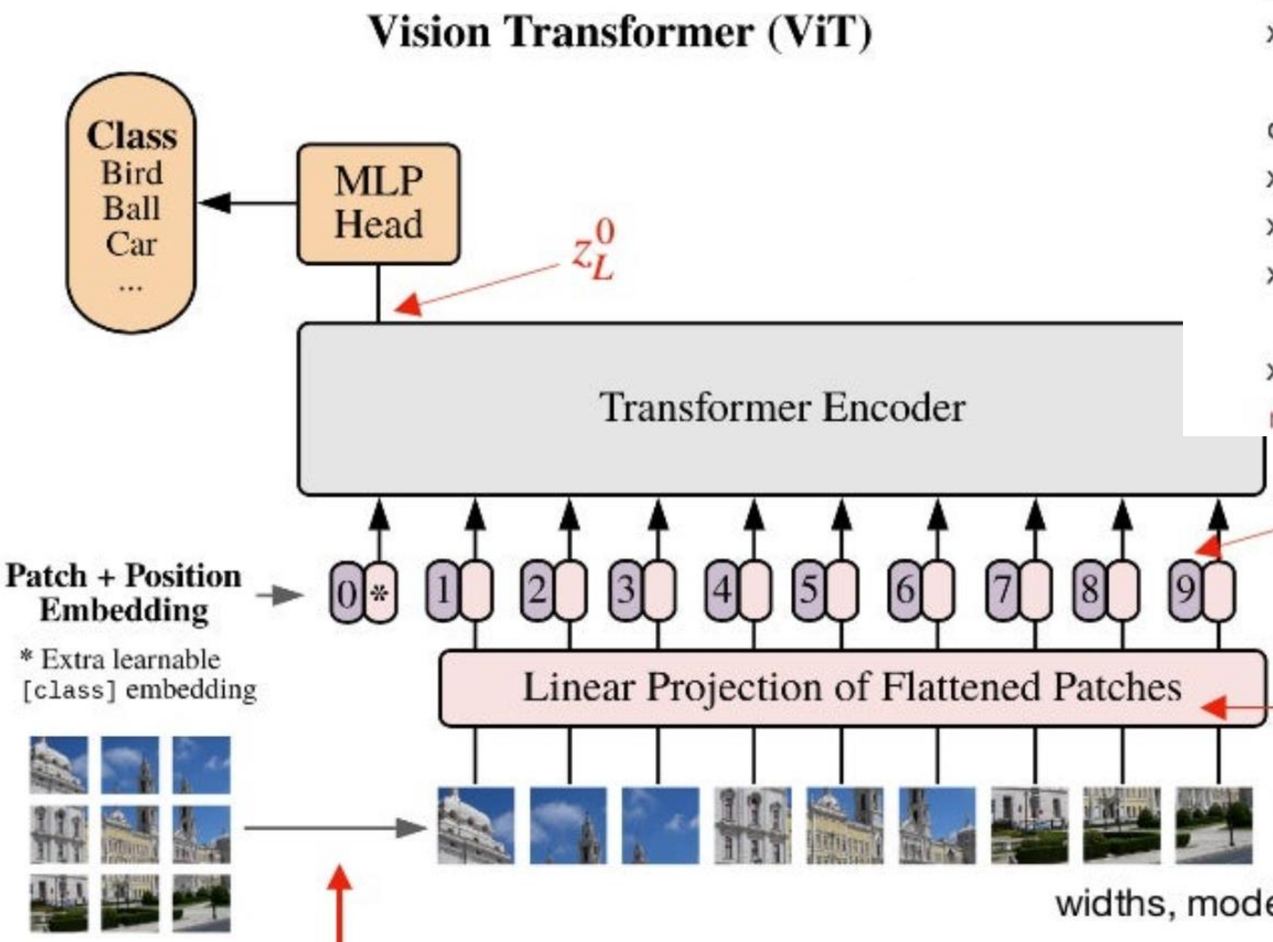
CNN에서 사용되는 inductive bias 없이 이미지 인식에서 성능향상을 보인 ViT

Visual Transformer (ViT)

Key idea



```python
def forward(self, img, mask = None):
    p = self.patch_size

    x = rearrange(img, 'b c (h p1) (w p2) -> b (h w) (p1 p2
    x = self.patch_to_embedding(x)

    cls_tokens = self.cls_token.expand(img.shape[0], -1, -1)
    x = torch.cat((cls_tokens, x), dim=1)
    x += self.pos_embedding
    x = self.transformer(x, mask)

    x = self.to_cls_token(x[:, 0])
    return self.mlp_head(x)
```



Learnable Position Embedding

$E_{pos} \in R^{(N+1)\times D}$

* to retain positional information

Trainable linear projection maps

$x_p \in R^{N\times(P^2\cdot C)} \rightarrow x_p E \in R^{N\times D}$

* Because Transformer uses constant widths, model dimension D, through all of its layers

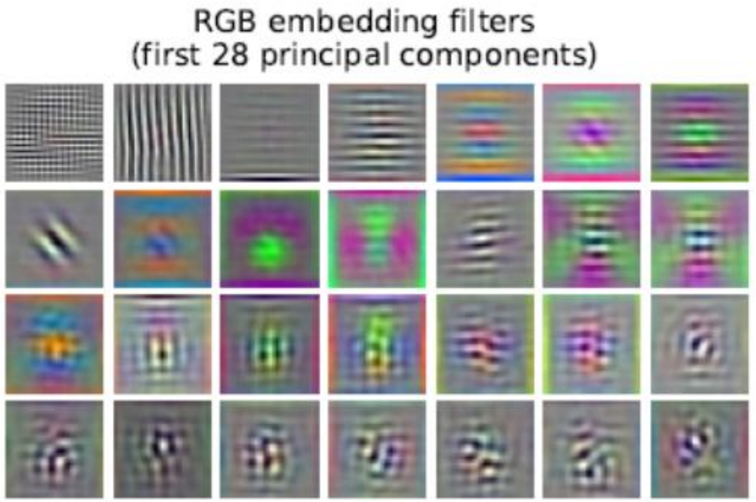
Image $x \in R^{H\times W\times C} \rightarrow$ A sequence of flattened 2D patches $x_p \in R^{N\times(P^2\cdot C)}$

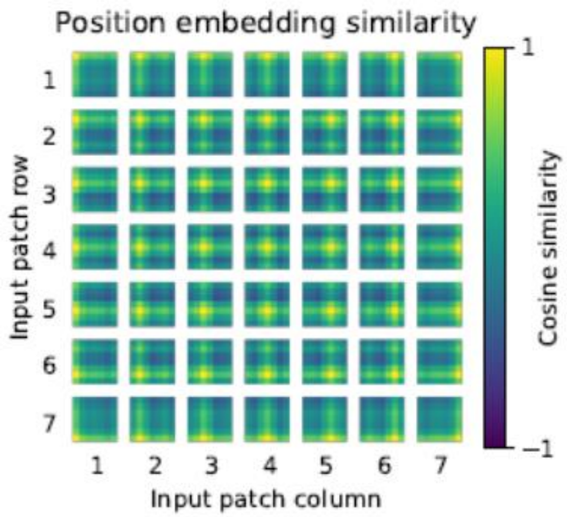
https://www.slideshare.net/DongminChoi6/vit-vision-transformer-review-cdm

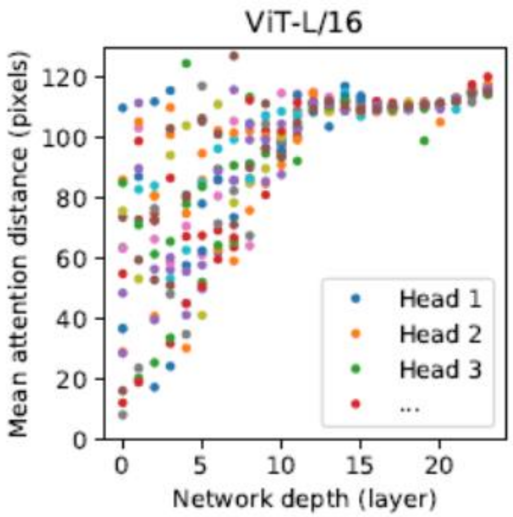
Table 1: Details of Vision Transformer model variants.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21K (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** $\pm$ 0.04	87.76 $\pm$ 0.03	85.30 $\pm$ 0.02	87.54 $\pm$ 0.02	88.4/88.5*
ImageNet ReaL	**90.72** $\pm$ 0.05	90.54 $\pm$ 0.03	88.62 $\pm$ 0.05	90.54	90.55
CIFAR-10	**99.50** $\pm$ 0.06	99.42 $\pm$ 0.03	99.15 $\pm$ 0.03	99.37 $\pm$ 0.06	–
CIFAR-100	**94.55** $\pm$ 0.04	93.90 $\pm$ 0.05	93.25 $\pm$ 0.05	93.51 $\pm$ 0.08	–
Oxford-IIIT Pets	**97.56** $\pm$ 0.03	97.32 $\pm$ 0.11	94.67 $\pm$ 0.15	96.62 $\pm$ 0.23	–
Oxford Flowers-102	99.68 $\pm$ 0.02	**99.74** $\pm$ 0.00	99.61 $\pm$ 0.02	99.63 $\pm$ 0.03	–
VTAB (19 tasks)	**77.63** $\pm$ 0.23	76.28 $\pm$ 0.46	72.72 $\pm$ 0.21	76.29 $\pm$ 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

- JFT300M dataset으로 pretrain 시키고 테스트







1. 자동으로 low-level의 특성을 학습하는 embedding filter
2. Positional embdding 이후의 이미지와 패치와의 유사도
3. Attention이 얼마나 먼 패치를 할당하고 있는지 (receptive field와 유사)

DETR

DETR

intro

End-to-End Object Detection with Transformers, ECCV 2020

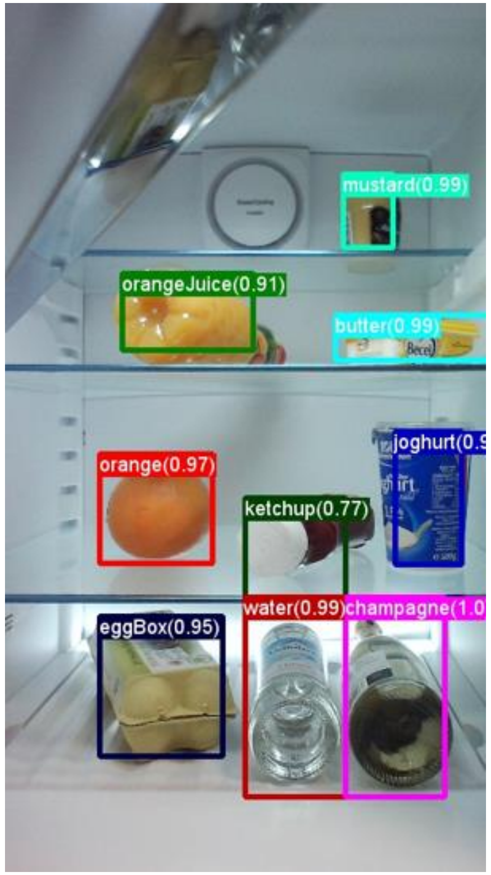
- Object Detection 문제를 direct set prediction으로 푸는 모델
- 기존의 Detection 에서 생기는 non-maximum suppression (NMS) 문제를 해결
- Transformer 기반의 prediction

DETR

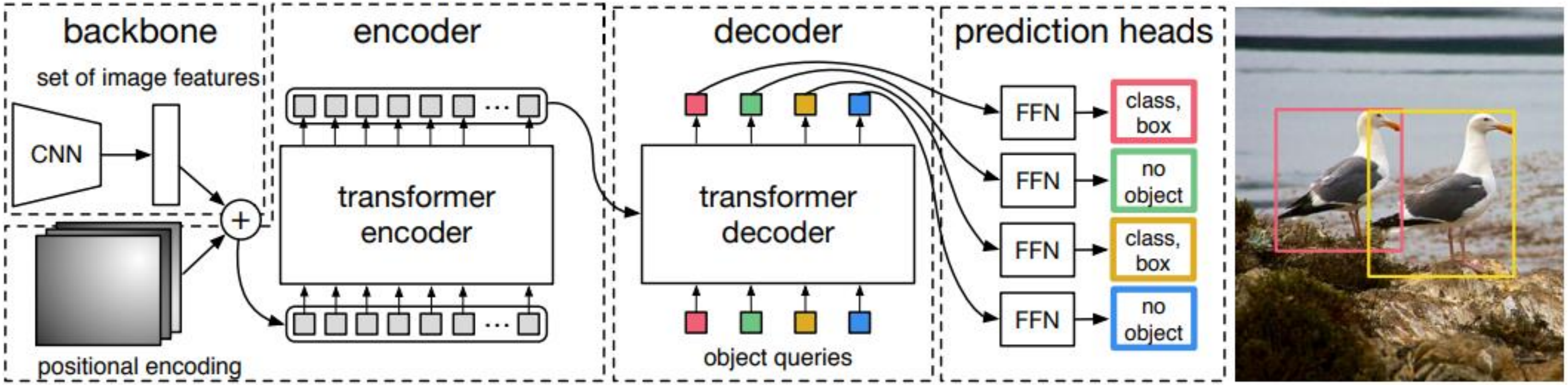
Object detection

DETR

NMS



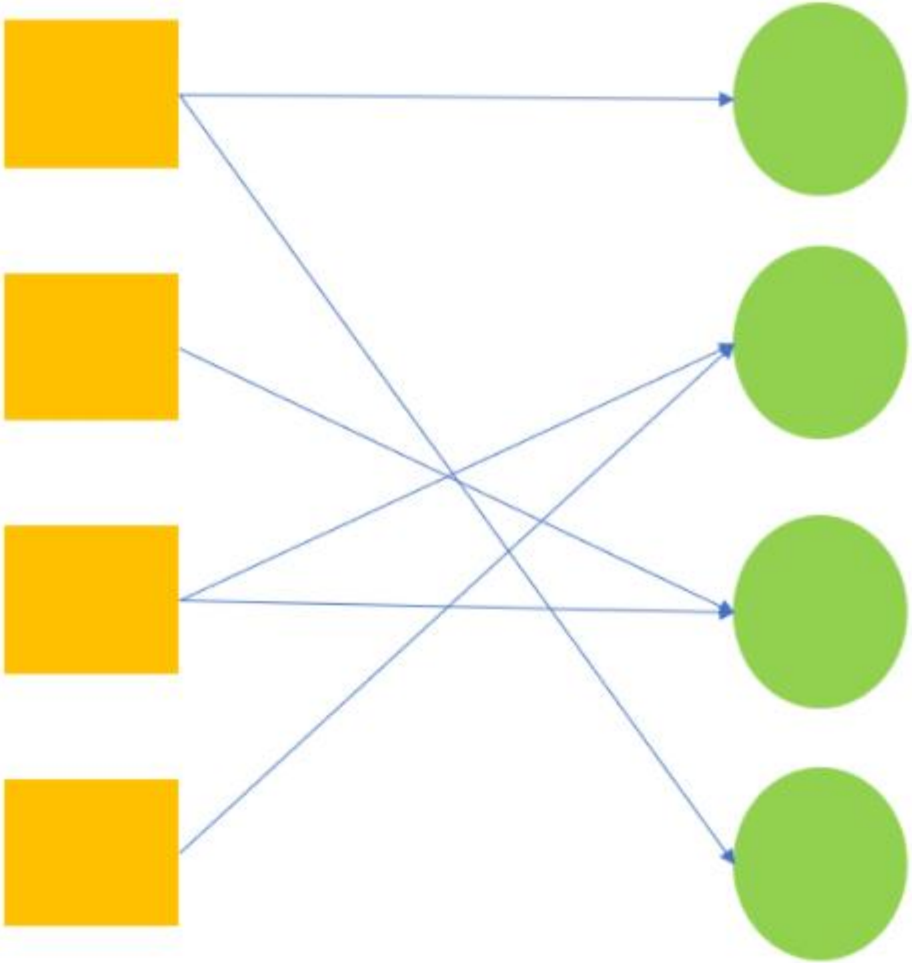
- 기존 방법은 non-maximum suppression (NMS) 문제를 해결하기 위해 post processing 필요
- DETR은 transformer 를 활용한 end-to-end 방식



backbone
set of image features
CNN
positional encoding
encoder
transformer encoder
decoder
transformer decoder
object queries
prediction heads
FFN
FFN
FFN
FFN
class, box
no object
class, box
no object

DETR

bipartite matching



- NMS 해결 -> 각각의 실제 레이블에 대해서 하나만 할당하자!
- bipartite matching

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

σ : index

$$\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = -\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})$$

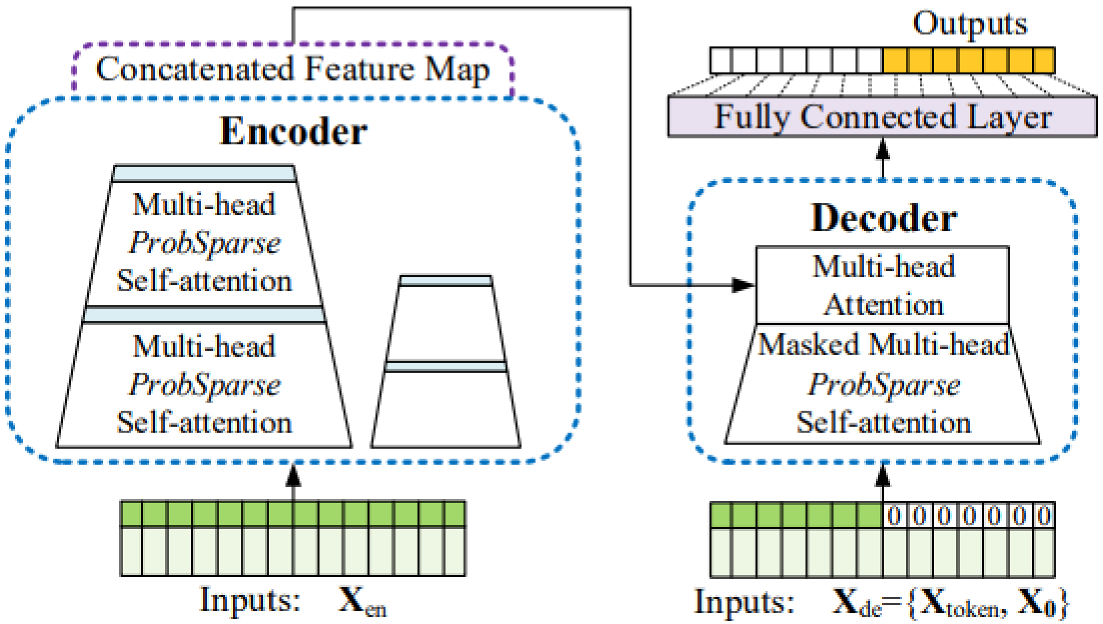
$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

```python
@torch.no_grad()
def forward(self, outputs, targets):

    bs, num_queries = outputs["pred_logits"].shape[:2]

    out_prob = outputs["pred_logits"].flatten(0, 1).softmax(-1)  # [batch_size * num_queries, num_classes]
    out_bbox = outputs["pred_boxes"].flatten(0, 1)  # [batch_size * num_queries, 4]

    tgt_ids = torch.cat([v["labels"] for v in targets])
    tgt_bbox = torch.cat([v["boxes"] for v in targets])

    cost_class = -out_prob[:, tgt_ids]
    cost_bbox = torch.cdist(out_bbox, tgt_bbox.type(torch.FloatTensor).to(out_prob.device), p=1)


    C = self.cost_bbox * cost_bbox + self.cost_class * cost_class
    C = C.view(bs, num_queries, -1).cpu()


    sizes = [len(v["boxes"]) for v in targets]
    indices = [linear_sum_assignment(c[i]) for i, c in enumerate(C.split(sizes, -1))]

    return [(torch.as_tensor(i, dtype=torch.int64), torch.as_tensor(j, dtype=torch.int64)) for i, j in indices]
```

시계열 데이터

시계열 데이터와 자연어 데이터

aa



Concatenated Feature Map
Encoder
Multi-head
ProbSparse
Self-attention
Multi-head
ProbSparse
Self-attention
Inputs: Xen
Outputs
Fully Connected Layer
Decoder
Multi-head
Attention
Masked Multi-head
ProbSparse
Self-attention
0 0 0 0 0 0 0
Inputs: Xde={Xtoken, X0}

Thank you