Learning Trajectory-Aware Transformer for Video Super-Resolution Supplementary Material

Chengxu Liu[1], Huan Yang[2], Jianlong Fu[2], Xueming Qian[1]
[1]Xi'an Jiaotong University [2]Microsoft Research Asia
liuchx97@gmail.com, {huayan, jianf}@microsoft.com, qianxm@mail.xjtu.edu.cn

In this supplementary material, Sec. 1 illustrates the details of the algorithm. Sec. 2 and Sec. 3 describe the detailed architecture and experimental settings, respectively. Sec. 4 provides the complexity analysis of trajectory-aware attention in TTVSR. Sec. 5 analyzes the limitations of TTVSR. Finally, Sec. 6 shows more comparison results.

1. Algorithm Details

In this section, we illustrate the details of the algorithm. It includes the pseudocode of the entire algorithm, the detailed analyzes of the proposed cross-scale feature tokenization module, and location map updating mechanism in our TTVSR.

Algorithm pseudocode. As shown in Alg. 1, we describe our proposed TTVSR in the form of pseudocode. Besides, we follow previous works [1, 2] to adopt a bidirectional propagation scheme, where the features in different frames can be propagated backward and forward. For clarity, in this algorithm, we only show the process of forward propagation, and the process of backward propagation is similar to forward propagation.

Cross-scale feature tokenization. As discussed in the main paper, in addition to the more complex motion in the long-range sequence, at the same time, the contents in sequences have the distinct scale changing as it moves. The use of texture from a larger scale often helps to recover more detailed texture on a smaller scale. Therefore, to adapt to the changes of different scales, we propose the cross-scale feature tokenization before trajectory-aware attention to obtain the unified features from the multi-scale. As shown in Fig. 1, we first use the successive unfold and fold operations to expand the receptive field of features, which capture features with more textures on a larger scale. Then, features of different scales are sampled to the same one by a pooling operation, which is used to unify features from different scales. Finally, the features are split by unfolding operation to obtain the output tokens. It is noteworthy that this process can extract features of the whole temporal sequence in parallel, and obtain features of larger scales without adding additional parameters unlike other tasks [6, 15].

Algorithm 1 The detailed algorithm describe of TTVSR.

Input: $\mathbf{I}_{LR}$: $\{I^t_{LR}, t \in [1, T]\}$; T: the length of sequence; $\mathcal{L}^{init}$: initialization by uniform discretization; H and W: the height and width of the feature maps; $\phi(\cdot)$ and $\varphi(\cdot)$: embedding networks; $\mathrm{S}(\cdot)$: spatial sampling operation; $\mathrm{R}(\cdot)$: reconstruction network. $\mathrm{P}(\cdot)$: pixel-shuffle. $\mathrm{U}(\cdot)$: upsampling operation. $\mathrm{H}(\cdot)$: motion estimation network with parameter θ and an average pooling operation. $\mathrm{A}_{traj}(\cdot)$: trajectory-aware attention.

Output: $\mathbf{I}_{SR}$: $\{I^t_{SR}, t \in [1, T]\}$;

$\mathcal{V} = \{\}$;
$\mathcal{L} = \{\}$;
for $t = 1; t <= T; t++$ **do**
  $O^t = \mathrm{H}(I^t_{LR}, I^{t-1}_{LR}; \theta)$;
  $\mathcal{L} = \mathrm{S}(\mathcal{L}, O^t)$;
  $\mathcal{L}$ add $\mathcal{L}^{init}$;
  $\mathcal{Q} = \phi(I^t_{LR}) = \{q_{\mathcal{L}^t_{m,n}}, m \in [1, H], n \in [1, W]\}$;
  $\mathcal{K} = \phi(\mathbf{I}_{LR}) = \{k_{\mathcal{L}^{t'}_{m,n}}, m \in [1, H], n \in [1, W], t' \in [1, t-1]\}$;
  $\mathcal{V} = \varphi(\mathbf{I}_{LR}) = \{v_{\mathcal{L}^{t'}_{m,n}}, m \in [1, H], n \in [1, W], t' \in [1, t-1]\}$;
  $F_{atten} = \mathrm{R}(\mathrm{A}_{traj_{t,m,n}}(q_{\mathcal{L}^t_{m,n}}, k_{\mathcal{L}^{t'}_{m,n}}, v_{\mathcal{L}^{t'}_{m,n}}))$
  $\mathcal{V}$ add F_{atten} (the process of obtaining F_{atten} can be represented as φ);
  $I^t_{SR} = \mathrm{P}(F_{atten}) + \mathrm{U}(I^t_{LR})$
end for

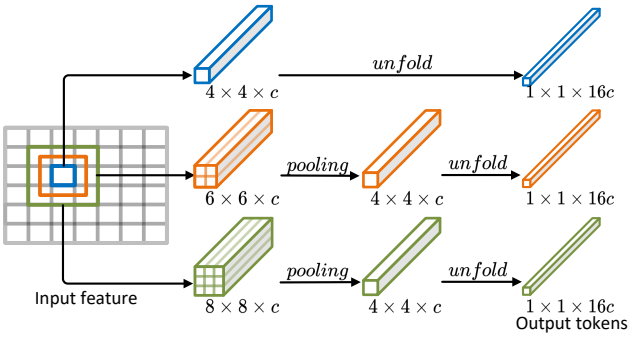

Figure 1. An illustration of the cross-scale feature tokenization.

Location map updating. As discussed in the main paper,

Table 1. Model Params and Runtime analysis.

Method	#Params	Runtime
Flow Estimator	1.4M	7.0ms
Feature Extraction	0.4M	0.1ms
Cross-scale Feature Tokenization	0.0M	8.0ms
Trajectory-aware Attention	0.1M	79.6ms
Reconstruction Network	4.8M	28.6ms
TTVSR Total	6.7M	142ms
MuCAN [5]	13.6M	1,202ms
BasicVSR [1]	6.3M	63ms
IconVSR [1]	8.7M	70ms

the location maps will change over time. We denotes the updated location maps as ${}^*\mathcal{L}^t$ and provides a more detailed and formulated description in this section.

When changing from time T to time $T+1$, first, based on Equ. 7 in the main paper, ${}^*\mathcal{L}^{T+1}_{m,n}$ represents the coordinate at time $T+1$ of a trajectory which is ended at (m, n) at time $T+1$, which can be expressed as:

$$ {}^*\mathcal{L}^{T+1}_{m,n} = (m, n). \tag{1} $$

Then we update the existing location maps $\{\mathcal{L}^1_{m,n}, \cdots, \mathcal{L}^T_{m,n}\}$. To build the connection of trajectories between time T and time $T+1$, we introduce a lightweight motion estimation network which computes a backward flow O^{T+1} from I^{T+1}_{LR} to I^T_{LR}. This process can be formualted as:

$$ O^{T+1} = \mathrm{H}(I^{T+1}_{LR}, I^T_{LR}; \theta), \tag{2} $$

where $\mathrm{H}(\cdot)$ is composed of the motion estimation network with parameter θ and an average pooling operation. The average pooling is used to ensure that the output of the motion estimation is the same size as $\mathcal{L}^t$.

Finally, we get the updated coordinates in location map $\mathcal{L}^t$ by interpolating between its adjacent coordinates:

$$ {}^*\mathcal{L}^t = \mathrm{S}(\mathcal{L}^t, O^{T+1}), \tag{3} $$

where $\mathrm{S}(\cdot)$ represents the spatial sampling operation on matrix $\mathcal{L}^t$ by spatial correlation O^{T+1} (i.e., $grid\_sample$ in PyTorch). Thus far, we have all the location maps for time $T+1$.

2. Details of Architecture and Runtimes

In this section, we will illustrate the detailed architecture and runtimes of TTVSR.

Architecture. We follow IconVSR [1], the feature extraction network uses five residual blocks, which are part of embedding operation $\phi(\cdot)$. The feature reconstruction network $\mathrm{R}(\cdot)$ uses 60 residual blocks, which are part of embedding operation $\varphi(\cdot)$. The channel number of feature is set to 64.

Runtimes. As shown in Tab. 1, we analyze the parameter size and runtime of each component in TTVSR. The runtime is computed on one LR frame with the size of 180×320 and $\times4$ upsampling, and all models are conducted on an NVIDIA Tesla V100 GPU.

As shown in Tab. 1, our method is much lighter and faster than other attention-based methods (e.g., MuCAN [5], which is the SOTA attention-based method), this is thanks to our efficient design of trajectory-aware attention. Compare with other methods without attention mechanisms, TTVSR achieves higher results with the smaller parameters size and comparable FLOPs (as shown in main paper Table 3). While it's worth noting that TTVSR is slower than IconVSR [1]. This is because the runtime highly depends on the code implementation and hardware platforms. Specific to our case, the attention mechanism contains a lot of small matrix multiplications which limits our method to run at the high efficiency (as shown in the fourth row of Tab. 1, the time used to calculate attention is more than half of the total time). However, we believe that with the optimization of code and hardware, TTVSR can achieve the comparable runtime to IconVSR [1] as demonstrated by FLOPs which is independent of hardware.

3. More Experimental Settings

Datasets. We use the **REDS** [9] and **Vimeo-90K** [11] datasets to construct our training data. For **REDS** [9], we apply the MATLAB bicubic downsample (BI) degradation on **REDS4** [9] to evaluate TTVSR. For **Vimeo-90K** [11], we use the Gaussian filter with a standard deviation of $\sigma = 1.6$ and downsampling (BD) degradation, and use **Vid4** [7], **UDM10** [12] and **Vimeo-90K-T** [11] as test sets along with it.

Settings. To leverage the information of the whole sequence, we follow previous works [1, 2] to adopt a bidirectional propagation scheme. In the process of bidirectional propagation, the same parameters are shared, and the final feature at each frame is obtained by cascading bidirectional output features.

For **REDS** [9], we use sequences with a length of 50 as inputs, and loss is computed for the 50 output frames. For **Vimeo-90K** [11], we augment the sequence by flipping twice to extend the length of the sequence to 28 as input, and the inference results on **Vimeo-90K-T** [11] are the average of output frames derived from the same frame extension.

The Charbonnier penalty loss [4] is applied on whole sequence between the ground-truth and restored SR frame. It can be formualted as:

$$ \ell = \frac{1}{T}\sum_{t=1}^{T}\sqrt{\|I^t_{HR} - I^t_{SR}\|^2 + \varepsilon^2}, \tag{4} $$

where $\varepsilon = 1\times10^{-8}$. T denotes the sequence length. All

Table 2. Quantitative comparison (PSNR↑, SSIM↑, and LPIPS↓) on the REDS4 [9] dataset for $4\times$ video super-resolution. The results are tested on RGB channels. Red indicates the best and blue indicates the second best performance (best view in color).

Method	Bicubic	RCAN [14]	CSNLN [8]	TOFlow [11]	DUF [3]	EDVR [10]	MuCAN [5]	BasicVSR [1]	IconVSR [1]	**TTVSR**
PSNR	26.14	28.78	28.83	27.98	28.63	31.09	30.88	31.42	31.67	32.12
SSIM	0.7292	0.8200	0.8196	0.7990	0.8251	0.8800	0.8750	0.8909	0.8948	0.9021
LPIPS	0.3395	0.2716	0.2668	0.2969	0.2911	0.2225	0.2112	0.1979	0.1890	0.1786

experiments are conducted on a server with Python 3.9, PyTorch 1.9, and $8\times$NVIDIA Tesla V100 GPUs.

4. Complexity Analysis

By introducing trajectories into Transformer in our proposed TTVSR, the attention calculation on $\mathcal{K}$ and $\mathcal{V}$ can be significantly reduced because it can avoid the computation on spatial dimension compared with vanilla vision Transformers. In this section, we analyze the complexity of trajectory-aware attention calculation in detail.

Take the attention process of one token in $\mathcal{Q}$ as an example, when a sequence of length T and size $C \cdot H \cdot W$ is input as $\mathcal{K}$, the size of tokens and the number of tokens can be expressed as $C \cdot D_h \cdot D_w$ and $T \cdot \frac{H}{D_h} \cdot \frac{W}{D_w}$, respectively. The similarity of attention mechanisms in vanilla vision Transformer has a computational cost of:

$$(T \cdot \frac{H}{D_h} \cdot \frac{W}{D_w}) \cdot (C \cdot D_h \cdot D_w). \tag{5}$$

The similarity of trajectory-aware attention mechanisms in TTVSR has a computational cost of:

$$(T \cdot 1 \cdot 1) \cdot (C \cdot D_h \cdot D_w), \tag{6}$$

here, the attention calculation on spatial dimension is avoided, so the number of tokens in the attention calculation is reduced from $(T \cdot \frac{H}{D_h} \cdot \frac{W}{D_w})$ to $(T \cdot 1 \cdot 1)$.

By expressing attention as the multiplication of tokens, we get a reduction in computation of:

$$\frac{(T \cdot 1 \cdot 1) \cdot (C \cdot D_h \cdot D_w)}{(T \cdot \frac{H}{D_h} \cdot \frac{W}{D_w}) \cdot (C \cdot D_h \cdot D_w)} = \frac{1}{(\frac{H}{D_h} \cdot \frac{W}{D_w})}. \tag{7}$$

In general, by introducing trajectories into Transformer, the computational complexity of attention is reduced by $(\frac{H}{D_h} \cdot \frac{W}{D_w})$ times, and provides a more efficient way to enable our TTVSR can directly leverage the information from a distant video frame.

5. Limitation Analysis

In this section, we discuss the limitations of TTVSR.

Length of sequence. The design of TTVSR is to fully utilize the temporal information of more frames in the sequence, so the performance improvement is limited for shorter sequences. (e.g., each sequence contains seven frames on Vimeo-90K [11]). Therefore, to enable long-range sequence capability, for **Vimeo-90K** [11], we augment the sequence by flipping twice to extend the length of the sequence to 28 as input. However, due to the limitation of original sequences, the improvement is still limited. We will add relevant features on spatial dimensions to further enhance the utilization of spatial information in the model.

Training Time. To make the model have the ability to model long-range sequences, we input the sequences as long as possible during training. However, longer sequences often lead to longer training times. Next, we will optimize the training process so that TTVSR can speed up the training process with long sequences input.

GPU memory. According to the model design, TTVSR needs to store the features of each moment during inferring and training. It inevitably takes up GPU memory. Therefore, the next step is to design an adaptive feature storage mechanism dynamically and selectively retain the useful features for reconstruction.

6. More Results

In this section, to further verify the effectiveness of our method, we compare the perceptual results and show more comparison results among the proposed TTVSR and other advanced methods on four different benchmarks.

Perceptual results. We use LPIPS [13] as a widely used metric to evaluate perceptual quality. Results, shown in the following Additional Tab. 2, demonstrate that TTVSR is still highly superior in the perceptual metrics.

Visualization results. We show more comparison results among the proposed TTVSR and other advanced methods on four different benchmarks. The results on **REDS4** [9], **Vid4** [7], **UDM10** [12] and **Vimeo-90K-T** [11] are shown in Fig. 2-3, Fig. 4, Fig. 5 and Fig. 6-7, respectively.

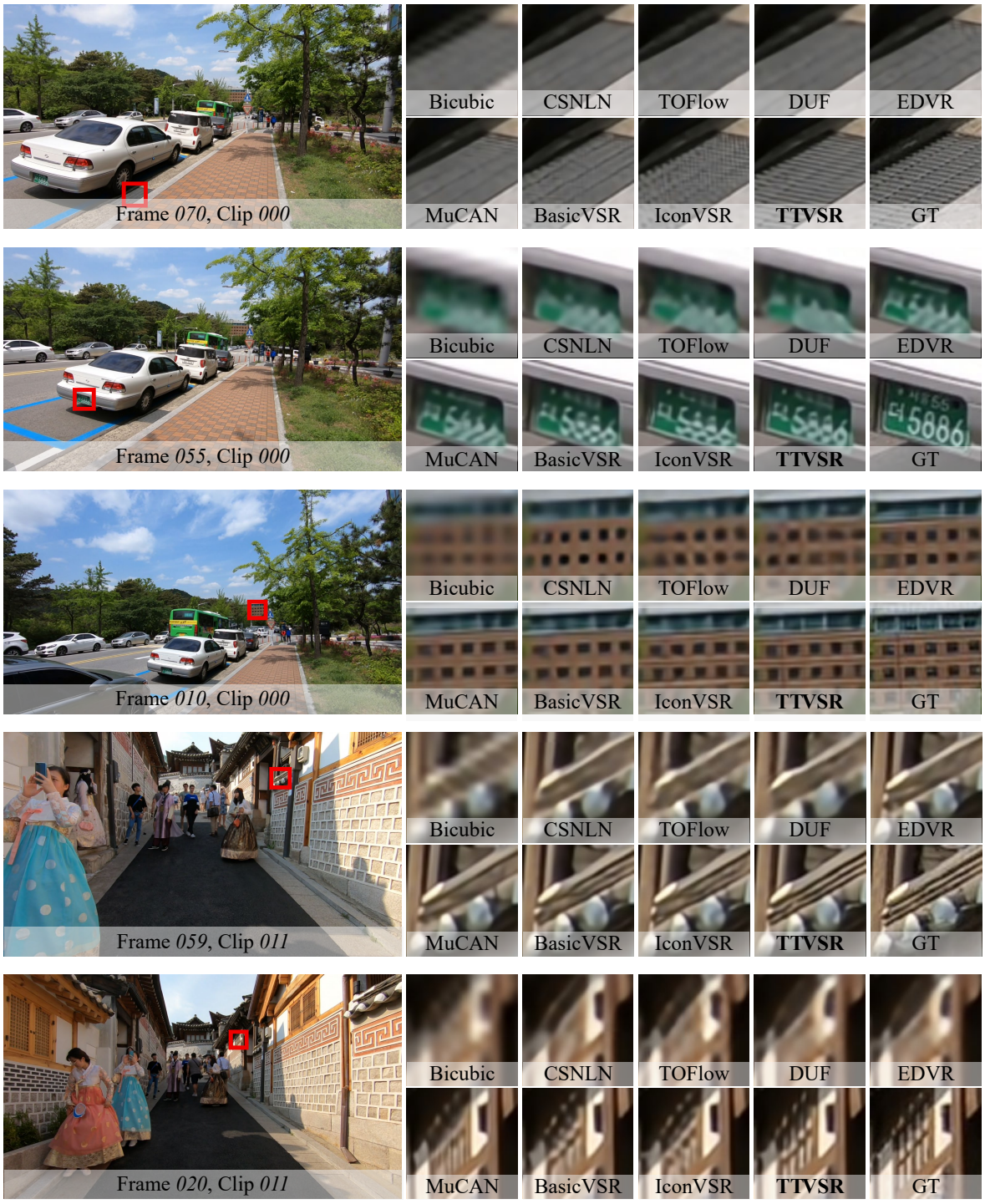

Figure 2. Visual results on REDS4 [9] for $4\times$ scaling factor. The frame number is shown at the bottom of each case. Zoom in to see better visualization.

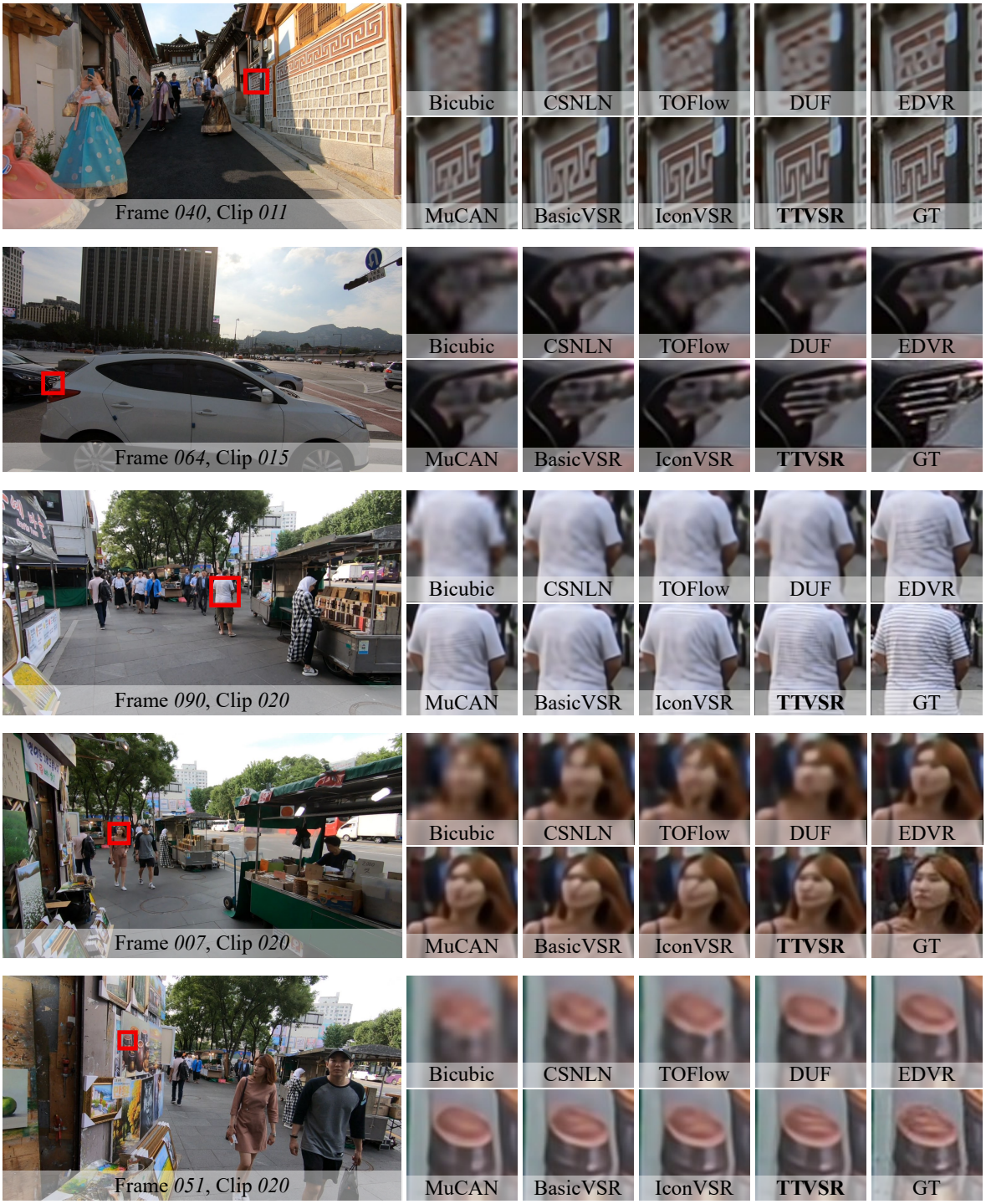

Figure 3. Visual results on REDS4 [9] for $4\times$ scaling factor. The frame number is shown at the bottom of each case. Zoom in to see better visualization.

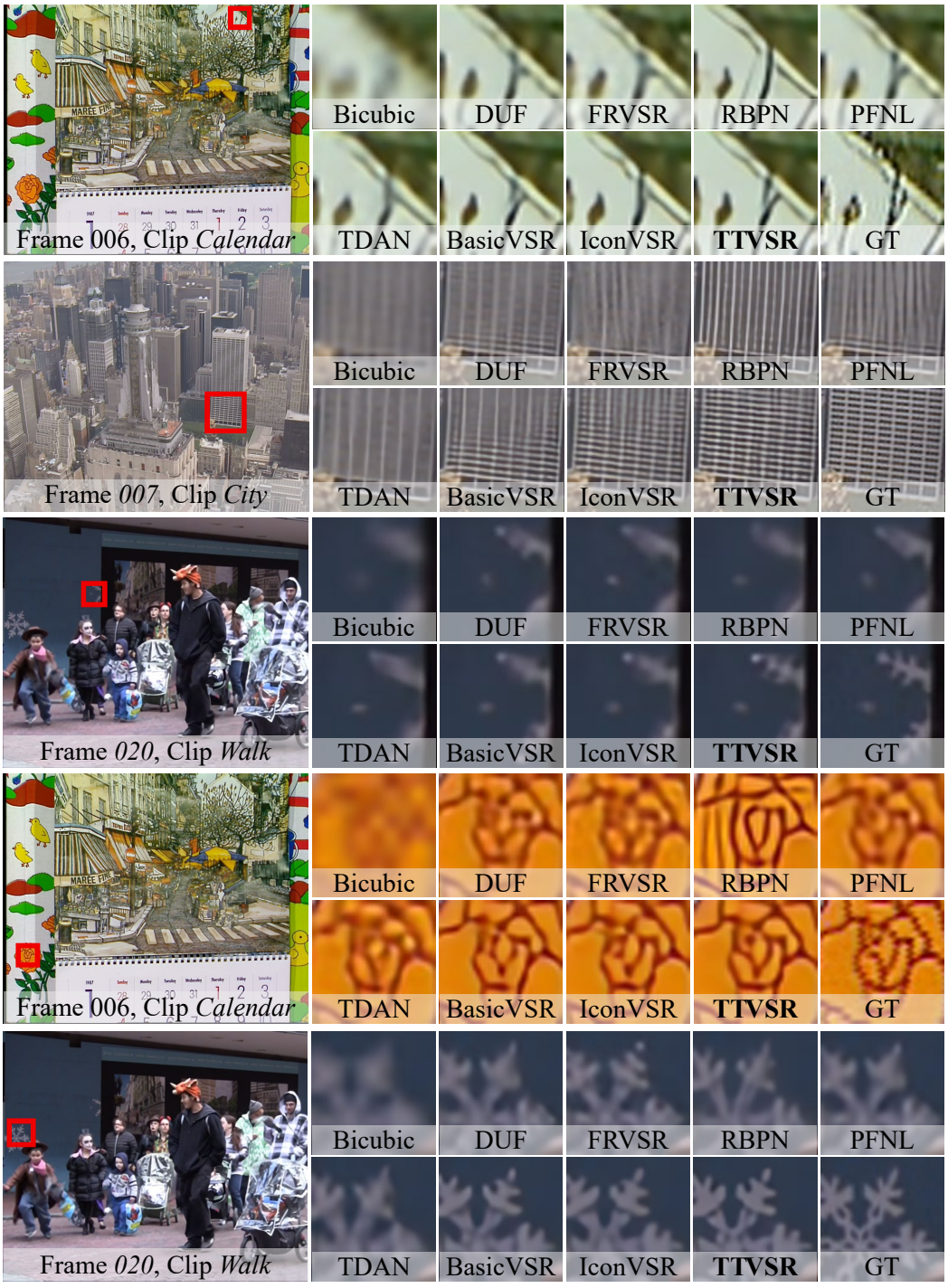

Figure 4. Visual results on Vid4 [7] for $4\times$ scaling factor. The frame number is shown at the bottom of each case. Zoom in to see better visualization.

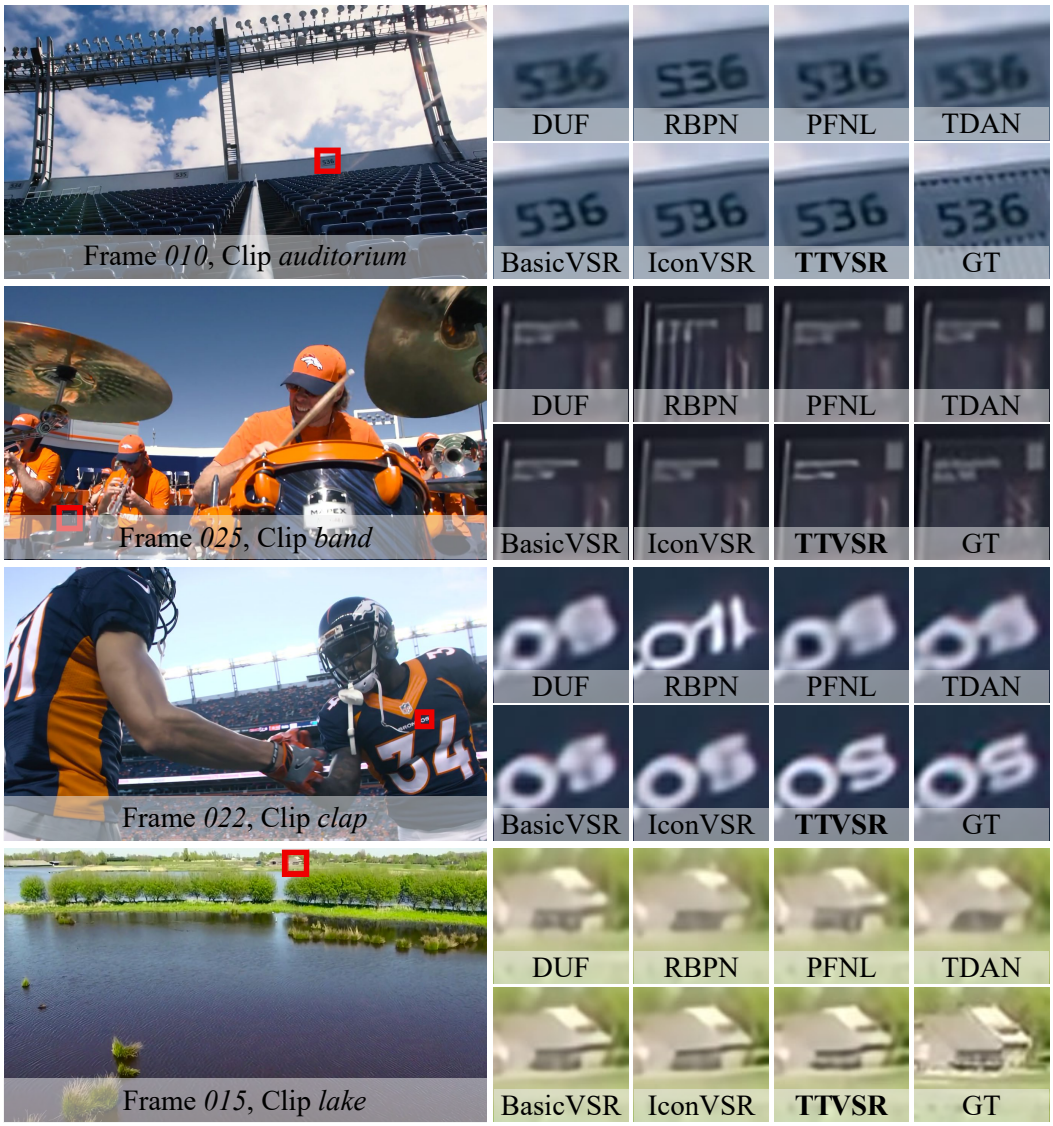

Figure 5. Visual results on UDM10 [12] for $4\times$ scaling factor. The frame number is shown at the bottom of each case. Zoom in to see better visualization.

Figure 6. Visual results on Vimeo-90K-T [11] for $4\times$ scaling factor. The frame number is shown at the bottom of each case. Zoom in to see better visualization.

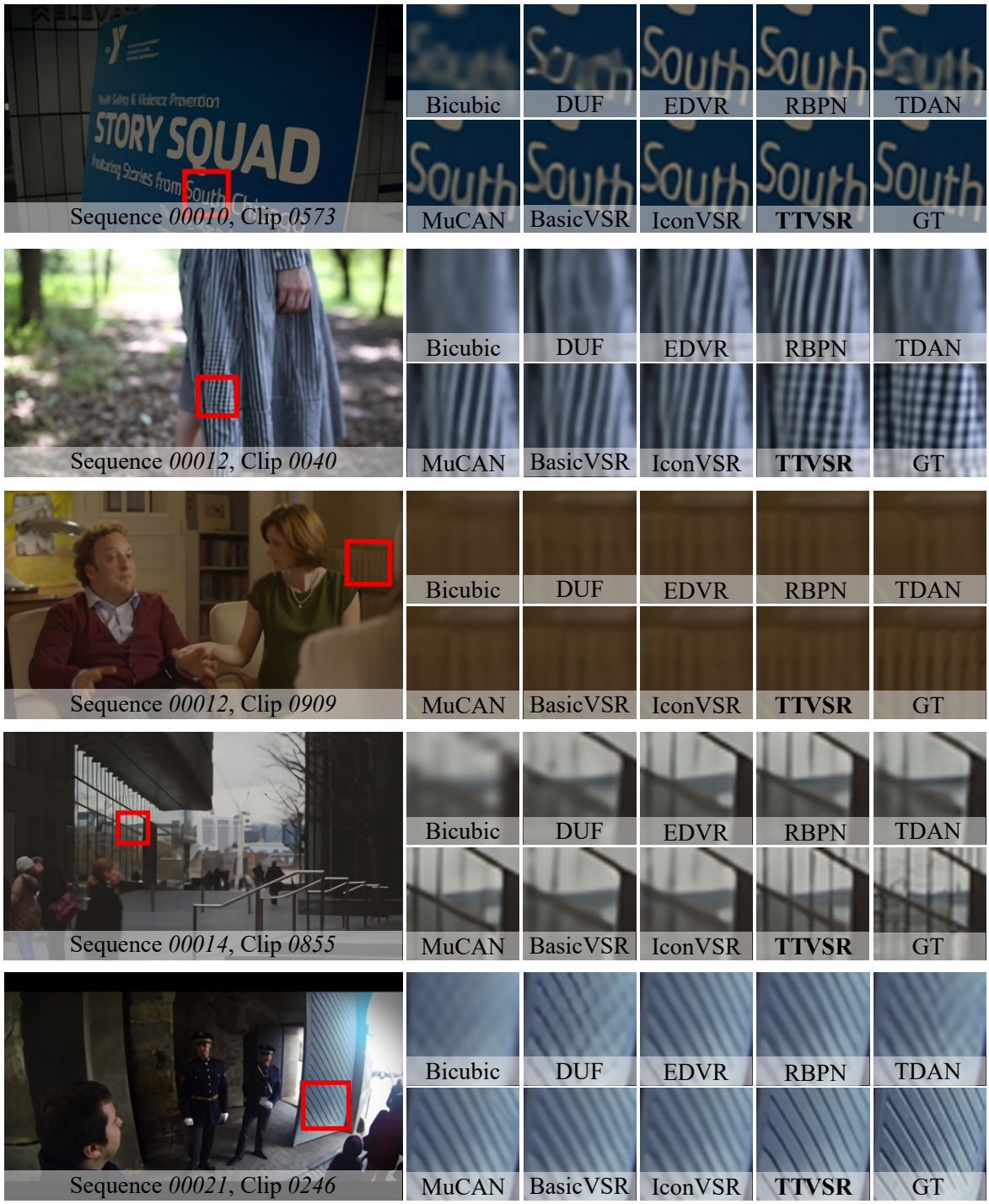

Figure 7. Visual results on Vimeo-90K-T [11] for $4\times$ scaling factor. The frame number is shown at the bottom of each case. Zoom in to see better visualization.

References

[1] Kelvin CK Chan, Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. BasicVSR: The search for essential components in video super-resolution and beyond. In *CVPR*, pages 4947–4956, 2021. 1, 2, 3

[2] Yan Huang, Wei Wang, and Liang Wang. Video super-resolution via bidirectional recurrent convolutional networks. *IEEE TPAMI*, 40(4):1015–1028, 2017. 1, 2

[3] Younghyun Jo, Seoung Wug Oh, Jaeyeon Kang, and Seon Joo Kim. Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation. In *CVPR*, pages 3224–3232, 2018. 3

[4] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *CVPR*, pages 624–632, 2017. 2

[5] Wenbo Li, Xin Tao, Taian Guo, Lu Qi, Jiangbo Lu, and Jiaya Jia. MuCAN: Multi-correspondence aggregation network for video super-resolution. In *ECCV*, pages 335–351, 2020. 2, 3

[6] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, pages 2117–2125, 2017. 1

[7] Ce Liu and Deqing Sun. On bayesian adaptive video super resolution. *IEEE TPAMI*, 36(2):346–360, 2013. 2, 3, 6

[8] Yiqun Mei, Yuchen Fan, Yuqian Zhou, Lichao Huang, Thomas S Huang, and Honghui Shi. Image super-resolution with cross-scale non-local attention and exhaustive self-exemplars mining. In *CVPR*, pages 5690–5699, 2020. 3

[9] Seungjun Nah, Sungyong Baik, Seokil Hong, Gyeongsik Moon, Sanghyun Son, Radu Timofte, and Kyoung Mu Lee. NTIRE 2019 challenge on video deblurring and super-resolution: Dataset and study. In *CVPRW*, pages 0–0, 2019. 2, 3, 4, 5

[10] Xintao Wang, Kelvin CK Chan, Ke Yu, Chao Dong, and Chen Change Loy. EDVR: Video restoration with enhanced deformable convolutional networks. In *CVPRW*, 2019. 3

[11] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *IJCV*, 127(8):1106–1125, 2019. 2, 3, 8, 9

[12] Peng Yi, Zhongyuan Wang, Kui Jiang, Junjun Jiang, and Jiayi Ma. Progressive fusion video super-resolution network via exploiting non-local spatio-temporal correlations. In *ICCV*, pages 3106–3115, 2019. 2, 3, 7

[13] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, pages 586–595, 2018. 3

[14] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *ECCV*, pages 286–301, 2018. 3

[15] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, pages 2881–2890, 2017. 1