Supplementary Material – XMem: Long-Term Video Object Segmentation with an Atkinson-Shiffrin Memory Model

Ho Kei Cheng and Alexander G. Schwing

University of Illinois Urbana-Champaign
{hokeikc2, aschwing}@illinois.edu

This supplementary material is structured as follows:

- We first provide a more detailed analysis of the memory consolidation process (Sec. A).
- We then provide qualitative results, comparing the proposed XMem to baselines (Sec. B).
- We demonstrate failure cases (Sec. C).
- We compare different limits on the size of the long-term memory and illustrate the processing rate over the number of processed frames (Sec. D).
- We give quantitative results when retraining the STCN baseline in our training setting (Sec. E).
- We provide results on the YouTubeVOS 2019 validation set (Sec. F).
- We provide detailed quantitative results when XMem is trained on different datasets (Sec. G).
- We explain our multi-scale (MS) evaluation methodology and provide the corresponding enhanced performance of XMem (Sec. H).
- We outline an efficient implementation of the proposed anisotropic L2 similarity function (Sec. I).

A Visualizing Memory Consolidation

Here, we visualize the memory consolidation process (Section 3.3) by showing the candidate frames, some of the selected prototypes, and the corresponding aggregation weights (columns of $\mathbf{W}(\mathbf{k}^c, \mathbf{k}^p)$, each mapping to a distribution over all the candidates). Recall that $\mathbf{W}(\mathbf{k}^c, \mathbf{k}^p)$ is used to aggregate candidate values $\mathbf{v}^c$ into prototype values $\mathbf{v}^p$. Figure S1 and Figure S2 show two examples. As illustrated in Figure 5 of the main paper we observe semantically meaningful regions to be grouped.

Fig. S1. Visualization of memory consolidation. The first row shows the candidate frames to be converted into long-term memory. Each of the following rows show a prototype position (indicated by a yellow cross), and the corresponding aggregation weights (visualized as a red overlay). Frames that contain a prototype are framed in red. The consolidation process aggregates information from semantically meaningful regions (top-to-bottom): the swan's beak, part of the vegetation, part of the riverbank, the transition between vegetation and river bank, and part of the water surface.

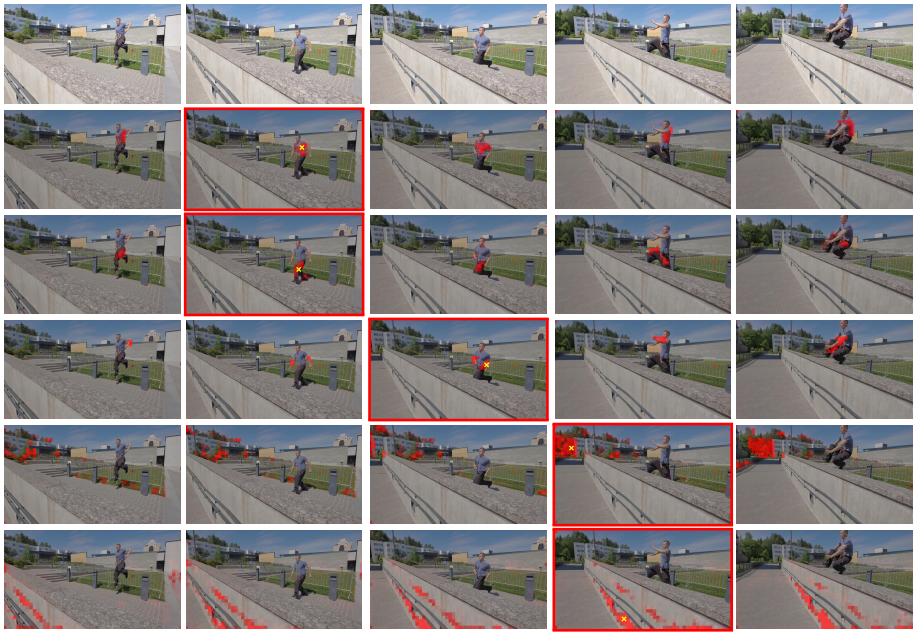

Fig. S2. Visualization of memory consolidation. The first row shows the candidate frames to be converted into long-term memory. Each of the following rows show a prototype position (indicated by a yellow cross), and the corresponding aggregation weights (visualized as a red overlay). Frames that contain a prototype are framed in red. The consolidation process aggregates information from semantically meaningful regions (top-to-bottom): torso, legs, arms, trees, and part of the wall.

B Qualitative Results

Here, we compare qualitatively to JOINT [8], AFB-URR [7], and STCN [4] using several long videos and using the same setting as in the paper. We show results on the `dressage` sequence (10,767 frames) which is part of the Long-time Video (3×) dataset [7], and two additional in-the-wild videos. `breakdance` contains a single foreground object with large and fast motion, and has 18,187 frames. `cans` is very challenging, contains five different objects, two of which (Dr. Pepper and Coca-Cola cans) are similar. The two cans are completely occluded for more than 2,000 frames, and our method can successfully capture them when they reappear. Figure S3, S4, and S5 compare results on these videos respectively. We show one potential application where an image layer is inserted between the foreground and the background using the predicted object mask on a snippet of the `breakdance` sequence.

Fig. S3. Results on the `dressage` sequence. JOINT [8] uses a temporally local feature window and loses track over time. AFB-URR [7] is stable but produces overall less accurate segmentations. STCN [4] uses a low memory insertion frequency to avoid memory explosion, and thus misses fast changes (2nd and 4th column). Ours is sometimes better than the provided ground-truth (last column, the horse's front legs).

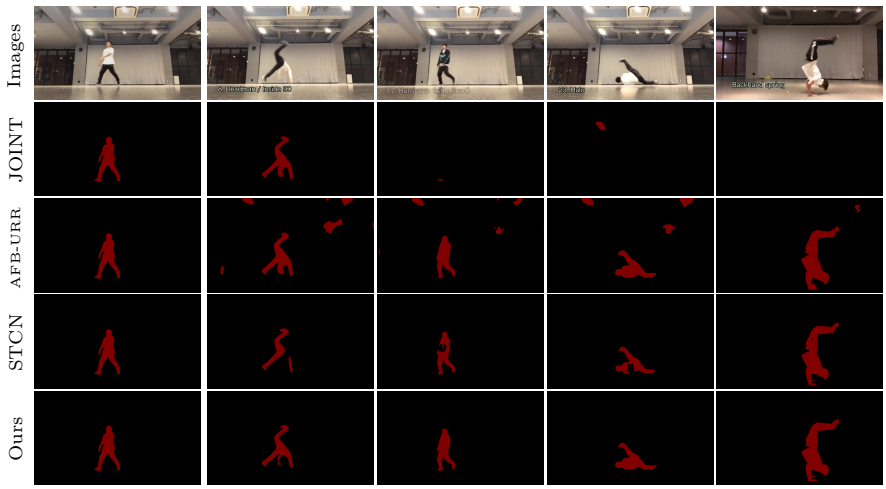

Fig. S4. Results on the `breakdance` sequence. We manually annotated the first frame as input. Similarly to the `dressage` sequence (Figure S3), JOINT [8] loses track over time, AFB-URR [7] is overall less accurate, STCN [4] struggles with fast motion, and our method performs well on this sequence.

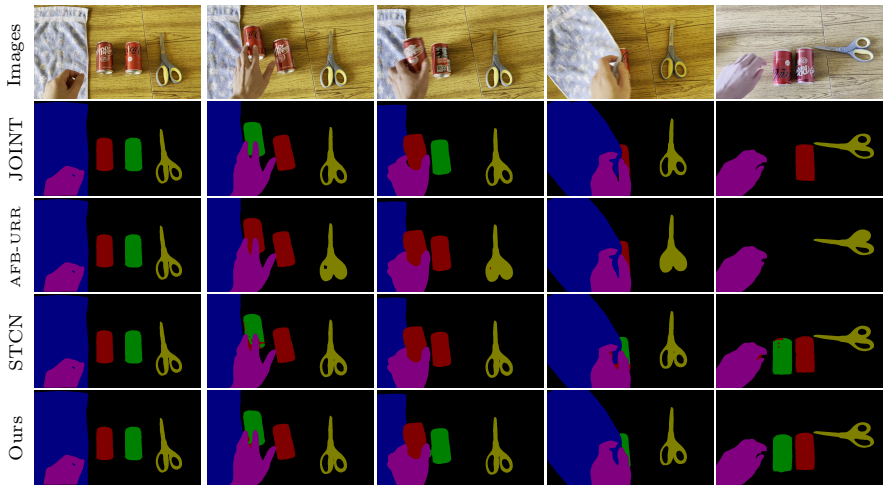

Fig. S5. Results on the `cans` sequence. We manually annotated the first frame as input. The Dr. Pepper can is labeled with red, and the Coca-Cola can is labeled with green. The two cans are completely occluded with a towel after frame 1,793, and reappear about 2,000 frames later. The color tone change is due to the camera's auto white balance. JOINT [8] misses the Coca-Cola can after occlusion. It still captures the Dr. Pepper can as the available first reference frame helps. AFB-URR [7] mixes up the two cans early on, and fails to capture them after they reappear. This is due to its eager feature compression and thus lower modeling capability. STCN [4] uses a low memory insertion frequency to avoid memory explosion which causes it to be less accurate when changes happen. Our method is the most accurate overall.

C Failure Cases

As mentioned in the limitation section of the main text, our method struggles with very fast moving objects. This is because even the fastest-updating sensory memory fails to track such objects, and the working memory fails to model objects with large motion blur. Figure S6 visualizes some failure cases.

Fig. S6. Failure cases. We point to objects of interest with an arrow. First row: multiple birds with similar appearances are flying. We fail to discriminate between some birds that are close to each other. Second row: a frisbee is being thrown. We cannot catch up as it is moving quickly with a large motion blur. Third row: two flags are being waved quickly. We fail to segment the whole left flag due to fast motion.

D Long-Term Memory Size and FPS Scaling

By default, we use a maximum long-term memory size of 10,000 which consumes a small amount of GPU memory and is reasonably capable – it can store information from around 3,900 frames after memory consolidation ($r = 10$). In practice, users might opt for a different upper limit of the long-term memory (LT_{max}), in consideration of any memory constraints, speed, and the complexity of the video. Here, we test the performance of different LT_{max} settings on the Long-time Video (3×) dataset [7] and show the results in Table S1. There is significant memory saving and speed-up when LT_{max} is decreased. While a smaller LT_{max} seems to be sufficient for this dataset, we expect using a higher LT_{max} can benefit more challenging videos with long-term occlusions.

We also plot the single-object FPS (1/time required to process a new frame) against the total number of processed frames for STCN [4] and different LT_{max} settings of XMem in Figure S7. We use a high-capacity 32GB V100 GPU in this experiment such that STCN can be run without out-of-memory errors. FPSs for XMem plateaue after reaching LT_{max}.

Table S1. Performance of XMem with different upper limits of the long-term memory LT_{max} on the Long-time Video (3×) dataset [7].

LT_{max}	$\mathcal{J}\&\mathcal{F}$	Max. GPU memory	FPS
500	87.2±4.7	1168 MB	35.3
1,000	89.5±0.3	1186 MB	34.1
2,500	89.8±0.2	1243 MB	31.1
5,000	89.9±0.2	1332 MB	27.5
10,000	90.0±0.4	1515 MB	23.4
20,000	90.0±0.4	1632 MB	21.1
30,000	90.0±0.4	1632 MB	20.9

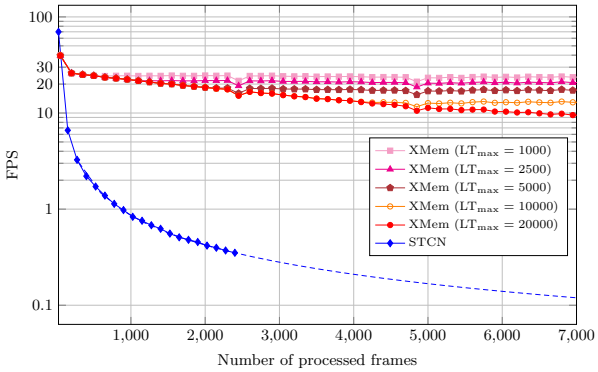

Fig. S7. FPS scaling of STCN [4] vs. variants of XMem. STCN starts off faster due to its simpler construction but slows down drastically as its memory bank expands. As STCN soon becomes too slow for practical use, we estimate its FPS by fitting a linear function to its processing time (i.e., inverse linear to FPS). This linear function is illustrated with a dashed line. XMem maintains a relatively stable and fast FPS throughout, thanks to our memory consolidation algorithm.

E Re-training STCN

We have changed the training schedule (see Section 3.6, Implementation Details) and adjusted parts of the network (including removing some convolutional layers and adding a feature fusion block [4] to the decoder for incorporating the sensory memory). For a fair comparison, we re-train the STCN [4] baseline under our setting. This is equivalent to removing the sensory memory, long-term memory, and both scaling terms. BL30K [3] is not used. Table S2 tabulates the results on the YouTubeVOS 2018 validation set [15] and the DAVIS 2017 validation set [11]. The re-trained method achieves a 1.2 higher $\mathcal{J}\&\mathcal{F}$ on DAVIS and a 1.0 higher $\mathcal{G}$ on YouTubeVOS. On average, the change is insignificant, i.e., our training schedule is not a sufficient condition for improved results.

Table S2. We compare the performance of STCN [4] to the re-trained version with our training setup. On average, there is no significant difference. The performance of XMem is provided as a reference.

Method	YouTubeVOS 2018 val $\mathcal{G}$	DAVIS 2017 val $\mathcal{J}\&\mathcal{F}$
STCN [4] (original)	83.0	85.4
STCN [4] (re-trained)	84.0	84.2
XMem (Ours)	**85.7**	**86.2**

F Results on YouTubeVOS 2019 validation

Table S3 tabulates our results on the YouTubeVOS [15] 2019 validation set. We compare the measured FPS on the 2018 version. The FPS on these two versions are highly correlated as their average number of objects and video length are similar.

Table S3. Quantitative comparisons on YouTubeVOS 2019 validation.

	YouTubeVOS 2019 val [15]					
Method	$\mathcal{G}$	$\mathcal{J}_s$	$\mathcal{F}_s$	$\mathcal{J}_u$	$\mathcal{F}_u$	FPS_{Y18}
CFBI [16]	81.0	80.6	85.1	75.2	83.0	3.4
SST [5]	81.8	80.9	-	76.7	-	-
MiVOS* [3]	82.4	80.6	84.7	78.1	86.4	-
HMMN [12]	82.5	81.7	86.1	77.3	85.0	-
CFBI+ [18]	82.6	81.7	86.2	77.1	85.2	4.0
STCN [4]	82.7	81.1	85.4	78.2	85.9	13.2
JOINT [8]	82.8	80.8	84.8	79.0	86.6	-
STCN* [4]	84.2	82.6	87.0	79.4	87.7	13.2
AOT [17]	85.3	83.9	88.8	79.9	88.5	6.4
XMem (Ours)	85.5	84.3	88.6	**80.3**	88.6	**22.6**
XMem* (Ours)	**85.8**	**84.8**	**89.2**	**80.3**	**88.8**	**22.6**

G Results with Different Training Datasets

Following prior works [9], we first pretrain our network on static images. As in the implementation of [3,4], we use a mix of single object datasets [13,14,6,19,2]. We compare with prior works that do not use pretraining in Table S4. We additionally present detailed results of our method when it is trained on 1) DAVIS 2017 [10] only, 2) YouTubeVOS 2019 [15] only, and 3) a mix of both, in the followings tables.

Table S4. Comparisons with methods without static image pretraining.

Method	Y_{18}	Y_{19}	D_{16}	D_{17}	D_{17td}	FPS_{D17}
LWL [1]	81.5	81.0	-	81.6	-	-
SST [5]	81.7	81.8	-	82.5	-	-
CFBI+ [18]	82.0	82.6	89.9	82.9	78.0	5.6
JOINT [8]	83.1	82.8	-	83.5	-	6.8
Ours$^-$	**84.3**	**84.2**	**90.8**	**84.5**	**79.8**	**20.2**

Table S5. Performance of XMem on DAVIS 2016 with different training data.

Training data	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$
DAVIS only	87.8	86.7	88.9
DAVIS+YouTubeVOS only	90.8	89.6	91.9
Static+DAVIS+YouTubeVOS	91.5	90.4	92.7
Static+BL30K+DAVIS+YouTubeVOS	92.0	90.7	93.2

Table S6. Performance of XMem on DAVIS 2017 validation with different training data.

Training data	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$
DAVIS only	76.7	74.1	79.3
DAVIS+YouTubeVOS only	84.5	81.4	87.6
Static+DAVIS+YouTubeVOS	86.2	82.9	89.5
Static+BL30K+DAVIS+YouTubeVOS	87.7	84.0	91.4

Table S7. Performance of XMem on DAVIS 2017 test-dev with different training data.

Training data	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$
DAVIS only	64.8	61.4	68.1
DAVIS+YouTubeVOS only	79.8	76.3	83.4
Static+DAVIS+YouTubeVOS	81.0	77.4	84.5
Static+BL30K+DAVIS+YouTubeVOS	81.2	77.6	84.7

Table S8. Performance of XMem on YouTubeVOS 2018 validation with different training data.

Training data	$\mathcal{G}$	$\mathcal{J}_s$	$\mathcal{F}_s$	$\mathcal{J}_u$	$\mathcal{F}_u$
YouTubeVOS only	84.4	83.7	88.5	78.2	87.2
DAVIS+YouTubeVOS only	84.3	83.9	88.8	77.7	86.7
Static+DAVIS+YouTubeVOS	85.7	84.6	89.3	80.2	88.7
Static+BL30K+DAVIS+YouTubeVOS	86.1	85.1	89.8	80.3	89.2

Table S9. Performance of XMem on YouTubeVOS 2019 validation with different training data.

Training data	$\mathcal{G}$	$\mathcal{J}_s$	$\mathcal{F}_s$	$\mathcal{J}_u$	$\mathcal{F}_u$
YouTubeVOS only	84.3	83.6	88.0	78.5	87.1
DAVIS+YouTubeVOS only	84.2	83.8	88.3	78.1	86.7
Static+DAVIS+YouTubeVOS	85.5	84.3	88.6	80.3	88.6
Static+BL30K+DAVIS+YouTubeVOS	85.8	84.8	89.2	80.3	88.8

H Multi-scale Evaluation

Multi-scale evaluation is a general trick used in segmentation tasks to boost accuracy by combining results from augmented inputs. Common augmentations include scale-change or vertical mirroring. Here, we show XMem's results with multi-scale evaluation as an attempt to achieve the best performance with a single model without retraining or using a better backbone. For these results, we use $P = 512$ for a relaxed compression. Vertical mirroring is used. Different augmentations are processed independently and the output probability maps are simply averaged.

On DAVIS, we note that a single large scale (720p) is better than merging multiple smaller scales. We use $r = 3$ for better results which has also been noted in STCN [4]. In the test-dev set, we additionally include results with $r = 5$ (i.e., multi-temporal-scale) in the merge. Table S10 tabulates our results.

Table S10. XMem with/without multi-scale evaluation on DAVIS. ‡: 600p evaluation.

	DAVIS 2017 val			DAVIS 2016 val			DAVIS 2017 test-dev		
Method	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$
XMem	86.2	82.9	89.5	91.5	90.4	92.7	81.0	77.4	84.5
XMem*	87.7	84.0	91.4	92.0	90.7	93.2	81.2	77.6	84.7
XMem*‡	-	-	-	-	-	-	82.5	79.1	85.8
XMem (MS)	88.2	85.4	91.0	92.7	92.0	93.5	83.1	79.7	86.4
XMem* (MS)	**89.5**	**86.3**	**92.6**	**93.3**	**92.2**	**94.4**	**83.7**	**80.5**	**87.0**

On YouTubeVOS, we adopt multiple scales: $\{480, 528, 576, 624\}$. Unlike on DAVIS, we find larger scales to be unhelpful – which might be due to the overall less accurate annotation of YouTubeVOS. We do not adopt multiple temporal scales here. Table S11 tabulates our results.

Table S11. XMem with/without multi-scale evaluation on YouTubeVOS.

	YouTubeVOS 2018 val					YouTubeVOS 2019 val				
Method	$\mathcal{G}$	$\mathcal{J}_s$	$\mathcal{F}_s$	$\mathcal{J}_u$	$\mathcal{F}_u$	$\mathcal{G}$	$\mathcal{J}_s$	$\mathcal{F}_s$	$\mathcal{J}_u$	$\mathcal{F}_u$
XMem	85.7	84.6	89.3	80.2	88.7	85.5	84.3	88.6	80.3	88.6
XMem*	86.1	85.1	89.8	80.3	89.2	85.8	84.8	89.2	80.3	88.8
XMem (MS)	86.7	85.3	89.9	**81.7**	89.9	86.4	84.9	89.2	**81.8**	89.8
XMem* (MS)	**86.9**	**85.6**	**90.3**	**81.7**	**90.2**	**86.8**	**85.5**	**89.8**	**81.8**	**89.9**

I Implementation of the Anisotropic L2 Similarity

STCN [4] decomposes the L2 similarity into a sequence of tensor operations for a memory- and compute-efficient implementation. For the proposed similarity function to be practical, a similar decomposition is required. Here, we derive and outline our implementation. Recall the definition of the anisotropic L2 similarity:

We are given key $\mathbf{k} \in \mathbb{R}^{C^{\mathrm{k}} \times N}$, value $\mathbf{v} \in \mathbb{R}^{C^{\mathrm{v}} \times N}$, and query $\mathbf{q} \in \mathbb{R}^{C^{\mathrm{k}} \times HW}$. The key is associated with a shrinkage term $\mathbf{s} \in [1, \infty)^N$ and the query is associated with a selection term $\mathbf{e} \in [0, 1]^{C^{\mathrm{k}} \times HW}$. Then, the similarity between the i-th key element and the j-th query element is computed via

$$\mathbf{S}(\mathbf{k}, \mathbf{q})_{ij} = -\mathbf{s}_i \sum_c^{C^{\mathrm{k}}} \mathbf{e}_{cj} \left(\mathbf{k}_{ci} - \mathbf{q}_{cj}\right)^2, \tag{S1}$$

which equates to the original L2 similarity [4] if $\mathbf{s}_i = \mathbf{e}_{cj} = 1$ for all i, j, and c.

We use ":" to denote all the elements in a dimension and "@" to denote a singleton dimension to be broadcasted.[1] $\odot$ denotes the Hadamard (element-wise) product. $\mathbf{1}$ is an all-ones row vector with length C^{k}.

$$\begin{aligned}
\mathbf{S}(\mathbf{k}, \mathbf{q})_{ij} &= -\mathbf{s}_i \sum_c^{C^{\mathrm{k}}} \mathbf{e}_{cj} \left(\mathbf{k}_{ci} - \mathbf{q}_{cj}\right)^2 \\
&= -\mathbf{s}_i \left(\sum_c^{C^{\mathrm{k}}} \mathbf{e}_{cj} \mathbf{k}_{ci}^2 - \sum_c^{C^{\mathrm{k}}} 2\mathbf{e}_{cj} \mathbf{k}_{ci} \mathbf{q}_{cj} + \sum_c^{C^{\mathrm{k}}} \mathbf{e}_{cj} \mathbf{q}_{cj}^2 \right) \\
&= -\mathbf{s}_i \left((\mathbf{k}_{:i} \odot \mathbf{k}_{:i})^T \mathbf{e}_{:j} - 2\mathbf{k}_{:i}^T (\mathbf{e}_{:j} \odot \mathbf{q}_{:j}) + \mathbf{1} (\mathbf{e}_{:j} \odot \mathbf{q}_{:j} \odot \mathbf{q}_{:j}) \right) \\
\Rightarrow \mathbf{S}(\mathbf{k}, \mathbf{q}) &= \mathbf{s}_{:@} \left(-(\mathbf{k} \odot \mathbf{k})^T \mathbf{e} + 2\mathbf{k}^T (\mathbf{e} \odot \mathbf{q}) - \mathbf{1} (\mathbf{e} \odot \mathbf{q} \odot \mathbf{q}) \right).
\end{aligned} \tag{S2}$$

This gives a fully vectorized implementation consisting of only element-wise operations and matrix multiplications with broadcasting.

[1] Boardcasting as in numpy.

References

1. Bhat, G., Lawin, F.J., Danelljan, M., Robinson, A., Felsberg, M., Van Gool, L., Timofte, R.: Learning what to learn for video object segmentation. In: ECCV (2020)
2. Cheng, H.K., Chung, J., Tai, Y.W., Tang, C.K.: Cascadepsp: Toward class-agnostic and very high-resolution segmentation via global and local refinement. In: CVPR (2020)
3. Cheng, H.K., Tai, Y.W., Tang, C.K.: Modular interactive video object segmentation: Interaction-to-mask, propagation and difference-aware fusion. In: CVPR (2021)
4. Cheng, H.K., Tai, Y.W., Tang, C.K.: Rethinking space-time networks with improved memory coverage for efficient video object segmentation. In: NeurIPS (2021)
5. Duke, B., Ahmed, A., Wolf, C., Aarabi, P., Taylor, G.W.: Sstvos: Sparse spatiotemporal transformers for video object segmentation. In: CVPR (2021)
6. Li, X., Wei, T., Chen, Y.P., Tai, Y.W., Tang, C.K.: Fss-1000: A 1000-class dataset for few-shot segmentation. In: CVPR (2020)
7. Liang, Y., Li, X., Jafari, N., Chen, J.: Video object segmentation with adaptive feature bank and uncertain-region refinement. In: NeurIPS (2020)
8. Mao, Y., Wang, N., Zhou, W., Li, H.: Joint inductive and transductive learning for video object segmentation. In: ICCV (2021)
9. Oh, S.W., Lee, J.Y., Xu, N., Kim, S.J.: Video object segmentation using space-time memory networks. In: ICCV (2019)
10. Perazzi, F., Pont-Tuset, J., McWilliams, B., Van Gool, L., Gross, M., Sorkine-Hornung, A.: A benchmark dataset and evaluation methodology for video object segmentation. In: CVPR (2016)
11. Pont-Tuset, J., Perazzi, F., Caelles, S., Arbeláez, P., Sorkine-Hornung, A., Van Gool, L.: The 2017 davis challenge on video object segmentation. In: arXiv:1704.00675 (2017)
12. Seong, H., Oh, S.W., Lee, J.Y., Lee, S., Lee, S., Kim, E.: Hierarchical memory matching network for video object segmentation. In: ICCV (2021)
13. Shi, J., Yan, Q., Xu, L., Jia, J.: Hierarchical image saliency detection on extended cssd. In: TPAMI (2015)
14. Wang, L., Lu, H., Wang, Y., Feng, M., Wang, D., Yin, B., Ruan, X.: Learning to detect salient objects with image-level supervision. In: CVPR (2017)
15. Xu, N., Yang, L., Fan, Y., Yue, D., Liang, Y., Yang, J., Huang, T.: Youtube-vos: A large-scale video object segmentation benchmark. In: ECCV (2018)
16. Yang, Z., Wei, Y., Yang, Y.: Collaborative video object segmentation by foreground-background integration. In: ECCV (2020)
17. Yang, Z., Wei, Y., Yang, Y.: Associating objects with transformers for video object segmentation. In: NeurIPS (2021)
18. Yang, Z., Wei, Y., Yang, Y.: Collaborative video object segmentation by multi-scale foreground-background integration. In: PAMI (2021)
19. Zeng, Y., Zhang, P., Zhang, J., Lin, Z., Lu, H.: Towards high-resolution salient object detection. In: ICCV (2019)