DMM-Net: Differentiable Mask-Matching Network for Video Object Segmentation

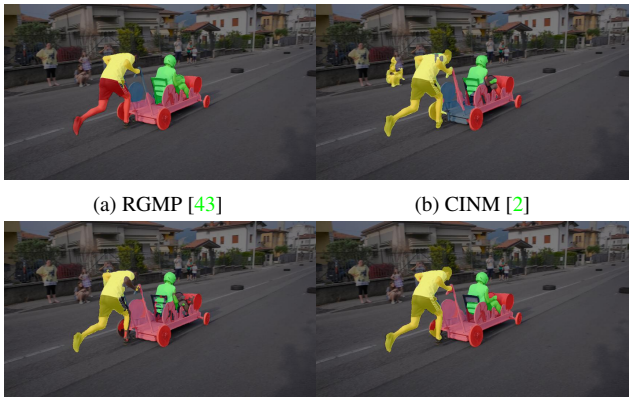
Xiaohui Zeng[1, 2*] Renjie Liao[1, 2, 3*] Li Gu[1] Yuwen Xiong[1, 2, 3]
Sanja Fidler[1, 2, 4] Raquel Urtasun[1, 2, 3, 5]
University of Toronto[1] Vector Institute[2] Uber ATG Toronto[3]
NVIDIA[4] Canadian Institute for Advanced Research[5]
{xiaohui, rjliao, yuwen, fidler}@cs.toronto.edu li.gu@mail.utoronto.ca urtasun@uber.com

Abstract

In this paper, we propose the differentiable mask-matching network (DMM-Net) for solving the video object segmentation problem where the initial object masks are provided. Relying on the Mask R-CNN backbone, we extract mask proposals per frame and formulate the matching between object templates and proposals at one time step as a linear assignment problem where the cost matrix is predicted by a CNN. We propose a differentiable matching layer by unrolling a projected gradient descent algorithm in which the projection exploits the Dykstra's algorithm. We prove that under mild conditions, the matching is guaranteed to converge to the optimum. In practice, it performs similarly to the Hungarian algorithm during inference. Meanwhile, we can back-propagate through it to learn the cost matrix. After matching, a refinement head is leveraged to improve the quality of the matched mask. Our DMM-Net achieves competitive results on the largest video object segmentation dataset YouTube-VOS. On DAVIS 2017, DMM-Net achieves the best performance without online learning on the first frames. Without any fine-tuning, DMM-Net performs comparably to state-of-the-art methods on SegTrack v2 dataset. At last, our matching layer is very simple to implement; we attach the PyTorch code (< 50 lines) in the supplementary material. Our code is released at https://github.com/ZENGXH/DMM_Net.

1. Introduction

Video object and instance segmentation problems have received significant attention [1, 35, 25] attributed to the recent availability of high-quality datasets, *e.g.*, YouTube-VOS [45], DAVIS [29, 30]. Given an input video, the aim is to separate the objects or instances from the background at the pixel-level. This is a fundamental computer vision task due to its wide range of applications including autonomous driving, video surveillance, and video editing.



(a) RGMP [43] (b) CINM [2]

(c) PReMVOS [24] (d) Ours

Figure 1. Visual comparison with some competitive methods on the last frame of the *soapbox* sequence of DAVIS 2017 dataset. Even though video segmentation quality generally degrades as time goes on, our model still provides better details.

Two main setups of this problem are *unsupervised* and *semi-supervised* which differ from each other in whether the ground-truth annotated masks of the object instances in the first frame of the video are provided [29, 30] during inference. In this paper, we focus on the *semi-supervised* setting, *i.e.*, the instance masks are provided for the first frames of the test videos. However, even with some annotated information at test time, this task is still very challenging. For example, the algorithm needs to deal with not only the dramatic appearance changes, occlusion and deformation but also potentially with large camera and object motion. Furthermore, the expectation from a good video instance segmentation model is to produce temporally cohesive and stable predictions, which presents an additional challenge.

Existing algorithms typically leverage pretrained deep neural networks to predict object instance masks. Some of them, *e.g.*, [25], directly predict the masks in a frame-independent way and achieve surprisingly good perfor-

*Equal contribution.

mance which verifies the great transfer ability of deep neural networks. Many algorithms leverage the previously predicted masks in various ways thus enabling the mask propagation over time. This strategy is demonstrated to improve temporal coherence and segmentation quality. Moreover, template matching between the reference and current frames is often exploited at pixel or mask level to deal with the object disappearance-reappearance phenomenon, occlusion, and fast motion. However, to the best of our knowledge, none of the existing work integrates the optimal matching algorithm into their framework, which may partly be due to the non-differentiable nature of the problem.

In this paper, we propose the differentiable mask-matching network (DMM-Net). We first extract mask proposals via a pretrained Mask R-CNN [13] in a frame-independent manner. For each time step, we then match proposals with the templates in the reference frame such that at most one mask proposal is assigned to one template instance. The matching cost between a pair of the template and proposal masks is determined based on the intersection-over-union (IoU) of masks and the cosine similarity between their feature maps predicted by a deep convolutional neural network (CNN). The key contribution of our paper is that we introduce a differentiable matching layer which solves the linear assignment problem.

Specifically, we unroll the projected gradient descent algorithm in which the challenging projection step is achieved by an efficient cyclic projection method known as *Dykstra*'s algorithm. The proposed double-loop matching algorithm is very simple to implement, guaranteed to converge, and achieves similar performance to the optimal matching obtained by the Hungarian algorithm [19, 26]. More importantly, it is fully differentiable which enables learning of the matching cost, thus having a better chance of handling large deformation and appearance changes. After matching, we adopt a refine head to refine the matched mask. Note that our main contribution is somewhat orthogonal to many existing work in a way that our differentiable matching can be integrated with other networks to potentially boost their performance. On DAVIS 2017 [30], SegTrack v2 [20] and YouTube-VOS [45] datasets, our model achieves either state-of-the-art or comparable performance.

2. Related Work

The problem of video object/instance segmentation has been extensively studied in the past [35, 1, 29, 30]. Many algorithms in this field rely on techniques like template matching which are popular in the object tracking and image matching literature [47, 23, 3, 31]. However, video object/instance segmentation is more challenging than tracking as it requires pixel-level object/instance mask as output rather than the bounding box. Meanwhile, it is also very different from matching in that it requires semantic understanding of the image rather than the similarities of low-level cues like color, motion and so on. Related literature can be classified based on the problem setup, *i.e.*, *unsupervised* vs. *semi-supervised*. Methods under the *unsupervised* category [5, 12, 42, 34] typically exploit the dense optical flow and appearance feature to group the pixels within the spatio-temporal neighborhood. In this paper, we focus on the *semi-supervised* setting. Based on whether an explicit matching between template and proposal mask is performed and at which level the matching is performed, we can further divide the related work into three sub-categories.

Pixel-level Matching Pixel-level matching network (PLM) [33] first exploits a Siamese type of CNN to extract features of the current frame and the masked reference frame. Based on the features, it computes the pixel-level similarity scores and the instance masks. VideoMatch [15] applies a CNN to extract features from the reference and the current frames, respectively. The feature of the reference frame is further split into the foreground and background ones which are used to compute the similarity with the feature of the current frame via a specially designed soft-matching module. The similarity-weighted combination of feature is used to predict the final mask. A fully convolutional Siamese network based approach (SiamMask) is proposed in [40]. It computes the depth-wise cross correlation between features of templates in the reference and the current frames. It also consists of mask, box, and score prediction heads similar to Mask R-CNN. Although these methods do not solve the exact matching problem, the pixel-level similarity scores output by a learnable CNN are still helpful for the task of mask prediction. However, since cross-correlation between different pixels from the template and the current frame is required, they tend to be intensive on computation and memory.

Mask-level Matching Instead of operating on the pixel-level, some methods including our DMM-Net resort to the mask-level matching. Based on pre-computed feature maps, DyeNet [21] iteratively uses the re-identification and the recurrent mask propagation modules to retrieve disappearing-reappearing objects and handle temporal variations of pose and scale separately. Authors in [9] propose to track the object parts in the video and also compute the similarity scores between the proposal and template parts in the reference frame in order to deal with the missing of tracking and background noises. In these work, matching is computationally light due to the number of masks/parts is significantly smaller than the number of pixels. However, they all exploit the greedy solution for matching, *i.e.*, for each template, it returns the maximum-scored assignment if the score is above some threshold otherwise returns no assignment. In contrast, we solve the matching problem via an

iterative solver which is better than the greedy solution in most cases as verified by the experiments.

No Explicit Matching Some of the recent works directly exploit deep neural networks to predict the masks. In [6, 25], a pretrained CNN is first fine-tuned to predict both the segmentation mask and contour per frame and then a boundary snapping step is applied to combine both results. Authors in [39] later extended this work by introducing an online adaptation step to bootstrap the foreground object segmentation. Video propagation network (VPN) [16] proposes a bilateral network along with a CNN to propagate the previously predicted masks and images. MaskRNN [14] exploits the optical flow, images and mask proposals in a recurrent fashion to predict the masks per frame. MaskTrack [28] uses a CNN which takes the last predicted instance mask and the current frame as input and outputs the refined mask. Spatial propagation network (SPN) [8] performs foreground segmentation and instance recognition simultaneously and then refines the instance masks using a spatial propagation module. Pixel-wise metric learning (PML) [7] formulates the video object segmentation as a pixel-wise retrieval problem where the embedding space is predicted by a CNN and then learned via triplet-constrained metric learning. Based on optical flow and a spatial CNN, a pixel-level spatio-temporal Markov random field (MRF) is built in [2] where approximate inference is achieved by a CNN. Authors in [46] propose two sub-networks to compute the visual feature of the templates and spatial attention map from the last frame respectively to guide the mask prediction. Relying on a U-Net, authors in [43] combine the concatenation of the current frame with last predicted mask and the concatenation of the reference frame with the template mask to predict the current mask. These works are orthogonal to ours in the sense that we can use some of their networks as our feature extractor, while our matching layer could also potentially improve their models.

3. Model

In this section, we introduce our approach which consists of two key components: differentiable mask matching and mask refinement. Our model assumes we have access to mask proposals in each frame. We first explain how we obtain the mask proposals. Then we describe our differentiable mask matching approach and discuss the mask refinement. Overview of our approach is illustrated in Fig. 2.

We assume a video has T frames. The mask templates in the first frame are denoted as $R = \{r_i | i = 1, \dots, n\}$ where n is the total number of instances throughout the video.

Mask Proposal Generation We first extract mask proposals independently per frame with a COCO-pretrained Mask R-CNN [13] (details in Sec. 4.1). We only keep the top-50 proposals based on their scores, ensuring that recall is sufficiently high. This step is performed off-line, *i.e.*, our method will run on top of these fixed proposals. We denote mask proposals in frame t as $P^t = \{p_j^t | j = 1, \dots, m^t\}$ where m^t is the total number of proposals at time t.

Algorithm 1 : Projected Gradient Descent for Matching

Input: $N_{\text{grad}}, N_{\text{proj}}, X, \alpha, C$
Initialization: $X^0 = X$
For $i = 1, 2, \dots, N_{\text{grad}}$:
    $X^i = X^{i-1} - \alpha C$
    $Y_0 = X^i, q_1 = 0, q_2 = 0, q_3 = 0$
    For $j = 1, 2, \dots, N_{\text{proj}}$:
        $Y_1 = \mathcal{P}_1(Y_0 + q_1), \quad q_1 = Y_0 + q_1 - Y_1$
        $Y_2 = \mathcal{P}_2(Y_1 + q_2), \quad q_2 = Y_1 + q_2 - Y_2$
        $Y_3 = \mathcal{P}_3(Y_2 + q_3), \quad q_3 = Y_2 + q_3 - Y_3$

    $X^i = Y_3$
Return $\hat{X} = \frac{1}{N_{\text{grad}}} \sum_{i=1}^{N_{\text{grad}}} X^i$

Differentiable Mask Matching The main motivation for performing object-level matching is to deal with the cases where large deformation, motion and dramatic appearance change are present. As aforementioned, proposal based matching is typically superior to optical flow based methods when motion is large. Moreover, we design a learnable matching cost which could potentially handle the dramatic appearance change and deformation.

In particular, at time step t, we use a CNN, denoted as f_θ, to extract features for the mask proposals P^t and the templates R in the first frame. Here θ denotes the learnable parameters. The details of the feature extractor is explained in Sec. 4.1. For the i-th mask template r_i (ground-truth mask in first frame) and the j-th mask proposal p_j^t, we compute their features as $f_\theta(r_i)$ and $f_\theta(p_j^t)$, respectively. The matching cost matrix C^t consists of the cosine similarity between features and IoU between masks as below,

$$C_{i,j}^t = (\lambda - 1) \cos\big(f_\theta(p_j^t), f_\theta(r_i)\big) - \lambda \text{IoU}(p_j^t, r_i), \quad (1)$$

where λ is a hyperparameter and $0 < \lambda < 1$. The overall cost matrix C^t is of size $n \times m^t$ where each row and column correspond to a template and a mask proposal, respectively. From here on out, we drop the superscript t for simplicity.

We now introduce how we solve the bipartite matching problem. In particular, we first formulate the minimum-cost bipartite matching as the following integer linear program-

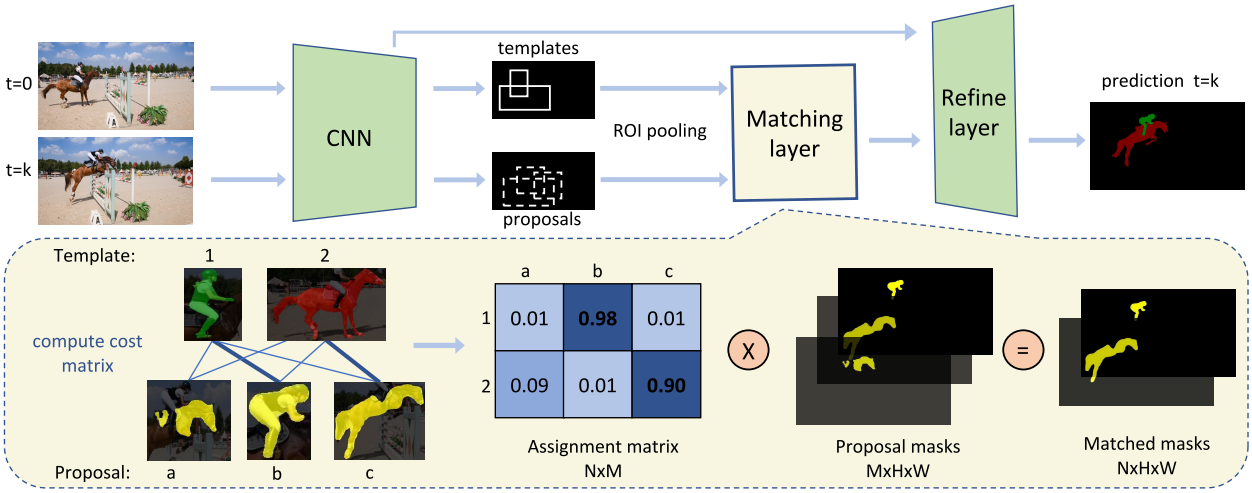


Figure 2. The overall architecture of our model. The yellow box in the bottom indicates the process of bipartite matching and the corresponding output masks as described in the Eq. (8).

ming (ILP) problem,

$$\begin{aligned} \min_{X} \quad & \mathrm{Tr}\left(CX^\top\right) \\ s.t. \quad & X\mathbf{1}_m = \mathbf{1}_n \\ & X^\top \mathbf{1}_n \leq \mathbf{1}_m \\ & X \geq 0 \\ & X_{i,j} \in \{0,1\} \qquad \forall (i,j) \end{aligned} \tag{2}$$

where $X \in \mathbb{R}^{n\times m}$ is the boolean assignment matrix. $\mathbf{1}_n$ and $\mathbf{1}_m$ are all one vectors with size n and m, respectively. Here we slightly abuse the notation such that subscript i, j denotes the (i, j)-th entry of the matrix. We add the constraint $X \geq 0$ which will be helpful in understanding the relaxed version introduced later. Note that the problem formulated in Eq. (2) and the standard linear assignment problem (LAP) are slightly different in that we replace $X^\top \mathbf{1} = \mathbf{1}$ with $X^\top \mathbf{1} \leq \mathbf{1}$. This is due to the fact that the number of proposals m is much larger than the number of templates n in our case, *i.e.*, X is a wide matrix.

To solve this ILP problem, one can introduce dummy variables to make X a squared matrix and then use the Hungarian method to optimize the standard LAP. However, this naive extension increases the time complexity to $O(m^3)$ and is not easy to back-propagate through. Also, we may not necessarily require the exact matching, *i.e.*, real-valued approximated assignment matrix X may be sufficient for the later stage. Therefore, we resort to the following linear programming (LP) relaxation,

$$\begin{aligned} \min_{X} \quad & \mathrm{Tr}\left(CX^\top\right) \\ s.t. \quad & X\mathbf{1}_m = \mathbf{1}_n \\ & X^\top \mathbf{1}_n \leq \mathbf{1}_m \\ & X \geq 0. \end{aligned} \tag{3}$$

Although there are many standard solvers for LP, *e.g.*, the simplex method and the interior point methods, we here introduce a differentiable and easy-to-implement projected gradient descent algorithm. The algorithm is presented in Alg. 1 where $N_{\text{grad}}, N_{\text{proj}}$ are the number of gradient decent (outer-loop) steps and the number of projection (inner-loop) steps, respectively.

At each iteration, we update X following the negative gradient direction. The major challenge lies in projecting the updated X onto the constraint set. It is not an easy task since the constraint set in Eq. (3) is the intersection between three closed convex sets.

To compute the projection, we adopt a cyclic constraint projection method, known as *Dykstra's algorithm* [11, 4] which is proved to be convergent for projection onto the non-empty intersection of finite closed convex sets. The key idea is to break the whole constraint set into multiple simple subsets such that we can easily find the projection operator. In particular, we can split the constraint set $\mathcal{C}$ into individual constraints, *i.e.*, $\mathcal{C} = \mathcal{C}_1 \bigcap \mathcal{C}_2 \bigcap \mathcal{C}_3$ where

$$\begin{aligned} \mathcal{C}_1 &= \{X | X\mathbf{1}_m = \mathbf{1}_n\} \\ \mathcal{C}_2 &= \{X | X^\top \mathbf{1}_n \leq \mathbf{1}_m\} \\ \mathcal{C}_3 &= \{X | X \geq 0\}. \end{aligned} \tag{4}$$

It is straightforward to derive the projection operators w.r.t. each constraint as follows,

$$\mathcal{P}_1(X) = X - \frac{1}{m}(X\mathbf{1}_m - \mathbf{1}_n)\mathbf{1}_m^\top \tag{5}$$

$$\mathcal{P}_2(X) = \begin{cases} X & \text{if } X^\top \mathbf{1}_n \leq \mathbf{1}_m \\ X - \frac{1}{n}\mathbf{1}_n(\mathbf{1}_n^\top X - \mathbf{1}_m^\top) & \text{otherwise} \end{cases} \tag{6}$$

$$\mathcal{P}_3(X) = X^+ \tag{7}$$

Note that $\mathcal{P}_3$ is just the ReLU operator. All these projection operators are differentiable and simple. *Dykstra's algorithm* works by iteratively projecting the corrected point onto individual constraint set in a cyclic order and then updating the correction by the difference between pre-projection and post-projection. The final solution is obtained by averaging the intermediate projected assignment matrices.

The convergence results of the Dykstra's algorithm are established in [11, 4, 10]. Relying on the convergence analysis under the framework of inexact projection primal first-order methods for convex optimization in [27], we derive the following convergence result of our projected gradient descent algorithm for matching.

Theorem 1. *Let $r_0 = \|X^0 - X^*\|_F$ where X^0 and X^* are the initial and optimal assignment matrices, respectively. Let the learning rate $0 < \alpha < \min(15r_0, r_0/\|C\|_F)$. There exists some constants $0 \leq c < 1$ and $\rho > 0$ such that at any iteration of outer loop i in Alg. 1, the error of projection $\|X^i - \mathcal{P}_{\mathcal{C}}(X^i)\|_F \leq \delta = \rho c^{N_{proj}}$ where X^i and $\mathcal{P}_{\mathcal{C}}(X^i)$ are the assignment matrix and its correct projection onto $\mathcal{C}$, respectively. Moreover, for any $0 < \epsilon < 1$, there exists a $N_{proj} \geq \log_{1/c}\left(\rho\sqrt{\frac{15K}{\alpha\epsilon}}\right)$ such that,*

$$\delta \leq \frac{\alpha\epsilon}{15r_0}$$

and after at most K iterations where

$$K = \left\lceil \frac{6r_0^2}{\alpha\epsilon} \right\rceil,$$

the output of Alg. 1 $\hat{X}$ is ϵ-optimal, i.e., *$\|\hat{X} - \mathcal{P}_{\mathcal{C}}(\hat{X})\|_F \leq \epsilon$ and $|\operatorname{Tr}\left(C\hat{X}^\top\right) - \operatorname{Tr}\left(CX^{*\top}\right)| \leq \epsilon$.*

We leave the proof to the supplementary material. In practice, the convergence is typically observed with moderately large N_{proj} and reasonable learning rate. The implementation of the overall algorithm is very simple. Please see an example implementation using PyTorch (less than 50 lines) and empirical convergence analysis with different hyperparameters in the supplementary material.

Mask Refinement After matching, for each template, we need to output one mask which is fed to the refinement stage. Recall that we obtain the optimal assignment $\hat{X}$ (approximately optimal if the previous algorithm does not run till convergence), we can compute a weighted combination of the proposal masks P. Specifically, we first resize mask proposals such that they have the same resolution as the input image. We then paste them into void images to get the full masks which have the same size as the input images, denoted as $\tilde{P}$. We obtain the matched mask $\hat{P}$ as:

$$\hat{P} = \hat{X} \otimes \tilde{P} \tag{8}$$

where $\hat{X} \in \mathbb{R}^{n \times m}$, $\tilde{P} \in \mathbb{R}^{m \times H \times W}$, $\hat{P} \in \mathbb{R}^{n \times H \times W}$, and $\otimes$ indicates the tensor contraction operator along the last and the first dimensions of $\hat{X}$ and $\tilde{P}$, respectively. Here H and W denote the height and width of the input image. Each spatial slice of $\hat{P}$ denotes the matched mask corresponding to a particular template. This process is shown in the yellow box of Fig. 2. The matched mask for the template will be used to compute the IoU score, shown in Eq. (1), at the next time step. Therefore, we propagate the latest mask information over time.

Given the output mask from matching, we then refine it using the template of the same instance. In particular, we construct the input by stacking multi-scale image features from the backbone, the matched mask and template mask. The multi-scale features are extracted from the last layer of the conv 2-5 blocks in the feature extractor backbone, respectively. Inspired by RVOS, we adopt a decoder containing four ConvLSTM [32] layers as the refinement module to predict the masks of all objects at each time step and carry over the memory and hidden states to the next time step.

4. Experiments

In this section, we compare our DMM-Net with a wide range of recent competitors on YouTube-VOS, DAVIS 2017 and SegTrack v2 datasets. YouTube-VOS has $3,471$ and 474 videos for training and validation, respectively. Among the 91 object categories in the validation set, 65 are seen in the training set while the other 26 are unseen. DAVIS 2017 has 60 and 30 video sequences for training and validation respectively and the average video length is around 70. The average number of instances per sequence are 2.30 and 1.97 for training and validation, respectively. For the SegTrack v2 dataset, there are 14 low resolution videos (947 frames in total) with 24 generic foreground objects. All of our experiments are conducted on NVIDIA Titan XP GPUs.

4.1. Implementation Details

We first introduce implementation details of our model.

Mask Proposal Generation In the stage of mask proposal generation, we use Mask R-CNN with ResNeXt-101-FPN as the backbone which is pretrained on COCO dataset [22]. Score threshold of the ROI head is set to 0. We resize the input image such that its short-side is no larger than 800. We first train the class-agnostic binary mask proposal network on COCO. Following the strategy used

	OL	$\mathcal{J}_\mathcal{S}$	$\mathcal{J}_\mathcal{U}$	$\mathcal{F}_\mathcal{S}$	$\mathcal{F}_\mathcal{U}$	$\mathcal{G}_\mathcal{M}$	FPS
OSMN [46]	✘	60.0	40.6	60.1	44.0	51.2	8.0
SiamMask [41]	✘	60.2	45.1	58.2	47.7	52.8	**55**
RGMP [43]	✘	59.5	-	45.2	-	53.8	7
OnAVOS [39]	✔	60.1	46.6	62.7	51.4	55.2	-
RVOS [37]	✘	63.6	45.5	67.2	51.0	56.8	24
S2S [44]	✘	66.7	48.2	65.5	50.3	57.7	6
OSVOS [6]	✔	59.8	54.2	60.5	60.7	58.8	-
S2S [44]	✔	**71.0**	**55.5**	**70.0**	**61.2**	**64.4**	-
DMM-Net	✔	59.2	47.6	62.6	53.9	55.8	-
DMM-Net+	✘	58.3	41.6	60.7	46.3	51.7	12
DMM-Net+	✔	60.3	50.6	63.5	57.4	58.0	-

Table 1. Results on YouTube-VOS (validation set) and frame-per-second (FPS) during inference for methods withour online learning. '$\mathcal{S}$' and '$\mathcal{U}$' denote the seen and unseen categories. 'OL': online learning. '+' means we use ResNet-101 as feature extractor instead of ResNet-50

Methods	$\mathcal{J}_\mathcal{M}$	$\mathcal{F}_\mathcal{M}$	$\mathcal{G}_\mathcal{M}$
MaskRNN [14]	45.5	-	-
OSMN [46]	52.5	57.1	54.8
FAVOS [9]	54.6	61.8	58.2
VideoMatch [15]	56.5	-	-
MSK [28]	63.3	67.2	65.3
RGMP [43]	64.8	68.6	66.7
FEELVOS [38]*	65.9	72.3	69.1
DyeNet [21]	67.3	71.0	69.1
DMM-Net	**68.1**	**73.3**	**70.7**

Table 2. Results without online learning on the validation set of DAVIS 2017 dataset. FEELVOS* also reports another better performed model which is trained on YouTube-VOS [45]. '-' means no public results available.

in [41], we then finetune the proposal network on the combination of COCO and YouTube-VOS with learning rate 0.02, batch size 8 and number of training iteration $200,000$.

Differentiable Mask Matching For the feature extractor f_θ, we use a COCO-pretrained Mask R-CNN with a ResNet-50-FPN backbone. We also try a ResNet-101 backbone of which the weights are initialized from the released model of RVOS [37]. We denote this model as DMM-Net+. Note that it is possible to share the backbone between the proposal network and the feature extractor of matching such that the overall model is more compact. However, sharing backbone leads to worse results in our experiments which may suggest that generating good proposals and learning good feature for matching require different representations. After we obtain the proposals for each frame, we perform ROI pooling for each proposal to extract multi-scale feature from the backbone and then average the feature spatially to obtain a single feature vector. Similar to the feature fed to the refinement layer, we obtain the proposals feature from the last layer of conv2-5 block in the backbone. Input image of the feature extractor is resized such that the short-side is no larger than 480 and 800 for DAVIS 2017 and SegTrack v2, respectively. For YouTube-VOS, we resize image to 255×448 in order to have a fair comparison with S2S [44] and RVOS [37]. The score weight λ used to compute the matching cost in Eq. (1) is set to 0.3 for DAVIS 2017 and YouTube-VOS, and 0.9 for SegTrack v2. For the mask matching, we set $N_{\text{grad}} = 40$, $N_{\text{proj}} = 5$ and learning rate $\alpha = 0.1$. Ablation study on the hyperparameters of the matching is left in the supplementary material. After the matching assignment matrix $\hat{X}$ is obtained, we also found it helpful to remove the non-confident matching by applying a differentiable masking $\hat{X} = \hat{X} \cdot \mathbf{1}[\hat{X} = \max(\hat{X})]$.

Methods	Online Learning	mIoU*	mIoU†
OSVOS [6]	✔	61.9	65.4
OFL [36]	✔	67.5	-
MSK [28]	✔	67.4	70.3
RGMP [43]		-	71.1
MaskRNN [14]		72.1	-
LucidTracker [17]	✔	-	77.6
DyeNet [21]		-	78.3
DyeNet [21]	✔	-	**78.7**
DMM-Net		**76.8**	76.7

Table 3. Results on the SegTrack v2 dataset. mIoU* is averaged over all frames whereas mIoU† is averaged over all instances.

Refinement Network A light version of refinement network containing only four ConvLSTM layers is used for experiments on YouTube-VOS to reduce the computational cost. The weights are randomly initialized and trained with the matching layer in an end-to-end manner. Input images are resized to be 255×448 for both training and inference. The refinement network outputs our final predicted masks.

A heavier version of refinement network is used for model trained on DAVIS 2017. We follow the U-Net style architecture as RGMP [43] and initialize the weights from their released model. The refinement network is also trained together with the matching layer.

Fine-tune Since our DMM-Net is end-to-end differentiable, we fine-tune it on the training sets of YouTube-VOS and DAVIS 2017. We use the Adam [18] optimizer with the learning rate $1.0e^{-4}$ for the weights initialized from pre-trained model and $1.0e^{-3}$ for the weights from random initialization. We set batch-size to 24 and train on YouTube-VOS dataset for 10 epochs in total. Data augmentation such as random affine transformation is applied during training. On DAVIS 2017, we use the same optimizer with learning rate $1.0e^{-7}$ and batch size 1. We fine-tune the model for 8 epochs in total.

Online Learning For online learning, we train both the proposal generator and the DMM-Net with the refinement module on the first frame of the validation set. We use the same learning rate and batch size as Section 4.1 except that the number of epochs is 100 for online learning.

Evaluation For YouTube-VOS and DAVIS 2017, we follow [30] and use the region ($\mathcal{J}$), boundary ($\mathcal{F}$) and their average ($\mathcal{G}$) score as the metrics. For SegTrack v2, there are two types of mean IoU adopted in the existing literature. Specifically, one can compute the IoU averaging over all instances per frame and then average over all frames as in [14], denoted as mIoU*. One can also compute the IoU per instance, average over the frames where the instance has appeared, and then average over all instances as in [21], denoted as mIoU†. We report both metrics on this dataset.

4.2. Main Results

YouTube-VOS We fine-tune both our DMM-Net and proposal net on YouTube-VOS. We first split the official training set into train-train, train-val and train-test splits. Our train-val split consists of 200 videos while the train-train split consists of 3000 videos. Our training is performed on the train-train split, and the best model is selected based on the performance on the train-val split. The final performance on the official validation set is reported in Table 1. Compared to the state-of-the-art method S2S, our model achieves competitive segmentation metrics with double speed. In general, we obtain a good trade off between the performance and running time. Moreover, we found that using a stronger backbone, *i.e.*, DMM-Net+, can further boost the performance.

DAVIS 2017 We compare with a wide range of recent competitors on the validation set of DAVIS 2017. For experiments on DAVIS, we only fine-tune our DMM-Net on the training set of DAVIS 2017 and use the proposal generator pretrained on COCO. The models without online learning are listed in Table 2. From the table, it is clear that without online learning, our method achieves the state-of-the-art performance. In Fig. 3, we show the qualitative results of our DMM-Net at different time steps (uniformly sampled percentage w.r.t. the whole video length) of each video sequence. From the figure, we can see that our model consistently keeps a very good segmentation quality as time goes on. We also show a visual comparison at the last frame of the *soapbox* sequence with other strong competitors in Fig. 1. It is clear that our model does a better job in segmenting the details of the persons and the soapbox. However, some failure cases still exist. For example, in the 100% column and 4th row of Fig. 3, the segmentation of the goldfish in the bottom-right corner is unsatisfying.

Matching	DMM-Net		Prop. Net		Train-val	
	Ft.	Unroll	Arch.	+ytb	$\mathcal{J}_\mathcal{M}$	$\mathcal{F}_\mathcal{M}$
Greedy	-	-	X101		57.1	68.1
Hungarian	-	-	X101		57.3	68.4
Ours	✘	-	X101		57.3	68.3
Ours	✔	2	R50		58.5	71.4
Ours	✔	2	X101		59.0	71.7
Ours	✔	3	X101		58.2	71.4
Ours	✔	2	X101	✔	**60.2**	**73.0**

Table 4. Ablation study evaluated on our train-val split of YouTube-VOS. Prop. Net: mask proposal network. '+ytb': using YouTube-VOS train-train split during the training of proposal net or not. 'Ft.': fine-tuning, 'Arch.': architecture for the proposal net, 'R50': ResNet-50, 'X101': ResNetXt-101.

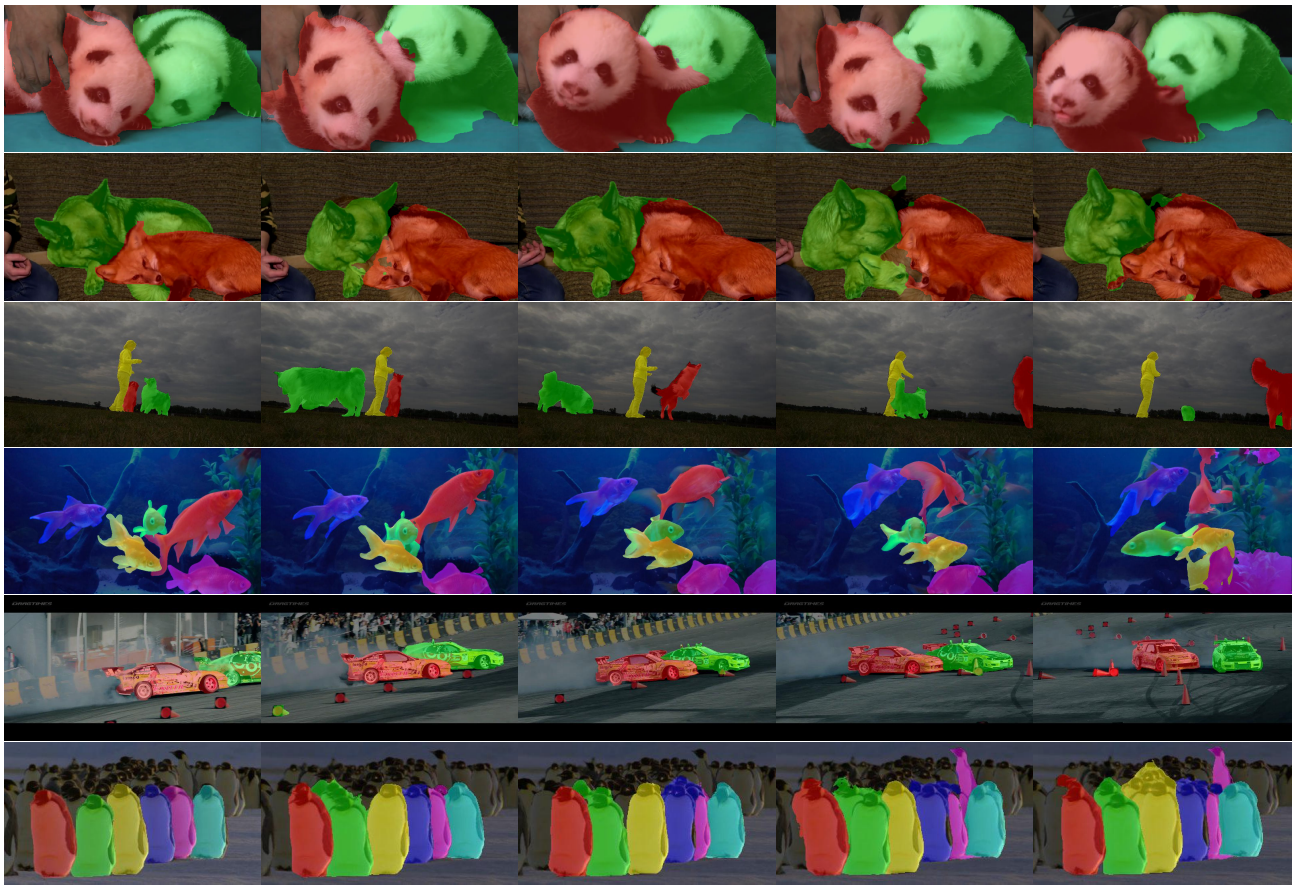
SegTrack v2 We test our DMM-Net model (fine-tuned on DAVIS 2017 training set) directly on the full SegTrack v2 dataset. We do not perform any fine-tuning on this dataset. Moreover, for simplicity, we again do not adopt any online learning such that we could fully test the generalization ability of our model. The quantitative results are listed in Table 3. From the table, we can see that without any fine-tuning and online learning, our DMM-Net achieves comparable performance to the state-of-the-art methods. We show some visual examples in the bottom two rows of Fig. 3. We can see that our model again has a consistently high segmentation quality across different time steps.

4.3. Ablation Study

In this section, we conduct thorough ablation study to justify the design choice and hyperparameters of our model.

Greedy vs. Hungarian vs. Our Matching Layer We first test the matching layer against the optimal matching using the Hungarian method and the popular greedy approximation during inference. For a fair comparison, we use the same set of mask proposals, the same feature extractor network pretrained on COCO dataset. We show the mean of $\mathcal{J}$ and $\mathcal{F}$ scores on YouTube-VOS dataset in Table 4. All the ablation results are obtained by training on our train-train split and evaluating on train-val split. From the table, we can see that our matching layer has similar performance compared to the optimal matching and is superior to the greedy one during inference.

End-to-End Fine-tuning We now study the effect of fine-tuning the whole model on the train-train split of YouTube-VOS. As shown in Table 4, the performance is improved significantly, which verifies the benefits of the end-to-end training and the differentiability of our matching layer.



(a) 0% (b) 25% (c) 50% (d) 75% (e) 100%

Figure 3. Visualization of our results on YouTube-VOS, DAVIS 2017 and SegTrack v2 at different time steps (percentage w.r.t. the whole video length). The first 2, the middle 2 and the last 2 rows correspond to the YouTube-VOS, DAVIS 2017 and SegTrack v2 datasets respectively.

Mask Proposal Network We also try different backbones for the mask proposal network and fine-tune the network on YouTube-VOS dataset. As shown in Table 4, the performance gain is pretty high by fine-tuning the proposal net on large scale video dataset such as YouTube-VOS.

Number of Unrolled Steps At last, we investigate the number of unrolled steps of the refinement network during training. As shown in Table 4, unrolling more than 2 step seems not helpful and increases the memory cost significantly. Note that during test we unroll from the beginning to the end of the video sequence.

5. Conclusion

In this paper, we propose the Differentiable Mask-Matching Network (DMM-Net) for solving the problem of video object segmentation. Relying on the pre-computed masks proposals, DMM-Net first conducts the mask matching between proposals and templates via a projected gradient descent method which is guaranteed to converge and fully differentiable. It enables the learning of the cost matrix of matching. Based on the template mask, we then refine the current matched mask to further improve the segmentation quality. We demonstrate that our model achieves the state-of-the-art or comparable performances under different settings of several challenging benchmarks. In the future, we would like to apply our differentiable matching layer to other backbone networks for the purpose of further boosting the performance. Moreover, exploring the mask matching in a longer temporal window, *i.e.*, a multi-partite matching problem similar to tracking, would be very interesting.

Acknowledgments

We gratefully acknowledge support from Vector Institute. RL was supported by Connaught International Scholarships, RBC Fellowship and NSERC. SF acknowledges the Canada CIFAR AI Chair award at Vector Institute. Part of this work was also supported by NSERC Cohesa grant, and Samsung. We are grateful to Relu Patrascu for infrastructure support.

References

[1] Vijay Badrinarayanan, Fabio Galasso, and Roberto Cipolla. Label propagation in video sequences. In *CVPR*, 2010. 1, 2

[2] Linchao Bao, Baoyuan Wu, and Wei Liu. CNN in MRF: Video object segmentation via inference in a CNN-based higher-order spatio-temporal MRF. In *CVPR*, 2018. 1, 3

[3] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *ECCV*. Springer, 2016. 2

[4] James P Boyle and Richard L Dykstra. A method for finding projections onto the intersection of convex sets in hilbert spaces. In *Advances in order restricted statistical inference*, pages 28–47. Springer, 1986. 4, 5

[5] Thomas Brox and Jitendra Malik. Object segmentation by long term analysis of point trajectories. In *ECCV*. Springer, 2010. 2

[6] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. One-shot video object segmentation. In *CVPR*, 2017. 3, 6

[7] Yuhua Chen, Jordi Pont-Tuset, Alberto Montes, and Luc Van Gool. Blazingly fast video object segmentation with pixel-wise metric learning. In *CVPR*, 2018. 3

[8] Jingchun Cheng, Sifei Liu, Yi-Hsuan Tsai, Wei-Chih Hung, Shalini De Mello, Jinwei Gu, Jan Kautz, Shengjin Wang, and Ming-Hsuan Yang. Learning to segment instances in videos with spatial propagation network. *arXiv preprint arXiv:1709.04609*, 2017. 3

[9] Jingchun Cheng, Yi-Hsuan Tsai, Wei-Chih Hung, Shengjin Wang, and Ming-Hsuan Yang. Fast and accurate online video object segmentation via tracking parts. In *CVPR*, 2018. 2, 6

[10] Frank Deutsch and Hein Hundal. The rate of convergence of dykstra's cyclic projections algorithm: The polyhedral case. *Numerical Functional Analysis and Optimization*, 15(5-6):537–565, 1994. 5, 11

[11] Richard L Dykstra. An algorithm for restricted least squares regression. *Journal of the American Statistical Association*, 78(384):837–842, 1983. 4, 5

[12] Matthias Grundmann, Vivek Kwatra, Mei Han, and Irfan Essa. Efficient hierarchical graph-based video segmentation. In *CVPR*, 2010. 2

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 2, 3

[14] Yuan-Ting Hu, Jia-Bin Huang, and Alexander Schwing. MaskRNN: Instance level video object segmentation. In *NIPS*, 2017. 3, 6, 7

[15] Yuan-Ting Hu, Jia-Bin Huang, and Alexander G Schwing. Videomatch: Matching based video object segmentation. In *ECCV*, 2018. 2, 6

[16] Varun Jampani, Raghudeep Gadde, and Peter V Gehler. Video propagation networks. In *CVPR*, 2017. 3

[17] Anna Khoreva, Rodrigo Benenson, Eddy Ilg, Thomas Brox, and Bernt Schiele. Lucid data dreaming for multiple object tracking. *arXiv preprint arXiv:1703.09554*, 2017. 6

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[19] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 2

[20] Fuxin Li, Taeyoung Kim, Ahmad Humayun, David Tsai, and James M Rehg. Video segmentation by tracking many figure-ground segments. In *ICCV*, 2013. 2

[21] Xiaoxiao Li and Chen Change Loy. Video object segmentation with joint re-identification and attention-aware mask propagation. In *ECCV*, 2018. 2, 6, 7

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 5

[23] David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004. 2

[24] Jonathon Luiten, Paul Voigtlaender, and Bastian Leibe. PReMVOS: Proposal-generation, refinement and merging for video object segmentation. *arXiv preprint arXiv:1807.09190*, 2018. 1

[25] Kevis-Kokitsi Maninis, Sergi Caelles, Yuhua Chen, Jordi Pont-Tuset, Laura Leal-Taixe, Daniel Cremers, and Luc Van Gool. Video object segmentation without temporal information. *IEEE TPAMI*, 2018. 1, 3

[26] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957. 2

[27] Andrei Patrascu and Ion Necoara. On the convergence of inexact projection primal first-order methods for convex minimization. *IEEE Transactions on Automatic Control*, 63(10):3317–3329, 2018. 5, 11, 12

[28] Federico Perazzi, Anna Khoreva, Rodrigo Benenson, Bernt Schiele, and Alexander Sorkine-Hornung.

Learning video object segmentation from static images. In *CVPR*, 2017. 3, 6

[29] Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *CVPR*, 2016. 1, 2

[30] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alex Sorkine-Hornung, and Luc Van Gool. The 2017 DAVIS challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*, 2017. 1, 2, 7

[31] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Deepmatching: Hierarchical deformable dense matching. *IJCV*, 120(3):300–323, 2016. 2

[32] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015. 5

[33] Jae Shin Yoon, Francois Rameau, Junsik Kim, Seokju Lee, Seunghak Shin, and In So Kweon. Pixel-level matching for video object segmentation using convolutional neural networks. In *ICCV*, 2017. 2

[34] Pavel Tokmakov, Karteek Alahari, and Cordelia Schmid. Learning motion patterns in videos. In *CVPR*, 2017. 2

[35] David Tsai, Matthew Flagg, Atsushi Nakazawa, and James M Rehg. Motion coherent tracking using multi-label MRF optimization. *IJCV*, 100(2):190–202, 2012. 1, 2

[36] Yi-Hsuan Tsai, Ming-Hsuan Yang, and Michael J Black. Video segmentation via object flow. In *CVPR*, 2016. 6

[37] 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

[38] Paul Voigtlaender, Yuning Chai, Florian Schroff, Hartwig Adam, Bastian Leibe, and Liang-Chieh Chen. Feelvos: Fast end-to-end embedding learning for video object segmentation. In *CVPR*, 2019. 6

[39] Paul Voigtlaender and Bastian Leibe. Online adaptation of convolutional neural networks for video object segmentation. *arXiv preprint arXiv:1706.09364*, 2017. 3, 6

[40] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. *arXiv preprint arXiv:1812.05050*, 2018. 2

[41] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *CVPR*, 2019. 6

[42] Wenguan Wang, Jianbing Shen, and Fatih Porikli. Saliency-aware geodesic video object segmentation. In *CVPR*, 2015. 2

[43] Seoung Wug Oh, Joon-Young Lee, Kalyan Sunkavalli, and Seon Joo Kim. Fast video object segmentation by reference-guided mask propagation. In *CVPR*, 2018. 1, 3, 6

[44] Ning Xu, Linjie Yang, Yuchen Fan, Jianchao Yang, Dingcheng Yue, Yuchen Liang, Brian L. Price, Scott Cohen, and Thomas S. Huang. Youtube-vos: Sequence-to-sequence video object segmentation. In *ECCV*, 2018. 6

[45] Ning Xu, Linjie Yang, Yuchen Fan, Dingcheng Yue, Yuchen Liang, Jianchao Yang, and Thomas Huang. Youtube-vos: A large-scale video object segmentation benchmark. *arXiv preprint arXiv:1809.03327*, 2018. 1, 2, 6

[46] Linjie Yang, Yanran Wang, Xuehan Xiong, Jianchao Yang, and Aggelos K. Katsaggelos. Efficient video object segmentation via network modulation. In *CVPR*, 2018. 3, 6

[47] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006. 2

6. Proof of Theorem 1

In this section, we prove our main result, *i.e.*, Theorem 1, based on the established convergence results of *Dykstra's algorithm* and the inexact projected gradient method for the general constrained convex minimization [27].

6.1. Convergence of Dykstra's Algorithm

First, we state the convergence result of *Dykstra's algorithm* from [10] without proof as Lemma 1 below.

Lemma 1. *Suppose K is the intersection of a finite number of closed half-spaces in a Hilbert space X. Starting with any point $x \in X$, there exists some constants $\rho > 0$ and $0 \leq c < 1$, such that the sequence of iterates $\{x_n\}$ generated by Dykstra's algorithm satisfied the inequality,*

$$\|x_n - \mathcal{P}_K(x)\| \leq \rho c^n \tag{9}$$

for all n, where $\mathcal{P}_K(x)$ is the nearest point in K to x.

6.2. Convergence of Inexact Projected Gradient Method

Now we turn to the inexact projected gradient method as described in [27] which aims at solving the following general constrained convex minimization,

$$\min_{x \in \mathbb{R}^n} \quad f(x) \qquad \text{s.t.} \quad x \in \mathcal{C}, \tag{10}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex and $\mathcal{C}$ is a general closed convex set. We only state the results which are relevant to our case, *i.e.*, f is differentiable and has Lipschitz continuous gradient with constant L_f. In particular, we have,

$$\|\nabla f(x) - \nabla f(y)\| \leq L_f \|x - y\| \qquad \forall x, y \in \mathbb{R}^n \tag{11}$$

The inexact projected gradient method is stated as below. Note that each projection step is inexact as it only requires

Algorithm 2 : Inexact Projected Gradient Method for General Constrained Convex Minimization

1: **Input:** $\delta > 0, N, x_0$
2: **For** $k = 1, 2, \ldots, N-1$:
3: $\quad z_k = x_{k-1} - \frac{1}{L_f} \nabla f(x_{k-1})$
4: $\quad$ Compute x_k such that $\|x_k - \mathcal{P}_{\mathcal{C}}(z_k)\| \leq \delta$
5: **Return** $\frac{1}{N} \sum_{k=0}^{N-1} x_k$

δ-approximation. Therefore, our proposed Alg. 1 can be regarded as a special instance of this general algorithmic framework of projected gradient method. We state the convergence result of this method from [27] without proof as Lemma 2 below.

Lemma 2. *Assuming the objective f is differentiable and has Lipschitz continuous gradient with constant L_f. Let $\{x_k\}$ be the sequence generated by the inexact projected gradient method. Given inner accuracy $\delta > 0$ and any $k \geq 0$, assuming $\|\nabla f(x^*)\| \leq L_f r_0$, then the following sublinear estimates for feasibility and suboptimality in $\hat{x}_k = \frac{1}{k} \sum_{i=0}^{k} x_i$ hold:*

$$\begin{aligned} &\|\hat{x}_k - \mathcal{P}_{\mathcal{C}}(\hat{x}_k)\| \leq \delta, \\ &f(\hat{x}_k) - f(x^*) \geq -\|\nabla f(x^*)\| \delta, \\ &f(\hat{x}_k) - f(x^*) \leq \frac{2 L_f r_0^2}{k} + 5 L_f r_0 \delta + 5 k L_f \delta^2, \end{aligned} \tag{12}$$

where $r_0 = \|x_0 - x^*\|$.

Note that the sublinear convergence rate is obtained provided that $\delta = O(\frac{1}{k})$.

6.3. Our Result

Now we are ready to prove our main result. Recall that our objective is $f(X) = \mathrm{Tr}(CX^\top)$. The gradient is $\nabla f(X) = C$. Therefore, any $L_f \geq 0$ is a Lipschitz constant satisfying Eq. (11). Also, our constraint set $\mathcal{C}$ is a closed convex set.

Theorem 2. *Let $r_0 = \|X^0 - X^*\|_F$ where X^0 and X^* are the initial and optimal assignment matrices respectively. Let the learning rate $0 < \alpha < \min(15 r_0, r_0 / \|C\|_F)$. There exists some constants $0 \leq c < 1$ and $\rho > 0$ such that the at any outerloop iteration i of Alg. 1, the error of projection $\|X^i - \mathcal{P}_{\mathcal{C}}(X^i)\|_F \leq \delta = \rho c^{N_{proj}}$ where X^i and $\mathcal{P}_{\mathcal{C}}(X^i)$ are the assignment matrix and its correct projection onto $\mathcal{C}$ respectively. Moreover, for any $0 < \epsilon < 1$, there exists a $N_{proj} \geq \log_{1/c} \left(\rho \sqrt{\frac{15K}{\alpha \epsilon}} \right)$ such that,*

$$\delta \leq \frac{\alpha \epsilon}{15 r_0}$$

and after at most K iterations where

$$K = \left\lceil \frac{6 r_0^2}{\alpha \epsilon} \right\rceil,$$

the output of Alg. 1 $\hat{X}$ is ϵ-optimal, i.e., *$\|\hat{X} - \mathcal{P}_{\mathcal{C}}(\hat{X})\|_F \leq \epsilon$ and $|\mathrm{Tr}\left(C\hat{X}^\top\right) - \mathrm{Tr}\left(CX^{*\top}\right)| \leq \epsilon$.*

Proof. At outer step i of Alg. 1, we run *Dykstra's algorithm* for N_{proj} inner steps. We know from Lemma 1 that there exists a $\rho_i > 0$ and $0 \leq c_i < 1$ such that,

$$\|X^i - \mathcal{P}_{\mathcal{C}}(X^i)\| \leq \rho_i c_i^{N_{\text{proj}}} \tag{13}$$

Denoting $\delta_i = \rho_i c_i^{N_{\text{proj}}}$, $\rho, c = \text{argmax}_{(\rho_i, c_i), i=1,\ldots,N_{\text{grad}}} \delta_i$, and $\delta = \rho c^{N_{\text{proj}}}$, we have,

$$\max_{i=1,\cdots,N_{\text{grad}}} \|X^i - \mathcal{P}_{\mathcal{C}}(X^i)\|_F \leq \delta, \tag{14}$$

where $\rho > 0$ and $0 \leq c < 1$.

Since δ is monotonically decreasing w.r.t. N_{proj}, there exists $N_{\text{proj}} \geq \log_{1/c}\left(\rho\sqrt{\frac{15K}{\alpha\epsilon}}\right)$, for any $0 < \epsilon < 1$, such that,

$$\delta \leq \frac{\alpha\epsilon}{15r_0}. \tag{15}$$

Our objective function $f(X) = \text{Tr}(CX^\top)$ is linear and constraint set $\mathcal{C}$ is closed and convex. Therefore, our Alg. 1 is an instance of the inexact projected gradient method. Moreover, since any $L_f \geq 0$ is a Lipschitz constant satisfying Eq. (11) in our case, we set $L_f = \frac{1}{\alpha}$ where α is is the learning rate and $0 < \alpha < \min(15r_0, \frac{r_0}{\|C\|_F})$. Now since all assumptions of Lemma 2 are satisfied, we apply it to our algorithm and obtain that,

$$\begin{aligned}
&\|\hat{X} - \mathcal{P}_{\mathcal{C}}(\hat{X})\|_F \leq \delta, \\
&f(\hat{X}) - f(X^*) \geq -\|C\|_F\delta, \\
&f(\hat{X}) - f(X^*) \leq \frac{2r_0^2}{K\alpha} + 5\frac{r_0}{\alpha}\delta + 5\frac{K\delta^2}{\alpha},
\end{aligned} \tag{16}$$

where the K-step output of our Alg. 1 is $\hat{X} = \frac{1}{K}\sum_{i=0}^{K-1} X^i$ and we use the fact that $\nabla f(X^*) = C$.

If $K \geq \frac{6r_0^2}{\alpha\epsilon}$, then we have,

$$\frac{2r_0^2}{K\alpha} \leq \frac{\epsilon}{3}. \tag{17}$$

Due to Eq. (15), we have,

$$5\frac{r_0}{\alpha}\delta \leq \frac{\epsilon}{3}. \tag{18}$$

Since $N_{\text{proj}} \geq \log_{1/c}\left(\rho\sqrt{\frac{15K}{\alpha\epsilon}}\right)$, we have,

$$\begin{aligned}
5\frac{K\delta^2}{\alpha} &= 5\frac{K}{\alpha}(\rho c^{N_{\text{proj}}})^2 \\
&= 5\frac{K}{\alpha}\left(\sqrt{\frac{\alpha\epsilon}{15K}}\right)^2 \\
&\leq \frac{\epsilon}{3}.
\end{aligned} \tag{19}$$

Therefore, with Eq. (17), Eq. (18) and Eq. (19), we prove that $f(\hat{X}) - f(X^*) \leq \epsilon$.

For the lower bound, from Eq. (16) we have,

$$\begin{aligned}
f(\hat{X}) - f(X^*) &\geq -\|C\|_F\delta \\
&\geq -\|C\|_F\frac{\alpha\epsilon}{15r_0} \\
&\geq -\frac{\epsilon}{15} \\
&\geq -\epsilon,
\end{aligned} \tag{20}$$

We prove the ϵ-optimality w.r.t. the objective function.

Again, from Eq. (16), we have,

$$\begin{aligned}
\|\hat{X} - \mathcal{P}_{\mathcal{C}}(\hat{X})\|_F &\leq \delta \\
&\leq \frac{\alpha\epsilon}{15r_0} \\
&\leq \epsilon.
\end{aligned} \tag{21}$$

We prove the ϵ-optimality w.r.t. the feasibility.

□

Note that the constants (*e.g.*, 6, 15) in Theorem 1 do not matter that much as the inequality still holds by properly changing the constants in Lemma 2 as discussed in [27].

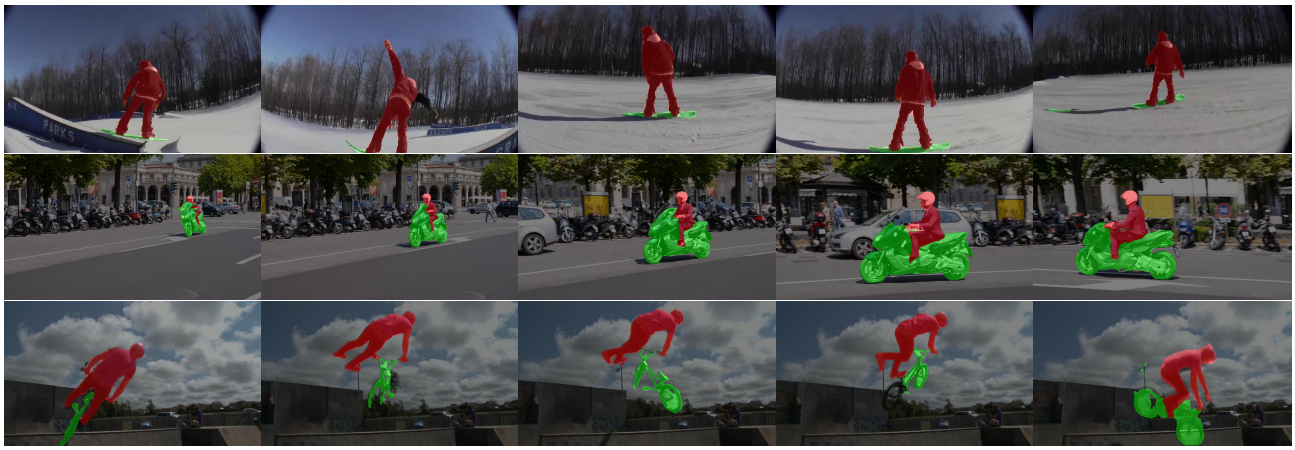
7. Additional Experiments

We show the hyperparameters of our matching algorithm on random cost matrices in Fig. 5. The random cost matrices have 5 rows and 100 columns, where the values are sampled from a uniform distribution over $[0, 1)$. Three random cost matrices are generated for the experiment. We plot the objective function value of the outer loop and the projection error of the inner loop under different sets of hypperparameters including number of steps of outer loop N_{grad}, number of steps of inner loop N_{proj} and the learning rate α. We choose the configuration which gives the best trade-off between the computational cost and the convergent rate: $N_{\text{grad}} = 400$, $N_{\text{proj}} = 50$, and $\alpha = 0.01$. From the figure, one can see that the objective value decreases as the number of outer loops increases and it reaches the minimum at about $N_{grad} = 400$ under different setting and different cost matrices. The inner projection error drops in a faster rate and in most of the case it is able to reach almost zero after about 50 rounds of projection. The figure also shows that the learning rate plays an important role for the optimization, *i.e.*, learning rate 0.01 generally leads to a faster convergence than learning rate 0.005. For the experiments on video object segmentation dataset, we start with $N_{\text{grad}} = 400$, $N_{\text{proj}} = 50$ for both training and inference. However, we reduce them to 40 and 5, respectively, for the trade off of performance and speed.

We show additional visual examples of our method on three datasets in Fig. 4.

8. Example Code

We show the example PyTorch code (less than 50 lines) of our differentiable matching algorithm as below.



(a) 0% (b) 25% (c) 50% (d) 75% (e) 100%

Figure 4. Visualization of our results on YouTube-VOS, DAVIS 2017 and SegTrack v2 at different time steps (percentage w.r.t. the whole video length). The three rows correspond to the YouTube-VOS, DAVIS 2017 and SegTrack v2 datasets respectively.

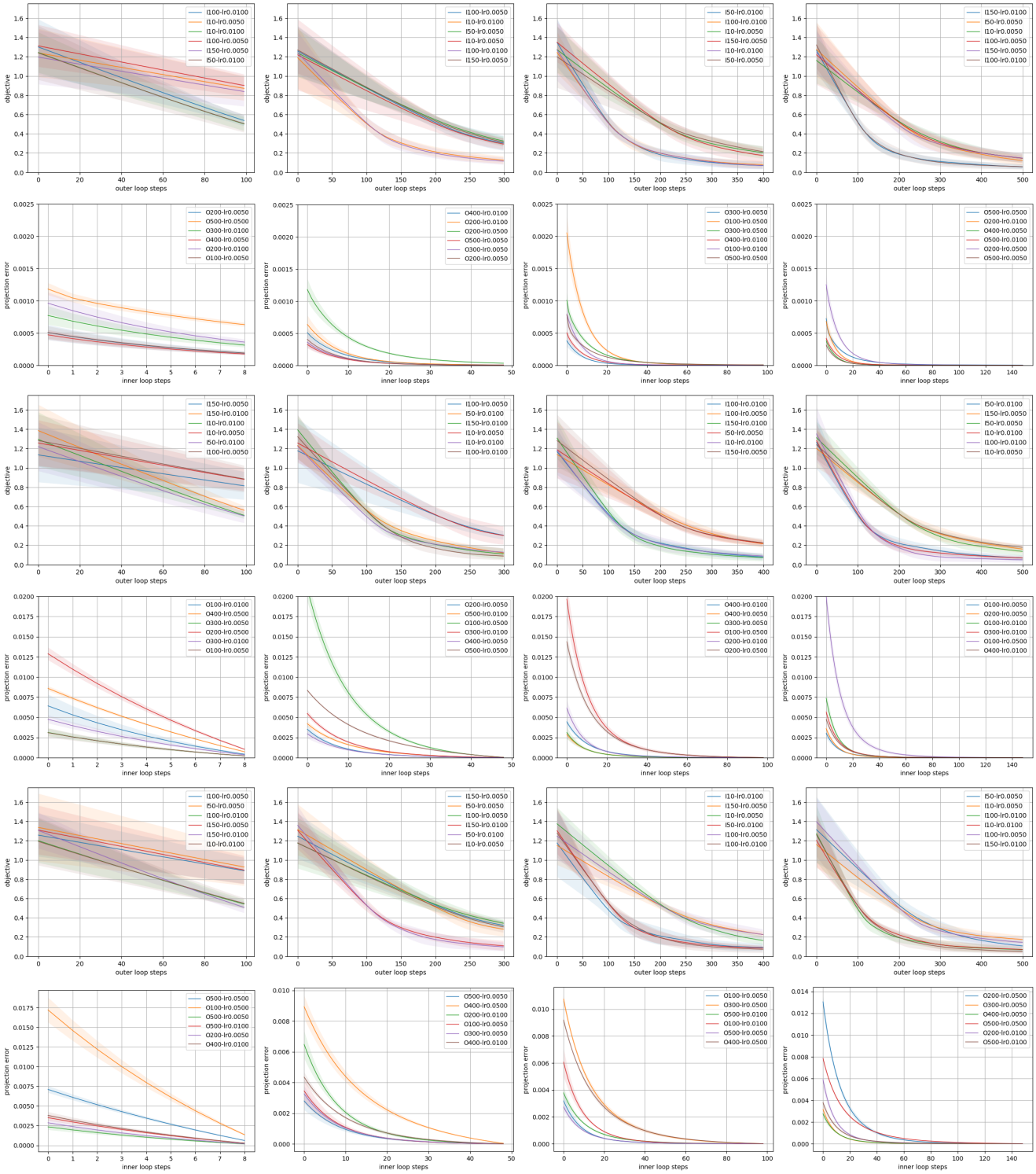


Figure 5. Hyperparameters (number of steps of outer loop N_{grad}, number of steps of inner loop N_{proj}, and learning rate α) tuning of our differentiable matching layer. We test with three random cost matrices, each of which corresponds to two consecutive rows in the figure, *e.g.*, 1-st and 2-nd rows correspond to the 1-st cost matrix. Within the two consecutive rows, the first one shows how the objective function varies with the number of steps in outer loop. The second one shows how projection error varies with the number of steps in inner loops. For each setting of hyperparameters, we perform 10 different random initialization of the assignment matrix and plot the mean and variance of the curves.

```python
def project_row(X):
    """
        p(X) = X - 1/m (X 1m - 1n) 1m^T
        X shape: n x m
    """
    X_row_sum = X.sum(dim=1, keepdim=True) # shape n x 1
    one_m = torch.ones(1, X.shape[1]).to(X.device) # shape 1 x m
    return X - (X_row_sum - 1).mm(one_m) / X.shape[1]

def project_col(X):
    """
        p(X) = X  if X^T 1n <= 1m else X - 1/n 1n (1n^T X - 1m^T)
        X shape: n x m
    """
    X_col_sum = X.sum(dim=0, keepdim=True)      # shape 1 x m
    one_n = torch.ones(X.shape[0], 1).to(X.device) # shape n x 1
    mask = (X_col_sum <= 1).float()
    P = X - (one_n).mm(X_col_sum - 1) / X.shape[0]
    return X * mask + (1 - mask) * P

def relax_matching(C, X_init, max_iter, proj_iter, lr):
    X = X_init
    P = [torch.zeros_like(C) for _ in range(3)]
    X_list = [X_init]

    for i in range(max_iter):
        X = X - lr * C # gradient step

        # project C onto the constrain set
        for j in range(proj_iter):
            X = X + P[0]
            Y = project_row(X)
            P[0] = X - Y

            X = Y + P[1]
            Y = project_col(X)
            P[1] = X - Y

            X = Y + P[2]
            Y = F.relu(X)
            P[2] = X - Y

            X = Y

        X_list += [X]
    return torch.stack(X_list, dim=0).mean(dim=0)
```

Listing 1. PyTorch example code