Supplementary Material

Learning What to Learn for Video Object Segmentation

Goutam Bhat*,[1], Felix Järemo Lawin*,[2], Martin Danelljan[1]
Andreas Robinson[2], Michael Felsberg[2], Luc Van Gool[1], and Radu Timofte[1]

[1] CVL, ETH Zürich, Switzerland [2] CVL, Linköping University, Sweden

In this supplementary material we provide additional results and further details about our method. First, in Section S1, we provide a derivation of the steepest descent iterations in Eq. (4). Next, we present more details about our label generator, weight predictor, box encoder and decoder modules in Section S2. Section S3 and Section S4 provide additional details about our inference and training procedures, respectively. We include a detailed runtime analysis in Section S5. A comparison of our approach on the YouTube-VOS 2019 validation set is provided in Section S6. We further provide detailed ablative analysis of our training and inference parameters in Section S7. Finally, in Section S8, we provide additional qualitative results, including outputs generated with our box-initialization setting, mask encoding outputs and a video presenting output segmentations on a number of sequences from YouTube-VOS 2018 and DAVIS 2017 validation sets. Note that the numbering of all sections, equations, figures, and tables in this supplementary material is prefixed with the letter 'S'. Any number without this prefix refer to the main paper.

S1 Derivation of Internal Learner Iteration Steps

In this section we derive the steepest decent iterations in Eq. (4) used in our few-shot learner to minimize the loss in Eq. (3). To simplify the derivation, we first convert the loss into a matrix formulation. We then derive expressions for the vectorized gradient $\bar{g}$ and step-length α, showing that these can be computed using simple neural network operations.

We use the fact that the convolution between the feature map $x_t \in \mathbb{R}^{H\times W\times C}$ and weights $\tau \in \mathbb{R}^{K\times K\times C\times D}$ can be written in matrix form as $\mathrm{vec}(x_t * \tau) = X_t\bar{\tau}$. Here, vec is the vectorization operator, $\bar{\tau} = \mathrm{vec}(\tau) \in \mathbb{R}^{K^2CD}$ and $X_t \in \mathbb{R}^{HWD\times K^2CD}$ is a matrix representation of $[x_t*]$. We further define $e_t = \mathrm{vec}(E_\theta(y_t)) \in \mathbb{R}^{HWD}$ as a vectorization of the label encoding and $W_t = \mathrm{diag}(\mathrm{vec}(W_\theta(y_t))) \in \mathbb{R}^{HWD\times HWD}$ is a diagonal matrix corresponding to the point-wise multiplication of the importance weights $W_\theta(y_t)$. We can now write Eq. (3) in matrix form as,

$$L(\bar{\tau}) = \frac{1}{2}\sum_t \left\|W_t(X_t\bar{\tau} - e_t)\right\|^2 + \frac{\lambda}{2}\left\|\bar{\tau}\right\|^2. \tag{S1}$$

In the steepest descent algorithm, we update the parameters by taking steps $\bar{\tau}^{i+1} = \bar{\tau}^i - \alpha^i\bar{g}^i$ in the gradient direction $\bar{g}^i$ with step length α^i. Setting $\bar{r}_t(\bar{\tau}) =$

$W_t(X_t\bar{\tau} - e_t)$, the gradient is obtained using the chain rule,

$$\bar{g} = \nabla L(\bar{\tau}_d) = \sum_t \Big(\frac{\partial \bar{r}_t}{\partial \bar{\tau}}\Big)^{\mathrm{T}} \bar{r}_t(\bar{\tau}) + \lambda\bar{\tau} = \sum_t X_t^{\mathrm{T}} W_t^2\big(X_t\bar{\tau} - e_t\big) + \lambda\bar{\tau}\,. \tag{S2}$$

We see that the gradient can be computed as,

$$\begin{aligned}\bar{g} &= \sum_t X_t^{\mathrm{T}} W_t^2\big(\operatorname{vec}(x_t * \tau) - e_t\big) + \lambda \operatorname{vec}(\tau) \\ &= \operatorname{vec}\left(\sum_t x_t *^{\mathrm{T}} W_\theta(y_t)^2 \cdot \big(x_t * \tau - E_\theta(y_t)\big) + \lambda\tau\right),\end{aligned} \tag{S3}$$

where the transposed convolution x_t*^{T} corresponds to the matrix multiplication with X_t^{T}. Thus,

$$g = \sum_t x_t *^{\mathrm{T}} \Big(W_\theta(y_t)^2 \cdot \big(x_t * \tau - E_\theta(y_t)\big)\Big) + \lambda\tau\,. \tag{S4}$$

We compute the step length α^i that minimizes L in the current gradient direction g^i

$$\alpha^i = \arg\min_\alpha L(\tau^i - \alpha\bar{g}^i)\,. \tag{S5}$$

Since the loss is convex, it has an unique global minimum obtained by solving for the stationary point $\frac{\mathrm{d}L(\bar{\tau}^i - \alpha\bar{g}^i)}{\mathrm{d}\alpha} = 0$. We set $v = \bar{\tau}^i - \alpha\bar{g}^i$, and use (S2) with the chain rule to obtain,

$$\begin{aligned}0 = \frac{\mathrm{d}L(v)}{\mathrm{d}\alpha} &= \Big(\frac{\mathrm{d}v}{\mathrm{d}\alpha}\Big)^{\mathrm{T}} \nabla_v L(v) \\ &= (\bar{g}^i)^{\mathrm{T}}\bigg(\sum_t X_t^{\mathrm{T}} W_t^2\big(X_t(\bar{\tau}^i - \alpha\bar{g}^i) - e_t\big) + \lambda(\bar{\tau}^i - \alpha\bar{g}^i)\bigg) \\ &= (\bar{g}^i)^{\mathrm{T}}\bar{g}^i - \alpha(\bar{g}^i)^{\mathrm{T}}\bigg(\sum_t X_t^{\mathrm{T}} W_t^2 X_t\bar{g}^i + \lambda\bar{g}^i\bigg) \\ &= \|\bar{g}^i\|^2 - \alpha\bigg(\sum_t \|W_t X_t\bar{g}^i\|^2 + \lambda\|\bar{g}^i\|^2\bigg)\,.\end{aligned} \tag{S6}$$

Thus, the step length is obtained as,

$$\alpha = \frac{\big\|\bar{g}^i\big\|^2}{\sum_t \big\|W_t X_t\bar{g}^i\big\|^2 + \lambda\big\|\bar{g}^i\big\|^2}\,. \tag{S7}$$

We note that, $\big\|\bar{g}^i\big\|^2 = \big\|g^i\big\|^2$ and $\big\|W_t X_t\bar{g}^i\big\|^2 = \big\|\operatorname{vec}(W_\theta(y_t)\cdot x_t * g^i)\big\|^2 = \big\|W_\theta(y_t)\cdot(x_t * g^i)\big\|^2$. The step length can therefore be computed as follows,

$$\alpha = \frac{\big\|g^i\big\|^2}{\sum_t \big\|W_\theta(y_t)\cdot(x_t * g^i)\big\|^2 + \lambda\big\|g^i\big\|^2}\,. \tag{S8}$$

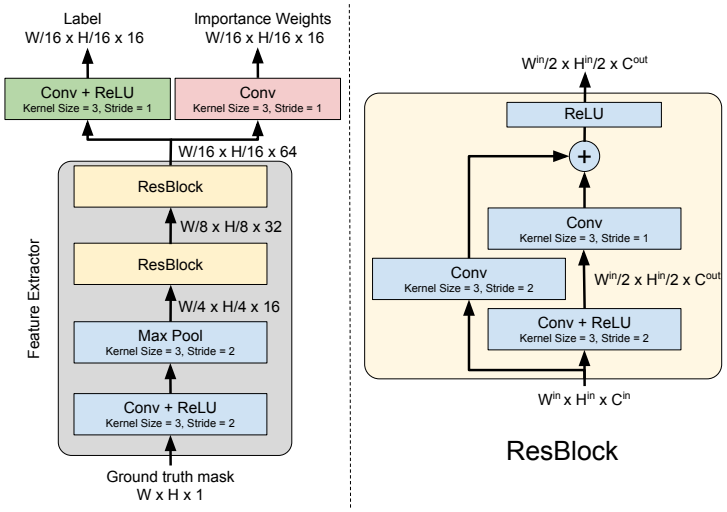


Fig. S1. The network architecture employed for the label generator E_θ and the importance weight predictor W_θ modules. Both modules share a common feature extractor (gray) consisting of a convolutional layer followed by two residual blocks (yellow). The feature extractor takes as input the ground truth segmentation mask, and outputs a deep representation of the mask containing 64 channels. The label generator E_θ (green), consisting of a single convolutional layer followed by ReLU activation, generates the ground truth label for the few-shot learner using the mask features as input. Similarly, the importance weight predictor W_θ (red), which consists of a single convolutional layer, predicts the importance weights using the mask features as input.

S2 VOS Architecture Details

S2.1 Few-shot Label Generator E_θ and Weight Predictor W_θ

Here, we describe in detail the network architecture employed for the label generator E_θ and the importance weight predictor W_θ. The network architecture is visualized in Figure S1. The label generator E_θ and the importance weight predictor W_θ share a common feature extractor consisting of a convolutional layer followed by two residual blocks. The feature extractor takes as input the ground truth segmentation mask, and outputs a deep representation of the mask. The mask features contain 64 channels, and have a spatial resolution 16 times lower than the input mask. The label generator module E_θ, which consists of a single convolutional layer followed by a ReLU activation, operates on the mask features to predict the ground truth label for the few-shot learner. Similarly, the importance weight predictor W_θ consists of a single convolutional layer and predicts the importance weights using the mask features as input. All the convolutional layers in the network employ 3×3 kernels. Note that the importance weights $W_\theta(y)$ are squared when computing the squared error between the target model output and the ground truth label $E_\theta(y)$ predicted by the label generator E_θ. Thus, we thus allow $W_\theta(y)$ to take negative values.

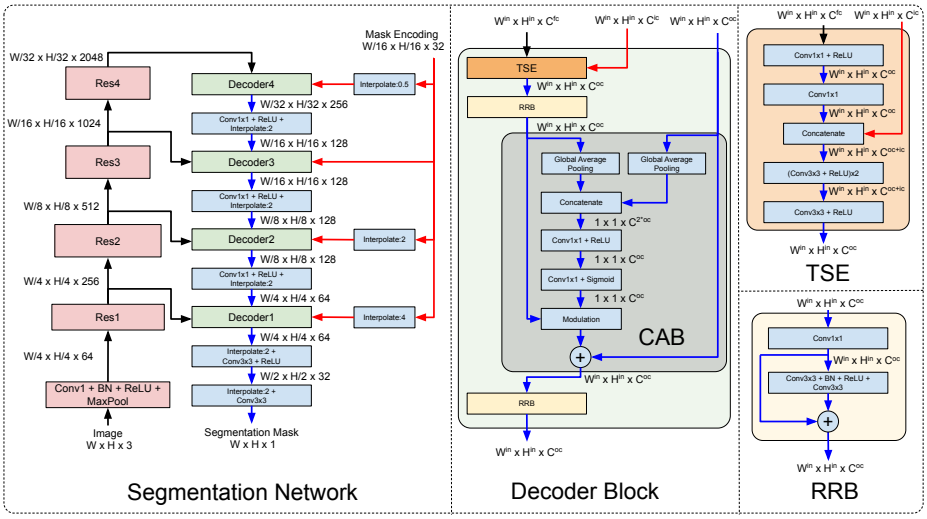


Fig. S2. The network architecture employed by our segmentation decoder D_θ. The segmentation decoder takes as input the mask encoding output by the target model T_τ (red arrow), along with the output of the four residual blocks of the backbone ResNet-50 (black arrow). The network has a U-Net based structure containing four decoder blocks (green) corresponding to the residual blocks in the ResNet-50 feature extractor. The TSE module (orange) in each decoder block first projects the backbone features to a lower-dimensional representation, which is concatenated with the mask encoding. We interpolate the mask encodings to the same spatial size as the backbone features before concatenations. The concatenated features are processed by three convolutional layers followed by a residual block (yellow). The resulting features are then merged with features from a deeper decoder module with a channel attention block (CAB) [8] (gray). Finally, the features are processed by another residual block before being passed to the next decoder level. The output from the final decoder block is up-sampled and processed by convolutional layer to obtain the final segmentation mask.

S2.2 Segmentation Decoder D_θ

Here, we detail the segmentation decoder D_θ architecture, visualized in Figure S2. We adopt a similar architecture as in [7]. The decoder module takes the mask encoding output by the target model T_τ, along with backbone ResNet features, in order to predict the final accurate segmentation mask. It has a U-Net based structure containing four decoder blocks corresponding to the residual blocks in the ResNet feature extractor. In each decoder block, we first project the backbone features into a lower-dimensional representation. Next, we concatenate the projected feature maps with the mask encoding output by the T_τ. These are processed by three convolutional layers followed by a residual block. The resulting features are then merged with features from a deeper decoder module with a channel attention block (CAB) [8]. Finally, the features are processed by another residual block before they are merged with features from a shallower level. The

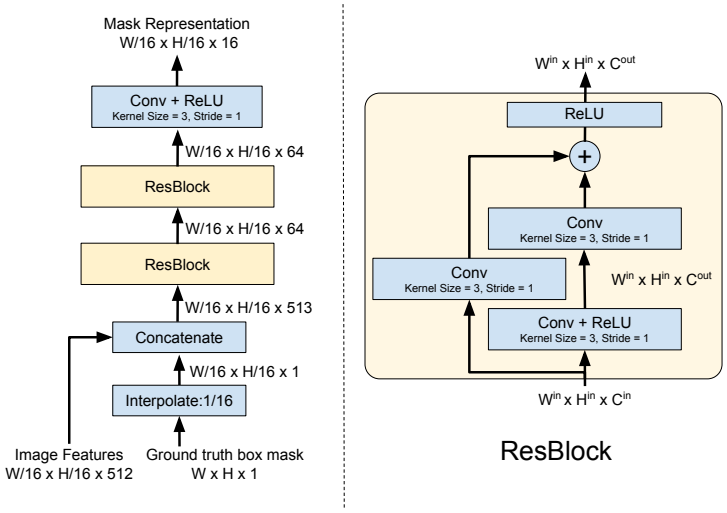


Fig. S3. The network architecture employed for the bounding box encoder $B_\theta(b_0, x_0)$. The network takes as input a mask b_0 denoting the input ground truth box, along with a deep feature representation x_0 of the image. The input mask is first downsampled by a factor of 16 and concatenated with the image features. These are then processed by two residual blocks (yellow). The output of the second residual block is passed to a convolutional block which predicts the mask representation of the target object. This mask representation is input to the segmentation decoder module D_θ to obtain the segmentation mask for the target.

output from the final decoder module is up-sampled and projected to a single channel target segmentation mask.

S2.3 Bounding Box Encoder B_θ

In this section we give a detailed description of the network architecture of the box encoder module. We provide an illustration of the architecture in Figure S3. The network takes a mask representation of the bounding box along with features from layer3 from the backbone feature extractor as input. The mask is down-sampled with bilinear interpolation to a 1/16th of the input resolution, to match the size of the backbone features. The backbone features are first processed by a convolutional layer that reduces the dimension to $C = 512$. Here, the weights of this convolutional layer is shared with the projection layer for the target model. The resulting features are first concatenated with the downsampled mask and then fed through a residual block, which also reduces the feature dimension to 64 channels. Next, the features are processed by an another residual block, before the final box encoding is generated by a convolutional layer. This convolutional layer reduces the number of dimensions to coincide with number of channels in the mask representation produced by the label generator module. The output of the box encoder can then be processed by the decoder network to produce a segmentation mask.

Algorithm 1 Segmentation Mask to Target Box

Input: Mask prediction $m(\mathbf{r}) \in [0,1]$ for every pixel $\mathbf{r} \in \Omega := \{0,\dots,W-1\} \times \{0,\dots,H-1\}$ in the image, Previous target size $\hat{\mathbf{b}} = (\hat{b}_w, \hat{b}_h)$

Output: Current target location $\mathbf{c} = (c_x, c_y)$ and size $\mathbf{b} = (b_w, b_h)$

$z = \sum_{\mathbf{r}\in\Omega} m(\mathbf{r})$ # Compute normalization factor

$\mathbf{c} = \frac{1}{z}\sum_{\mathbf{r}\in\Omega} \mathbf{r}\cdot m(\mathbf{r})$ # Estimate target center

$\sigma^2 = \frac{1}{z}\sum_{\mathbf{r}\in\Omega} (\mathbf{r}-\mathbf{c})^2 \cdot m(\mathbf{r})$ # Estimate target size

$\mathbf{b} = 4\sigma$

$\Delta_{\text{size}} = \sqrt{\frac{b_w b_h}{\hat{b}_w \hat{b}_h}}$ # Change in target size from prev. frame

$\Delta_{\text{size}} = \min(\max(\Delta_{\text{size}}, 0.95), 1.1)$ # Limit change in target size

$\mathbf{b} = \Delta_{\text{size}}\hat{\mathbf{b}}$

S3 Inference Details

In this section, we provide more details about our inference procedure. Instead of operating on the full image, we process only a local region around the previous target location in each frame. This allows us to effectively segment objects of any size. The local search region is obtained by cropping a patch that is 5 times larger than the previous estimate of target, while ensuring the maximal size to be equal to the image itself. The cropped region is resized to 832×480 with preserved aspect ratio. An estimate of the target location and size is obtained from the predicted segmentation mask, as detailed in Algorithm 1. The target center is determined as the center of mass of the predicted target mask, while the target size is computed using the variance of the segmentation mask. We additionally prevent drastic changes in the target size between two consecutive frames by limiting the target scale change between two frames to be in the range $[0.95, 1.1]$. This allows the inference to be robust to incorrect mask predictions in one or few frames.

S4 Training Details

Here, we provide more details about our offline training procedure. Our network is trained using the YouTube-VOS 2019 training set (excluding the 300 validation videos) and the DAVIS 2017 training set. We sample sequences from both datasets without replacement, using a 6 times higher probability for YouTube-VOS, as compared to the DAVIS 2017 training set, due to the higher number of sequences in the former dataset. Our network is trained using the ADAM [4] optimizer.

Our final networks, that are used for state-of-the-art comparisons, are trained using the *long* strategy. In this setting, the networks are trained for $150k$ iterations in total, with a base learning rate of 10^{-2}. The learning rate is reduced by a factor of 5 after 40k, 95k, and 145k iterations. For the first 70k iterations, we freeze the weights of our backbone feature extractor, and only train the newly added layers. The complete network, excluding the first convolutional

and residual blocks in the feature extractor, are then trained for the remaining 80k iterations. We use a mini-batch size of 20 throughout our training. For evaluation on the DAVIS dataset, we found it beneficial to use $D = 32$ output channels in our target model. We use the same initial training schedule as for Youtube-VOS. Additionally we fine-tune our network for $2k$ more iterations using only the DAVIS 2017 training set. The entire training takes 48 hours on 4 Nvidia V100 GPUs.

Due to resource constraints, we use a *shorter* schedule when training different versions of our proposed approach for the ablation study. Here, we train the network for 70k iterations, using a mini-batch size of 10. We use a base learning rate of 10^{-2}, which is reduced by a factor of 5 after 25k, and 50k iterations. In this training setting, we keep the weights of the backbone feature extractor fixed and only train the newly added layers. The entire training takes 24 hours on a single Nvidia V100 GPU.

The bounding box encoder is trained on YouTube-VOS 2019 (excluding the 300 validation videos) and MSCOCO [5]. The mini-batches are constructed by sampling images with twice as high probability from MSCOCO compared to YouTube-VOS. We train the network for $50k$ iterations with a batch size of 8, only updating the weights of the convolutional layers in the box encoder. We use a base learning rate of 10^{-2} and reduce it by a factor of 5 after 20k and 40k iterations.

S5 Runtime Analysis

In this section we provide a run-time analysis for our VOS algorithm. The results are summarized in Table S1. We note that the algorithm runs in two phases; the first frame initialization phase, where we learn the target module parameters on the initial frame, and the inference phase applied on the subsequent test frames. Within the initial phase, the few-shot learner takes 71.6% of the runtime and the remainder is due to feature extraction. The impact of the initial phase on the total runtime for the full sequence depend on the sequence length. For instance, in a sequence with 100 frames, the initial phase accounts for 3.4% of the total runtime. In the inference phase, updating the target module has the highest impact on the runtime at 69.7%. Most of the remaining time is due to feature extraction (14.4%) and the decoder (14.7%). Applying the target module on the extracted features has almost negligible impact at 0.13%. The remaining time of the algorithm, denoted as other in Table S1, is due to overhead processes such as image cropping and memory management. Note that, by updating the target module less frequently we can reduce its influence on the runtime significantly.

S6 YouTube-VOS 2019

We evaluate our approach on the YouTube-VOS 2019 validation set consisting of 507 sequences. The dataset contains 1063 unique object instances belonging to 91 object categories, of which 26 are *unseen* in the training dataset. The results are

Table S1. Runtime analyis of our VOS algorithm. The two left columns show the percentages taken by the steps of the frame initialization phase. The remainder right columns show the percentages taken by the steps in the inference phase. In both phases, the few-shot learner has a dominating impact on the run-time.

First frame initialization		Test frame inference				
Feature extractor	Few-shot learner	Feature extractor	Few-shot learner	Target model	decoder	other
28.4%	71.6%	14.4%	69.7%	0.13%	14.7%	1.07%

Table S2. Comparison of our approach with the recently introduced STM [6] on the large-scale YouTube-VOS 2019 validation dataset. Results are reported in terms of mean Jaccard ($\mathcal{J}$) and boundary ($\mathcal{F}$) scores for object classes that are *seen* and *unseen* in the training set, along with the overall mean ($\mathcal{G}$). Our approach outperforms STM with a large margin of +1.8 points in terms of the overall $\mathcal{G}$ score.

Method	$\mathcal{G}$(%) overall	$\mathcal{J}$(%) seen | unseen	$\mathcal{F}$(%) seen | unseen
Ours	81.0	79.6 | 76.4	83.8 | 84.2
STM [6]	79.2	79.6 | 73.0	83.6 | 80.6

obtained through the online evaluation server. The benchmark reports Jaccard $\mathcal{J}$ and boundary $\mathcal{F}$ scores for *seen* and *unseen* categories. Methods are ranked by the overall $\mathcal{G}$-score, obtained as the average of all four scores. We compare our approach with results shown on the leaderboard of the evaluation server for the recently introduced STM [6] method. The results are shown in Table S2. Our approach achieves at overall $\mathcal{G}$ score of 81.0, outperforming STM with a large margin of +1.8.

S7 Detailed Ablative Study

In this section, we analyze the impact different components in our architecture. As in the main paper, our analysis is performed on the validation set of 300 sequences generated from the YouTube-VOS 2019 training set. Unless otherwise mentioned, we use the shorter training schedule (see Section S4) for training the networks compared in this section and the default inference parameters (see Section 3.5) in the evaluations. The networks are evaluated using the mean Jaccard $\mathcal{J}$ index (IoU).

Impact of initial backbone weights: Here, we analyse the impact of using the Mask-RCNN [3] weights for initializing our backbone feature extractor. We compare our approach with a baseline network which uses ImageNet [2] pre-trained weights for initializing the backbone. The results of this comparison is shown in Table S3. Using the Mask-RCNN weights for initializing the backbone feature extractor provides an improvement of +1.2 in $\mathcal{J}$ score over the version which uses ImageNet trained weights.

Table S3. Impact of the weights used for initializing the backbone feature extractor. We compare a network using Mask-RCNN weights for initializing the backbone feature extractor with a network using ImageNet pre-trained weights. The results are reported over a validation set of 300 videos sampled from YouTube-VOS 2019 training set, in terms of mean Jaccard $\mathcal{J}$ score.

	$\mathcal{J}(\%)$
Mask-RCNN weights	79.8
ImageNet weights	78.6

Table S4. Impact of the segmentation loss employed during training. We compare a network trained using the Lovasz [1] loss function, with a network trained using the binary cross-entropy loss.

	$\mathcal{J}(\%)$
Lovasz Loss	79.8
Binary Cross-Entropy Loss	79.2

Table S5. Comparison of a network trained using the shorter training strategy (see Section S4) with the network trained using the long training strategy. In contrast to the shorter training strategy, the backbone feature extractor is also trained when using the long training strategy, while employing a larger batch size. The long training however requires 8 times more GPU hours, as compared to the shorter training.

	$\mathcal{J}(\%)$
Short training strategy	79.8
Long training strategy	81.2

Impact of training loss: We investigate the impact of using the Lovasz [1] loss as our segmentation loss in Eq. (5) by evaluating a version which uses the binary cross-entropy (BCE) loss. The results in Table S4 show that using the Lovasz loss provides as improvement of +0.6 in $\mathcal{J}$ score over the baseline trained using the BCE loss.

Impact of long training: Here, we analyze the impact of using the long training procedure, as described in Section S4. Table S5 shows a comparison between the shorter and longer training strategy. The long training provides an improvement of +1.4 in $\mathcal{J}$ score over the version using the shorter training, albeit taking 8 times more GPU hours for training.

Impact of number of mask encoding channels D: We investigate the impact of the number of output channels D in the mask encoding $T_\tau(x)$ predicted by the target model T_τ. The $\mathcal{J}$ score for different values of mask encoding channels D are plotted in Figure S4. Using a larger number of channels (≥ 4) allows the target model to output a richer representation of the target mask, leading to an improvement of +2.0 in $\mathcal{J}$ score over the baseline employing a single channel.

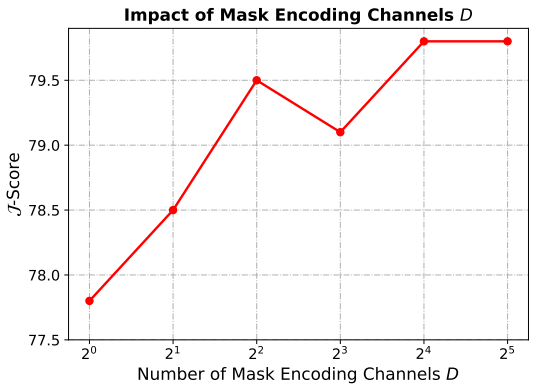
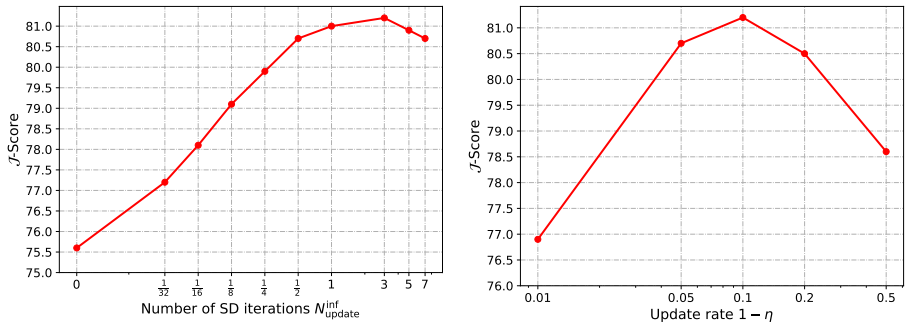


Fig. S4. Impact of the number of output channels D in the mask encoding $T_\tau(x)$ predicted by the target model T_τ. We plot the $\mathcal{J}$ score (y-axis) computed over a validation set of 300 videos sampled from YouTube-VOS 2019 training set for different values of mask encoding channels D (x-axis). Using a larger number of channels (>= 4) allows the target model to output a richer representation of the target mask, leading to improved results over the baseline employing a single channel.

Impact of number of update iterations: We analyze the impact of number of steepest-descent (SD) iterations $N^{\text{inf}}_{\text{update}}$ employed during inference. The analysis is performed using the final network trained using the long training strategy. The $\mathcal{J}$ score for different number of update iterations $N^{\text{inf}}_{\text{update}}$ are plotted in Figure S5a. The fractional values $\frac{1}{n}$ for the update iterations $N^{\text{inf}}_{\text{update}}$ in Figure S5a imply that a single steepest-descent iteration is performed after every n frames. The setting $N^{\text{inf}}_{\text{update}} = 0$ corresponds to not updating the target model parameters τ during inference. That is, the target model estimated using the initial frame is used for the whole sequence. We observe that performing only a single steepest-descent iteration in each frame significantly improves the performance by more than 5 points in $\mathcal{J}$ score, compared to the version with no model update. This demonstrates that our few-shot learner can quickly adapt the target model to the changes in the scene, thanks to the fast convergence of steepest-descent updates. The setting $N^{\text{inf}}_{\text{update}} = 3$ provides the best performance with a $\mathcal{J}$ score of 81.2. Performing a higher number of SD iterations (> 3) results in overfitting of the target model to the training set, leading to a slight degradation in performance.

Impact of update rate: Here, we investigate the impact of the update rate $1 - \eta$ employed when setting the global importance weights for the samples in the few-shot training set $\mathcal{D}$ (see Section 3.5). Figure S5b shows the $\mathcal{J}$ score for different values of the update rate $1 - \eta$. A higher update rate implies that the recent samples get a larger global importance weights when updating the target model τ during inference. We observe that an update rate of 0.1 gives the best results with a $\mathcal{J}$ score of 81.2.



(a) Impact of update iterations (b) Impact of update rate $1-\eta$

Fig. S5. Impact of the number of steepest-descent update iterations (a), and the update rate $1-\eta$ (b) employed during inference. In plot (a), fractional values $\frac{1}{n}$ for the update iterations $N^{\text{inf}}_{\text{update}}$ imply that a single steepest-descent iteration is performed after every n frames. The results are shown in terms of the mean Jaccard $\mathcal{J}$ score over a validation set of 300 videos randomly sampled from the YouTube-VOS 2019 training set.

Table S6. Impact of different evaluation modes during inference. We compare a version in which the network operates on a local search region with a version in which the network operates on the full image. The search region in the first version is obtained by using the estimate of the target mask in the previous frame.

	$\mathcal{J}$(%)
Local search region evaluation	81.2
Full frame evaluation	80.2

Impact of maximum training set size $K_{\max}$: Here, we analyze the impact of the maximum few-shot training set size $K_{\max}$. Figure S6 shows the $\mathcal{J}$ score for different values of $K_{\max}$ over the 300 sequences in our validation set. The initial annotated sample is always included in our training set, in addition to $K_{\max}-1$ previous frames. Thus, the setting $K_{\max}=1$ corresponds to using only the initial frame, while $K_{\max}=2$ corresponds to using the initial frame and the previous frame for updating the target model. Using a larger training set leads to improved performance until a training set size of $K_{\max}=16$, at which point the performance gain begins to saturate.

Impact of evaluation mode: We analyze the impact of different modes for running the network on the test frames during inference. We compare our approach in which the network operates on a local search region with a baseline version in which the network operates on the full image. The search region in our approach is obtained by using the estimate of the target mask in the previous frame, as described in Section S3. The results of this comparison is shown in Table S6. Operating on a local search region allows the network to better handle small objects, leading to an improvement of +1.0 points in $\mathcal{J}$ score over the baseline operating on the full image.

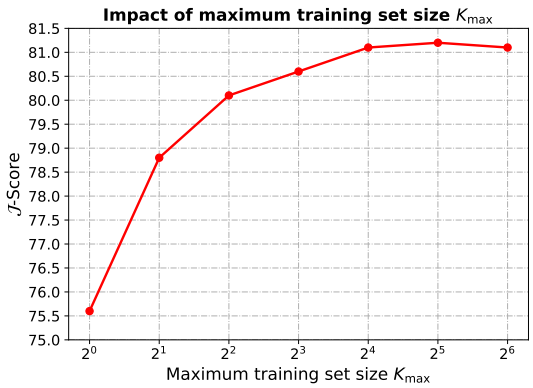


Fig. S6. Impact of the maximum few-shot training set size $K_{\max}$ employed during inference. The results are shown in terms of the mean Jaccard $\mathcal{J}$ score over a validation set of 300 videos randomly sampled from the YouTube-VOS 2019 training set.

S8 Qualitative Results

Qualitative Comparison: We show a qualitative comparison between our approach and STM [6] in Figure S7. In the comparison we include two frames from each of the sequences shooter, loading, pigs and soapbox from the DAVIS 2017 validation set. These sequences contain several challenges such as occlusions, changes in appearance and distractor objects. We observe that these challenges are handled very differently by STM and our approach. In the shooter sequence, STM fails to segment the gun in the late frame, while our approach successfully segments all targets. Further, our approach struggles with segmenting the box in the loading sequence and one of the piglets in the pigs sequence. Finally, in contrast to our approach, STM manages to segment the thin structured handles of the wagon in the soapbox sequence. On the other hand, STM falsely predicts segments on the wagon with the label of the running man, while our approach segments all targets without any major false predictions.

Label encoding: In Figure S8 we show a number of selected channels from the labels produced by the label generator and the absolute values of the corresponding importance weights. As we can see, the channels in labels model different aspects of the target mask, such as boundaries, background and a low resolution approximation of the mask. The weights alternate between being higher or lower on the background compared to the foreground regions. However, there seems to be consistently higher weighting around the edges of the target.

Bounding box initialization: In Figure S9 we show some example outputs generated by our approach with bounding box initialization. The sequences are sampled from YouTube-VOS 2018 and DAVIS 2017 validation sets. As these examples demonstrate, our decoder network manages to segment the target in the first frame (second column), given a mask representation generated by our bounding box encoder module. This mask prediction is then used as a pseudo

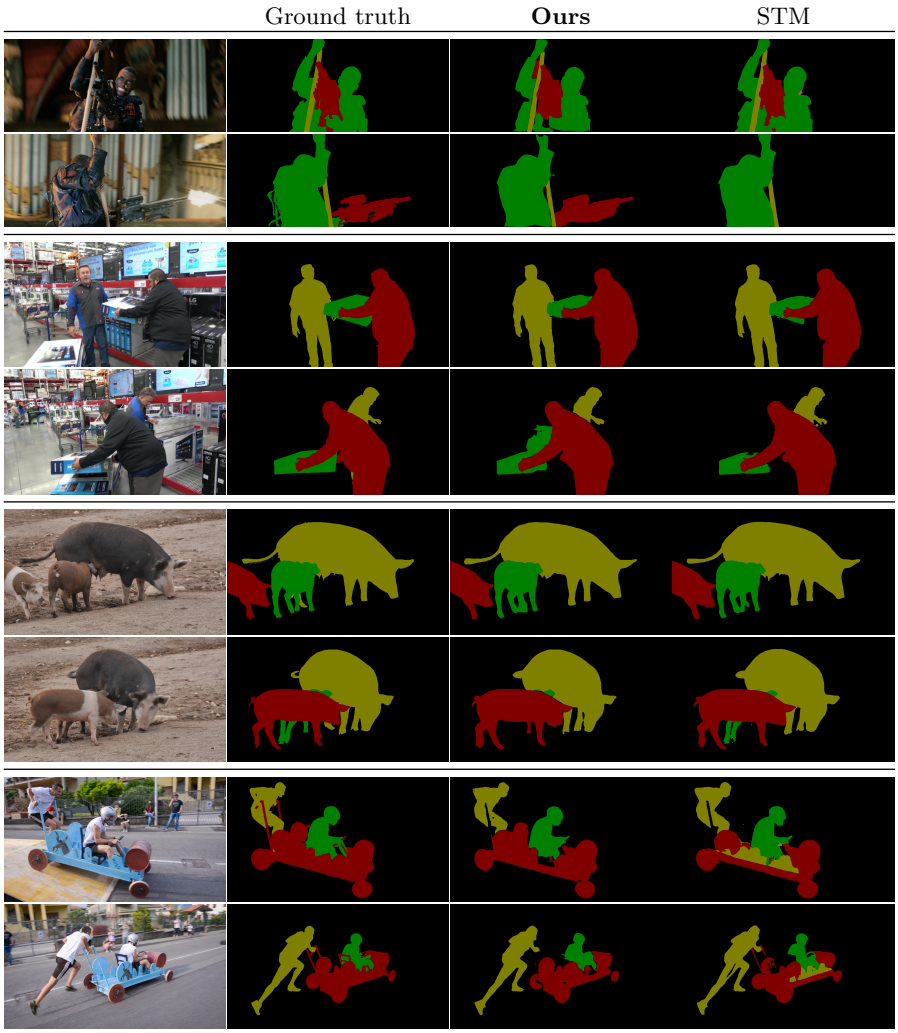


Fig. S7. Qualitative comparisons between our approach and STM [6] on the sequences shooter, loading, pigs and soapbox from the DAVIS 2017 validation set. For each sequence, we have selected two specific frames with distinct scene appearance.

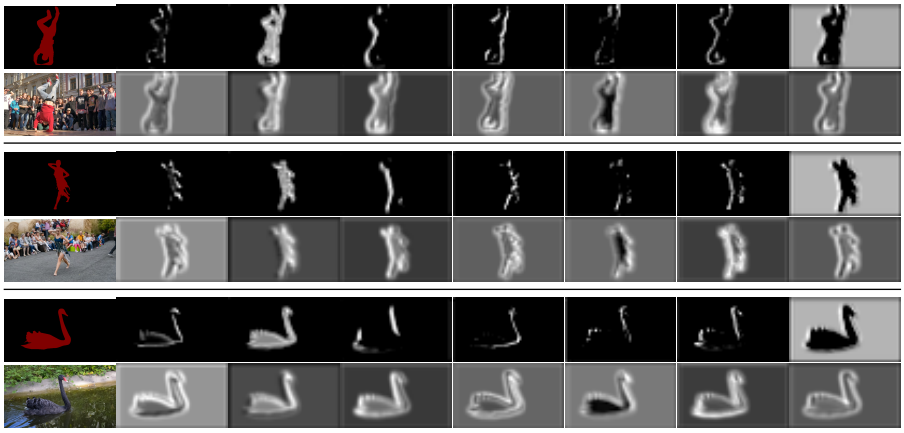


Fig. S8. Visualization of learned mask encodings. The left most column contain the ground truth masks with the corresponding image below. In the other columns we show labels generated by the label generator in the top rows and absolute values of the corresponding importance weights in the rows below.

ground truth annotation of the initial target mask in our VOS approach. Although the predicted initial mask is less accurate than the ground truth annotation, our approach has the ability to generate high quality segmentation masks in the subsequent frames.

Video: We provide further qualitative results in a a video. The video contains example segmentation outputs generated by our approach with mask initialization on sequences from YouTube-VOS 2018 and DAVIS 2017 validation sets. The following sequences are included from YouTube-VOS 2018 validation set:

`00f88c4f0a, 30fe0ed0ce, 39bce09d8d, 3f99366076, 63a68c6741, 67e397b1f2, 6a75316e99, 77bec90101, 7fb4f14054, 8e2e5af6a8, 9a38b8e463, 9c693f291b, 9ce299a510, 9fd2d2782b`

The following sequences are included from DAVIS 2017 validation set:

`bike-packing, blackswan, bmx-trees, dance-twirl, drift-chicane, drift-straight, goat, horsejump-high, judo, kite-surf, motocross-jump, soapbox`

Fig. S9. Box initialization example results. The first column shows the initial frame with the given bounding box annotation. The second column shows the first frame segmentation, predicted by the decoder network given a mask representation generated by our box encoder. The right four columns show predicted segmentation masks from our approach.

S9 Internal Validation Set

Here, we describe the internal validation set used for our ablation study in Section 4.1 of the main paper. The validation set was constructed by uniformly sampling 300 videos from the YouTube-VOS 2019 training set. The sequences included in our validation set are listed below.

d82a0aa15b, 691a111e7c, 97ab569ff3, d4a607ad81, f46c364dca, 4743bb84a7,
1295e19071, 267964ee57, df59cfd91d, c557b69fbf, 927647fe08, 88f345941b,
8ea6687ab0, 444aa274e7, ae93214fe6, b6e9ec577f, de30990a51, acb73e4297,
6cccc985e0, ebc4ec32e6, f34a56525e, 2b351bfd7d, a43299e362, 733798921e,
feda5ad1c2, 103f501680, da5d78b9d1, 634058dda0, 34d1b37101, 73c6ae7711,
a8f78125b9, e1495354e4, 4fa9c30a45, c3457af795, fe3c02699d, 878a299541,
a1193d6490, d69967143e, d6917db4be, bda224cb25, 621584cffe, 7a5f46198d,
35195a56a1, 204a90d81f, e0de82caa7, 8c3015cccb, 4e3f346aa5, 5e418b25f9,
4444753edd, c7bf937af5, 4da0d00b55, 48812cf33e, 35c6235b8d, 60c61cc2e5,
9002761b41, 13ae097e20, ec193e1a01, d3987b2930, 72f04f1a38, 97e59f09fa,
d0ab39112e, 9533fc037c, 2b88561cf2, 6c4387daf5, e1d26d35be, 0cfe974a89,
0eefca067f, 887a93b198, 4bc8c676bb, 6f49f522ef, a9c9c1517e, 8dcfb878a8,
1471274fa7, 53cad8e44a, 46146dfd39, 666b660284, 51e85b347b, ec3d4fac00,
1c72b04b56, 2ba621c750, d123d674c1, bd0e9ed437, dd61d903df, 80c4a94706,
b4d0c90bf4, 52c8ec0373, 7bc7761b8c, 25f97e926f, e72a7d7b0b, 9f913803e9,
8bf84e7d45, a9cbf9c41b, 7abdff3086, ae13ee3d70, a68259572b, 081ae4fa44,
8d064b29e2, 41dab05200, 6024888af8, 5110dc72c0, b0dd580a89, 2ff7f5744f,
45c36a9eab, ec4186ce12, 72cac683e4, c2a35c1cda, 11485838c2, 5675d78833,
55c1764e90, bfd8f6e6c9, 7ecd1f0c69, 90c7a87887, 4f414dd6e7, 211bc5d102,
3299ae3116, 827cf4f886, 5665c024cb, 08aa2705d5, 8e1848197c, d7bb6b37a7,
9d01f08ec6, fad633fbe1, 11ce6f452e, 644bad9729, ae3bc4a0ef, b2ce7699e3,
f7e0c9bb83, 52c7a3d653, 7806308f33, fed208bfca, 9198cfb4ea, 8c469815cf,
731b825695, c52bce43db, 0d2fcc0dcd, 1917b209f2, b274456ce1, d44e6acd1d,
7e0cd25696, 8909bde9ab, 68ea4a8c3d, 69ea9c09d1, 5a4a785006, b73867d769,
f0c34e1213, 84044f37f3, 479f5d7ef6, 3cc37fd487, f8fcb6a78c, f0ad38da27,
d0c65e9e95, 3b6c7988f6, f9ae3d98b7, e4d4872dab, 14dae0dc93, 86a40b655d,
4eb6fc23a2, 15617297cc, 4b67aa9ef6, 3e7d2aeb07, 4ea77bfd15, 2719b742ab,
f04cf99ee6, 75285a7eb1, 74ef677020, c9b3a8fbda, 62d6ece152, 536096501f,
3355e056eb, 6a48e4aea8, 04259896e2, 189ac8208a, ba98512f97, 223bd973ab,
a3f51855c3, 8b4fb018b7, 0ea68d418b, 6d4bf200ad, c130c3fc0c, 8a31f7bca5,
f8b4ac12f1, f85796a921, ef45ce3035, e4f8e5f46e, d5b6c6d94a, c760eeb8b3,
0b9d012be8, 1f4ec0563d, 2df005b843, dc32a44804, 1cada35274, 4cfdd73249,
b8f34cf72e, 53af427bb2, 1329409f2a, 1b8680f8cd, 2bbde474ef, 2f5b0c89b1,
6693a52081, 684bcd8812, e1f14510fa, 72a810a799, 70c3e97e41, 7c4ec17eff,
8a75ad7924, fd77828200, 53d9c45013, 968c41829e, d39934abe3, 6e1a21ba55,
bc4f71372d, 57246af7d1, f49e4866ac, 1e1a18c45a, a14ef483ff, d92532c7b2,
aab33f0e2a, f3325c3338, 4cf5bc3e60, c98b6fe013, 619812a1a7, f8c8de2764,
6dd2827fbb, f277c7a6a4, 1ca240fede, 16e8599e94, b554843889, df0638b0a0,
d664c89912, c5ab1f09c8, d38d1679e2, 31bbd0d793, b24fe36b2a, c1c830a735,
75504539c3, a74b9ca19c, c6bb6d2d5c, 99dc8bb20b, 92c46be756, 7a626ec98d,
0891ac2eb6, 7f54132e48, c47d551843, 4122aba5f9, 5aeb95cc7d, 8ca1af9f3c,
4019231330, 8f320d0e09, 5851739c15, b69926d9fa, b132a53086, 135625b53d,

05d7715782, e3e4134877, d3069da8bb, 747c44785c, 59a6459751, 5a75f7a1cf,
63936d7de5, d301ca58cc, 9c404cac0c, 78613981ed, d072fda75b, 390c51b987,
571ca79c71, 67cfbff9b1, 7a8b5456ca, efe5ac6901, c4571bedc8, 57a344ab1a,
d205e3cff5, 39befd99fb, 3b23792b84, 6a5de0535f, ced7705ab2, 06ce2b51fb,
dd415df125, 2f710f66bd, 0f6c2163de, e470345ede, 6b2261888d, 6671643f31,
de74a601d3, f14c18cf6a, f38e5aa5b4, 57427393e9, 6da21f5c91, 738e5a0a14,
0f2ab8b1ff, 4a4b50571c, a263ce8a87, 031ccc99b1, ab45078265, 01e64dd36a,
e0c478f775, b5b9da5364, 72acb8cdf6, c922365dd4, df11931ffe, ad3fada9d9

References

1. Berman, M., Rannen Triki, A., Blaschko, M.B.: The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4413–4421 (2018)
2. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09 (2009)
3. He, K., Gkioxari, G., Dollár, P., Girshick, R.B.: Mask r-cnn. 2017 IEEE International Conference on Computer Vision (ICCV) pp. 2980–2988 (2017)
4. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (12 2014)
5. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
6. Oh, S.W., Lee, J.Y., Xu, N., Kim, S.J.: Video object segmentation using space-time memory networks. Proceedings of the IEEE International Conference on Computer Vision (2019)
7. Robinson, A., Lawin, F.J., Danelljan, M., Khan, F.S., Felsberg, M.: Learning fast and robust target models for video object segmentation (2020)
8. Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., Sang, N.: Learning a discriminative feature network for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1857–1866 (2018)