AdaMixer: A Fast-Converging Query-Based Object Detector

论文分享

Ziteng Gao[1] Limin Wang[1] Bing Han[2] Sheng Guo[2]

[1]State Key Laboratory for Novel Software Technology, Nanjing University, China
[2]MYbank, Ant Group, China

15 Nov, 2022

Contents

Introduction

- Query-based object detectors:
  - 使用 queries 表示目标
  - 例如 Sparse R-CNN[1]、DETR[2]和 Deformable DETR[3]
- 挑战：
  - 更高的计算复杂度
  - 更慢的收敛速度
  - 小目标上更差的性能

[1] Peize Sun et al. "Sparse r-cnn: End-to-end object detection with learnable proposals". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 14454–14463, p. 1.

[2] Nicolas Carion et al. "End-to-end object detection with transformers". In: *European conference on computer vision*. Springer. 2020, pp. 213–229, p. 2.

[3] Xizhou Zhu et al. "Deformable detr: Deformable transformers for end-to-end object detection". In: *arXiv preprint arXiv:2010.04159* (2020), p. 3.

Motivation

Key

adapt limited queries to varying visual objects in decoding queries

- 提升两个方面：
  - positional adaptability
    - translation & scale variance
  - content adaptability
    - visual appearance variance

positional adaptability

content adaptability

Motivation

- 将交叉注意力看作两个阶段：
  - sample features
  - decode sampled features
- 交叉注意力的变体：
  - adaptive 3D feature sampling for positional adaptability
  - adaptive mixing for content adaptability

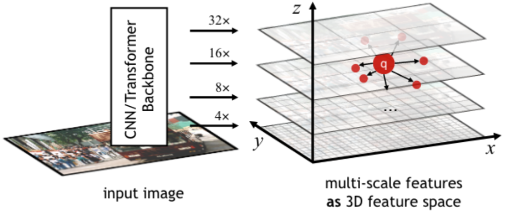


Figure: 3D feature sampling

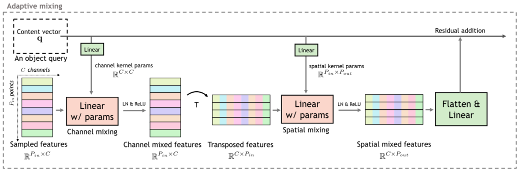


Figure: adaptive mixing

Contents

Contents

Object Query Definition

- 同 Sparse R-CNN、Anchor DETR[4]和 Deformable DETR 中的做法一样，将 query 解耦为内容向量和位置向量：
  - content vector $\mathbf{q} \in \mathbb{R}^{d_q}$，$d_q$ 为通道数
  - positional vector (x, y, z, r)，分别代表中心点的 x、y 轴坐标以及尺度和横纵比的对数。
- 从位置向量中可以解码出 bounding box 的中心点坐标 (x_B, y_B)、宽度 w_B 和高度 h_B：

$$x_B = s_{base} \cdot x, y_B = s_{base} \cdot y$$

$$w_B = s_{base} \cdot 2^{z-r}, h_B = s_{base} \cdot 2^{z+r}$$

[4] Yingming Wang et al. "Anchor detr: Query design for transformer-based detector". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. 3. 2022, pp. 2567–2575, p. 4.

Adaptive Location Sampling

Multi-scale features as the 3D featrue space

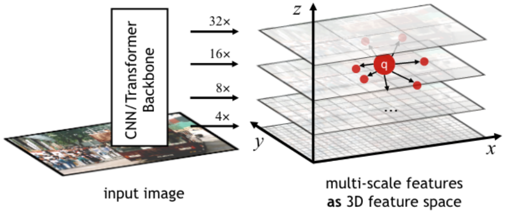


Figure: 3D feature sampling

- Decoder 不仅要在 (x, y) 空间内具有自适应性，在潜在目标的尺度上也要具有自适应性。将多尺度特征空间看做一个 3D 特征空间：

$$z_j^{feat} = \log_2(s_j^{feat}/s_{base})$$

- 将不同步长的特征图的高和宽重新缩放到相同的 $H/s_{base}, W/s_{base}$，其中 H, W 是输入图像的高和宽，并将它们放在 3D 空间中的 x 轴和 y 轴上对齐。

Adaptive Location Sampling

Adaptive 3D Feature Sampling Process

- 首先，利用 query i 的内容向量 $\mathbf{q}$ 通过一个线性层生成针对点的偏移向量集合 P_{in}：

$$\{(\Delta x_i, \Delta y_i, \Delta z_i)\}_{P_{in}} = \mathrm{Linear}(\mathbf{q})$$

- 然后，根据 query i 的的位置向量将这些偏移量转化为采样位置：

$$\begin{cases} \tilde{x}_i = x + \Delta x_i \cdot 2^{z-r}, \\ \tilde{y}_i = y + \Delta y_i \cdot 2^{z+r}, \\ \tilde{z}_i = z + \Delta z_i, \end{cases}$$

- 得到 3D 空间内的采样点坐标后，先在 (x, y) 平面内进行双线性插值，然后再对 $\tilde{z}$ 进行高斯权重插值：

$$\tilde{w}_j = \frac{\exp(-(\tilde{z} - z_j^{feat})^2/\tau_z)}{\sum_j \exp(-(\tilde{z} - z_j^{feat})^2\tau_z)}$$

Adaptive Location Sampling

Group Sampling

- 为了尽可能采样多点的特征，引入了分组采样机制，类似于多头注意力和分组卷积。
- 首先将 3D 特征空间的通道数 d_{feat} 分成 g 组，每组为 d_{feat}/g，每组分别进行 3D 采样。
- 对于每个 query，decoder 可以生成 $g \cdot P_{in}$ 组偏移向量，来丰富采样点的多样性，获得更丰富的空间结构特征。
- 采样特征矩阵 $\mathbf{x}$ 的形状就变成了 $\mathbb{R}^{g \times P_{in} \times (d_{feat}/g)}$。

Adaptive Location Sampling

Code

```python
def sampling_3d(
    sample_points: torch.Tensor,
    multi_lvl_values,
    featmap_strides,
    n_points: int = 1,
    num_levels: int = None,
    tau=2.0,
):
    B, n_queries, _t, n_groups_points, _ = sample_points.shape
    assert _t == 1
    B, C_feat, _, _ = multi_lvl_values[0].shape

    n_groups = n_groups_points//n_points
    n_channels = C_feat//n_groups

    if num_levels is None:
        num_levels = len(featmap_strides)
```

```python
sample_points_xy = sample_points[..., 0:2]

sample_points_z = sample_points[..., 2].clone()
sample_points_lvl_weight = translate_to_linear_weight(
    sample_points_z, num_levels,
    tau=tau, featmap_strides=featmap_strides)

sample_points_lvl_weight_list = sample_points_lvl_weight.unbind(-1)

out = sample_points.new_zeros(
    B, n_queries, n_groups, n_points, n_channels)

for i in range(num_levels):
    value = multi_lvl_values[i]
    lvl_weights = sample_points_lvl_weight_list[i]

    stride = featmap_strides[i]

    mapping_size = value.new_tensor(
        [value.size(3), value.size(2)]).view(1, 1, 1, 1, -1) * stride
    normalized_xy = sample_points_xy/mapping_size

    out += sampling_each_level(normalized_xy, value,
                               weight=lvl_weights, n_points=n_points)

return out, None
```

Adaptive Location Sampling

Code

```
def sampling_each_level(sample_points: torch.Tensor,
                        value: torch.Tensor,
                        weight=None,
                        n_points=1):
    B1, n_queries, _t, n_groups_points, _ = sample_points.shape
    assert _t == 1
    B2, C_feat, H_feat, W_feat = value.shape
    assert B1 == B2
    B = B1

    n_groups = n_groups_points//n_points
    n_channels = C_feat//n_groups

    sample_points = sample_points \
        .view(B, n_queries, n_groups, n_points, 2) \
        .permute(0, 2, 1, 3, 4).flatten(0, 1)
    sample_points = sample_points*2.0-1.0

    # `sampling_points` now has the shape [B*n_groups, n_queries, n_points, 2]
```

```
value = value.view(B*n_groups, n_channels, H_feat, W_feat)
out = F.grid_sample(
    value, sample_points,
    mode='bilinear', padding_mode='zeros', align_corners=False,
)

# `out`` now has the shape [B*n_groups, C, n_queries, n_points]

if weight is not None:
    weight = weight.view(B, n_queries, n_groups, n_points) \
        .permute(0, 2, 1, 3).flatten(0, 1).unsqueeze(1)
    # `weight`` has the shape [B*n_groups, 1, n_queries, n_points]
    out *= weight

return out \
    .view(B, n_groups, n_channels, n_queries, n_points) \
    .permute(0, 3, 1, 4, 2)

# `out`` has shape [B, n_queries, n_groups, n_points, n_channels]
```

Contents

Adaptive Content Decoding

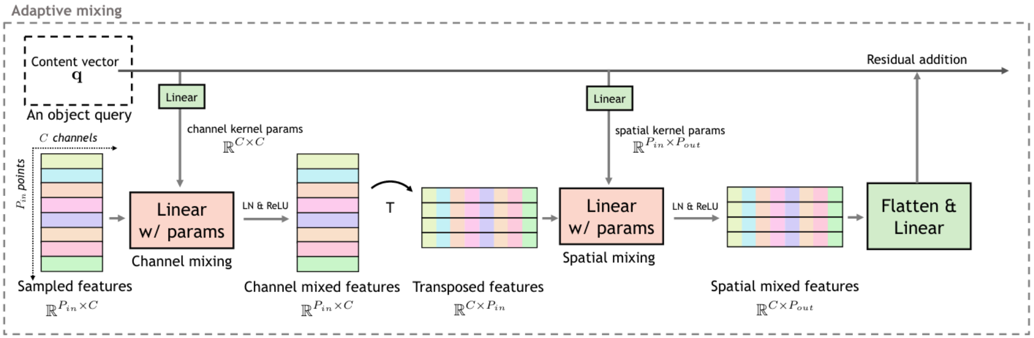


Figure: adaptive mixing

Adaptive Content Decoding

Adaptive Channel Mixing

- 给定一个 query 的采样特征矩阵 $\mathbf{x} \in \mathbb{R}^{P_{in} \times C}$，其中 $C = d_{feat}/g$，自适应通道混合（ACM）会使用基于 $\mathbf{q}$ 的动态权重在通道维数对 $\mathbf{x}$ 进行转换以自适应地增强通道语义：

$$M_c = \text{Linear}(\mathbf{q}) \in \mathbb{R}^{C \times C}$$

$$\text{ACM}(\mathbf{x}) = \text{ReLU}(\text{LayerNorm}(\mathbf{x}M_c))$$

- 其中 $\text{ACM}(\mathbf{x}) \in \mathbb{R}^{P_{in} \times C}$ 是通道混合特征的输出，线性层对于每一组都是独立的。然后对混合输出的所有维度都进行层正则化。
- 在这一步中，动态权重在 3D 空间中的不同采样点之间是共享的，类似于 Sparse R-CNN 中 RoI 特征的 1×1 的自适应卷积。

Adaptive Content Decoding

Adaptive Spatial Mixing

- 为了使 query 对采样特征的空间结构自适应，引入了自适应空间混合（ASM）。先对通道混合特征矩阵进行转置，然后对其空间维度应用动态核：

$$M_s = \mathrm{Linear}(\mathbf{q}) \in \mathbb{R}^{P_{in} \times P_{out}}$$

$$\mathrm{ASM}(\mathbf{x}) = \mathrm{ReLU}(\mathrm{LayerNorm})(\mathbf{x}^T M_s)$$

- 其中 $\mathrm{ASM}(\mathbf{x}) \in \mathbb{R}^{C \times P_{out}}$ 是空间混合输出，P_{out} 是空间混合输出数量。动态权重在不同通道间是共享的。

Adaptive Mixing

Code

```
    M = M.reshape(
        B*N, G, self.eff_in_dim, self.eff_out_dim)
    S = S.reshape(
        B*N, G, self.out_points, self.in_points)

    '''adaptive channel mixing'''
    out = torch.matmul(out, M)
    out = F.layer_norm(out, [out.size(-2), out.size(-1)])
    out = self.act(out)

    '''adaptive spatial mixing'''
    out = torch.matmul(S, out)  # implicitly transpose and matmul
    out = F.layer_norm(out, [out.size(-2), out.size(-1)])
    out = self.act(out)

'''linear transfomation to query dim'''
out = out.reshape(B, N, -1)
out = self.out_proj(out)

out = query + out
```

Contents

Overall AdaMixer Detector

- 类似于 DETR 和 Deformable DETR 中的 decoder 结构，将 query 间的自注意力机制、提出的自适应混合器和 FFN 按顺序放在 decoder 中

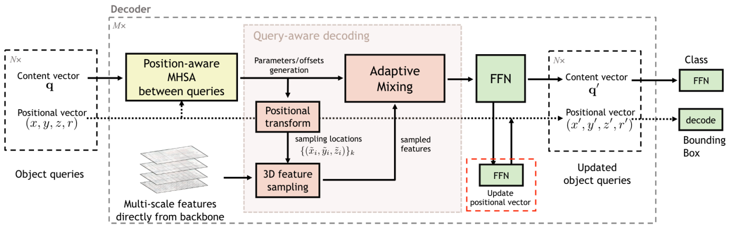


Figure: Decoder 的结构

- Query 的位置向量在每个阶段结束后由 FFN 更新：

$$x' = x + \Delta x \cdot 2^z, y' = t + \Delta y \cdot 2^z,$$

$$z' = z + \Delta z, r' = r + \Delta r$$

Overall AdaMixer Detector

Position-aware multi-head self-attentions

- 将内容向量和位置信息 (x, y, z, r) 嵌入到自注意力中。
- 将前景交集（IoF）作为 query 间注意力权重的偏置嵌入到 query 中，以显式地包含 query 间被包含的关系。
- 对于每个注意力头：

$$\text{Attn}(Q, K, V) = \text{Softmax}(QK^T/\sqrt{d_q} + \alpha B)V$$

- 其中 $B_{ij} = \log(|box_i \cap box_j|/|box_i| + \epsilon)$， $\epsilon = 10^{-7}$
- $B_{ij} = 0$ 代表 box i 完全包含在 box j 中， $B_{ij} = \log \epsilon \ll 0$ 代表 box i 和 j 之间没有重叠。

Overall AdaMixer Detector

Code

```python
with torch.no_grad():
    rois = decode_box(query_xyzr)
    roi_box_batched = rois.view(N, n_query, 4)
    iof = bbox_overlaps(roi_box_batched, roi_box_batched, mode='iof')[
        :, None, :, :]
    iof = (iof + 1e-7).log()
    pe = position_embedding(query_xyzr, query_content.size(-1) // 4)

'''IoF'''
attn_bias = (iof * self.iof_tau.view(1, -1, 1, 1)).flatten(0, 1)

query_content = query_content.permute(1, 0, 2)
pe = pe.permute(1, 0, 2)
'''sinusoidal positional embedding'''
query_content_attn = query_content + pe
query_content = self.attention(
    query_content_attn,
    attn_mask=attn_bias,
)
query_content = self.attention_norm(query_content)
query_content = query_content.permute(1, 0, 2)
```

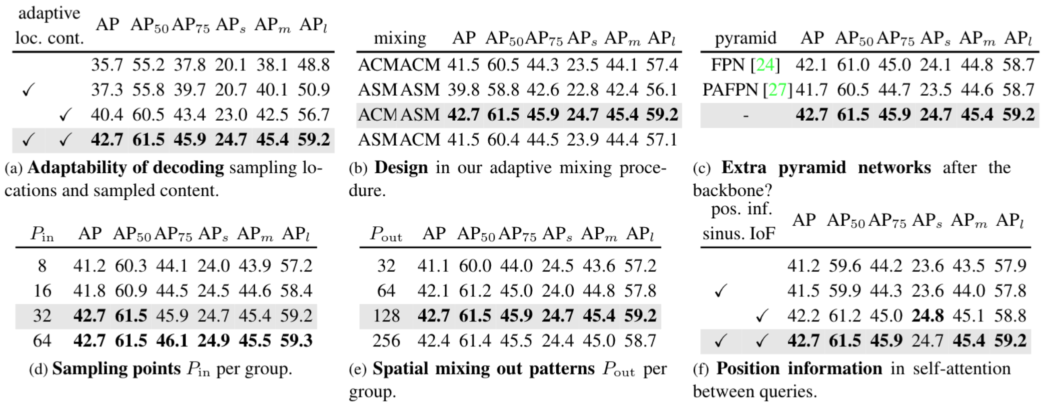
```python
''' adaptive 3D sampling and mixing '''
query_content = self.sampling_n_mixing(
    x, query_content, query_xyzr, featmap_strides)

# FFN
query_content = self.ffn_norm(self.ffn(query_content))

cls_feat = query_content
reg_feat = query_content

for cls_layer in self.cls_fcs:
    cls_feat = cls_layer(cls_feat)
for reg_layer in self.reg_fcs:
    reg_feat = reg_layer(reg_feat)

cls_score = self.fc_cls(cls_feat).view(N, n_query, -1)
xyzr_delta = self.fc_reg(reg_feat).view(N, n_query, -1)

return cls_score, xyzr_delta, query_content.view(N, n_query, -1)
```

Contents

Contents

西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Ablation Studies



adaptive loc.	adaptive cont.	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
		35.7	55.2	37.8	20.1	38.1	48.8
✓		37.3	55.8	39.7	20.7	40.1	50.9
	✓	40.4	60.5	43.4	23.0	42.5	56.7
✓	✓	**42.7**	**61.5**	**45.9**	**24.7**	**45.4**	**59.2**

(a) **Adaptability of decoding** sampling locations and sampled content.

mixing		AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
ACM	ACM	41.5	60.5	44.3	23.5	44.1	57.4
ASM	ASM	39.8	58.8	42.6	22.8	42.4	56.1
ACM	ASM	**42.7**	**61.5**	**45.9**	**24.7**	**45.4**	**59.2**
ASM	ACM	41.5	60.4	44.5	23.9	44.4	57.1

(b) **Design** in our adaptive mixing procedure.

pyramid	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
FPN [24]	42.1	61.0	45.0	24.1	44.8	58.7
PAFPN [27]	41.7	60.5	44.7	23.5	44.6	58.7
-	**42.7**	**61.5**	**45.9**	**24.7**	**45.4**	**59.2**

(c) **Extra pyramid networks** after the backbone?

P_{in}	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
8	41.2	60.3	44.1	24.0	43.9	57.2
16	41.8	60.9	44.5	24.5	44.6	58.4
32	**42.7**	**61.5**	45.9	24.7	45.4	59.2
64	**42.7**	**61.5**	**46.1**	**24.9**	**45.5**	**59.3**

(d) **Sampling points** P_{in} per group.

P_{out}	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
32	41.1	60.0	44.0	24.5	43.6	57.2
64	42.1	61.2	45.0	24.0	44.8	57.8
128	**42.7**	**61.5**	**45.9**	**24.7**	**45.4**	**59.2**
256	42.4	61.4	45.5	24.4	45.0	58.7

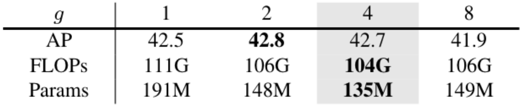
(e) **Spatial mixing out patterns** P_{out} per group.

pos. inf. sinus.	pos. inf. IoF	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
		41.2	59.6	44.2	23.6	43.5	57.9
✓		41.5	59.9	44.3	23.6	44.0	57.8
	✓	42.2	61.2	45.0	**24.8**	45.1	58.8
✓	✓	**42.7**	**61.5**	**45.9**	24.7	**45.4**	**59.2**

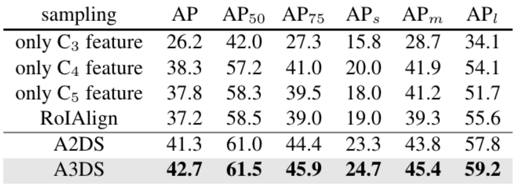
(f) **Position information** in self-attention between queries.

Figure: AdaMixer Ablation Studies

Ablation Studies



g	1	2	4	8
AP	42.5	**42.8**	42.7	41.9
FLOPs	111G	106G	**104G**	106G
Params	191M	148M	**135M**	149M



sampling	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
only C_3 feature	26.2	42.0	27.3	15.8	28.7	34.1
only C_4 feature	38.3	57.2	41.0	20.0	41.9	54.1
only C_5 feature	37.8	58.3	39.5	18.0	41.2	51.7
RoIAlign	37.2	58.5	39.0	19.0	39.3	55.6
A2DS	41.3	61.0	44.4	23.3	43.8	57.8
A3DS	**42.7**	**61.5**	**45.9**	**24.7**	**45.4**	**59.2**

a. 位置和内容自适应性都非常重要。
b. 通道语义和空间结构对于混合设计都很重要。两种混合交换效果不好的结果可能是因为空间混合特征的通道语义信息不足。
c. 使用了额外网络的模型可能需要更长的训练时间和更多训练样本才能获得更好的效果。
d. 性能在 $P_{in} = 32$ 时达到饱和
e. 性能在 $P_{out} = 128$ 时出现下降
f. 框与框之间的 IoF 描述了对应 query 的包含关系，对于自注意力模仿 NMS 操作很重要

Contents

Comparison

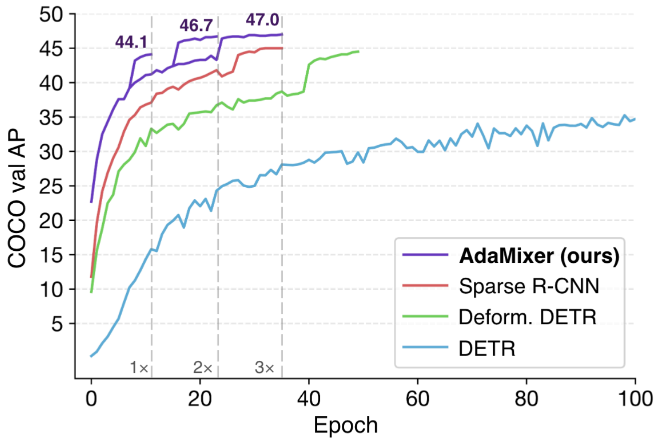
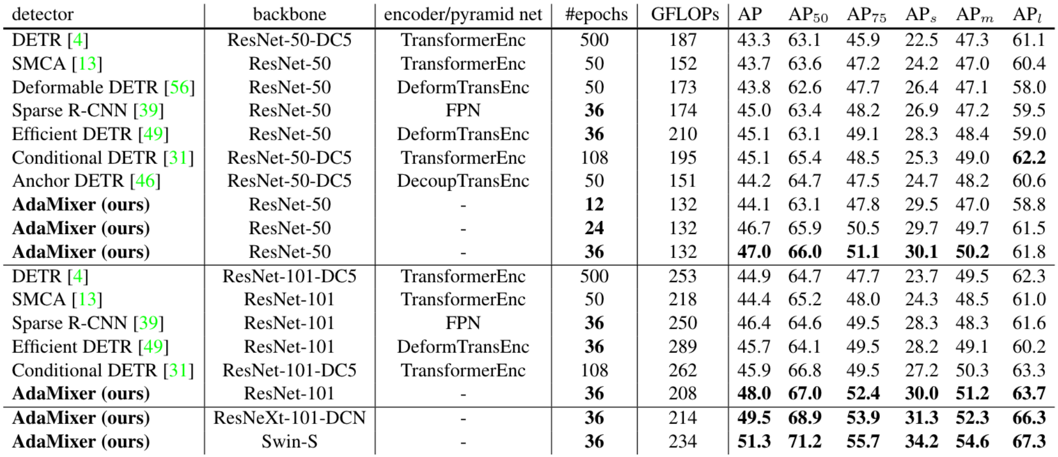


Figure: 收敛曲线

Comparison



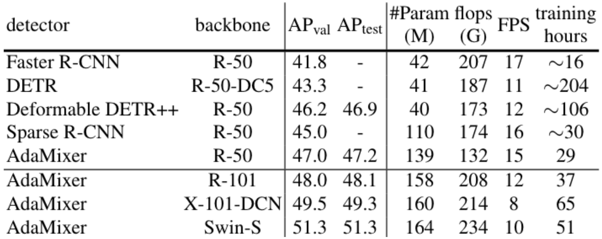
detector	backbone	encoder/pyramid net	#epochs	GFLOPs	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
DETR [4]	ResNet-50-DC5	TransformerEnc	500	187	43.3	63.1	45.9	22.5	47.3	61.1
SMCA [13]	ResNet-50	TransformerEnc	50	152	43.7	63.6	47.2	24.2	47.0	60.4
Deformable DETR [56]	ResNet-50	DeformTransEnc	50	173	43.8	62.6	47.7	26.4	47.1	58.0
Sparse R-CNN [39]	ResNet-50	FPN	**36**	174	45.0	63.4	48.2	26.9	47.2	59.5
Efficient DETR [49]	ResNet-50	DeformTransEnc	**36**	210	45.1	63.1	49.1	28.3	48.4	59.0
Conditional DETR [31]	ResNet-50-DC5	TransformerEnc	108	195	45.1	65.4	48.5	25.3	49.0	**62.2**
Anchor DETR [46]	ResNet-50-DC5	DecoupTransEnc	50	151	44.2	64.7	47.5	24.7	48.2	60.6
AdaMixer (ours)	ResNet-50	-	**12**	132	44.1	63.1	47.8	29.5	47.0	58.8
AdaMixer (ours)	ResNet-50	-	**24**	132	46.7	65.9	50.5	29.7	49.7	61.5
AdaMixer (ours)	ResNet-50	-	**36**	132	**47.0**	**66.0**	**51.1**	**30.1**	**50.2**	61.8
DETR [4]	ResNet-101-DC5	TransformerEnc	500	253	44.9	64.7	47.7	23.7	49.5	62.3
SMCA [13]	ResNet-101	TransformerEnc	50	218	44.4	65.2	48.0	24.3	48.5	61.0
Sparse R-CNN [39]	ResNet-101	FPN	**36**	250	46.4	64.6	49.5	28.3	48.3	61.6
Efficient DETR [49]	ResNet-101	DeformTransEnc	**36**	289	45.7	64.1	49.5	28.2	49.1	60.2
Conditional DETR [31]	ResNet-101-DC5	TransformerEnc	108	262	45.9	66.8	49.5	27.2	50.3	63.3
AdaMixer (ours)	ResNet-101	-	**36**	208	**48.0**	**67.0**	**52.4**	**30.0**	**51.2**	**63.7**
AdaMixer (ours)	ResNeXt-101-DCN	-	**36**	214	**49.5**	**68.9**	**53.9**	**31.3**	**52.3**	**66.3**
AdaMixer (ours)	Swin-S	-	**36**	234	**51.3**	**71.2**	**55.7**	**34.2**	**54.6**	**67.3**

Figure: Comparison

Contents

Conclusion and Limitation

- AdaMixer 改进了 query-based detectors 的 decoder，提出了自适应 3D 采样和自适应通道和空间混合，避免了在 backbone 和 decoder 之间引入额外的网络结构，降低了计算开销，加快了收敛速度。模型取得了优越的性能，特别是在小目标的检测上。
- 不足之处是，尽管使用了分组机制，但是仍保有了巨大的参数量。这是由于在线性层中生成动态混合权重产生了大量的参数。



detector	backbone	AP_{val}	AP_{test}	#Param (M)	flops (G)	FPS	training hours
Faster R-CNN	R-50	41.8	-	42	207	17	∼16
DETR	R-50-DC5	43.3	-	41	187	11	∼204
Deformable DETR++	R-50	46.2	46.9	40	173	12	∼106
Sparse R-CNN	R-50	45.0	-	110	174	16	∼30
AdaMixer	R-50	47.0	47.2	139	132	15	29
AdaMixer	R-101	48.0	48.1	158	208	12	37
AdaMixer	X-101-DCN	49.5	49.3	160	214	8	65
AdaMixer	Swin-S	51.3	51.3	164	234	10	51

谢谢大家！