StageInteractor: Query-based Object Detector with Cross-stage Interaction

Yao Teng[1] Haisong Liu[1] Sheng Guo[3] Limin Wang[1,2,✉]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, China
[2]Shanghai AI Lab, China [3]MYbank, Ant Group, China

A. Spatial Mixing

The process of applying filter reusing onto the spatial mixing is depicted in Fig. 1. It is very similar to the filter reusing on channel mixing, but the reused filter is used in the combination with the generated filter rather than cascade mixing. This combination is performed along the output dimension.

In the cascade mixing, the lightweight static linear layers are placed between the activation function and the dynamic mixing. Apart from the static channel mixing, the linear layers can also achieve the *efficient* spatial mixing, as shown in Code 1. Specifically, we split the sampling points into K groups, and perform affine transformation within and across groups, like [1]. The parameters cost on this operation is $(K^2 + (\frac{P_{\text{in}}^{(i)}}{K})^2) \cdot D_C^2$. Since the number of sampling points is set as the power of 2 and the number of spatial blocks K is set close to the square root of the number of sampling points, we use the formula $K = 2^{\lfloor \log_2 \sqrt{P_{\text{in}}^{(i)}} \rfloor}$ for calculation. Thus, an upper-bound of the parameter cost is $O(3P_{\text{in}}^{(i)} D_C^2)$, and thus this module is still more lightweight than those related to dynamic filter generation.

```
# K: spatial group size, P: the number of sampling points
# G: channel group size, Dc: channel dimension per group
# N: the number of queries
# I: the sampled feaures with shape (N*G, K, P//K, Dc)

I_1 = I.reshape(N*G*K, P//K*Dc)
I_2 = I.permute(0, 2, 1, 3).reshape(N*G*P//K, K*Dc)
I_1 = Linear(I_1).reshape(N*G, K, P//K, Dc)
I_2 = Linear(I_2).reshape(N*G, P//K, K, Dc)
I = I + I_1 + I_2.permute(0, 2, 1, 3)
I = ChannelMixing(I)
```

Code 1: PyTorch-like Pseudocode for the static mixing.

B. Feature Sampling

For the feature sampling, according to [4, 8, 2], we first generate a set of sampling points via content vectors, and then use these points to capture the desired image features

✉: Corresponding author (lmwang@nju.edu.cn).

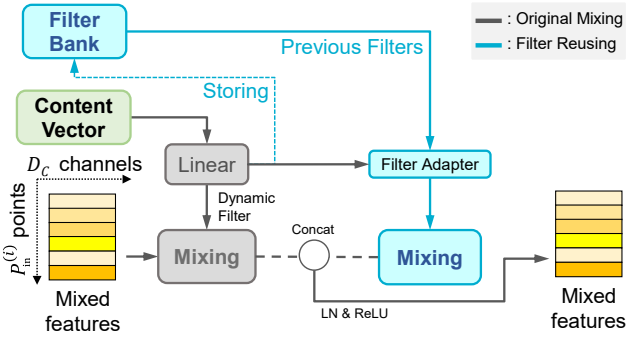


Figure 1: The overview of our spatial dynamic mixing.

with bilinear interpolation. Since the sampling points are organized into K groups, the feature sampler is correspondingly designed to generate points in groups. The PyTorch-like Pseudo-code is illustrated in Code 2. Specifically, we first use the content vectors to generate two sets of offsets to the positional vectors by linear layers. Then, the offsets is formed into the sampling points to extract features.

```
# K: spatial group size, P: the number of sampling points
# G: channel group size, N: the number of queries
# v: the content vector, b: the positional vector
# F(): the bilinear interpolation on multi-scale image features
# Im: the multi-scale image feaures, I: the sampled feaures

xy = b[..., 0:2], z = b[..., 2:3], r = b[..., 3:4]

p_1 = Linear(v).reshape(N*G, K, 1, 3)
p_2 = Linear(v).reshape(N*G, 1, P//K, 3)

dxy_1 = p_1[..., 0:2], dz_1 = p_1[..., 2:3]
dxy_2 = p_2[..., 0:2], dz_2 = p_2[..., 2:3]

p_xy = xy + 2**(z - 0.5*r) * (dxy_1 + 2**dz_1 * dxy_2)
p_z = z + dz_1 + dz_2
I = F(Im, p_xy, p_z)
```

Code 2: PyTorch-like Pseudocode of the sampler.

C. Additional Ablation Studies

The modules in the cascade mixing. Both the reused heavy dynamic filters and the lightweight static linear layers are crucial to our method. As shown in Tab. 1, only when these two mixing approaches are combined can the large performance gain be achieved. Moreover, as shown

Dynamic	Static	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
		44.1	62.3	47.6	25.5	47.5	60.3
	✓	43.5	61.7	46.8	25.3	47.0	59.0
✓		43.9	62.2	47.6	26.6	46.9	60.8
✓	✓	**44.8**	**63.0**	**48.4**	**27.5**	**48.0**	**61.3**

Table 1: The modules in the cascade mixing.

Static Mixing	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
	43.9	62.2	47.6	26.6	46.9	60.8
Channel	44.0	62.3	47.8	26.2	47.0	61.1
Spatial	43.7	61.8	47.2	25.7	47.0	59.9
Channel-spatial	**44.8**	**63.0**	**48.4**	**27.5**	**48.0**	**61.3**

Table 2: The type of static mixing in our detector.

Sampling	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
Vanilla	43.7	62.1	47.2	25.6	47.4	59.7
Group init.	44.1	62.3	47.9	26.2	47.0	60.8
Ours	**44.8**	**63.0**	**48.4**	**27.5**	**48.0**	**61.3**

Table 3: The type of feature sampling. `Vanilla` denotes the original feature sampling [4]. `Group-init.` means the group-wise initialization on the original sampler.

in Tab. 2, we find that inserting static channel-spatial aggregation into the lightweight linear layers is more beneficial than solely performing channel or spatial mixing.

Feature sampling. Different from the feature sampling in [4], the sampler in our detector is required to generate points in groups. Therefore, in this part, we explore whether the original feature sampling method (*i.e*. directly generating all sampling points) is feasible. As shown in Tab. 3, we report the results of our detector with vanilla feature sampling in the first line. Compared to the first line, the results in the last line show that our sampling is more compatible with our detector than 3D feature sampling. To find whether the weight initialization of 3D feature sampler causes this phenomenon, we modify the initialization of sampler so that its outputs at the first iteration are identical with the our sampler, and report the corresponding performance in the second line. The results are still worse than our sampling. Therefore, we speculate that our two-stage sampler is consistent with our dynamic mixing, thereby boosting the performance.

More sampling points in the first stage. As shown in Tab. 4, we conduct ablation studies on adding more sampling points of the first stage. The motivation is to use more points to cover the whole image as much as possible, enlarging the receptive field. Both of our baseline and our method can get slight benefits from more sampling points.

StageInter	$P_{\text{in}}^{(1)}$	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
	32	42.5	61.4	45.7	25.0	45.1	58.2
	64	42.6	61.4	45.7	24.4	45.7	58.2
✓	32	44.6	62.6	48.3	26.2	**48.1**	61.1
✓	64	**44.8**	**63.0**	**48.4**	**27.5**	48.0	**61.3**

Table 4: The number of sampling points at the first stage.

NMS	Threshold	AP	AP_{75}
✓	0.5	44.1 (-0.7)	47.1
✓	0.75	44.8 (+0.0)	48.4
✓	0.9	44.8 (+0.0)	48.5
-	-	44.8	48.4

Table 5: The performance of our StageInteractor with or without NMS.

Backbone	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
ResNet-50	49.0	67.4	53.7	30.2	51.7	62.3
ResNet-101	50.4	68.8	55.2	31.0	53.1	64.1
ResNeXt-101-DCN	51.3	70.1	56.0	32.1	53.9	65.2
Swin-S	52.7	71.8	57.7	33.3	55.1	67.1

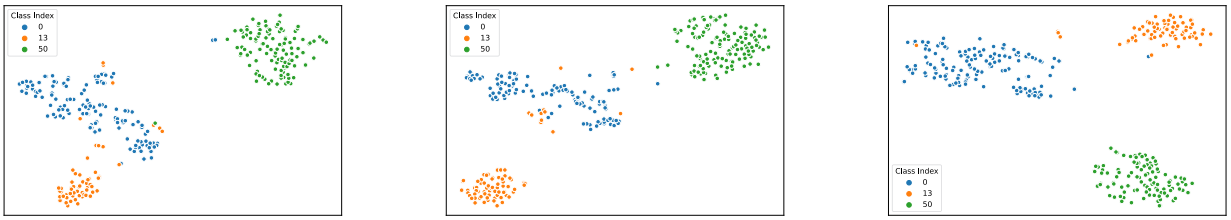
Table 6: The performance of StageInteractor on COCO `test-dev` set with 300 queries, 36 training epochs and single model single scale testing.

D. Duplicate Removal

According to [6], the strict one-to-one label assignment can ensure the object detector to have the ability to remove duplicate predictions. However, our cross-stage label assigner actually does not strictly follow one-to-one matching even in the last few stages, *i.e*., it has the potential to assign one ground-truth object to multiple predicted boxes on the classification task. Therefore, we explore whether our label assigner influences the performance of duplicate removal in query-based object detectors. As shown in Tab. 5, the results show that the performance our detector is relatively stable on AP with or without NMS. We consider this is because the coordinates of the most predicted boxes in the last few stages change little, and the operation of *gathering-and-selecting* labels in our assigner is performed adaptively.

E. MS COCO Test

As shown in Tab. 6, we report the performance of StageInteractor on COCO `test-dev` set. Here, the performance is evaluated with the same models that are used for the comparison with other state-of-the-art query-based detectors. Because the labels of COCO `test-dev` set are not publicly available, so the evaluation is performed on the online server.



(a) Baseline + CSLA. (b) Baseline + Reusing. (c) StageInteractor.

Figure 2: t-SNE [7] visualization of features of each query on MS COCO dataset [5] learned by various model variants. Each point denotes a feature vector, and colors denote different categories. `CSLA`: cross-stage label assignment.

F. Analysis about dynamic channel mixing

In vanilla AdaMixer [4], the FLOPs for the channel mixing is $B \times N \times G \times P_{\text{in}}^{(i)} \times (2D_C - 1) \times D_C$, whereas the FLOPs for generating a dynamic channel filter is $B \times N \times G \times (2D - 1) \times D_C \times D_C$. Therefore, the ratio between these two FLOPs is:

$$\frac{B \times N \times G \times (2D-1) \times D_C \times D_C}{B \times N \times G \times P_{\text{in}}^{(i)} \times (2D_C - 1) \times D_C} \approx 8 \quad (1)$$

Therefore, generating channel filters consumes more computational costs than performing channel mixing.

G. Qualitative Analysis

To verify the discriminability of our detector, we use t-SNE [7] visualization for the query features in various models. As depicted in Fig. 2, we select some representative categories with corresponding features to show the effectiveness of our structures. Compared with Fig. 2a and Fig. 2b, the distance between each group of categories is wider in Fig. 2c, and the points are more separate.

H. Limitation

Although our two cross-stage structures are effective, their designs are simple. In the future, we hope the following topics could be explored in the future: (1) the optimal designs of each decoder layer in a query-based detector; (2) more elaborate and effective cross-stage interactions; (3) the theoretical properties and the essence of the cascade structures.

I. Societal Impact

Object detection is a classical vision task and we adopt the open dataset: MS COCO [5], so there is no negative social impact if the method is used properly.

J. Model Implementation Details

Hyper-parameters. The cross-stage label assignment is performed on each stage, and its application scope is $[i-1, L]$. The threshold for selecting labels is set to 0.5. The reuse of dynamic filters do not perform on the first two decoder layers, and in other stages, all the generated filters for channel mixing are reused. The number of spatial blocks K is set close to the square root of the number of sampling points, *i.e.*, we use the formula $K = 2^{\lfloor \log_2 \sqrt{P_{\text{in}}^{(i)}} \rfloor}$ for calculation. Other parameters of our model are in line with [4]. Other parameters of our model are in line with the vanilla AdaMixer [4] and DETRs [3].

Initialization. Following [4], the initial weights of linear layers generating dynamic filters are set to zero, and the biases of these linear layers are initialized as expected. The initial weights of linear layers in the feature sampler are also set to zero, and the biases of these linear layers are initialized as follows: (1) the bias corresponding to $\mathrm{dxy\_1}$ in Code 2 is uniformly initialized within [-0.5, 0.5]. (2) the one corresponding to $\mathrm{dxy\_2}$ is uniformly initialized within [$-\frac{0.5}{\sqrt{2}}$, $\frac{0.5}{\sqrt{2}}$]. (3) The parts corresponding to the $\mathrm{dz\_1}$ and $\mathrm{dz\_2}$ are initialized as zeros. The initialization of other modules are set following [4, 3].

References

[1] Shoufa Chen, Enze Xie, Chongjian Ge, Ding Liang, and Ping Luo. Cyclemlp: A mlp-like architecture for dense prediction. *arXiv preprint arXiv:2107.10224*, 2021. 1

[2] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017. 1

[3] detrex contributors. detrex: An research platform for transformer-based object detection algorithms, 2022. 3

[4] Ziteng Gao, Limin Wang, Bing Han, and Sheng Guo. Adamixer: A fast-converging query-based object detector. In *Proceedings of the IEEE/CVF Conference on Computer Vi-*

sion and Pattern Recognition, pages 5364–5373, 2022. 1, 2, 3

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 3

[6] Peize Sun, Yi Jiang, Enze Xie, Wenqi Shao, Zehuan Yuan, Changhu Wang, and Ping Luo. What makes for end-to-end object detection? In *International Conference on Machine Learning*, pages 9934–9944. PMLR, 2021. 2

[7] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 3

[8] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020. 1