Memory Based Video Scene Parsing

Zhenchao Jin[1], Dongdong Yu[2], Kai Su[2], Zehuan Yuan[2], Changhu Wang[2]
[1]University of Science and Technology of China [2]ByteDance
blwx@mail.ustc.edu.cn
{yudongdong, sukai, yuanzehuan, wangchanghu}@bytedance.com,

Abstract

Video scene parsing is a long-standing challenging task in computer vision, aiming to assign pre-defined semantic labels to pixels of all frames in a given video. Compared with image semantic segmentation, this task pays more attention on studying how to adopt the temporal information to obtain higher predictive accuracy. In this report, we introduce our solution for the 1st Video Scene Parsing in the Wild Challenge, which achieves a mIoU of 57.44 and obtained the 2nd place (our team name is CharlesBLWX).

1. Overview

Our goal is to build an accurate 2D segmentor for video scene parsing. As the performance is the most important term for this challenge, we adopt BEiT [1] as our backbone network, *i.e.*, the encoder structure. Then, we take the upernet decoder [8] as our baseline method and introduce several improvements to build the final video semantic segmentation framework. Below we first present the main idea of our method in Section 2, followed by the network structure used in the proposed segmentor in Section 3. Then, we describe the train and inference strategy used to further boost our segmentation performance in Section 4. Finally, we report the experiments and detailed ablation analysis in Section 5.

2. Our Approach

Although video semantic segmentation is different from image semantic segmentation, a stronger image semantic segmentor usually can achieve the better performance in video semantic segmentation. Therefore, we adopt BEiT backbone network [1] and upernet decoder [8] as our baseline method, whose performance on ADE20K [10] has achieved a state-of-the-art mIoU, *i.e.*, 57.00%.

Based on this, we improve the decoder structure by using two kinds of memory strategies. One is to set up a feature memory module and store the dataset-level representations of various classes in it. Then, we adopt these dataset-level representations to augment the pixel representations of current input image. The other is to incorporate the temporal memory attention module (*i.e.*, TMA module) [7] into the first decoder structure.

After training both segmentors on VSPW [5], we ensemble the output class probability distributions of the segmentors to obtain the final prediction results.

3. Model Structure

Encoder Structure. We leverage BEiT [1] as our encoder structure, in which we set $layer = 24$, $hidden = 1024$, $FFNfactor = 4\times$, $head = 16$ and $patch = 16 \times 16$. The initialized weights are self-supervised pretrained and then intermediate fine-tuned on ImageNet22k following [1].

Decoder Structure A. Here we introduce the first decoder structure designed in this paper. This structure is based on our paper *Mining Contextual Information Beyond Image for Semantic Segmentation* accepted by ICCV 2021. As illustrated in Figure 1, given the input $Frame\ N$, we first leverage the BEiT backbone network to obtain the multi-level representations $\mathcal{R} = \{\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4\}$. Then, we leverage the upernet head consisted of pyramid pooling module [9] and feature pyramid network [4] to extract the basic representations $\mathcal{C}_{wi}$. Next, we will set up a feature memory module $\mathcal{M}$ which is dynamically updated during training. To be more specific, $\mathcal{M}$ of size $K \times C$ is introduced to store the dataset-level representations of various classes, where K is the number of the classes and the dimension of a pixel representation is C. To set up the feature memory, we first randomly select one pixel representation for each category from the train set to initialize $\mathcal{M}$, where the representations are calculated by leveraging the backbone network BEiT. Then, the values in $\mathcal{M}$ are updated by leveraging moving average after each training iteration:

$$\mathcal{M}_t = (1 - m_{t-1}) \cdot \mathcal{M}_{t-1} + m_{t-1} \cdot \mathscr{T}(\mathcal{R}_{t-1}), \quad (1)$$

where m is the momentum, t denotes for the current number of iterations and $\mathscr{T}$ is used to transform $\mathcal{R}$ to have the same size as $\mathcal{M}$.

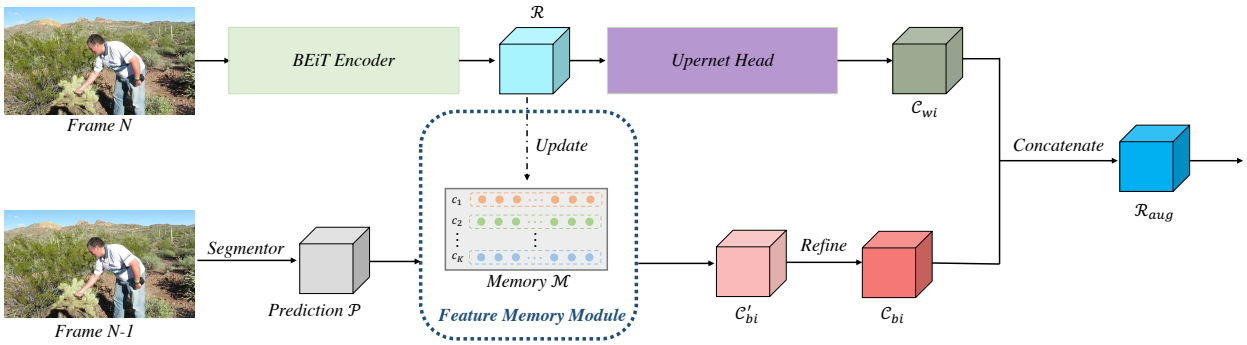

Figure 1. Illustrating the pipeline of Decoder A.

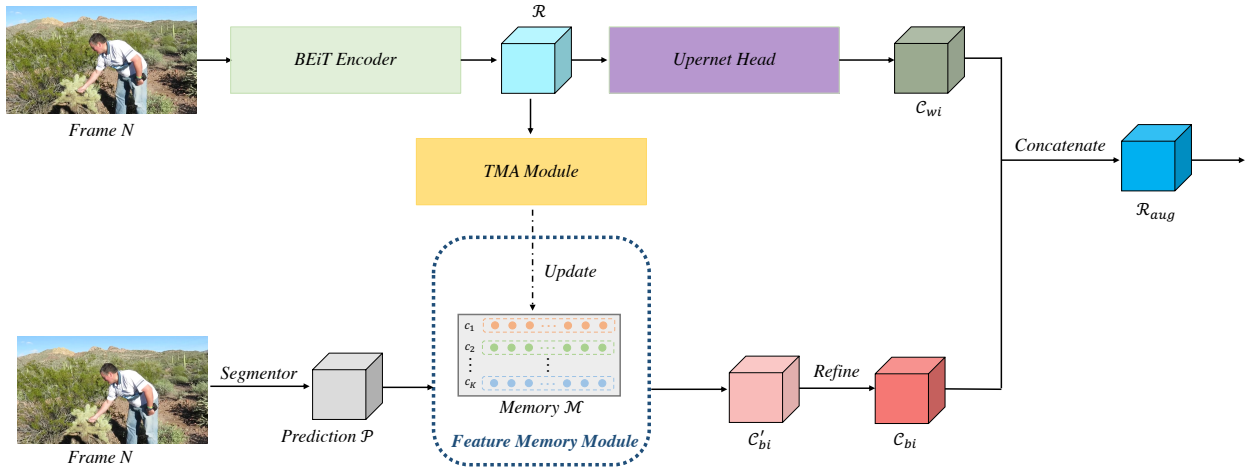

Figure 2. Illustrating the pipeline of Decoder B.

To implement $\mathscr{T}$, we first setup a matrix $\mathcal{R}'$ of size $K \times C$ and initialize it by using the values in $\mathcal{M}$. For the convenience of presentation, we leverage the subscript $[i, j]$ or $[i, *]$ to index the element or elements of a matrix. $\mathcal{R}$ is upsampled and permuted as size $HW \times C$, *i.e.*, $\mathcal{R}^{HW\times C}$. Subsequently, for each category c_k existing in the input image, we have:

$$\mathcal{R}_{c_k} = \{\mathcal{R}^{HW\times C}_{[i,*]} \mid (\mathcal{GT}_{[i]} = c_k) \wedge (1 \leq i \leq HW)\}, \quad (2)$$

where $\mathcal{GT}$ of size HW stores the ground truth category labels of $\mathcal{R}^{HW\times C}$. $\mathcal{R}_{c_k}$ of size $N_{c_k} \times C$ stores the representations of category c_k of $\mathcal{R}^{HW\times C}$. N_{c_k} is the number of pixels labeled as c_k in the input image. Next, we calculate the cosine similarity matrix $\mathcal{S}_{c_k}$ of size N_{c_k} between $\mathcal{R}_{c_k}$ and $\mathcal{M}_{[c_k,*]}$:

$$\mathcal{S}_{c_k} = \frac{\mathcal{R}_{c_k} \cdot \mathcal{M}_{[c_k,*]}}{\|\mathcal{R}_{c_k}\|_2 \cdot \|\mathcal{M}_{[c_k,*]}\|_2}. \quad (3)$$

Finally, the representation of c_k in $\mathcal{R}'$ is updated as:

$$\mathcal{R}'_{[c_k,*]} = \sum_{i=1}^{N_{c_k}} \frac{1 - S_{c_k,[i]}}{\sum_{j=1}^{N_{c_k}}(1 - S_{c_k,[j]})} \cdot \mathcal{R}_{c_k,[i,*]}. \quad (4)$$

The output of $\mathscr{T}$ is $\mathcal{R}'$ which has been updated by all the representations of various classes in $\mathcal{R}^{HW\times C}$.

Then, $\mathcal{M}$ can be used to augment the original representations $\mathcal{R}$ of current input image by leveraging $\mathcal{P}$. In our implementations, $\mathcal{P}$ is the predicted segmentation mask of previous frame, *i.e.*, $Frame\ N-1$. Specifically, we select the dataset-level representations for each pixel according to $\mathcal{P}$ so that we can obtain $\mathcal{C}'_{bi}$. Then, we calculate the relations between $\mathcal{R}$ and $\mathcal{C}'_{bi}$ so that we can obtain a position confidence weight to further refine $\mathcal{C}'_{bi}$. Specifically, we first calculate the relations $\mathcal{O}$ as follows:

$$\mathcal{O} = Softmax(\frac{g_q(permute(\mathcal{R})) \otimes g_k(\mathcal{C}'_{bi})^T}{\sqrt{\frac{C}{2}}}), \quad (5)$$

where $permute$ is adopted to let $\mathcal{R}$ have size of $\frac{HW}{64} \times C$. Then, $\mathcal{C}'_{bi}$ is refined as follows:

$$\mathcal{C}_{bi} = permute(g_o(\mathcal{O} \otimes g_v(\mathcal{C}'_{bi}))), \quad (6)$$

where g_q, g_k, g_v and g_o are introduced to adjust the dimension of each pixel representation, implemented by a 1×1

Table 1. Ablation study of model ensemble on the validation set of VSPW. All the models are trained on the train set and tested under single-scale.

Method	Backbone	Epochs	mIoU
Upernet	BEiT-Large	240	60.02
Decoder A (*ours*)	BEiT-Large	240	61.24
Decoder B (*ours*)	BEiT-Large	240	61.18
Decoder A+B (*ours*)	BEiT-Large	240	**62.12**

Table 2. Ablation study of multi-stage inference on the validation set of VSPW. All the models are trained on the train set and tested under single-scale.

Method	Backbone	Inference Stage	mIoU
Decoder A	BEiT-Large	stage0	61.24
Decoder A	BEiT-Large	stage1	61.43
Decoder A	BEiT-Large	stage2	**61.44**
Decoder A	BEiT-Large	stage3	61.43

convolutional layer. $permute$ is used to let the output have size of $C \times \frac{H}{8} \times \frac{W}{8}$.

Finally, we concatenate the output of upernet head $\mathcal{C}_{wi}$ and the refined representations $\mathcal{C}_{bi}$ to predict the segmentation mask of current frame.

Decoder Structure B. Here we introduce the second decoder structure designed in this paper. This decoder is similar to decoder structure A and there are two main differences:

- We introduce the TMA module [7] to better model the temporal information. Specifically, the input of feature memory module is the output of TMA rather than $\mathcal{R}$.
- $\mathcal{P}$ is the predicted segmentation mask of current frame N rather than $N-1$.

After calculating $\mathcal{C}'_{bi}$, we also leverage the output of TMA to refine it so that we can obtain $\mathcal{C}_{bi}$. Following decoder structure A, we finally concatenate the output of upernet head $\mathcal{C}_{wi}$ and the refined representations $\mathcal{C}_{bi}$ to predict the segmentation mask of current frame.

Model Ensemble. We simply add the outputs (*i.e.*, the class probability distribution of each pixel representation) of both decoders to generate the final segmentation mask.

4. Train and Inference Strategy

Train Strategy. Here, we introduce several tricks used in training the proposed decoders.

- In the initial stage of training, considering the instability of the segmentor, we replace $\mathcal{P}$ with the ground truth label of the selected frames.
- We use the data augmentaton to further boost the performance of our segmentor. Specifically, we adopt random scaling, horizontal flipping and color jitter following the default setttings in MMSegmentation [2].
- Synchronized batch normalization implemented by pytorch is enabled during training.
- AdamW is used as our optimizer.

Test Strategy. Here, we introduce several tricks adopted in testing the proposed decoders.

- We use multi-scale and flipping testing technology to obtain the best segmentation performance where the selected scales are $[0.75, 1.0, 1.25, 1.5, 1.75]$.
- To further boost the segmentation performance, we also save the predicted segmentation masks in the first-stage testing. And then, we replace $\mathcal{P}$ with the saved segmentation masks to performance the second-stage predicting. Similarly, we can perform the third-stage, fourth-stage testing and so on.

5. Experiments and Analysis

5.1. Training Configuration

Our method is implemented in PyTorch ($version \geq 1.3$) [6] and trained on 8 NVIDIA Tesla V100 GPUs with a 32 GB memory per-card. The overall learning consists of two stages: pre-training stage and fine-tuning stage.

pre-training stage. We learn the backbone network of our framework by leveraging ImageNet22K dataset [3] following the settings described in [1].

fine-tuning stage. The initial learning rate is set as 0.00002 and the weight decay is 0.05. We set the crop size of the input image as 512×512 and batch size as 16 by default. Besides, the networks are fine-tuned for 240 epochs on the train set. For each iteration, we randomly select one frame from the videos to train our framework.

5.2. Abaltion Study

Model Ensemble. As indicated in Table 1, the designed Decoder A outperforms the baseline model by 1.22% mIoU and the proposed Decoder B is 1.16% mIoU higher than the baseline model. Finally, by the technology of model ensemble, we obtain the final video segmentation framework with a mIoU of 62.12%.

Multi-stage Inference. As illustrated in Table 2, we show the ablation study on the multi-stage inference. To be more specific, the stage0 means the original output $Mask_0$ of Decoder A. And stage1 means the predicted segmentation mask $Mask_1$ by replacing $\mathcal{P}$ with $Mask_0$. Similarly, stage2 and stage3 use $Mask_1$ and $Mask_2$ to replace the $\mathcal{P}$ to obtain the predicted segmentation mask, respectively.

5.3. Video Segmentation Framework

Our final video segmentation framework is built by ensembling the Decoder A and Decoder B as well as leveraging the trick of multi-stage inference and we finally achieve a mIoU of 57.44% on the test set of VSPW.

References

[1] Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

[2] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. `https://github.com/open-mmlab/mmsegmentation`, 2020.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[5] Jiaxu Miao, Yunchao Wei, Yu Wu, Chen Liang, Guangrui Li, and Yi Yang. Vspw: A large-scale dataset for video scene parsing in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4133–4143, 2021.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[7] Hao Wang, Weining Wang, and Jing Liu. Temporal memory attention for video semantic segmentation. *arXiv preprint arXiv:2102.08643*, 2021.

[8] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 418–434, 2018.

[9] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[10] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017.