Accelerating Video Object Segmentation with Compressed Video

Kai Xu Angela Yao
National University of Singapore
{kxu, ayao}@comp.nus.edu.sg

Contents

Algorithm Details

Given a compressed video sequence of length T, let $\{i, i \in [1,T]\}$ be the natural presentation index of the frames in time. Additionally, we denote with $\mathbf{\Gamma} = \{\Gamma_i, i \in [1,T]\}$ the decoding order, where Γ_i is the decoding index for frame i. Recall that $\{(I_i, M_i, \mathbf{e}_i), i, i \in [1,T]\}$ is the reconstructed image sequence, where I_i is the reconstructed RGB image of frame i and $\{(M_i, \mathbf{e}_i)\}$ are the associated motion fields and residuals. $\mathcal{I}$, $\mathcal{P}$, $\mathcal{B}$ are sets of I-, P- and B-frame indices, respectively. $W_{MV}(\cdot)$ is the motion vector warping operation defined in Eq. 6 and Eq. 7 of the main paper. The overall pipeline can be summarized in Algorithm 1.

Algorithm 1 Compressed video object segmentation.

Model: Base Model $\{F, G\}$, Decoder Model $\mathcal{D}$.
Input: Decoding order $\{\Gamma_t, t \in [1,T]\}$; Reconstructed RGB $\{(I_i), i \in [1,T]\}$; Motion fields and residuals $\{(M_i, \mathbf{e}_i), i \in [1,T]\}$; Residual binarization threshold τ; $\mathcal{I}$, $\mathcal{P}$, $\mathcal{B}$ are sets of indices for I-, P- and B-frames respectively. ψ denotes *softmax aggregation* which is used in RGMP [3]. $\mathbb{1}[\cdot]$ is indicator function.

for $i = \Gamma_1, ..., \Gamma_T$ **do**
  if $i \in \mathcal{I}$ *or* $\mathcal{P}$ **then**
    Get key frame's softmax prediction from base model: $P_i = G(F(I_i))$, low level features: $V_i = F(I_i)$.
  else
    Propagation:
      Warp predictions: $\hat{P}_i = W_{MV}(M_i, P_*)$
        Buffer for multi-hop reference: $P_* \leftarrow \hat{P}_i$
      Warp features: $\hat{V}_i = W_{MV}(M_i, V_*)$
    Confidence Re-weighting:
      Compute features: $V_i = F(I_i)$
      Compute confidence-weighted prediction: $\dot{P}_i = S(V_i, \hat{V}_i) \cdot \hat{P}_i$
    Correction:
      Binarize residual: $\mathbf{e}_b = |greyscale(\mathbf{e})| > \tau$
      Get warped foreground mask: $\hat{\mathbf{S}} = \mathbb{1}[argmax(\psi(\hat{P})) > 0]$
      Get dilation of the foreground mask: $\hat{\mathbf{S}}_+ = dilate(\hat{\mathbf{S}})$
      Get selected pixels to correct: $\tilde{\mathbf{S}} = U(\mathbf{e}_b, \hat{\mathbf{S}}_+)$
      Feature matching: $\bar{P}_i = \text{feature-matching}(\tilde{\mathbf{S}}, P_{k*}, V_i, V_{k*}) + \hat{P}_i$
    Decoder:
      $P_i = \mathcal{D}([\hat{P}_i, V_i, \dot{P}_i, \bar{P}_i])$

Output: All segmentations $\mathcal{S} = argmax(\psi(P_i))$

Additional Implementation Details

Low-level feature encoder F: With MiVOS [1] as the base model, we use the layers before the second residual stage of their RGB encoder (ResNet 50) as the low-level feature encoder. The encoder outputs features of 1/4 spatial size and 256 channels.

Training: All training data is generated with MiVOS as the base model. We generate and store the warped features, warped predictions, and residuals in advance using the weights for the model released by the authors. We use a pixel-wise *cross entropy* loss for training. The training is performed on an RTX A5000 GPU for half a day. To reuse the weights on other base models, we simply compute the low-level features of the keyframes with the low-level feature encoder from MiVOS. The corresponding overhead is already included in the reported times.

Soft Motion Vector Propagation Module

Decoder $\mathcal{D}$: The lightweight decoder in the soft propagation module as given in Equation 9 of the main paper is composed of two convolutional layers and three residual layers. Specifically, it has

```
Conv2d(289,128)->Conv2d(128,64)->
ResBlock(64,64)->ResBlock(64,64)->
ResBlock(64,64)->Conv2d(64,11)->Upsampling.
```

For all the layers, the kernel size is 3 with a padding of 1. We use BasicBlock of ResNet18 here as the ResBlock without the batch normalization. Final predictions are upsampled by 4 with bi-linear interpolation.

Residual Correction Module

Feature matching $\tilde{\mathbf{S}}$ provides an indication of which areas in the propagated mask will require correction. For each pixel in $\tilde{\mathbf{S}}$ indexed by a at frame n, we search in the temporally closest keyframe k^* and match between V_n and V_{k^*}. Specifically, we define $\mathbf{W}^{ak}$ as the affinity between the feature at pixel a in V_n, *i.e.* V_n^a, and all pixels in V_{k^*}. The corrected mask prediction at pixel a is then obtained by $P_n^a = \mathbf{W}^{ak} P_{k^*}$. Following STCN [2], we use an L2-similarity function to compute the affinity matrix, where $\mathbf{W}^{ak} \in \mathbb{R}^{1\times HW}$:

$$(\mathbf{W}^{ak})^b = \frac{f(V_n^a, V_{k^*}^b)}{\sum_m (f(V_n^a, V_{k^*}^m))}. \tag{1}$$

f is the L2 similarity function:

$$f(V_n^a, V_{k^*}^b) = exp(-||V_n^a - V_{k^*}^b||_2^2). \tag{2}$$

Timing Analysis

As analyzed in Section 5.4, we estimate the amortized per frame inference time via

$$T_{\text{base}} \cdot R + (T_{\text{propagation}} + T_{\text{correction}}) \cdot (1-R), \tag{3}$$

where R denotes the ratio of keyframe. T_{base} denotes the inference times of the base segmentation model, while $T_{\text{propagation}}$ and $T_{\text{correction}}$ denote the time for motion-vector-based propagation, and time for residual correction, respectively. We measured the propagation and correction time on DAVIS17 with an RTX-2080Ti, and the sum $(T_{\text{propagation}} + T_{\text{correction}})$ is 12ms.

Base model timing T_{base}

One of the key variable components in the timing is T_{base}. Our reported values do not match the published FPS of the respective works for the base models. When only keyframes are fed into the base model, the T_{base} for fine-tuning-based models will be higher but shorter for memory-based models. We elaborate on the details below.

Fine-tuning based model contains two components for timing: online fine-tuning and segmentation. Applying the model to fewer frames, *i.e.* only on the keyframes, will reduce the segmentation time, but the online fine-tuning time remains the same. This leads to a higher T_{base} than the published FPS, as online fine-tuning dominates and is less amortized over the various frames. Table 1 shows T_{base} for a fine-tuning based base model FRTM [4]. Adding on our framework is less ideal, because the overall FPS largely depends on how much time the model spends on online fine-tuning, and reducing the number of frames processed by the base model does not provide much speed-up.

Table 1. FRTM [4] base network timing analysis on DAVIS17

Frames used by base model FRTM [4]	T_{base}(ms)
All frames in sequence	71.0
Only keyframes (37.2% of all frames)	115.3

Memory-based models also have two components for timing: memory reading/update and segmentation. Using fewer keyframes can reduce both components. The total memorized frames are directly proportional to the total number of frames applied to the base model multiplied by the update frequency. Keeping the same update frequency as the original base model results in a memory proportional to the percentage of keyframes. This is equivalent to increasing the interval between the stored frames in the memory for the original video sequence.

On DAVIS17, decreasing the update frequency reduces the keyframe accuracy. As shown in Table 2, keeping the same update frequency for keyframes uses only 36% of default memory, but results in a near 0.5 point drop on both $\mathcal{J}$ and $\mathcal{F}$ scores. Yet, updating with every keyframe would exceed the original frequency interval used during training. This incompatibility causes the accuracy to drop. Additionally, it exceeds the original memory by 81% and is nearly 46ms slower than the default memory setting. In our reported result in Section 5.2 of the main paper, we make a trade-off between efficiency and accuracy and update every two keyframes. This gives base times and a memory size approximately equal to the original base model.

In Table 3, we find that for YouTube-VOS, updating the memory every five keyframes achieves comparably accurate segmentations. As the original base model has a setting of updating every five frames over the entire sequence, this is the setting we choose for our results in Section 5.2 of the main paper. Updating every keyframe would result in 35% more memory and 49ms slower at segmentation.

There is a difference in the memory update frequency between DAVIS and the YouTube-VOS dataset mainly be-

Table 2. Time and accuracy analysis on DAVIS17 on RTX 2080Ti (default encoder settings, 36.1% keyframes). The star denotes the setting used for our main result in Section 5.2. The frame % in memory is tabulated as a percentage of the original base model when applied to all frames of the sequence.

Model	Update frequency	Frame % in memory	T_{base}(ms)	$\mathcal{J}$	$\mathcal{F}$
MiVOS [1]	every 5 frames	100% (by default)	89.3	81.7	87.4
MiVOS on keyframes	every keyframes	181%	134.8	78.9	83.9
MiVOS on keyframes	every 2 keyframes*	90%	87.9	79.7	84.6
MiVOS on keyframes	every 3 keyframes	60%	78.9	79.3	84.2
MiVOS on keyframes	every 5 keyframes	36%	73.8	79.1	84.1

Table 3. Time analysis for base model on the first 30 sequences of YouTube-VOS on RTX A5000 (default encoder preset, 27.2% keyframes). We do not have access to the per-frame segmentation accuracies as evaluation scores and tallied privately over a test server. Here, we report the combined score $\mathcal{G}$ averaged over all the frames for all sequences. The star denotes the setting used for our main result in Section 5.2. The frame % in memory is tabulated as a percentage of the original base model when applied to all frames of the sequence.

Model	Update frequency	Frames % in memory	T_{base}(ms)	$\mathcal{G}$
MiVOS [1]	every 5 frames	100% (by default)	77.0	82.6
MiVOS on keyframes	every keyframe	135%	96.3	79.3
MiVOS on keyframes	every 2 keyframe	68%	58.3	79.4
MiVOS on keyframes	every 3 keyframe	45%	49.6	79.5
MiVOS on keyframes	every 5 keyframes*	27%	47.0	79.3

cause 1) YouTube-VOS has a higher frame rate than DAVIS (30FPS vs. 25FPS), and 2) YouTube-VOS is less dynamic in terms of object appearance. The I-/P-frames ratio between YouTube-VOS and DAVIS is 27.2% and 36.1%, respectively. Under the same encode setting, YouTube-VOS results in 10% fewer I-/P-frames than DAVIS. Lower temporal density makes YouTube-VOS require less frequent memory updating.

Table 4. Warping method towards target objects of different sizes on propagated frames on DAVIS17. Each entry shows $\mathcal{J}\&\mathcal{F}$ scores of non-keyframes. Keyframes are selected under the default encoder preset.

	DAVIS17		
	small	medium	large
Optical Flow	60.9	75.1	81.0
MV to Flow	60.0	73.0	77.8
Ours	72.7	89.7	91.9
Base model without propagation [1]	76.8	91.9	92.3

Impact of Object Size

To better understand the capabilities of mask propagation using motion vector warping, we further evaluate according to different sized objects. We split the object masks into three categories based on their total pixels and report the segmentation accuracy in Table 4 according to object size. Small objects have less than 10k pixels, *i.e*. approximately 100×100 or smaller, medium objects are between 10k and 30k pixels, *i.e*. between approximately 100×100 and 170×170 pixels, and large objects have more than 30k pixels, *i.e*. larger than 170×170 pixels. This results in an approximately even split of 32%, 35%, 33% for DAVIS17. As expected, segmentation performance is proportional to the object size category, as smaller objects have less support in the scene and are therefore more challenging.

Similar to the results in Table 2 of the main paper, we observe that our bi-directional and multi-hop motion vector propagation's accuracy exceeds "Optical Flow" warping and naive "MV to Flow" [5] by a large margin on DAVIS17. Compared to per-frame inference without any propagation, our proposed propagation and correction achieve excellent segmentation accuracies on medium and large objects. Interestingly, for the large objects, we are able to achieve comparable results with the original base model. This result validates the effectiveness of our motion vector-based soft propagation and residual-based correction.

Our propagation and refinement scheme is less effective on small objects (100×100). Small objects are challenging for many segmentation methods, and given the lightweight nature of our decoder, we observe a similar weakness in our system. We believe that the results can be improved if we choose to use deeper features and a more powerful decoder, but more computational cost will be applied.

Qualitative Comparison

Fig. 1 highlights success cases compared with other state-of-the-art methods and the base model, where our method can provide a full mask and recover most of the boundary detailing compared to the base model.

Fig. 2 shows some sample cases of our residual correction module. In the *bmx-trees* sequence, we are able to recover the bike from the residual. Similarly, in the *kite-surf* sequence, where most of the human body is missing in the propagation, we successfully recover it from the residual.

However, we fail to estimate the correct label of the band due to its similar appearance to the human body.

Fig. 3 shows some challenging cases for our methods. In the *motocross-jump* sequence shown in the first and second row, the motion vector fails to capture the fast scene change, resulting in inaccurate segmentations as highlighted with white rectangles.

We show more qualitative examples on the DAVIS dataset in Figure 4 and 5.

Licence

The annotations in DAVIS belong to the organizers of the challenge and are licensed under the BSD License. The annotations in YouTube-VOS belong to the organizers of the challenge and are licensed under a Creative Commons Attribution 4.0 License. Code and pretrained weights for MiVOS, STCN and FRTM-VOS are under GNU General Public License v3.0.

References

[1] Ho Kei Cheng, Yu-Wing Tai, and Chi-Keung Tang. Modular interactive video object segmentation: Interaction-to-mask, propagation and difference-aware fusion. In *CVPR*, 2021.

[2] Ho Kei Cheng, Yu-Wing Tai, and Chi-Keung Tang. Rethinking space-time networks with improved memory coverage for efficient video object segmentation. In *NeurIPS*, 2021.

[3] Seoung Wug Oh, Joon-Young Lee, Kalyan Sunkavalli, and Seon Joo Kim. Fast video object segmentation by reference-guided mask propagation. In *CVPR*, 2018.

[4] Andreas Robinson, Felix Jaremo Lawin, Martin Danelljan, Fahad Shahbaz Khan, and Michael Felsberg. Learning fast and robust target models for video object segmentation. In *CVPR*, 2020.

[5] Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Compressed video action recognition. In *CVPR*, 2018.

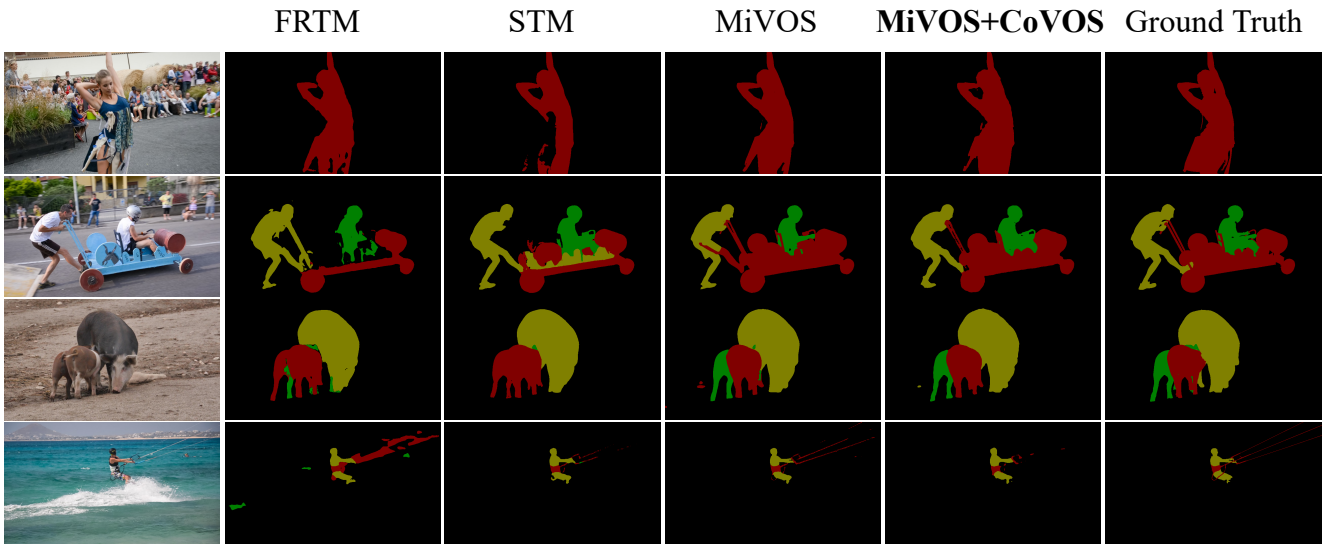

Figure 1. Qualitative comparison with state-of-the-art methods. With MiVOS as the base model, we propagate most of the segmentation, while competing methods FRTM and STM exhibit several failures, e.g. fail to provide a precise mask (row 1: body of dancer, row 2: cart), distinguish similar objects (row 3: two piglets get merged) or provide more precise boundaries (row 4: kite handle).

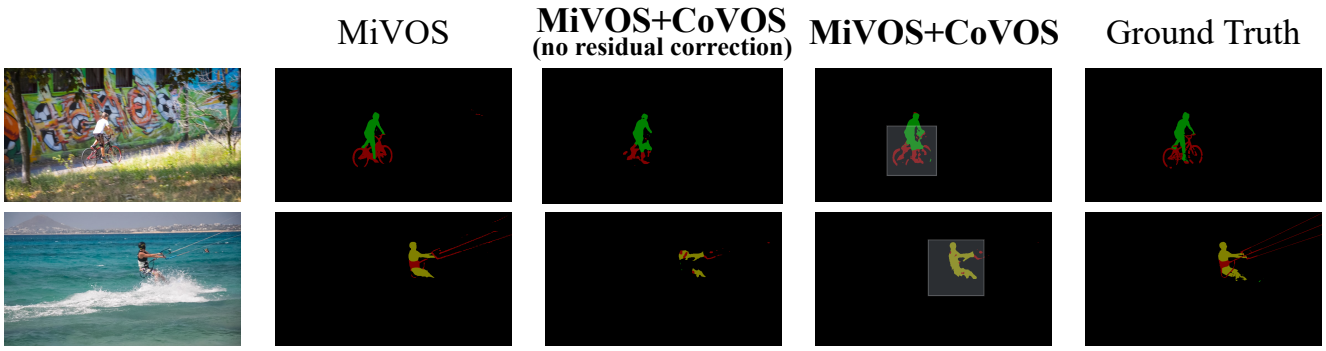

Figure 2. Cases where motion vectors fail to correctly propagate the segmentation masks, but we recover the correct masks through residual correction. Label recovery through feature matching sometimes fails to distinguish similar regions as we only use very low-level features (second row).

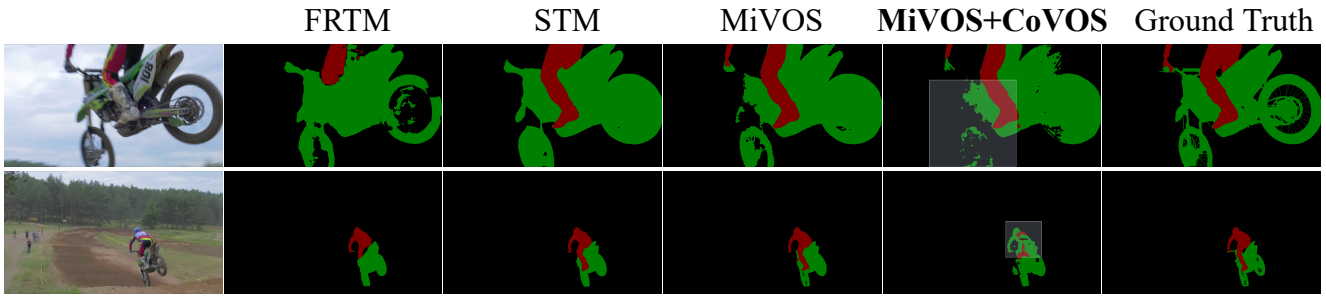

Figure 3. Our method struggles in cases of abrupt motions, which lead to inaccurate motion vectors (see white rectangle highlights).

Figure 4. Qualitative comparison on DAVIS17.

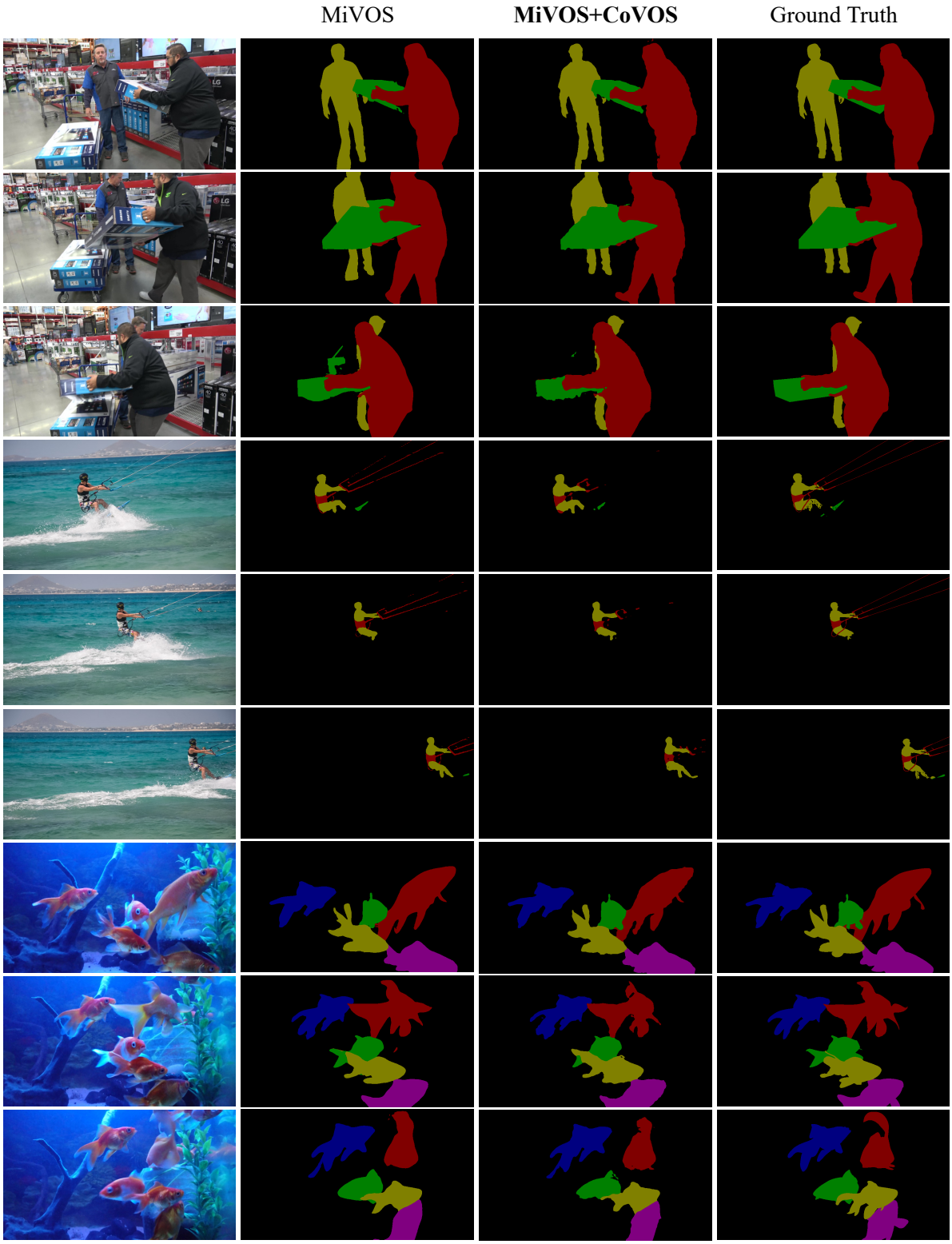

Figure 5. Qualitative comparison on DAVIS17.