Hierarchical Memory Matching Network for Video Object Segmentation

Hongje Seong
Seoung Wug Oh
Joon-Young Lee
Seongwon Lee
Suhyeon Lee
Euntai Kim

1Yonsei University 2Adobe Research

Abstract
We present Hierarchical Memory Matching Network (HMMN) for semi-supervised video object segmentation. Based on a recent memory-based method [33], we propose two advanced memory read modules that enable us to perform memory reading in multiple scales while exploiting temporal smoothness. We first propose a kernel guided memory matching module that replaces the non-local dense memory read, commonly adopted in previous memory-based methods. The module imposes the temporal smoothness constraint in the memory read, leading to accurate memory retrieval. More importantly, we introduce a hierarchical memory matching scheme and propose a top-k guided memory matching module in which memory read on a fine-scale is guided by that on a coarse-scale. With the module, we perform memory read in multiple scales efficiently and leverage both high-level semantic and low-level fine-grained memory features to predict detailed object masks. Our network achieves state-of-the-art performance on the validation sets of DAVIS 2016/2017 (90.8% and 84.7%) and YouTube-VOS 2018/2019 (82.6% and 82.5%), and test-dev set of DAVIS 2017 (78.6%). The source code and model are available online: https://github.com/Hongje/HMMN.

1. Introduction
Semi-supervised video object segmentation (VOS) aims to predict the foreground object mask in every frame of a video given an object mask at the first frame. Recently, memory-based VOS methods [33, 39, 27, 21, 22, 23] have achieved great success. A key idea of the memory-based methods is matching densely between query (i.e., current frame) and memory (i.e., past frames with given or predicted masks) to retrieve the memory at a pixel-level. Since the camera’s field of view or objects in a video may move, spatio-temporal non-local and dense matching was performed to compute similarity for all matching possibilities.

There are two limitations of the existing memory-based methods: temporal smoothness and fine-grained memory information. Temporal smoothness is one of the strong constraints that we can assume for the VOS task. Previous VOS methods without memory often applied a local matching [43, 50] or local refinement [34, 16, 32, 49, 13, 53] between two adjacent frames for temporal smoothness. However, in the memory-based method [33], the non-local matching completely ignores the constraint and it raises the risk of false matches (e.g., when multiple similar instances exist, see Fig. 3). Another weakness is the lack of fine-grained memory information. In the memory-based methods, a query encoder only takes the current frame without any target information. Thus the memory matching is the only source to get information of the target object mask. The previous memory-based methods conduct the memory matching only at the coarsest resolution, (e.g., 1/16 of the input resolution [33]), as shown in Fig. 1 (a). At the low resolution, while accurate matching is possible with high-level
semantic features, we cannot expect fine-grained information that is also important to predict fine-detailed masks.

In this paper, we propose Hierarchical Memory Matching Network (HMMN) with two novel memory matching modules. To exploit the temporal smoothness, we propose kernel guided memory matching module. We restrict possible correspondences between two adjacent frames to a local window and apply kernel guidance to the non-local memory matching that imposes the temporal smoothness constraint. For long-range matching between distant frames, we track the most probable correspondence for each memory pixel to a query pixel and apply relaxed kernel guidance according to the temporal distance, resulting in a smooth transition from local to global memory matching. This module replaces the non-local memory reading in the previous memory-based networks.

To retrieve fine-grained memory information, we propose top-\(k\) guided memory matching module. The computational cost for the dense memory matching grows quadratically with increasing search space. Naively performing the memory reading at fine-scales [51] (Fig. 1 (b)) requires prohibitively heavy computation. Also, memory matching with the low-level features at a fine-scale is susceptible to noisy matches. Our top-\(k\) guided memory matching solves both the computational cost and the matching robustness issues. We first sample the top-\(k\) candidate memory locations for each query pixel using the matching similarity score at the coarse-scale. Then, we conduct fine-scale memory matching between each query pixel and the corresponding candidate memory locations, as shown in Fig. 1 (c). The top-\(k\) guided memory matching reduces the matching complexity at high-resolution significantly from \(O(T^2H^2W^2)\) to \(O(kHW)\), where \(T\), \(H\), and \(W\) are the time, height, and width of the feature map, and \(k\) is a constant. The coarse-to-fine hierarchical matching scheme makes our fine-scale memory matching robust even with low-level features. We note that some previous works [19, 55] also reduce memory matching complexity by extracting \(k\) matching candidates but they select candidates using features at the same scale. In contrast, we selected \(k\) matching candidates from high-level (i.e., coarse-scale) semantic features, thus semantically more accurate matching candidates would be selected.

Our contributions are summarized as follows:

- We propose kernel guided memory matching module, imposing the temporal smoothness constraint to the non-local matching with all memory frames.
- We propose top-\(k\) guided memory matching module, resulting in efficient and robust fine-scale memory matching.
- With the two novel memory matching modules, we present Hierarchical Memory Matching Network (HMMN) that performs coarse-to-fine hierarchical memory matching effectively.

- Our network achieves state-of-the-art performance on both DAVIS and YouTube-VOS benchmarks.

2. Related Work

Semi-supervised Video Object Segmentation: Semi-supervised VOS [35, 36, 48] has been tackled in two ways: online-learning method and offline-learning method. The online-learning methods [5, 3, 44, 1, 28, 29, 47, 7, 30] fine-tune networks at test time using the given ground-truth mask at the first frame. The objective of fine-tuning is to let networks detect target objects for each video. Therefore, the online-learning method can expect accurate results by training a target-specific network, but they are subject to severe disadvantages at run-time because the network needs to be trained multiple times on the first frame during testing.

Offline-learning methods aim to train a network that works well for any input videos without test-time training. It has usually been solved by mask propagation or pixel-wise matching. The propagation-based methods [34, 16, 12, 20, 32, 15, 4, 54, 11] train a network to propagate the given mask sequentially from the first frame. Since the propagation is conducted in a short-time interval, the methods often exploit the temporal smoothness constraint but are not robust to occlusion. The matching-based methods [41, 13, 52, 43, 14, 50] predict a foreground mask in the current frame based on matching with previously predicted or given mask. Recently, STM [33] introduced a memory-based method for offline-learning VOS and demonstrated a significantly improved performance while achieving a fast run-time. Our approach follows the memory-based method, and we address the main limitations of existing methods.

Memory-based Video Object Segmentation: Memory networks [42, 31, 18] memorize external information as key and value, then the value is retrieved by query via non-local matching with the key. It was first proposed for natural language processing, and STM [33] repurposed the memory networks to memory-based VOS. STM retrieves memory using non-local and dense memory matching and finds the target object in the query using the retrieved memory. Extended from STM, EGMN [27] proposed the graph memory networks to update memory using query. GC [21] introduced a new global matching method for fast memory matching. Liang et al. [23] proposed an adaptive memory update scheme to reduce redundant computation at memory copying. Li et al. [22] explored a cyclic mechanism for both training and inference to boost performance. KMN [39] additionally conducted memory-to-query matching then applied 2D Gaussian kernels on the query for robust matching. The previous memory-based methods overlook the temporal smoothness, one of the most important cues for VOS, as they performed memory matching in a non-local manner. In addition, the previous works conduct memory matching only at the coarsest resolution, which
hard to expect to take fine mask information. We address the problems by introducing two matching modules, kernel guided memory matching and top-k guided memory matching. Note that our kernel guided memory matching is completely different from kernelization used in KMN [39], which generates kernel based on non-local matching thus does not exploit temporal smoothness.

3. Method

Our method, Hierarchical Memory Matching Network (HMMN), is based on STM [33]. Given a ground-truth object mask at the first frame, we sequentially predict the target object mask from the second frame to the last frame. The past frames concatenated with predicted or given masks are set to memory, and the current frame is used as a query.

The main distinction comes from the construction and the use of hierarchical memory. The objective of the hierarchical memory is to leverage memories in multiple scales, from low-resolution semantic features to high-resolution detailed features, on the memory-based VOS architectures. To efficiently read the information from the hierarchical memory, we design two types of memory matching modules based on the feature map’s scale: kernel guided dense memory matching at the coarsest scale, and top-k guided sparse memory matching at fine scales. At the coarsest scale, we perform dense and non-local query-memory matching similar to STM [33] and other variants. But, we improve the robustness of the global matching through the kernel guidance that exploits temporal smoothness as an additional cue. At the finer scales followed by the coarsest level, we perform a sparse query-memory matching making use of the matching results from the coarsest level as guidance. Specifically, we take the top-k memory matching for each query point at the coarsest scale and use them to guide the sparse matching at the finer scales. In this way, we can retrieve fine-detailed memory information while taking a fractional computational cost compared to dense memory matching.

The overview of our network is shown in Fig. 2. In our network, memory and query frames are first fed into two independent ResNet50 [10]-based encoders. Both encoders extract multi-scale features – \( Q_S \) for the query frame and \( M_S \) for the memory frames – from ResNet50’s \( S \)-th \( \text{res} \) block. We use three scales where \( S \in \{2, 3, 4\} \) with the output scale of \( \{1/4, 1/8, 1/16\} \) with respect to the input image. At each scale, in the order of coarse-to-fine scales, we perform a memory read by matching the query and memory features, and then the outputs further go through the decoder to predict an object mask.

For the memory matching in the coarsest scale, the embedded query and memory \( \{Q_4, M_4\} \) are fed into kernel guided memory matching module, and it outputs the updated feature \( Z_4 \) and a guidance \( g_4 \) which is the similarity matrix used for memory retrieval. For the finer scales (\( S = 2 \) or 3), top-k guided memory matching module is used instead. It takes a pair of embedded query and memory \( \{Q_S, M_S\} \) along with the guidance \( g_4 \), and outputs the updated feature \( Z_S \). Finally, the decoder takes all the output features \( Z_S \) (either as the input or through a skip-connection), and makes a mask prediction. Note that, except for the new
3.1. Kernel Guided Memory Matching

With the embedded memory and query \((M_4, Q_4)\), extracted from each encoder at \(\text{res4}\) stage, we first encode keys \((k_{M4}, k_{Q4})\) and values \((v_{M4}, v_{Q4})\) via four independent \(3 \times 3\) convolutional layers. Then, a non-local matching between memory and query is performed using keys as follows:

\[
M_4 = k_{M4}^\top k_{Q4},
\]

where \(^\top\) indicates a matrix transpose. Based on the non-local matching \((M_4)\), we compute the attention map \((g_4)\) by

\[
g_4 = L_1 (\mathcal{K}(M_4) \odot \text{softmax}(M_4)),
\]

where \(\odot\) indicates an element-wise multiplication, \(L_1(.)\) is L1 normalization which normalizes along the memory dimension, and \(\mathcal{K}(.)\) is 2D Gaussian kernel. Then, the memory value is retrieved using the attention map \((g_4)\) as follows:

\[
v'_{M4} = v_{M4}^\top g_4.
\]

Finally, the query value \((v_{Q4})\) is concatenated with the retrieved value \((v'_{M4})\) along the feature dimension to be the output.

Here, we impose the temporal smoothness, that is the common and strong constraint for videos, on the memory matching through the kernel prior \((\mathcal{K})\). If \(\mathcal{K}(.) = 1\), the output \((Z_4)\) will be the same as the output from vanilla memory read block used in STM [33]. In other words, STM [33] retrieves memory solely based on non-local query-to-memory matching \((i.e., \text{softmax}(M_4))\), as illustrated in Fig. 3 (a). Thus, the fact that objects are likely to appear in similar local positions between adjacent frames \((i.e., \text{temporal smoothness})\) is completely ignored. To harness this behavior, we additionally generate a kernel guidance \((\mathcal{K}(\cdot))\) based on spatio-temporal local matching. As illustrated in Fig. 3 (c), we conduct memory-to-query matching between two adjacent frames for every memory pixel. Here, we constrained the matching to perform only within a local region with a window size of \(s\). Between every two adjacent frames, we track every pixel by selecting a single pixel within a local window that has the highest similarity score. This way, every memory pixel can reach to the best-matching query pixel by connecting local pixel-level tracking frame-by-frame. Based on the resulting memory-to-query matching, we generate 2D Gaussian kernels for every memory pixel with the standard deviation of \(\sigma^t\). As the temporal distance of memory-to-query increases, the tracking error can be accumulated and the temporal smoothness constraint weakens. Thus, we relaxed the kernel guidance...
by controlling the standard deviation according to the temporal distance by $\sigma^t = \sigma_{\text{init}} + (T - t)\sigma_{\text{factor}}$. This results in a smooth transition from local to global memory matching according to the temporal distance between query and memory features. A detailed implementation of kernel guided memory matching module is shown in Fig. 4.

Note that our kernel guidance is inspired by KMN [39], but the objective is completely different. KMN [39] used kernel only for robust matching from bi-directional attention, thus the kernels were generated based on non-local matching, as illustrated in Fig. 3 (b). Our kernel guidance, however, is based on fully local matching, and it effectively exploits the temporal smoothness as shown in Fig. 3 (c).

### 3.2. Top-$k$ Guided Memory Matching

The main objective of computing a dense spatio-temporal attention map in memory matching module is to find when-and-where each query pixel attends to memory pixel. However, computing the dense attention map in high resolution requires prohibitively large computing resources as its computational complexity grows quadratically with regard to the feature map size. Thus, computing dense attention maps for finer levels of the feature hierarchy (res3 and res2) is computationally too expensive. We address this issue by reducing the number of matching candidates in memory using top-$k$ guidance.

Here, we assume that the matching result at high-resolution should be similar to that at low-resolution. By this assumption, we reuse the dense matching result at low-resolution as guidance for matching in higher resolution. An illustration of selecting $k$ pixels and guiding to high-resolution for each query pixel is depicted in Fig. 5. Based on the low-resolution attention map ($g_4$), which comes from res4 stage, we select $k$ best matching memory pixels for each query pixel via top-$k$ operation. Then, only a sparse matching to selected pixels from the memory is performed.

Note that the selected $k$ pixels in res4 correspond to $4k$ and $16k$ pixels at res3 and res2 stages, respectively, thus we take $k$ and $k/4$ for guiding each at res3 and res2 stage in order to have a similar computational overhead. This memory read module based on sparse matching can be efficiently implemented with a combination of common tensor operations. A detailed implementation of the top-$k$ guided memory matching module is shown in Fig. 6. The outputs of top-$k$ guided memory matching modules ($Z_4$, $Z_3$, $Z_2$) are fed into the decoder through shortcut connections at the corresponding scale.

Note that, in the module, rather than directly using the retrieved values as the output, we place one convolutional layer followed by Dropout layer before added to the query value as residual. This design choice is due to the following observation. Without dropout, the model tends to converge to a sub-optimal state that does not make use of the matching results at the coarsest scale (i.e., memory at res4). This sub-optimal model appears to take a shortcut for easier solutions, simply relying on low-level mask information (i.e., memory at res2 and res3) ignoring the high-level semantic matching. We were able to prevent this behavior by delivering the information in a restrictive way through a residual connection after a dropout layer that randomly drops the whole input feature during training. In this way, the network has to consider the output of top-$k$ guided memory matching module as supplementary information to refine the memory matching at the coarsest resolution.

### 4. Experiments

#### 4.1. Implementation Details

**Training.** For a fair comparison with STM [33], we follow the same training strategies. We initialize the encoders with ImageNet [38] pre-trained weights and randomly initialize the other layers. Then, we take the images with object masks in [8, 25, 9, 40, 6, 45] and pretrain HMMN on the...
and the learning rate is set to 1e-5. We use an input size of 384 × 384 and a mini-batch size of 4. According to [33], we set the standard deviation of \( \sigma_{\text{init}} \) and \( \sigma_{\text{factor}} \) into 3 and 0.5, respectively, and we used window size \( s \) of 7. We measure our run-time using a single NVIDIA GeForce 1080 Ti GPU.

### 4.2. Comparisons

We compare our HMMN against state-of-the-art methods on DAVIS [35, 36] and YouTube-VOS [48] benchmarks. For DAVIS benchmarks, 60 videos from DAVIS 2017 training set are used during main training following the common evaluation protocol [32, 49, 33, 39]. In addition, we report our results on DAVIS benchmarks using additional training videos from YouTube-VOS for a fair comparison with some recent methods [33, 39, 2, 50, 27, 37]. For YouTube-VOS benchmarks, the training set of 3471 videos are used. For all experiments, we either use the official evaluation code or upload our results to the evaluation server.

**DAVIS** [35, 36] is a densely annotated VOS dataset and mostly adopted benchmark to evaluate VOS models. To evaluate HMMN on DAVIS benchmarks, we use an input size of 480p resolution for all experiments. DAVIS dataset is divided into two sets: (1) DAVIS 2016, which is an object-level annotated dataset (single object); and (2) DAVIS 2017, which is an instance-level annotated dataset (multiple objects). The official metrics, region similarity \( J \) and contour accuracy \( F \), are measured for comparison. As shown in Table 1, our HMMN achieves state-of-the-art performance while taking a fast run-time on DAVIS 2016 validation set. Further, even without an additional YouTube-VOS dataset to train HMMN, we surpass most state-of-the-art methods.

We also conduct comparisons on DAVIS 2017 validation and test-dev sets, and the results are given in Table 2 and Table 3. As shown in the Tables, our HMMN significantly outperforms the current best results by 1.9% and 1.4% of \( J & F \) scores on DAVIS 2017 validation and test-dev sets, respectively. We omitted some comparable works in the tables. The full comparison tables are available in the supplementary material.

**YouTube-VOS** [48] is a large-scale benchmark for VOS. To evaluate our HMMN on YouTube-VOS benchmarks, we

<table>
<thead>
<tr>
<th>Method</th>
<th>OL</th>
<th>( J &amp; F )</th>
<th>( J )</th>
<th>( F )</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-OSVOS [30]</td>
<td>✓</td>
<td>86.8</td>
<td>86.6</td>
<td>87.0</td>
<td>3.4s</td>
</tr>
<tr>
<td>DyeNet [20]</td>
<td>✓</td>
<td>-</td>
<td>86.2</td>
<td>-</td>
<td>2.32s</td>
</tr>
<tr>
<td>RaNet [46]</td>
<td>✓</td>
<td>87.1</td>
<td>86.6</td>
<td>87.6</td>
<td>4s</td>
</tr>
<tr>
<td>STM (+YV) [33]</td>
<td></td>
<td>89.3</td>
<td>88.7</td>
<td>89.9</td>
<td>0.16s</td>
</tr>
<tr>
<td>CFBI (+YV) [50]</td>
<td></td>
<td>89.4</td>
<td>88.3</td>
<td>90.5</td>
<td>0.18s</td>
</tr>
<tr>
<td>KMN (+YV) [39]</td>
<td></td>
<td>90.5</td>
<td>89.5</td>
<td>91.5</td>
<td>0.12s</td>
</tr>
<tr>
<td>HMMN (+YV)</td>
<td></td>
<td>89.4</td>
<td>88.2</td>
<td>90.6</td>
<td>0.10a</td>
</tr>
<tr>
<td>HMMN (+YV)</td>
<td></td>
<td>90.8</td>
<td>89.6</td>
<td>92.0</td>
<td>0.10a</td>
</tr>
</tbody>
</table>

Table 1. Comparison on DAVIS 2016 validation set. (+YV) indicates YouTube-VOS is additionally used for training, and OL denotes the use of online-learning strategies during test-time. Time measurements reported in this table are directly from the corresponding papers.

image datasets. Specifically, we generate three frames by augmenting each image via random affine transforms. The random affine transforms include rotation, shearing, zooming, translation, and cropping. During the pre-training, the dropout rate in top-\( k \) guided memory matching module (§3.2) is gradually decreased from 1 to 0.5.

After the pre-training on image datasets, the main training is done using either DAVIS 2017 [36] or YouTube-VOS 2019 [48] training set depending on the target benchmark. During main training, three frames are randomly sampled from a video with the gradually increasing maximum interval (from 0 to 25). The dropout rate in top-\( k \) guided memory matching module is gradually decreased from 0.5 to 0.

During both pre-training and main training, we minimize pixel-wise cross-entropy loss with Adam optimizer [17], and the learning rate is set to 1e-5. We use an input size of 384 × 384 and a mini-batch size of 4. According to [33], we employ the soft aggregation operation when multiple target objects exist in a video.

**Inference.** As in [33, 39], we take the first frame, the previous frame, and the intermediate frames sampled at every 5 frames for the memory in the coarsest scale (\( M_2 \)). For the fine-scale memories (\( M_3, M_4 \)), we do not use the intermediate frames to avoid GPU memory overflow unless mentioned otherwise. We use the same number of \( k \) for top-\( k \) guided memory matching during training and inference, which is set to 32. The kernel guidance in §3.1 is used only during inference, as in KMN [39]. We have tried to use the kernel guidance during training, but there was no noticeable improvement. We set the standard deviation of \( \sigma_{\text{init}} \) and \( \sigma_{\text{factor}} \) into 3 and 0.5, respectively, and we used window size \( s \) of 7. We measure our run-time using a single NVIDIA GeForce 1080 Ti GPU.

### 4.2. Comparisons

We compare our HMMN against state-of-the-art methods on DAVIS [35, 36] and YouTube-VOS [48] benchmarks. For DAVIS benchmarks, 60 videos from DAVIS 2017 training set are used during main training following the common evaluation protocol [32, 49, 33, 39]. In addition, we report our results on DAVIS benchmarks using additional training videos from YouTube-VOS for a fair comparison with some recent methods [33, 39, 2, 50, 27, 37]. For YouTube-VOS benchmarks, the training set of 3471 videos are used. For all experiments, we either use the official evaluation code or upload our results to the evaluation server.

**DAVIS** [35, 36] is a densely annotated VOS dataset and mostly adopted benchmark to evaluate VOS models. To evaluate HMMN on DAVIS benchmarks, we use an input size of 480p resolution for all experiments. DAVIS dataset is divided into two sets: (1) DAVIS 2016, which is an object-level annotated dataset (single object); and (2) DAVIS 2017, which is an instance-level annotated dataset (multiple objects). The official metrics, region similarity \( J \) and contour accuracy \( F \), are measured for comparison. As shown in Table 1, our HMMN achieves state-of-the-art performance while taking a fast run-time on DAVIS 2016 validation set. Further, even without an additional YouTube-VOS dataset to train HMMN, we surpass most state-of-the-art methods.

We also conduct comparisons on DAVIS 2017 validation and test-dev sets, and the results are given in Table 2 and Table 3. As shown in the Tables, our HMMN significantly outperforms the current best results by 1.9% and 1.4% of \( J \& F \) scores on DAVIS 2017 validation and test-dev sets, respectively. We omitted some comparable works in the tables. The full comparison tables are available in the supplementary material.

**YouTube-VOS** [48] is a large-scale benchmark for VOS. To evaluate our HMMN on YouTube-VOS benchmarks, we

<table>
<thead>
<tr>
<th>Method</th>
<th>OL</th>
<th>( J &amp; F )</th>
<th>( J )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CINN [1]</td>
<td>✓</td>
<td>67.5</td>
<td>64.5</td>
<td>70.5</td>
</tr>
<tr>
<td>DyeNet [20]</td>
<td>✓</td>
<td>68.2</td>
<td>65.8</td>
<td>70.5</td>
</tr>
<tr>
<td>PRenMVOS [28]</td>
<td>✓</td>
<td>71.6</td>
<td>67.5</td>
<td>75.7</td>
</tr>
<tr>
<td>STM (+YV) [33]</td>
<td>✓</td>
<td>72.2</td>
<td>69.3</td>
<td>75.2</td>
</tr>
<tr>
<td>CFBI (+YV) [50]</td>
<td>✓</td>
<td>74.8</td>
<td>71.1</td>
<td>78.5</td>
</tr>
<tr>
<td>KMN (+YV) [39]</td>
<td>✓</td>
<td>77.2</td>
<td>74.1</td>
<td>80.3</td>
</tr>
<tr>
<td>HMMN (+YV)</td>
<td>✓</td>
<td>78.6</td>
<td>74.7</td>
<td>82.5</td>
</tr>
</tbody>
</table>

Table 2. Comparison on DAVIS 2017 validation set.

<table>
<thead>
<tr>
<th>Method</th>
<th>OL</th>
<th>( J &amp; F )</th>
<th>( J )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRTM (+YV) [37]</td>
<td>✓</td>
<td>76.7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>e-OSVOS [30]</td>
<td>✓</td>
<td>77.2</td>
<td>74.4</td>
<td>80.0</td>
</tr>
<tr>
<td>PRenMVOS [28]</td>
<td>✓</td>
<td>77.8</td>
<td>73.9</td>
<td>81.7</td>
</tr>
<tr>
<td>LWL (+YV) [2]</td>
<td>✓</td>
<td>81.6</td>
<td>79.1</td>
<td>84.1</td>
</tr>
<tr>
<td>STM (+YV) [33]</td>
<td>✓</td>
<td>81.8</td>
<td>79.2</td>
<td>84.3</td>
</tr>
<tr>
<td>CFBI (+YV) [50]</td>
<td>✓</td>
<td>81.9</td>
<td>79.1</td>
<td>84.6</td>
</tr>
<tr>
<td>EGMM (+YV) [27]</td>
<td>✓</td>
<td>82.8</td>
<td>80.2</td>
<td>85.2</td>
</tr>
<tr>
<td>KMN (+YV) [39]</td>
<td>✓</td>
<td>82.8</td>
<td>80.0</td>
<td>85.6</td>
</tr>
<tr>
<td>HMMN (+YV)</td>
<td>✓</td>
<td>84.7</td>
<td>81.9</td>
<td>87.5</td>
</tr>
</tbody>
</table>

Table 3. Comparison on DAVIS 2017 test-dev set.
reduce the input image to 480p resolution. We measured region similarity ($J_S$, $J_U$) and contour accuracy ($F_S$, $F_U$) for 65 of seen and 26 of unseen object categories separately.

In Table 4, we compare HMMN with state-of-the-art methods on YouTube-VOS 2018 and 2019 validation sets. Note that only CFBI [50] officially reported for comparison on YouTube-VOS 2019 validation set, so we additionally report our reproduced results of STM [33] and KMN [39] using our training setup. As shown in Table 4, our HMMN surpasses the state-of-the-art methods in all official metrics on both YouTube-VOS 2018 and 2019.

**Qualitative Comparison.** Fig. 7 shows qualitative comparison with STM [33] and KMN [39]. In the figure, STM [33] almost failed to predict target objects when multiple similar objects have appeared or several occlusion occurred (DAVIS example). KMN [39] failed to predict a very small object (YouTube-VOS example). On the other hand, our HMMN predicted the target objects accurately in the challenging cases. More qualitative results are provided in the supplementary material.

### 4.3. Ablation Experiments

**Module ablation.** We conduct an ablation study on our two proposed memory matching modules to demonstrate the efficacy of those. We also compare our kernel guided memory matching with the kernelization method proposed in KMN [39]. As shown in Table 5, our kernel guidance is more effective than one from KMN, and the use of fine-scale memories through top-$k$ guided memory module greatly boosts the performance to the state-of-the-art.

**Temporal stability ($T$).** To validate the effectiveness of our HMMN on temporal smoothness quantitatively, we evaluate temporal stability ($T$) on DAVIS 2016 validation set. STM [33], KMN [39], and our HMMN achieved $T$ scores (lower is better) of 17.2%, 15.2%, and 13.0%, respectively. This implies that our method significantly improves temporal stability over STM and KMN.

**$k$-pixel selection strategies.** To validate the effectiveness of our top-$k$ guidance (§3.2), we study various strategies to sample $k$ memory pixels. As can be seen in Table 6 (a), fine-scale memory with simple sampling methods (random, stride) do not provide consistent improvement over the baseline ($k=0$). However, fine-scale memory with our top-$k$ guidance yields significant performance improvement even with a small number of $k$.

**The effect of $k$.** We further study the effect of $k$ during both training and inference by increasing the number of $k$ from 32 to $\infty$. Here, $k=\infty$ indicates using dense memory without sampling. As shown in Table 6 (b), using a dense fine-scale memory either in training and/or inference degrades the overall performance compared to using top-$k$ sampled memory. We conjecture that, in fine-scale, the feature is not robust enough for the global and dense matching. In this case, top-$k$ guidance could be beneficial to rejecting noises by restricting the search space into few reliable options. While our default setting is to set $k=32$ for both training and inference, we observed that the performance could be further improved by tuning $k$.

**Dropout for high-resolution memory.** Table 6 (c) shows the effect of dropout in top-$k$ guided memory module. As we discussed in §3.2, our dropout strategy makes our network learn with hierarchical memories effectively.

**Fine-scale memory management.** Table 6 (d) shows that we can further boost our performance by exploiting fine-scale memories from the intermediate frames sampled from every 5 frames. However, this configuration requires too much GPU memory to store memory features, while performance improvement is marginal. We use the first and previous frames for fine-scale memory by default to run HMMN. Note that we use the intermediate frames for the coarse-scale memory.

<table>
<thead>
<tr>
<th>Method</th>
<th>OL</th>
<th>$G$</th>
<th>$J_S$</th>
<th>$J_U$</th>
<th>$F_S$</th>
<th>$F_U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>YouTube-VOS 2018 validation set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGSS-VOS [24]</td>
<td>✓</td>
<td>71.3</td>
<td>71.3</td>
<td>65.5</td>
<td>75.2</td>
<td>73.1</td>
</tr>
<tr>
<td>e-OSVOS [30]</td>
<td>✓</td>
<td>71.4</td>
<td>71.7</td>
<td>74.3</td>
<td>66.0</td>
<td>73.8</td>
</tr>
<tr>
<td>FRTM [37]</td>
<td>✓</td>
<td>72.1</td>
<td>72.3</td>
<td>65.9</td>
<td>76.2</td>
<td>74.1</td>
</tr>
<tr>
<td>STG-Net [26]</td>
<td>✓</td>
<td>73.0</td>
<td>72.7</td>
<td>69.1</td>
<td>75.2</td>
<td>74.9</td>
</tr>
<tr>
<td>STM [33]</td>
<td>✓</td>
<td>79.4</td>
<td>79.7</td>
<td>72.8</td>
<td>84.2</td>
<td>80.9</td>
</tr>
<tr>
<td>AFB+URR [23]</td>
<td>✓</td>
<td>79.6</td>
<td>78.8</td>
<td>74.1</td>
<td>83.1</td>
<td>82.6</td>
</tr>
<tr>
<td>EGNN [27]</td>
<td>✓</td>
<td>80.2</td>
<td>80.7</td>
<td>74.0</td>
<td>85.1</td>
<td>80.9</td>
</tr>
<tr>
<td>CFBI [50]</td>
<td>✓</td>
<td>81.4</td>
<td>81.1</td>
<td>75.3</td>
<td>85.8</td>
<td>83.4</td>
</tr>
<tr>
<td>KMN [39]</td>
<td>✓</td>
<td>81.4</td>
<td>81.4</td>
<td>75.3</td>
<td>85.6</td>
<td>83.3</td>
</tr>
<tr>
<td>LWL [2]</td>
<td>✓</td>
<td>81.5</td>
<td>80.4</td>
<td>76.4</td>
<td>84.9</td>
<td>84.4</td>
</tr>
<tr>
<td>HMMN</td>
<td>✓</td>
<td>82.6</td>
<td>82.1</td>
<td>76.8</td>
<td>87.0</td>
<td>84.6</td>
</tr>
</tbody>
</table>

Table 4. Comparison on YouTube-VOS validation sets. $G$ is an average of $J_S$, $J_U$, $F_S$, and $F_U$. * denotes our reproduced result using our training setup.

<table>
<thead>
<tr>
<th>Method</th>
<th>OL</th>
<th>$G$</th>
<th>$J_S$</th>
<th>$J_U$</th>
<th>$F_S$</th>
<th>$F_U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>YouTube-VOS 2019 validation set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STM* [33]</td>
<td>✓</td>
<td>79.3</td>
<td>79.8</td>
<td>73.0</td>
<td>83.8</td>
<td>80.5</td>
</tr>
<tr>
<td>KMN* [39]</td>
<td>✓</td>
<td>80.0</td>
<td>80.4</td>
<td>73.8</td>
<td>84.5</td>
<td>81.4</td>
</tr>
<tr>
<td>CFBI [50]</td>
<td>✓</td>
<td>81.0</td>
<td>80.6</td>
<td>75.2</td>
<td>85.1</td>
<td>83.0</td>
</tr>
<tr>
<td>HMMN</td>
<td>✓</td>
<td>82.5</td>
<td>81.7</td>
<td>77.3</td>
<td>86.1</td>
<td>85.0</td>
</tr>
</tbody>
</table>

Table 5. Module ablation study. We report $J$, $F$ and $G$ scores for DAVIS and YouTube-VOS, respectively. The run-time is measured on DAVIS 2016 validation set. The baseline model is STM [33]. K* [39] denotes the kernelization proposed in [39], and $K$ and $T$ indicate our kernel guided memory matching and top-$k$ guided memory matching modules, respectively.

12895
Table 6. Ablation Study. For each setting, we report results of $J$ & $F$ and $G$ scores on DAVIS 2017 and YouTube-VOS 2019 validation sets, respectively.

Fine-scale memory stages. We ablate hierarchical memory stage-by-stage, and the results are given in Table 6 (e). As shown in the table, using memory hierarchies in both stages shows the best performance. If the hierarchical memory is used only in a single stage, interestingly, taking finer-scale memory (i.e., res2 stage) achieves better performance even we reduced the number of $k$ to $k/4$ at res2 stage. It is thought that the finer-scale memory can provide more complementary information to the memory from the coarsest scale.

Window size $s$ & standard deviation $\sigma_{\text{init}}$. Tables 6 (f) and 6 (g) show the parameter search experiments for guidance kernel (§3.1). Choosing a too large and too small value for $s$ and $\sigma_{\text{init}}$ has degraded the performance. Therefore, we select proper window size $s$ and standard deviation of Gaussian kernel $\sigma_{\text{init}}$ as $7 \times 7$ and 3, respectively.

5. Conclusion

We presented two advanced memory matching modules that exploit temporal smoothness and hierarchical memory effectively. We demonstrated the efficacy of our HMMN through extensive experiments and achieved state-of-the-art performance on all evaluated benchmarks while keeping a fast run-time. We believe that our proposed two memory matching modules can be further extended to other matching-based vision applications such as video saliency detection, video instance segmentation, and semantic correspondence.

Acknowledgement. This work was supported by the Industry Core Technology Development Project, 20005062, Development of Artificial Intelligence Robot Autonomous Navigation Technology for Agile Movement in Crowded Space, funded by the Ministry of Trade, industry & Energy (MOTIE, Republic of Korea).
References


