MAMBA: Multi-level Aggregation via Memory Bank for Video Object Detection

Guanxiong Sun[1, 2], **Yang Hua**[1], **Guosheng Hu**[2, 1], **Neil Robertson**[1]
[1]EEECS/ECIT, Queen's University Belfast, UK
[2]Anyvision, Belfast, UK
{gsun02, y.hua, n.robertson}@qub.ac.uk, huguosheng100@gmail.com

Abstract

State-of-the-art video object detection methods maintain a memory structure, either a *sliding window* or a *memory queue*, to enhance the current frame using attention mechanisms. However, we argue that these memory structures are not efficient or sufficient because of two implied operations: (1) concatenating *all* features in memory for enhancement, leading to a heavy computational cost; (2) *frame-wise* memory updating, preventing the memory from capturing more temporal information. In this paper, we propose a multi-level aggregation architecture via memory bank called MAMBA. Specifically, our memory bank employs two novel operations to eliminate disadvantages of existing methods: (1) *light-weight* key-set construction which can significantly reduce the computational cost; (2) fine-grained *feature-wise* updating strategy which enables our method to utilize knowledge from the whole video. To better enhance features from complementary levels, i.e., feature maps and proposals, we further propose a generalized enhancement operation (GEO) to aggregate multi-level features in a unified manner. We conduct extensive evaluations on the challenging ImageNetVID dataset. Compared with existing state-of-the-art methods, our method achieves superior performance in terms of both speed and accuracy. More remarkably, MAMBA achieves mAP of 83.7%/84.6% at 12.6/9.1 FPS with ResNet-101.

Introduction

Object detection is a fundamental task in computer vision and plays a critical role in many real-world applications. Recently, deep convolutional neural networks (CNNs) based object detectors (Girshick et al. 2015; Girshick 2015; Ren et al. 2015; Dai et al. 2016; Redmon et al. 2016; Tian et al. 2019) have achieved excellent performance on still images. However, the success of still-image detectors is hard to transfer to video data directly, because of the quality deterioration of video frames, caused by severe motion blur, rare poses, defocus, occlusions, etc.. To solve these issues, recent methods (Zhu et al. 2017b, 2018, 2017a; Deng et al. 2019a; Wu et al. 2019; Deng et al. 2019b; Chen et al. 2020; Shvets, Liu, and Berg 2019) utilize temporal information to enhance video frames, a.k.a., feature-level enhancement methods. Specifically, feature-level enhancement methods construct a memory structure that contains the features of other frames. Then, either alignment modules, e.g., FlowNet (Dosovitskiy et al. 2015; Ilg et al. 2017) or relation modules, e.g., attention mechanisms (Hu et al. 2018; Vaswani et al. 2017), are employed to enhance the current frame using features stored in the memory structure. Depending on how the memory structure is constructed and what features are stored in the memory, existing feature-level enhancement methods can be categorized into two groups: *sliding window* methods

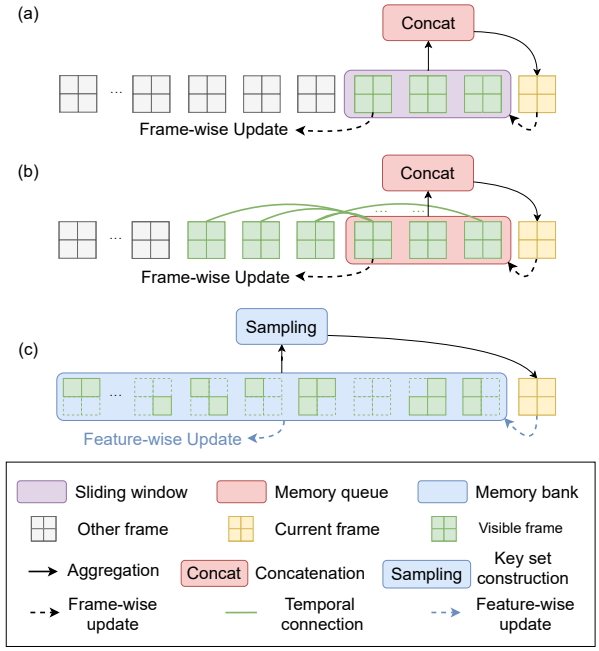

Figure 1: Comparisons of the memory construction process in three memory structures. (a) Sliding window stores raw features of neighbour frames. (b) Memory queue stores features of the enhanced frames. One enhanced frame contains the temporal information of its previous frames. As a result, the number of visible frames is enlarged by temporal connections. (c) The proposed memory bank contains two novel operations: light-weight key-set construction and fine-grained feature-wise updating, which help enlarge the number of visible frames to the length of the whole video. *Best viewed in color.*

and *memory queue* methods.

A *sliding window* (Zhu et al. 2017a; Wang et al. 2018; Bertasius, Torresani, and Shi 2018; Wu et al. 2019; Deng et al. 2019b; Shvets, Liu, and Berg 2019) stores raw features of several neighbour frames of the current frame. Note that the sliding window may contain future frames (offline methods). For demonstration, we show an online version of sliding window in Figure 1 (a). The number of visible frames, which denotes the amount of information the current frame can gather from, is equal to the length of the sliding window. To enlarge the number of visible frames, *memory queue* methods (Chen et al. 2020) utilize recurrent temporal connections to aggregate more temporal information from additional frames, shown in Figure 1 (b). Instead of storing raw features, a memory queue stores the intermediate enhanced features. Thanks to the stacked enhancement stages, a memory queue enlarges the number of visible frames several times, e.g., double in (Chen et al. 2020). The increased number of visible frames contributes to better performance.

However, these memory structures are not efficient or sufficient enough because of two implied operations: (1) Concatenating *all* features in memory for enhancement, leading to heavy computational cost. (2) Updating the memory in a coarse-grained manner, i.e., *frame-wise updating*. Deleting features of the oldest frame at every time step. This limits the number of visible frames to a fixed number, e.g., 20-30, and prevents the model from capturing temporal information from the whole video.

To address these issues, we propose a *memory bank* for video object detection. As shown in Figure 1 (c), our memory bank contains two novel operations. Firstly, unlike the existing methods (Chen et al. 2020; Deng et al. 2019b; Wu et al. 2019; Deng et al. 2019a) that use *all* the features in the memory, we introduce a light-weight key-set construction strategy to select a subset of features in the memory bank for enhancement, significantly reducing the computational cost and leading to a higher speed. Secondly, instead of the widely used holistic frame-wise memory updating strategy, we propose a fine-grained feature-wise updating strategy, which can partially delete features from multiple frames. As a result, our method is able to capture and store information from more frames under the same memory size.

In addition, several RFCN-based (Dai et al. 2016) methods, e.g., MANet (Wang et al. 2018) and OGEM (Deng et al. 2019a), demonstrate that the enhancement in different levels, i.e., pixel-level (deep feature maps) and instance-level (position-sensitive score maps), are complementary. More recent FasterRCNN-based (Ren et al. 2015) methods (Chen et al. 2020; Deng et al. 2019b; Wu et al. 2019; Shvets, Liu, and Berg 2019) leverage relation networks (Hu et al. 2018) to perform better instance-level enhancements and improve the performance significantly. However, relation networks cannot receive pixel-level feature as input. To solve this, we introduce a generalized enhancement operation (GEO), which can enhance features in both pixel-level and instance-level in a unified way. By introducing multi-level aggregation via proposed memory bank (MAMBA), our method achieves superior performance in terms of both speed and accuracy. To sum up, our contribution is threefold:

- We propose a memory bank for video object detection. Specifically, we introduce a light-weight key-set construction strategy and a more fine-grained feature-wise updating mechanism, greatly reducing the computational costs and achieving a flexible framework for different accuracy-speed trade-offs.
- We present a generalized enhancement operation (GEO), which can enhance complementary multi-level (pixel-level and instance-level) features in a unified way.
- We conduct extensive experiments on ImageNet VID dataset (Russakovsky et al. 2015). Compared with state-of-the-art methods, our method achieves better performance and faster speed at the same time.

Related Work

Object Detection. Still image object detectors (Girshick 2015; Ren et al. 2015; Redmon et al. 2016; Redmon and Farhadi 2017; Dai et al. 2016) have been remarkably improved due to the development of deep convolutional neural networks (CNNs) (He et al. 2016; Xie et al. 2017). In two-stage detectors (Ren et al. 2015; Dai et al. 2016), firstly, a backbone network (He et al. 2016; Simonyan and Zisserman 2014; Szegedy et al. 2015) is used to extract deep feature maps of an image and then the deep feature maps are passed into the Region Proposal Networks (RPN) (Ren et al. 2015) to generate object proposals. Secondly, the sub-networks further classify the proposals and regress the bounding boxes. Our proposed memory bank is a general module and can be easily applied to different detectors.

Video Object Detection. Existing video object detection methods can be divided into two categories: box-level methods and feature-level methods. Box-level methods leverage LSTM (Kang et al. 2017a) or tracking (Kang et al. 2017b) to model the temporal associations between detected bounding boxes. These methods either introduce heavy computational cost or serve as a post-processing manner (Han et al. 2016; Feichtenhofer, Pinz, and Zisserman 2017). In contrast, the feature-level methods enhance the current frame with other frames end-to-endly. Based on how to compute the correlation features between frames, feature-level methods can be divided into two subcategories: optical flow based (Dosovitskiy et al. 2015) and attention (Vaswani et al. 2017) based methods. Specifically, FGFA (Zhu et al. 2017a) uses optical flow to align neighbor frames onto the current frame at every time step. THP (Zhu et al. 2018) does partial aggregation in a recurrent manner. MANet (Wang et al. 2018) averages the optical flow within proposals to address the poor flow estimation caused by occlusion. Recent methods (Deng et al. 2019a; Wu et al. 2019; Deng et al. 2019b; Chen et al. 2020) employ attention mechanisms to do feature enhancement. Most of them (Deng et al. 2019b; Wu et al. 2019; Chen et al. 2020) conduct instance-level enhancement by reasoning the object relations across frames. OGEM (Deng et al. 2019a) proposes an object guided strategy to partially store and sparsely update the object features.

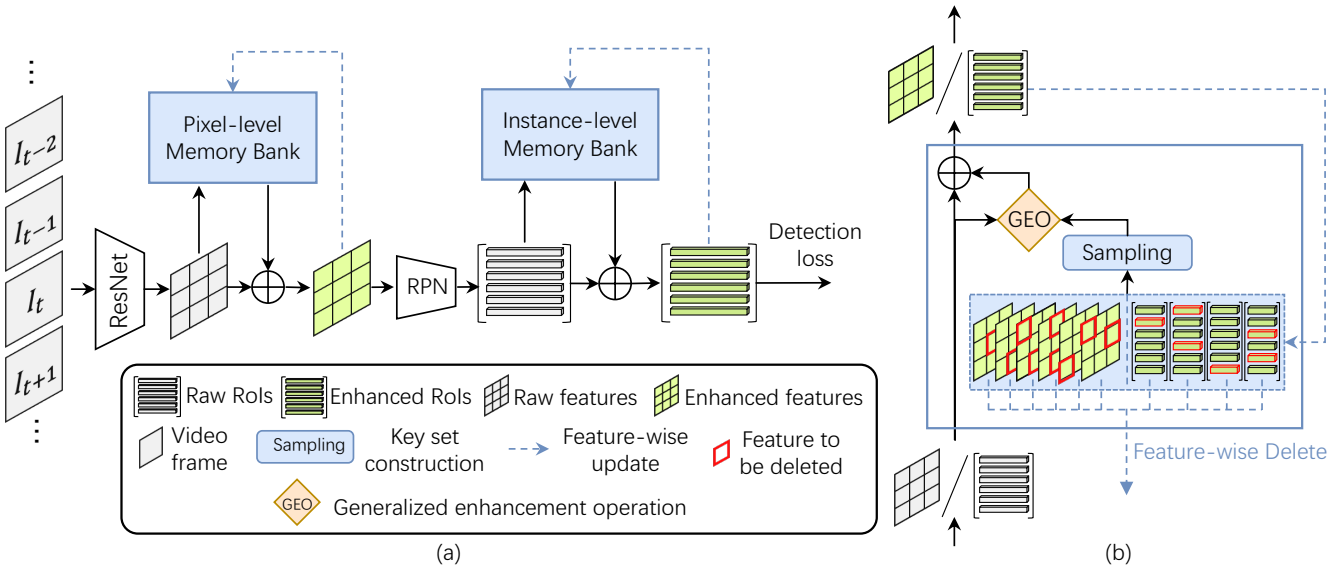

Figure 2: (a) Overview of our framework. Given an input frame I_t, firstly, I_t is passed through the backbone networks. Secondly, the extracted feature maps is enhanced by the pixel-level memory bank. Thirdly, the Region Proposal Networks (RPN) is used to extract proposals on the enhanced feature maps. Finally, the proposals are further enhanced by the instance-level memory bank and then enhanced proposals are used to compute the detection loss. (b) Illustration of the enhancement process of the memory bank. The input can either be feature maps or proposals. *Best viewed in color.*

Our Approach

In this section, we introduce a novel framework using Multi-level Aggregation with Memory Bank (MAMBA) for video object detection. In order to enhance the current frame (i.e., query) $Q = \{q_i\}_{i=1}^{N_q}$ (where N_q denotes the number of features in the current frame), we first construct a key set $K = \{k_j\}_j^{N_k}$ (where N_k is the total number of feature in the key set) by a light-weight key set construction strategy. The total number of features stored in memory bank $\mathcal{MB}$ is N_m ($N_m \gg N_k$). Then, we apply the generalized enhancement operation (GEO) to enhance Q with K. Note that the proposed GEO supports both pixel-level and instance-level features. Finally, we utilize a feature-wise updating strategy to update memory bank $\mathcal{MB}$. An overview of our framework is shown in Figure 2 and the inference procedure is described in Algorithm 1.

Light-weight Key Set Construction

Existing methods (Chen et al. 2020; Deng et al. 2019a; Wu et al. 2019; Shvets, Liu, and Berg 2019; Deng et al. 2019b) concatenate *all* the features in memory to build the key set (K), which not only increases computational cost but also restricts memory to store more diverse features to further improve the performance. In contrast, we design a light-weight key set construction strategy for memory bank to construction the key set, which can be formulated as:

$$K = Sampling(\mathcal{MB}), \tag{1}$$

where $Sampling$ denotes the sampling strategy and $\mathcal{MB}$ denotes memory bank. Specifically, we implement three sampling strategies: score ranking, frequency select, and random select. For score ranking strategy, we select the top-N_k features in the memory bank according to their confidence score, either classification score for pixel-level enhancement or objectness score for instance-level enhancement. For frequency-guided selection, we normalize confidence scores of all features in the memory bank by softmax. The normalized scores are used as the frequency, which can guide the sampling process. As a result, features with higher scores have higher possibility to be selected, but it is not hard restricted as the score ranking strategy. We also implement a random selection strategy. Using random selection, the sampled subset of features approximately follows the same distribution of all features stored in memory. In our experiments, all three sampling strategies can effectively improve the performance. Among them, random selection achieves a sightly higher accuracy. For simplicity, we use the random selected strategy by default.

It is worth noting that using sampling strategy in the memory bank, we can flexibly control N_k to achieve a controllable speed-accuracy trade-off. In our experiments, we can construct a much larger memory (e.g., N_m=96k while the largest N_m for existing methods is only $6k$) to store more diverse features in order to improve performance. Meanwhile, we can select a much smaller key set to achieve a faster speed. For example, the smallest N_k used in existing methods is 2,775 reported in RDN (Deng et al. 2019b) and RDN achieves 81.8% mAP with a speed of 128.0 ms. An extreme version of our method with $N_k = 50$ and $N_m = 20,000$ achieves a much higher accuracy of 83.1% mAP meanwhile a much faster runtime speed at 75.8 ms. Detailed speed-accuracy trade-offs our method are shown in Table 6.

Algorithm 1: Inference Algorithm with Memory Bank in a PyTorch-like style.

```
# offline_test: a bool value denotes whether enable
  offline testing
# n_feat: networks for feature extraction
# n_rpn: region proposal networks
# n_head: detection head networks for proposals
# GEO: generalized enhancement operation.
# V: video frames {I_t} of length T
# MB_pix, MB_ins: pixel/instance memory bank.
def enhance_via_mem_bank (q, MB):
    K = MB.sample ()    # key set construction
    q_hat = q + GEO (q, K)    # generalized enhancement
    return q_hat
if offline_test:    # shuffle if do offline testing
    Random.shuffle(V)
for I_t in V:    # load a frame in the video
    # feature extraction networks
    f = n_feat.forward(I_t)
    # enhance with pixel-level memory bank
    f_hat_pix = enhance_via_mem_bank (f_pix, MB_pix)
    # region proposal networks
    f_ins = n_rpn.forward(f_hat_pix)
    # enhance with instance-level memory bank
    f_hat_ins = enhance_via_mem_bank (f_ins, MB_ins)
    # detection head networks
    results = n_head.forward(f_hat_ins)
    # feature-wise updating
    MB_pix.update(f_hat_pix)
    MB_ins.update(f_hat_ins)
```

Unified Multi-level Enhancement

In this section, we introduce details of the generalized enhancement operation (GEO) which enables multi-level enhancements to be performed in a unified manner. Given a query set Q and a key set K, the GEO augments each $q_i \in Q$ by measuring M relation features which are achieved by the weighted sum of all key k samples in K, where M denotes the number of attention heads. Specifically, the m-th relation feature f_R^m of a query sample q_i is calculated as:

$$f_R^m(q_i, K) = \sum_j w_{ij}^m \cdot (W_V^m \cdot k_j), \quad m = 1, \cdots, M, \quad (2)$$

where W_V^m denotes a linear transformation matrix, M is the number of relation features calculated by attention operation and w_{ij} is an element in the correlation matrix W computed based on the similarity of all q-k pairs. Precisely, w_{ij} is computed as

$$w_{ij} = \frac{\exp(S(q_i, k_j))}{\sum_k \exp(S(q_i, k_j))}, \quad (3)$$

$$S(q_i, k_j) = \frac{dot(W_Q \cdot q_i, W_K \cdot k_j)}{\sqrt{d}}, \quad (4)$$

where S(q_i, k_j) represents the similarity of q_i and k_j, dot denotes the dot product, W_Q and W_K are two transformation matrix, and d is the feature dimension. The total of M relation features are then aggregated by concatenation. Finally, the GEO outputs the augmented feature by adding the original feature q_i and the aggregated relation feature:

$$GEO(q_i, K) = q_i + concat[f_R^m(q_i, K)_{m=1}^M]. \quad (5)$$

The enhancement process can be recursively performed. Formally, for the k-th GEO of enhancement, the augmented feature of q_i is computed as

$$q_i^k = GEO(h(q_i^{k-1}), K), \quad k = 1, \cdots, N_g, \quad (6)$$

where $h(\cdot)$ denotes the feature transformation function implemented with a fully-connected layer plus ReLU and N_g denotes times of GEO for enhancement recursively. With the GEO, we can easily achieve multi-level enhancement, i.e., pixel-level and instance-level feature enhancement, which proves to be effective to utilize complementary feature to further improve the performance.

Feature-wise Updating Strategy

Existing approaches (Deng et al. 2019b; Wu et al. 2019; Chen et al. 2020; Deng et al. 2019a) update the memory by a frame-wise operation, which deletes all features of the oldest frame. For the memory bank, we present a fine-grained feature-wise memory updating strategy, which is more flexible and efficient. The feature-wise memory updating strategy can also improve the diversity of features stored in the memory leading to a better performance. To implement the feature-wise updating strategy, we use the three sampling methods introduced in § to select features in memory to be updated.

Analysis of Sampling Strategy

In video object detection, there are many redundant features because adjacent frames are very similar. If the key set has many redundancies, the entropy of information will be small, which means the key set is less informative. The score ranking strategy and frequency selection tend to sample a large portion of features from few frames and decrease the entropy of information. On the contrary, random selection generates a more diverse and informative key set.

Experiments

Experimental Settings

Dataset and Evaluation. We evaluate our method on the ImageNet (Russakovsky et al. 2015) VID dataset which contains 3862 training and 555 validation videos. We follow the previous approaches (Zhu et al. 2017b,a; Wang et al. 2018; Deng et al. 2019a; Wu et al. 2019) and train our model on the overlapped 30 classes of ImageNet VID and DET set. Specifically, we sample 15 frames from each video in VID dataset and at most 2,000 images per class from DET dataset as our training set. Then we report the mean average precision (mAP) on the validation set.

Backbone and Detection Architecture. Following (Zhu et al. 2017b,a; Deng et al. 2019a,b) we use the ResNet-101

Methods	Memory	Base detector	Backbone	mAP(%)
D&T (Feichtenhofer, Pinz, and Zisserman 2017)	-	RFCN	ResNet-101	75.8
FGFA (Zhu et al. 2017a)	Window	RFCN	ResNet-101	76.3
MANet (Wang et al. 2018)	Window	RFCN	ResNet-101	78.1
THP (Zhu et al. 2018)	Queue	RFCN	ResNet-101+DCN	78.6
STSN (Bertasius, Torresani, and Shi 2018)	Window	RFCN	ResNet-101+DCN	78.9
PSLA (Guo et al. 2019)	Queue	RFCN	ResNet-101+DCN	80.0
OGEM (Deng et al. 2019a)	Queue	RFCN	ResNet-101	79.3
Ours	Bank	RFCN	ResNet-101	**80.8**
STCA (Luo et al. 2019)	Window	FasterRCNN	ResNet-101	80.3
SELSA (Wu et al. 2019)	Window	FasterRCNN	ResNet-101	80.3
LRTR (Shvets, Liu, and Berg 2019)	Window	FPN	ResNet-101	81.0
RDN (Deng et al. 2019b)	Window	FasterRCNN	ResNet-101	81.8
MEGA (Chen et al. 2020)	Queue	FasterRCNN	ResNet-101	82.9
Ours	Bank	FasterRCNN	ResNet-101	**84.6**
LRTR (Shvets, Liu, and Berg 2019)	Window	FPN	ResNeXt-101	84.1
RDN (Deng et al. 2019b)	Window	FasterRCNN	ResNeXt-101	83.2
MEGA (Chen et al. 2020)	Queue	FasterRCNN	ResNeXt-101	84.1
Ours	Bank	FasterRCNN	ResNeXt-101	85.4
Ours†	Bank	FasterRCNN	ResNeXt-101	**86.7**

Table 1: Comparison with state-of-the-art end-to-end methods on ImageNet VID validation set. † denotes using random crop and random scale data augmentations for training.

(He et al. 2016) as our backbone. Apart from ResNet-101, we also use a stronger backbone ResNeXt-101 (Xie et al. 2017) for some comparisons. For detection network, early methods (Zhu et al. 2017b,a; Wang et al. 2018; Deng et al. 2019a; Bertasius, Torresani, and Shi 2018) use RFCN (Dai et al. 2016) as the baseline detector, while more recent methods (Wu et al. 2019; Deng et al. 2019b; Shvets, Liu, and Berg 2019; Chen et al. 2020) use FasterRCNN (Ren et al. 2015). Since our memory bank is a general module and can be applied to different detectors, we implement memory bank on top of both RFCN and FasterRCNN for fair comparisons. We apply RPN on the extracted deep feature maps. We use 12 anchors with 4 scales $64^2, 128^2, 256^2, 512^2$ and 3 aspect ratios $1:2, 1:1, 2:1$. Non-maximum suppression (NMS) is applied to generate 300 proposals for each image with an IoU threshold 0.7. Finally, NMS is applied to clean the detection results, with IoU threshold 0.5.

Training and Inference Details. To reduce the redundancy and improve the quality of the stored features, we select K samples of enhanced features to update the memory bank. Following (Deng et al. 2019b; Chen et al. 2020), we select K=75 proposals with highest objectness score for instance-level memory bank. For pixel-level memory bank, we randomly select K=100 pixels within each detected bounding box. In both training and test phases, the images are resized to a shorter side of 600 pixels. The whole architecture is trained on 4 Titan RTX GPUs with SGD (momentum: 0.9, weight decay: 0.0001). In the first phase, we only train the pixel-level enhancement. Each GPU contains one mini-batch consisting of two frames, the key frame I_k and a randomly selected frame from the video to approximately form pixel-level memory. Both RPN losses and Detection losses are only computed on the key frame. We train the pixel-level model for 60K iterations. The learning rate is 0.001 for the first 40K iterations, and 0.0001 for the last 20k iterations. In the second phase, we end-to-end train both pixel-level enhancement and instance-level enhancement for 120K iterations. The learning rate is 0.001 for the first 80K iterations and 0.0001 for the last 40K iterations.

Comparison

End-to-end performance. We compare our method with the state-of-the-art methods in Table 1. To make fair comparisons with other methods, we implement our method on top of two base detectors: RFCN (Dai et al. 2016) and FasterRCNN (Ren et al. 2015). Table 1 shows that we achieve the best performance with both RFCN and FasterRCNN setting. Specifically, for RFCN setting, our method also outperforms the best competitor OGEM by 1.5% mAP and achieve 80.8% mAP. For FasterRCNN setting, our method achieves 84.6% of mAP and outperforms its best competitor MEGA by 1.7% using ResNet-101 backbone. By replacing the backbone with a stronger network ResNeXt-101, our method achieves 85.4% of mAP. By adding random crop and random scale data augmentations for training, our method finally achieves 86.7% of mAP.

Speed-accuracy trade-off. To analyze the speed-accuracy trade-off, we re-implement many state-of-the-art methods and make comparisons in Table 1. All results are obtained on Titan RTX GPUs. Our lite-version model Ours$_{ins}$ which

Methods	Base detector	mAP(%)	Published		Our Impl.
			Runtime(ms)	Device	Runtime(ms)
FGFA (Zhu et al. 2017a)	RFCN	76.3	733	K40	-
MANet (Wang et al. 2018)	RFCN	78.1	269.7	Titan X	-
OGEM (Deng et al. 2019a)	RFCN	79.3	112	1080 TI	89.1
Our$_{pix}$	RFCN	80.2	-		**81.3**
Our	RFCN	**80.8**			90.1
STCA (Luo et al. 2019)	FasterRCNN	80.3	322.2	Titan X	-
SELSA (Wu et al. 2019)	FasterRCNN	80.3	-		91.2
RDN (Deng et al. 2019b)	FasterRCNN	81.8	94.2	V100	128.0
MEGA (Chen et al. 2020)	FasterRCNN	82.9	114.5	2080 TI	182.7
Ours$_{ins}$	FasterRCNN	83.7	-		**79.6**
Ours	FasterRCNN	**84.6**			110.3

Table 2: Speed-accuracy trade-off with ResNet-101 backbone. The last column shows the runtime(ms) of our implementations. All our results are obtained on Titan RTX GPUs.

Methods	Pixel	Instance	mAP(%)	Runtime (ms)
RFCN			73.8	46.7
Ours$_{pix}$	✓		80.2$_{\uparrow 6.4}$	81.3
Ours$_{ins}$		✓	76.7$_{\uparrow 2.9}$	56.0
Ours	✓	✓	80.8$_{\uparrow 6.8}$	90.1
FasterRCNN			75.4	51.8
Ours$_{pix}$	✓		81.8$_{\uparrow 6.4}$	81.6
Ours$_{ins}$		✓	83.7$_{\uparrow 8.3}$	79.6
Ours	✓	✓	84.6$_{\uparrow 9.2}$	110.3

Table 3: Ablation study of pixel-level and instance-level memory bank on single frame baselines. The first part represents the results using RFCN (Dai et al. 2016) as the base detector. The second part shows the results using FasterRCNN (Ren et al. 2015) as the base detector.

only uses instance-level memory bank achieves both higher accuracy and faster speed than its best competitor MEGA. Specifically, Our$_{ins}$ achieves 83.7% mAP which outperforms MEGA by 1.7% mAP. Meanwhile, the speed of Ours$_{ins}$ is 79.6 ms much faster than MEGA. When both pixel-level and instance-level enhancements are performed, the accuracy of our method is further improved to 84.6% mAP and the speed is slightly decreased to 110.3 ms.

Ablation Study

To demonstrate the effect of key components in our memory bank, we conduct extensive experiments to study how they contribute to the final performance.

Multi-level enhancement. In this part, we carefully analyze every component of our method. Table 3 shows our results using two different base detectors, RFCN (shown in upper rows) and FasterRCNN (shown in lower rows). The single frame baseline achieve 73.8% mAP and 75.4% mAP for RFCN and FasterRCNN, respectively. By introducing the pixel-level memory bank, performances of two baselines are improved to 80.2% mAP and 81.8% mAP, respectively. The improvements introduced by pixel-level enhancement are equal for the two baselines. By introducing instance-level memory bank, the FasterRCNN baseline is hugely improved by 8.3% mAP and achieves 83.7% mAP. However, for the RFCN baseline, the improvement, 2.9% mAP, is relatively low. We believe that the improvement gap is caused by the difference of semantic information. Specifically, the proposals of FasterRCNN have more semantic information than the psroi-pooled features of RFCN. By utilizing both pixel-level and instance-level memory banks, the performance is further improved to 80.8% mAP and 84.6% mAP for the two base detectors. We also show the runtime speed of every model in the last column of Table 3.

Methods	(a)	(b)	(c)	(d)
Frame-wise updating?		✓		
Feature-wise updating?			✓	✓
Class-wise memory?				✓
mAP(%)	80.3	81.7	82.4	82.7

Table 4: Effect of feature-wise updating strategy and class-wise memory.

Feature-wise updating. Instead of updating the memory frame-wisely, we propose a feature-wise updating strategy which updates the memory in a more fine-grained manner. Specifically, we reuse the three different strategies for key set construction to select features in memory to be deleted. The three strategies achieve similar improvements. For simplicity, we use random selection for both key set construction and memory updating in our experiments. We use SELSA (Wu et al. 2019) as a baseline and denote it as $Model(a)$ in Table 4. $Model(b)$ incorporates frame-wise updating (memory queue). $Model(c)$ incorporates the pro-

N_m	Concatenation		Sampling	
	mAP(%)	Runtime(ms)	mAP(%)	Runtime(ms)
3k	82.3	83.4	83.2	79.4
6k	82.7	92.6	83.4	79.3
12k	82.7	104.6	83.5	79.4
24k	82.7	142.9	83.7	79.6
48k	OOM		83.7	80.3
96k			83.8	81.0

Table 5: Effect of light-weight key set construction with different N_m, where N_m denotes the total number of stored samples in the memory bank. OOM denotes out of GPU memory errors on Titan RTX (24GB) devices.

N_k	50	200	1k	2k*	5k
mAP(%)	83.1	83.5	83.6	83.7	**83.8**
runtime(ms)	**75.8**	75.9	77.3	79.6	86.7

Table 6: Analysis of different number of N_k.

N_{pix}	0	1*	2	3
mAP(%)	75.4	81.8	81.8	81.7
Runtime (ms)	51.8	81.6	112.2	143.1

Table 7: Analysis of different number of N_{pix}.

N_{ins}	0	1	2*	3
mAP(%)	75.4	82.0	83.7	83.8
Runtime (ms)	51.8	65.1	79.6	94.1

Table 8: Analysis of different number of N_{ins}.

Pixel	AR^5	AR^{10}	AR^{100}
	77.9	83.8	94.3
✓	**79.5**$_{\uparrow 1.6}$	**85.7**$_{\uparrow 1.9}$	**96.3**$_{\uparrow 2.0}$

Table 9: Effect of pixel-level enhancement to RPN.

posed feature-wise updating strategy. As shown in Table 4, comparing with frame-wise updating, feature-wise updating improves the performance by 0.7% mAP and achieves 82.4% mAP. Previous methods (Deng et al. 2019b; Chen et al. 2020) only maintain a video-wise memory that deletes all memory by the end of a video. In contrast, we introduce a class-wise memory which is maintained for the whole dataset (multiple videos). In this way, the enhancement process can utilize the information from other videos. By adding the class-wise memory, $Model(d)$ further improves the performance by 0.3% to 82.7% mAP.

Light-weight key set construction. We compare our light-weight key set construction strategy with the widely used concatenation. We conduct experiments on $Model(d)$ which concatenates all 6k stored features in memory as the key set. We test the overall performance under different number of stored features N_m, from $3k$ to $96k$. For every N_m, we keep the number of selected keys to $N_k = 2k$. Table 5 shows that our light-weight key set construction strategy works always better than concatenation in both speed and accuracy under all N_m settings. Using our light-weight key set construction strategy the runtime of the model can always roughly stay the same with the increasing of N_m. In contrast, using concatenation, the speed becomes slower and slower as the increasing of N_m. Specifically, when N_m is increased to $48k$, concatenation strategy occurs the Out Of Memory (OOM) error on Titan RTX GPU (24GB). We use $N_m = 24k$.

Size of the key set. We evaluate how the size of the key set N_k effects the performance. In this part, we conduct experiments on the our lite-version method Ours$_{ins}$ which only performs two times of instance-level enhancement $N_{ins} = 2$ and no pixel-level enhancement $N_{pix} = 0$. From Table 6, our method is very robust to the number of sampled keys. In practice, we set $N_k = 2000$ for better speed-accuracy trade-off. Surprisingly, even sampling a very small number of keys $N_k = 50$, our method still achieves 83.1% mAP and outperforms the best competitor MEGA by 0.2% mAP.

Number of pixel-level enhancements. As discussed in §, the enhancement with memory bank can be performed multiple times. We evaluate the effect of the number of pixel-level enhancements N_{pix}. When $N_{pix} = 0$, no pixel-level enhancement is performed, our method degenerates to the FasterRCNN baseline. In Table 7, we vary N_{pix} from 0 to 3. With $N_{pix} = 1$, the performance is improved by 6.4% to 81.8% mAP. By further increasing N_{pix}, the performance is merely improved. We use $N_{pix} = 1$ by default.

Number of instance-level enhancements. Similarly, we evaluate the effect of the number of instance-level enhancements N_{pix} by performing instance-level enhancement on top of the FasterRCNN. When $N_{ins} = 0$, no instance-level enhancement is performed. From Table 8, we can see that $N_{ins} = 2$ achieves the best speed-accuracy trade-off. Specifically, the performance is improved by 8.3% to 83.7% mAP with $N_{ins} = 2$. We use $N_{ins} = 2$ by default.

Effect of pixel-level memory bank to RPN. Recent instance-level methods (Chen et al. 2020; Wu et al. 2019; Deng et al. 2019b) do not enhance deep feature maps used in RPN. Thus, the RPN potentially misses some low-quality objects. We evaluate the effect of pixel-level memory bank to RPN. Specifically, the metric Average Recall (AR) is used for comparison. We select top k of the proposals generated by RPN to calculate the AR^k. Specifically, we tested with $k = \{5, 10, 100\}$. As shown in Table 9, with pixel-level enhancement, AR^5, AR^{10}, and AR^{100} are all improved.

Conclusions

In this paper, we propose a multi-level aggregation framework via memory bank (MAMBA). The memory bank contains two novel operations: (1) light-weight key-set construction and (2) fine-grained feature-wise memory updating. Experiment results demonstrate MAMBA achieves superior performance on the challenging ImageNet VID dataset in terms of both speed and accuracy.

References

Bertasius, G.; Torresani, L.; and Shi, J. 2018. Object detection in video with spatiotemporal sampling networks. In *ECCV*.

Chen, Y.; Cao, Y.; Hu, H.; and Wang, L. 2020. Memory Enhanced Global-Local Aggregation for Video Object Detection. In *CVPR*.

Dai, J.; Li, Y.; He, K.; and Sun, J. 2016. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*.

Deng, H.; Hua, Y.; Song, T.; Zhang, Z.; Xue, Z.; Ma, R.; Robertson, N.; and Guan, H. 2019a. Object Guided External Memory Network for Video Object Detection. In *ICCV*.

Deng, J.; Pan, Y.; Yao, T.; Zhou, W.; Li, H.; and Mei, T. 2019b. Relation Distillation Networks for Video Object Detection. In *ICCV*.

Dosovitskiy, A.; Fischer, P.; Ilg, E.; Hausser, P.; Hazirbas, C.; Golkov, V.; Van Der Smagt, P.; Cremers, D.; and Brox, T. 2015. Flownet: Learning optical flow with convolutional networks. In *ICCV*.

Feichtenhofer, C.; Pinz, A.; and Zisserman, A. 2017. Detect to track and track to detect. In *ICCV*.

Girshick, R. 2015. Fast r-cnn. In *ICCV*.

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. 2015. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38(1): 142–158.

Guo, C.; Fan, B.; Gu, J.; Zhang, Q.; Xiang, S.; Prinet, V.; and Pan, C. 2019. Progressive Sparse Local Attention for Video Object Detection. In *ICCV*.

Han, W.; Khorrami, P.; Paine, T. L.; Ramachandran, P.; Babaeizadeh, M.; Shi, H.; Li, J.; Yan, S.; and Huang, T. S. 2016. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Hu, H.; Gu, J.; Zhang, Z.; Dai, J.; and Wei, Y. 2018. Relation networks for object detection. In *CVPR*.

Ilg, E.; Mayer, N.; Saikia, T.; Keuper, M.; Dosovitskiy, A.; and Brox, T. 2017. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *CVPR*.

Kang, K.; Li, H.; Xiao, T.; Ouyang, W.; Yan, J.; Liu, X.; and Wang, X. 2017a. Object detection in videos with tubelet proposal networks. In *CVPR*.

Kang, K.; Li, H.; Yan, J.; Zeng, X.; Yang, B.; Xiao, T.; Zhang, C.; Wang, Z.; Wang, R.; Wang, X.; et al. 2017b. T-cnn: Tubelets with convolutional neural networks for object detection from videos. *IEEE Transactions on Circuits and Systems for Video Technology* 28(10): 2896–2907.

Luo, H.; Huang, L.; Shen, H.; Li, Y.; Huang, C.; and Wang, X. 2019. Object Detection in Video with Spatial-temporal Context Aggregation. *arXiv preprint arXiv:1907.04988* .

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *CVPR*.

Redmon, J.; and Farhadi, A. 2017. YOLO9000: better, faster, stronger. In *CVPR*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115(3): 211–252.

Shvets, M.; Liu, W.; and Berg, A. C. 2019. Leveraging Long-Range Temporal Relationships Between Proposals for Video Object Detection. In *ICCV*.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* .

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*.

Tian, Z.; Shen, C.; Chen, H.; and He, T. 2019. FCOS: Fully Convolutional One-Stage Object Detection. *arXiv preprint arXiv:1904.01355* .

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS*.

Wang, S.; Zhou, Y.; Yan, J.; and Deng, Z. 2018. Fully motion-aware network for video object detection. In *ECCV*.

Wu, H.; Chen, Y.; Wang, N.; and Zhang, Z. 2019. Sequence Level Semantics Aggregation for Video Object Detection. In *ICCV*.

Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; and He, K. 2017. Aggregated residual transformations for deep neural networks. In *CVPR*.

Zhu, X.; Dai, J.; Zhu, X.; Wei, Y.; and Yuan, L. 2018. Towards high performance video object detection for mobiles. *arXiv preprint arXiv:1804.05830* .

Zhu, X.; Wang, Y.; Dai, J.; Yuan, L.; and Wei, Y. 2017a. Flow-guided feature aggregation for video object detection. In *ICCV*.

Zhu, X.; Xiong, Y.; Dai, J.; Yuan, L.; and Wei, Y. 2017b. Deep feature flow for video recognition. In *CVPR*.