# Learning to Better Segment Objects from Unseen Classes with Unlabeled Videos — Supplementary Material —

Yuming Du Yang Xiao Vincent Lepetit
LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-vallée, France
{yuming.du, yang.xiao, vincent.lepetit}@enpc.fr
https://dulucas.github.io/gbopt/

In this supplementary material:

- We give the derivation of our objective function appearing in Eq. (2) of the main paper (Section 1);
- We detail our two-stage optimization method (Section 2);
- We detail our method to compute the "synthetic optical flow" $\overline{F}_t$ in Eq. (4) of the main paper (Section 3);
- We provide implementation details of the evaluation of our method and its comparison with related methods (Section 4).

## 1. Formulation

As discussed in Section 1 of the main paper, our goal is to improve the performance of a pre-trained class-agnostic instance segmentation on unseen classes. We do this by first using this model for mask generation on unlabeled video sequences. Then we apply our method to automatically select the correct masks and use them to train the instance segmentation model and boost its performance on unseen classes.

Given a video of $T$ frames, we start from a set of mask candidates $\mathcal{M}_t = \{M_{t,1}..M_{t,N}\}$ for each frame $I_t$ obtained using a pre-trained class-agnostic instance segmentation model, with $N$ the number of masks candidates in $I_t$. To select the mask candidates that actually correspond to objects, we want to exploit several cues and a constraint:

- The **"Background cue"**: Segmenting typical backgrounds such as sky or grass gives us a cue about where the objects are, whether they move or not.
- The **"Flow cue"**: The optical flow between consecutive frames gives us a cue about the moving objects.
- The **"Consistency cue"**: The selected masks should be consistent not only between consecutive frames, but also over long sequences.
- The **"Non-overlapping constraint"**: An additional constraint that is usually overlooked is that the masks should not overlap: Ideally, one pixel in the image can belong to at most one mask.

To combine these cues to select the correct masks, we rely on a Bayesian framework. The selection problem can be formalized as maximizing:

$$P\big(\mathbf{C}_1, .., \mathbf{C}_T | I_1, .., I_T\big) \,, \tag{1}$$

where $\mathbf{C}_t$ is a set of binary random variables, with $\mathbf{C}_{t,i} = 1$ corresponding to the event that the mask $M_{t,i}$ is selected and 0 that it is not. By applying Bayes' theorem, the problem becomes the maximization of:

$$P\big(I_1, .., I_T | \mathbf{C}_1, .., \mathbf{C}_T\big) P\big(\mathbf{C}_1, .., \mathbf{C}_T\big) \,. \tag{2}$$

To keep both the image and optical flow cues, we use a Product-of-Experts [11] with cliques made of two consecutive images to model the first term as:

$$P\big(I_1, .., I_T | \mathbf{C}_1, .., \mathbf{C}_T\big) \propto \prod_t P\big(I_t, I_{t+1} | \mathbf{C}_1, .., \mathbf{C}_T\big) \,. \tag{3}$$

We make the standard assumptions that the successive states (the $\mathbf{C}_t$) follow a Markov process, and that the measurements at time $t$ (the $I_t$) depend only upon the current state ($\mathbf{C}_t$). We denote $\delta_{t,i}$ the realization of the random binary variable $\mathbf{C}_{t,i}$ (thus $\delta_{t,i} \in \{0, 1\}$) and $\Delta_t = \{\delta_{t,1}, .., \delta_{t,N}\}$ the realization of $\mathbf{C}_t$. After standard derivations, the optimization problem becomes:

$$\underset{\{\Delta_1, .., \Delta_T\}}{\arg\max}\; P(\mathbf{C}_0) \prod_t P\big(I_t, I_{t+1} | \mathbf{C}_t, \mathbf{C}_{t+1}\big) P(\mathbf{C}_{t+1} | \mathbf{C}_t) \,. \tag{4}$$

$P\big(I_t, I_{t+1} | \mathbf{C}_t, \mathbf{C}_{t+1}\big)$ is high when the image likelihoods $P\big(I_t | \mathbf{C}_t\big)$ and $P\big(I_{t+1} | \mathbf{C}_{t+1}\big)$ are high and when the object motions between $I_t$ and $I_{t+1}$ are consistent with states $\mathbf{C}_t$ and $\mathbf{C}_{t+1}$. $P\big(I_t, I_{t+1} | \mathbf{C}_t, \mathbf{C}_{t+1}\big)$ is thus directly proportional to:

$$P\big(I_t | \mathbf{C}_t\big) P\big(I_{t+1} | \mathbf{C}_{t+1}\big) P\big(F_t | I_t, I_{t+1}, \mathbf{C}_t, \mathbf{C}_{t+1}\big) \,, \tag{5}$$

where $F_t$ is the optical flow for the pair of frames $(I_t, I_{t+1})$. After taking the log of Eq. (4), the optimization problem can be written as:

$$\arg\min_{\{\Delta_1,..,\Delta_T\}} \sum_t \lambda_I \mathcal{L}_I\big(I_t, \Delta_t\big) + \lambda_F \mathcal{L}_F\big(F_t, I_t, I_{t+1}, \Delta_t, \Delta_{t+1}\big) + \lambda_p \mathcal{L}_p(\Delta_t, \Delta_{t+1}) \,, \quad (6)$$

under the non-overlapping constraint. $\lambda_I$, $\lambda_F$, and $\lambda_p$ are weights. We take $\lambda_I = \lambda_F = 1$ and $\lambda_p = 0.5$ in all our experiments.

## 2. Two-Stage Optimization

In this section, we detail our efficient two-stage optimization of the objective function in Eq. 6. Recall that, in the first stage, we select the most promising combinations of masks for each frame independently using the background loss $\mathcal{L}_I$ and under the constraint that the selected masks should not overlap, we formulate this problem as a K-shortest path search problem and detail our method below in Section 2.1. Then, we detail in Section 2.2 the second stage where we optimise the full objective function on the whole video while considering only the top combinations selected during the first stage.

### 2.1. Image-Level Optimization

Given a frame $I_t$, we obtain a set of mask candidates $\mathcal{M}_t$ using MP R-CNN. To identify the top-$K$ combinations of masks according to $\mathcal{L}_I\big(I_t, \Delta_t\big)$ without having to perform an exhaustive evaluation, we iteratively apply Dijkstra's algorithm [6]. To do this, we first construct a binary tree $\mathcal{B}(\mathcal{V}, \mathcal{E})$.

As shown in Figure 1 for the easy case where the masks in $\mathcal{M}_t$ do not overlap, nodes $\mathcal{V}_i = \{\mathcal{V}_{i,j}\}_j$ at the same level in the tree correspond to the state of the $i$-th mask $M_{i,t} \in \mathcal{M}_t$ *i.e.* if it is selected or not, and edges $\mathcal{E}_i = \{\mathcal{E}_{i,j}\}_j$ at the same level are weighted with the contribution to $\mathcal{L}_I\big(I_t, \Delta_t\big)$ when mask $M_{i,t}$ is selected or not. In this way, the problem of finding the most promising combinations of masks for a frame thus becomes to find the K-shortest paths in the binary tree $\mathcal{B}(\mathcal{V}, \mathcal{E})$.

The time complexity for finding these K-shortest paths by exhaustive search is $O(2^N)$, for $N$ masks in each image, which is computationally prohibitive. We thus iteratively apply Dijkstra's algorithm [6] to efficiently find the top-$K$ shortest paths. We get the shortest path $\mathcal{H}_1$ by directly applying Dijkstra's algorithm on the binary tree $\mathcal{B}(\mathcal{V}, \mathcal{E})$.

To find the next shortest path, we consider a set $U$ of paths, initialized with $\mathcal{H}_1$ as its only element. The whole procedure consists of two nested loops. The first loop is over each path $\mathcal{H}$ in $U$. Given $\mathcal{H}$, the second loop is over each edge $\mathcal{E}_{i,j}$ of $\mathcal{H}$. We set the weight of $\mathcal{E}_{i,j}$ to infinity and apply Dijkstra's algorithm on the resulting tree. Each time we run Dijkstra's algorithm, we obtain an "intermediate shortest path" different from $\mathcal{H}$. After exiting the two loops, we add all the intermediate shortest paths to $U$ and remove $\mathcal{H}_1$ from it. The shortest path in $U$ is now the second shortest path $\mathcal{H}_2$ in the tree. We can then repeat the above procedure to get the next shortest paths.

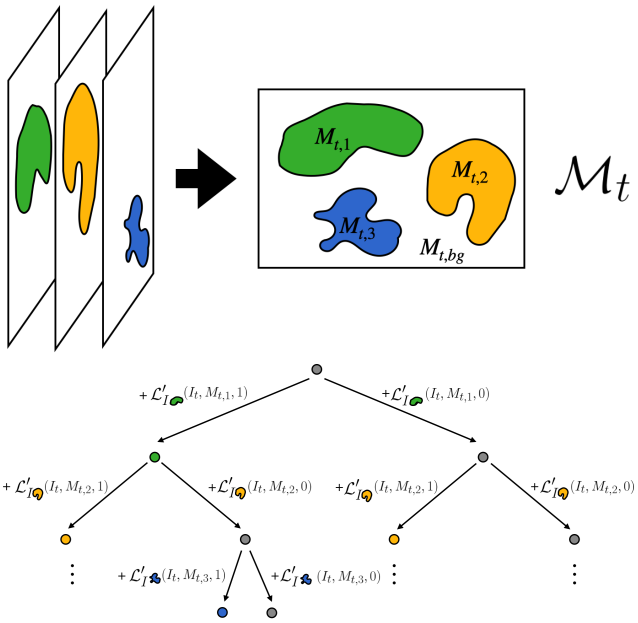


Figure 1. **Image-Level Optimization. Example of a tree $\mathcal{B}(\mathcal{V}, \mathcal{E})$ in the non-overlapping case.** Nodes at the $i$-th level correspond to the selection ($\delta_{t,i} = 1$) or non-selection ($\delta_{t,i} = 0$) of the $i$-th mask. Edges at the $i$-th level are weighted according to the image term $\mathcal{L}_I$ and depending whether or not the $i$-th mask is selected. As there is no overlaps among the three masks, the weights of edges can be calculated independently. A colored node corresponds to the selection of the mask with the same color; a gray node corresponds to the case where the mask is not selected.

This method decreases the time complexity from $2^N$ to $O(KN^3)$, where $N$ is the number of masks per image and $K$ is the number of optimal combinations required.

Below, we start with the easier case, where there is no overlap among the mask candidates in $\mathcal{M}_t$. In this case, each pixel in the image belongs to at most one mask candidate. The non-overlapping constraint is naturally satisfied, and the contributions of the masks to $\mathcal{L}_I\big(I_t, \Delta_t\big)$ are independent of each other. Then, we dive into the overlapping case and show that we can transform the overlapping case to the non-overlapping case with a decomposing-then-hashing operation, and calculate $\mathcal{L}_I\big(I_t, \Delta_t\big)$ exactly.

**Non-Overlapping Case.** We first consider the easier case where there is no overlap between the mask candidates, illustrated by Figure 1. In such case, image term $\mathcal{L}_I\big(I_t, \Delta_t\big)$ can be written as a sum over the masks in $\mathcal{M}_t$ plus a special additional mask made of the pixels that do not belong to any

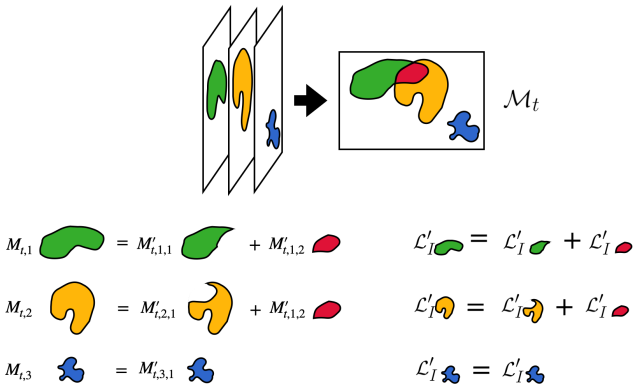


Figure 2. **Image-Level Optimization. Decomposing-then-hashing for the overlapping case.** The red mask represents the overlapping region between $M_{t,1}$ and $M_{t,2}$. The contribution of $M_{1,t}$ can thus be decomposed into the sum of contributions of the two non-overlapping green and red sub-masks.

mask(recall that $\Delta_t = \{\delta_{t,1}, .., \delta_{t,N}\}$):

$$\begin{aligned}\mathcal{L}_I\big(I_t, \Delta_t\big) &= \mathrm{CE}\big(\mathrm{Bg}(I_t), 1 - \overline{\mathrm{Fg}}(\Delta_t)\big) \\ &= \textstyle\sum_{M_{t,i} \in \mathcal{M}_t} \mathcal{L}'_I(I_t, M_{t,i}, \delta_{t,i}) + \bar{\mathcal{L}}_I(M_{t,bg})\end{aligned} \tag{7}$$

with

$$\begin{aligned}\mathcal{L}'_I(I_t, M_{t,i}, \delta_{t,i}) = &- \textstyle\sum_{x \in M_{t,i}} (1 - \delta_{t,i}) \log \mathrm{Bg}(I_t)(x) \\ &+ \delta_{t,i} \log\big(1 - \mathrm{Bg}(I_t)(x)\big)\,,\end{aligned} \tag{8}$$

where the sum is over image locations $x$ in mask $M_{t,i}$, and

$$\bar{\mathcal{L}}_I(M_{t,bg}) = - \textstyle\sum_{x \in M_{t,bg}} \log \mathrm{Bg}(I_t)(x)\,. \tag{9}$$

Term $\bar{\mathcal{L}}_I(M_{t,bg})$ in Eq. (7) is constant. $M_{t,bg}$ represents the background image generated for the selected masks. We can ignore it here for the selection of the top-$K$ mask combinations. Note that it still needs to be included in $\mathcal{L}_I$ for the Video-Level Optimization. We can use the value of $\mathcal{L}'_I(I_t, M_{t,i}, 1)$ as weight for an edge corresponding to the selection of mask $M_{i,t}$, $\mathcal{L}'_I(I_t, M_{t,i}, 0)$ as weight for an edge when mask $M_{i,t}$ is not selected.

**Overlapping Case.** In general, the mask candidates predicted by MP R-CNN overlap, and the contributions of masks $\{M_{i,t}\}_i$ to $\mathcal{L}_I\big(I_t, \Delta_t\big)$ are no longer independent. We propose a decomposing-then-hashing method which first decomposes each mask into a group of non-overlapping sub-masks, then identify the contribution of each mask using the sum of the sub-masks. When adding a novel node to the path, we fit the non-overlapping constraint by inspecting if there are some sub-masks selected in both the current node and the nodes in this path.

As shown in Figure 2, we first decompose each mask $M_{t,i}$ into a combination of sub-masks $\{M'_{t,i,j}\}_j$ depending on whether the pixels in the mask belong to other masks

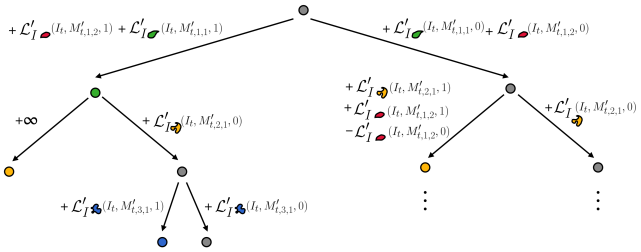


Figure 3. **Image-Level Optimization. Example of $\mathcal{B}(\mathcal{V}, \mathcal{E})$ in the overlapping case.** Each time a node is added in the path, the weight of the edge is calculated by considering all the previous nodes in the path.

or not. The whole image can thus be decomposed into the combination of all sub-masks $\{M'_{t,i,j}\}_{i,j}$ and $M_{t,bg}$, with no overlaps among the sub-masks in $\{M'_{t,i,j}\}_{i,j}$. Then, we can construct a hash table where the key is the index of each mask $M_{t,i}$ and the value is the index of corresponding sub-masks $\{M'_{t,i,j}\}_j$ for each mask $M_{t,i}$.

Similar to the non-overlapping case, with the help of non-overlapping sub-masks, we have:

$$\begin{aligned}&\mathcal{L}'_I(I_t, M_{t,i}, \delta_{t,i}) = \\ &\quad - \textstyle\sum_{S \in \{M'_{t,i,j}\}_{i,j}} \sum_{x \in S} \Big( (1 - \delta_{t,i}) \log \mathrm{Bg}(I_t)(x) + \\ &\qquad \delta_{t,i} \log\big(1 - \mathrm{Bg}(I_t)(x)\big)\Big)\end{aligned} \tag{10}$$

which means that the contribution of each mask to $\mathcal{L}_I\big(I_t, \Delta_t\big)$ can be decomposed into the sum of the contributions of its corresponding sub-masks. Eq. (7) can thus be written as:

$$\mathcal{L}_I\big(I_t, \Delta_t\big) = \textstyle\sum_{S \in \{M'_{t,i,j}\}_{i,j}} \mathcal{L}'_I(I_t, S) + \bar{\mathcal{L}}_I(M_{t,bg})\,. \tag{11}$$

A tree for an overlapping case is shown in Figure 3. When searching for the shortest path in the graph, given a path, each time a node is added to the path, with the help of the hash table constructed aforementioned, we first check whether there are any sub-masks $M'_{t,i,j}$ already being assigned a certain "state" in the previous nodes (by state, we mean if it has been selected as foreground or selected as background).

If not, like the non-overlapping case, we consider the $\mathcal{L}'_I(I_t, M_{t,i}, \delta_{t,i})$ for the weight of the edge $B, \forall i > 0$.

If a sub-mask has already been assigned some state, there are two possible cases. The first case is that the sub-masks $M'_{t,i,j}$ selected both in the current node and a previous node have the same state. When the state is "background", as we have already counted the contribution of sub-masks in previous edges, the weight of the current edge is assigned to the sum of the contribution of the rest of masks. When the state is "foreground", since the masks should not overlap, the weight of the current edge is set to infinity to prohibit this path from becoming one of the top-$K$ shortest paths.

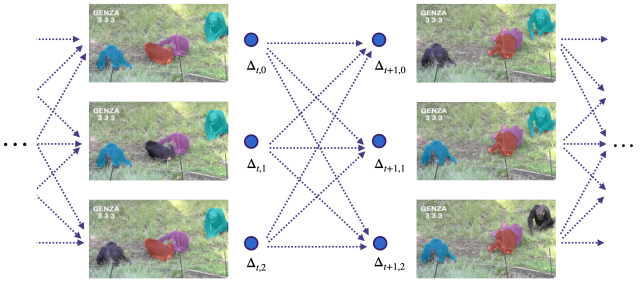

Figure 4. **Video-Level Optimization. Example of graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$,** represented only between two consecutive frames $I_t$ and $I_{t+1}$ for a video sequence of chimpanzees. Each image in the column represents a combination of masks $\Delta_t$, different masks are shown in different colors (Best seen in color).

The second case is that some sub-masks $M'_{t,i,j}$ selected both in the current node and a previous node possess different states. In order to calculate the weight of the current edge, along with the contribution of each sub-mask by considering the state assigned by the current node, we subtract the contribution of the shared sub-masks according to their assigned state in the previous node.

### 2.2. Video-Level Optimization

As shown in Figure 4, after the top-$K$ combinations of masks for each frame have been found by the first stage, we construct a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ to find the best combinations of masks for each frame by optimizing the objective function over the whole video. Node $\mathcal{V}_{t,k}$ represents the k-th combination of masks of frame $I_t$. $\mathcal{E}_{t,t+1,k,k'}$ represents the edge that connects node $\mathcal{V}_{t,k}$ and node $\mathcal{V}_{t+1,k'}$. The weight of edge is a sum over the image term, the flow term and the motion model term:

$$\begin{aligned} \text{Weight}_{\mathcal{E}_{t,t+1,k,k'}} = \\ \lambda_I \mathcal{L}_I\big(I_t, \Delta_t\big) + \\ \lambda_F \mathcal{L}_F\big(F_t, I_t, I_{t+1}, \Delta_t, \Delta_{t+1}\big) + \\ \lambda_p \mathcal{L}_p(\Delta_t, \Delta_{t+1})\,. \end{aligned} \tag{12}$$

In our case, as the weight of every edge is non-negative, we can use Dijkstra's algorithm [6] to select the combination of masks among the top-$K$ combinations for each frame.

## 3. Generation of Synthetic Flow $\overline{F}_t$

The motion of the selected objects should be consistent with the optical flow. To deal with camera motion, we consider that the optical flow of the background can be different from 0 (but uniform). Given two consecutive frames $I_t$ and $I_{t+1}$, together with two combinations of masks $\Delta_t$ and $\Delta_{t+1}$ for these frames, we propose a method to generate a "synthetic optical flow" which is then compared against the optical flow predicted by some optical flow model.

An overview of the generation pipeline is shown in Figure 5. We first generate two synthetic images $I'_t$ and $I'_{t+1}$ by cropping and pasting the content of selected masks in frame $I_t$ and $I_{t+1}$ to a background image $I_{bg}$ randomly selected from the Internet. Then we pass the pair of synthetic images into a optical flow model $g$ to generate a synthetic $\overline{F}'_t$. The pixels outside the selected masks are assigned the average flow in $F_t$ computed over the background.

The motivation for using a background image rather than a uniform background color which seems more natural is to make sure the objects are visible before computing the flow. For example, in the case of Figure 4, using a uniform black background would make optical flow prediction fail for the dark chimpanzees. Using a real image is a simple way to prevent this problem.

## 4. Implementation Details

### 4.1. Training Details

All our experiments were carried out on 4 Nvidia RTX 2080Ti GPUs. During the training, mixed precision training is used to reduce memory consumption and accelerate training.

**Pre-training.** For the pre-training on the COCO dataset, we use the open source repo of Mask-RCNN [17] and follow the training setting of [9], except that we adopt the class-agnostic setting, where all 80 classes are merged into a single "object" category. We use ResNet-50-FPN [15] as our backbone. As in [19], we call the Mask R-CNN model trained using this class-agnostic setting "MP R-CNN" for Mask Proposal R-CNN. Our backbone network is initialized with weights pre-trained on ImageNet [5]. During training, the shorter edge of images are resized to 800 pixels. Each GPU has 4 images and each image has 512 sampled RoIs, with a ratio of 1:3 of positives to negatives. We train our Mask R-CNN for 90k iterations. The learning rate is set to 0.02 at the beginning and is decreased by 10 at the 60k and 80k iteration. We use a weight decay of 0.0001 and momentum of 0.9.

**Fine-tuning.** We finetune MP R-CNN on the masks of Unseen-VIS-train generated by our method for 1k iterations. The initial learning rate is set to 0.002. The rest of the parameters and data augmentation strategies are set to be the same with the parameters used for pre-training. For the *DAVIS Unsupervised* benchmark, we finetune MP R-CNN on the masks generated on the 60 training videos of DAVIS2017 dataset, using the same hyper-parameters as for Unseen-VIS-train.

| Pretrained on COCO | Unseen-VIS-train GT | Unseen-VIS-train Generated Masks | Training Strategy | Unseen-VIS-test $AP$ | $AP_{50}$ | $AP_{75}$ | $AR_1$ | $AR_3$ | $AR_5$ | COCO val $AP$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ | | | – | 35.8 | 61.2 | 38.1 | 33.3 | 47.3 | 50.3 | 35.3 | 62.6 | 35.9 | 20.8 | 40.6 | 52.4 |
| ✓ | ✓ | | retrain | 51.2 | 80.2 | 56.8 | 45.0 | 56.8 | 59.5 | 34.9 | 61.8 | 35.7 | 20.6 | 40.2 | 52.3 |
| ✓ | | ✓ | retrain | 38.9 | 67.8 | 41.2 | 35.2 | 48.9 | 51.4 | 35.0 | 62.0 | 35.8 | 20.5 | 40.2 | 52.3 |
| ✓ | ✓ | | finetune | 50.8 | 80.9 | 54.6 | 43.6 | 58.6 | 60.6 | 24.0 | 44.9 | 23.4 | 15.3 | 28.7 | 33.6 |
| ✓ | | ✓ | finetune | 39.0 | 67.9 | 41.3 | 35.2 | 48.9 | 51.4 | 32.1 | 57.0 | 32.8 | 17.5 | 39.3 | 51.1 |

Table 1. Results on the COCO2017 validation dataset and Unseen-VIS-test adopting different training strategies. The performance of MP R-CNN fine-tuned on our generated masks on Unseen-VIS-train is much less affected compared to MP R-CNN fine-tuned on the ground truth masks. By training on the dataset created by naively mixing the training dataset of COCO with our generated masks on Unseen-VIS-train, we can achieve the same results on Unseen classes as fine-tuning while preserving the performance on Seen classes.

## 4.2. Implementation Details for the Related Methods

### 4.2.1 Foreground/Background Segmentation

In order to estimate the background regions present in the image, similar to [13], we adopt a FPN based network for segmentation. We use ResNest-200 [28] as our backbone. The regions that contain instances are considered as foreground and background otherwise to generate binary mask for training. We set the initial learning rate to 0.02, then decrease it by 10 at the 240k and 255k iteration. The batch size is set to 16. Scale augmentation is applied during training to further improves results.

### 4.2.2 Optical Flow Estimation

We use the model released by [24] trained on FlyingThings [18]. FlyingThings is a large-scale synthetic dataset for optical flow estimation. The dataset is generated by randomizing the movement of the camera and synthetic objects collected from the ShapeNet dataset [3]. The model for optical flow estimation is pre-trained on FlyingThings for 100k iterations with a batch size of 12, then for 100k iterations on FlyingThings3D with a batch size of 6.

### 4.2.3 Video Object Segmentation

**NLC [8].** We use the re-implementation code provided by [21]. In the original implementation of the NLC algorithm, an edge detector pre-trained on labeled edge images [7] is used, while the re-implementation of [21] replaces the trained edge detector with unsupervised superpixels. In order to obtain a motion segmentation for each frame, a per-frame saliency map based on motion is computed then averaged over superpixels calculated using method from [1]. Meanwhile, a nearest neighbor graph is computed over the superpixels in the video using location and appearance as features. Finally, the saliency is propagated across frames using a nearest neighbor voting scheme. For more details, please refer to [8, 21].

**FST [20].** We use the official code released by the authors to generate masks on the videos from Unseen-VIS-train.

**IOA [4].** We use the official code released by the authors. We first use the official code of the VideoPCA algorithm [23] to find the saliency regions which stand out the background in terms of motion and appearance. Then we follow the same approach as in [4] to keep only the top 20% saliency maps based on the mean score of the non-zero pixels. The remaining saliency maps are used for training of the PSPNet [29] with ResNet-50 [10] as backbone, and initialized with ImageNet pre-trained weights [5].

**UnOVOST [16].** We use the official code released by the authors, and use our MP R-CNN for mask proposals generation. All the hyper-parameters are kept the same as in the original work [16].

**TWB [2].** We use the official code released by the authors, and replace the human detector with our MP R-CNN. Since the confidence score for objects of unseen classes are sometimes low, we lower the thresholds for filtering detection proposals and regression results to 0.1. Note that [2] applies a Re-ID network to re-identify the persons that are missing in previous frames, in order to re-identify the animals in Unseen-VIS-train, we replace the Re-ID network trained on the pedestrian dataset with a ResNet-50 pre-trained on ImageNet [5].

**Data Distillation [22].** We use the official code released in Detectron2 [26] and follow the same pipeline as stated in [22] for filtering and retraining.

**UVC [14].** We use the model released by the authors for mask wrapping. The model is trained in a self-supervised manner on the Kinetics [12] dataset. During inference, we follow the same setting of UVC to resize the shorter edge of image to 480 pixels. For Zero-Shot UVC, the masks for the first frame are obtained by selecting the masks predicted

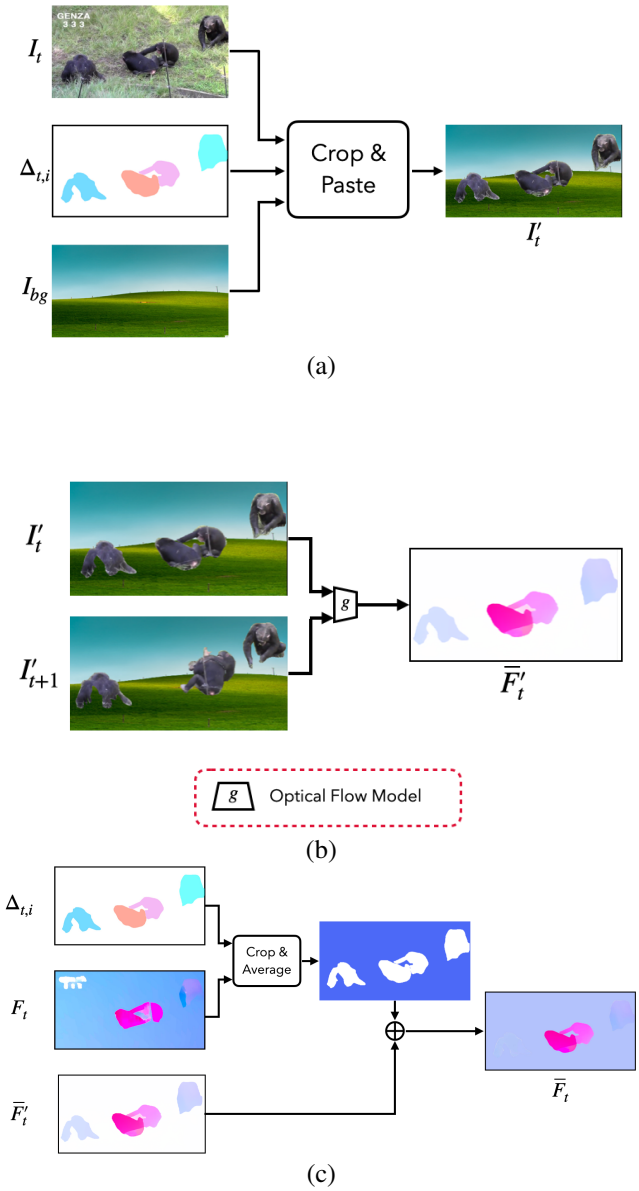


Figure 5. **Generation pipeline for "synthetic optical flow" $\overline{F}_t$.** **(a)** We first generate two synthetic images $I'_t$ and $I'_{t+1}$ by cropping and pasting the content of selected masks in frame $I_t$ and $I_{t+1}$ to a background image $I_{bg}$ randomly selected from the Internet. **(b)** We then feed the pair of synthetic images to a optical flow model $g$ to generate a synthetic $\overline{F}'_t$. **(c)** The pixels outside the selected masks are assigned the average flow in $F_t$ computed over the background.

by MP-RCNN on the first frame whose confidence score is larger than 0.1.

**RVOS [25].** We use the official codereleased by the authors, and replace the backbone of RVOS by ResNet-50. Input images are resized to 256×448. Each batch is composed with 4 clips of 5 consecutive frames. The model is trained on the 1089 videos of seen classes on YouTube VIS [27] for 50 epochs. The Adam optimizer is used to train our network and the initial learning rate is set to $1e{-}6$.

### 4.3. Results on COCO dataset

Detailed results are shown in Table 1. The performance of MP R-CNN on COCO-2017-val degrades dramatically after finetuning on either the ground truth of Unseen-VIS-train or the masks generated by our approach. Our guess is that this degradation results from the significant domain gap between images in YouTube-VIS and COCO. Interestingly, compared with the ground truth of Unseen-VIS-train, the performance of MP R-CNN is much less affected by finetuning on the masks generated by our approach, which demonstrates that our approach helps to preserve to some extent the information of the dataset used for pre-training.

For retraining, we use a naive strategy to train MP R-CNN from scratch on the mixture of COCO dataset and labels from Unseen-VIS-train. We add the images and annotations of Unseen-VIS-train into the COCO dataset, the annotations can be either ground truth or masks generated with our approach. Then we follow the training strategy given in Section 4.1. Mask R-CNN is trained on 120k training images of COCO for 90k iterations with batch size 16. The mixture of COCO and Unseen-VIS-train has 20k additional training images from Unseen-VIS-train, we thus train the our MP R-CNN on the mixture dataset for 110k iterations.

## References


[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC Superpixels Compared to State-Of-The-Art Superpixel Methods. *IEEE TPAMI*, 34(11), 2012. 5

[2] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixé. Tracking Without Bells and Whistles. In *CVPR*, 2019. 5

[3] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, and Others. Shapenet: An Information-Rich 3D Model Repository. In *arXiv*, 2015. 5

[4] Ioana Croitoru, Simion-Vlad Bogolin, and Marius Leordeanu. Unsupervised Learning from Video to Detect Foreground Objects in Single Images. In *ICCV*, 2017. 5

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 4, 5

[6] Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1), 1959. 2, 4

[7] Piotr Dollár and C. Lawrence Zitnick. Structured Forests for Fast Edge Detection. In *ICCV*, 2013. 5

[8] Alon Faktor and Michal Irani. Video Segmentation by Non-Local Consensus Voting. In *BMVC*, 2014. 5

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 4


Figure 6. **Qualitative results for the mask selection by UnOVOST and by our approach on new classes.** We show in this figure masks extracted from a sequence (from Unseen-VIS-train). First row and fourth row: Masks detected by baseline network MP R-CNN; Second row and fifth row: Masks selected by UnOVOST [16]; Third row and sixth row: Masks selected by our approach. Our method chooses the correct masks among the raw detections, while UnoVOST makes several mistakes. *For more examples, please refer to the video.*

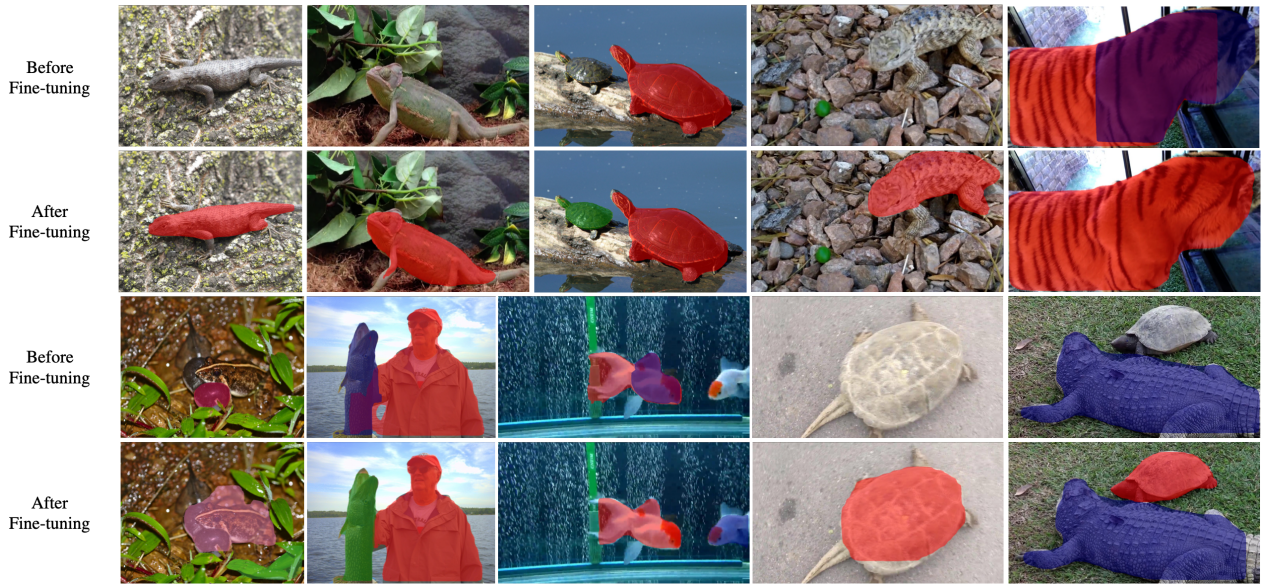


Figure 7. **Qualitative results for detected masks of new classes, before and after fine-tuning on our generated masks.** We show in this figure masks detected in *still images*. First row and third row: Detections by MP R-CNN before fine-tuning; Second row and fourth row: Detections by MP R-CNN after fine-tuning on our selected masks; The masks generated by our method results in a significantly better model for the new classes. *For more examples, please refer to the video.*

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 5

[11] Geoffrey E. Hinton. Products of Experts. In *International Conference on Artificial Neural Networks*, 1999. 1

[12] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, and Others. The Kinetics Human Action Video Dataset. In *arXiv*, 2017. 5

[13] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic Segmentation. In *CVPR*, 2019. 5

[14] Xueting Li, Sifei Liu, Shalini De mello, Xiaolong Wang, Jan Kautz, and Ming-Hsuan Yang. Joint-Task Self-Supervised Learning for Temporal Correspondence. In *NeurIPS*, 2019. 5

[15] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *CVPR*, 2017. 4

[16] Jonathon Luiten, Idil Esen Zulfikar, and Bastian Leibe. Un-

OVOST: Unsupervised Offline Video Object Segmentation and Tracking. In *WACV*, 2020. 5, 7

[17] Francisco Massa and Ross Girshick. Maskrcnn-Benchmark: Fast, Modular Reference Implementation of Instance Segmentation and Object Detection Algorithms in PyTorch. https://github.com/facebookresearch/maskrcnn-benchmark, 2018. 4

[18] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. In *ICCV*, 2016. 5

[19] Aljoša Ošep, Paul Voigtlaender, Mark Weber, Jonathon Luiten, and Bastian Leibe. 4D Generic Video Object Proposals. In *ICRA*, 2020. 4

[20] Anestis Papazoglou and Vittorio Ferrari. Fast Object Segmentation in Unconstrained Video. In *ICCV*, 2013. 5

[21] Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. Learning Features by Watching Objects Move. In *CVPR*, 2017. 5

[22] Ilija Radosavovic, P. Dollár, Ross B. Girshick, Georgia Gkioxari, and Kaiming He. Data Distillation: Towards Omni-Supervised Learning. In *CVPR*, 2018. 5

[23] Otilia Stretcu and Marius Leordeanu. Multiple Frames Matching for Object Discovery in Video. In *BMVC*, 2015. 5

[24] Zachary Teed and Jia Deng. RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. In *ECCV*, 2020. 5

[25] Carles Ventura, Miriam Bellver, Andreu Girbau, Amaia Salvador, Ferran Marques, and Xavier Giro-I-Nieto. RVOS: End-To-End Recurrent Network for Video Object Segmentation. In *CVPR*, 2019. 6

[26] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 5

[27] Linjie Yang, Yuchen Fan, and Ning Xu. Video Instance Segmentation. In *ICCV*, 2019. 6

[28] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, and Others. Resnest: Split-Attention Networks. In *arXiv*, 2020. 5

[29] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid Scene Parsing Network. In *CVPR*, 2017. 5