LDIP: Long Distance Information Propagation for Video Super-Resolution

Supplementary Material

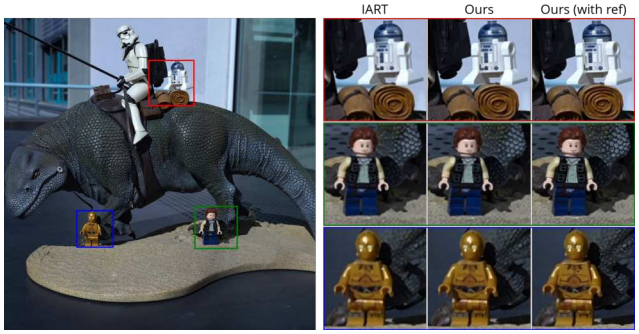

Figure 6. Visual example illustrating our methods capability to optionally inject information from available reference images. In the input video the camera pans from left to right. Even without reference images our method manages to reconstruct more fine details than IART. Injecting additional high-quality reference images allows our method to further increase visual quality.

	MRefSR	Ours	w/o ref	w/o ICA
PSNR ↑	30.98	32.55	32.03	32.54
LPIPS ↓	0.172	0.102	0.111	0.107

Table 5. Evaluation on frames 20 to 80 of the REDS evaluation dataset. Frames 0 and 99 are available in high-resolution as reference images. We include ablation of the Implicit Cross Attention. Our results were generated using the BasicVSR++ backbone.

6. Additional Results

In Figure 6 we showcase our methods ability to optionally use high quality reference images if they are available. The first two columns show the results produced by IART and our method only using the input video without any reference images. With its improved long range temporal propagation our method already produces slightly higher quality results. Using a small number (5) of high quality reference images depicting the same scene, our method is able to inject information throughout the sequence which results in significantly higher quality outputs.

In Table 5 we further illustrate the effectiveness of using high quality reference images numerically. Here, we upscale frames 20 to 80 from each sequence in the REDS test dataset while providing frames 0 and 99 as HR references. We add MRefSR [30] as a comparison in this scenario as, to the best of our knowledge, it is the only existing method applicable to this scenario. The results clearly show that using high quality reference images significantly improves quality. We also take this opportunity to ablate the effectiveness of our ICA module. In the last column (w/o ICA) we test our method where the ICA module is replaced by standard cross attention. We see that ICA improves improves both PSNR and LPIPS scores.

7. Architecture Details

In this section we give a detailed description of our methods architecture.

7.1. Reference Feature Extractor

Our **Ref-Align** module makes use of a feature extraction network to extract a feature map from a given reference image. We implement this network as a single convolution going from 3 to 64 channels followed by two residual dense blocks each using 64 intermediate channels, a growth rate of 16 and a depth of 4. The extracted feature map is then aligned to the anchor feature map by warping it according to W. Warping is performed similarly as in the arbitrary scaling module. That is, the feature map is warped using nearest neighbor sampling and the sampling offsets are appended along the channel dimension.

7.2. Multi Reference Fusion (Ref-Fusion) Module

The **Ref-Fusion** module takes a query feature map F^Q and an arbitrary number of KV feature maps $F_1^{KV}, \cdots F_N^{KV}$ as input and returns a refined version of the query feature map F^Q. Here, $F^Q \in \mathbb{R}^{C_q \times H \times W}$ corresponds to the HR feature map that is being refined and $F_i^{KV} \in \mathbb{R}^{C_{kv} \times H \times W}$ corresponds to the aligned reference feature map produced by the RA module from the reference image I_i^{ref}. The **Ref-Fusion** module is parametrized by the C_q, C_{kv}, the number of attention heads (H_n) and their size H_s, and depth D. In our method we set $C_q = 64$, $C_{kv} = 64 + 2 + 1$, $H_n = 4$, and $H_s = 16$. The **Ref-Fusion** module operates as follows.

$F'^Q \leftarrow F^Q$
$F^Q \leftarrow \text{Conv2d}_{qin}(F^Q)$
for $i \in [1, N]$ **do**
  $F_i^{KV} \leftarrow \text{Conv2d}_{kin}(F_i^{KV})$
end for
for $i \in [1, D]$ **do**
  $\hat{F}^Q \leftarrow \text{ICA}_i(F^Q, F_1^{KV}, \cdots, F_N^{KV})$
  $F^Q \leftarrow \text{fusion}_i(\text{cat}(F^Q, \hat{F}^Q))$
end for
$F^Q \leftarrow F'^Q + \text{Conv2d}_r(F^Q)$

Here, Conv2d_{qin} / Conv2d_{kin} are convolutions going from C_q / C_{kv} to $E = H_n \cdot H_s$ channels. Conv2d_r is a convolution going from C_q to C_q channels and is initialized with zeros. Each fusion_i is a chain of 3 convolutions going

from $2E$ to $4E$ to E channels with LeakyReLU activation between the layers. Finally, each Implicit Cross Attention (ICA_i) layer uses H_n attention heads each with size H_s.

Implicit Cross Attention (ICA). The ICA operates as follows. Note that everything in the ICA operates in a pixel-wise manner.

$F^Q \leftarrow \text{layer\_norm}(F^Q)$
for $i \in [1, N]$ **do**
  $F_i^{KV} \leftarrow \text{MLP}(\text{cat}(F^Q, F_i^{KV}))$
  $F_i^{KV} \leftarrow \text{layer\_norm}(F_i^{KV})$
end for
$F^Q \leftarrow \text{mha}(F^Q, F_1^{KV}, \cdots, F_N^{KV})$

Here, the MLP has 1 hidden layer of size $4E$ and uses LeakyReLU activation. By mha we refer to standard multi-head attention using H_n attention heads each with size H_s. Again note that the mha operates for each spatial location independently.

7.3. High-Resolution Iterative Propagation (HRIP)

We design our HRIP module similarly to the LRIP modules used by IART and BasicVSR++. To reduce computational complexity we only perform one forward and one backward pass (instead of two each) and we use first-order instead of second-order propagation. Feature maps from the previous frame are aligned to the current feature map using nearest neighbor sampling and appending the sampling offset to the warped feature map. The current feature map and the warped previous feature map are concatenated along the channel dimension and then passed to a SwinTransformer network to compute a residual to the current feature map.

7.4. Arbitrary Scaling

We implement the neural network $\mathcal{N}$ in our arbitrary scaling layer using two sequential residual dense blocks. Both RDBs operate on 64 channels with a growth rate of 16 and a depth of 4.

7.5. Number of Parameters

In total our LRRF module adds an additional 2.1 million trainable parameters on top of the VSR backbone. For comparison, IART consists of 13 million parameters.

8. Training Details

In this section we present further details regarding our methods training procedure.

8.1. Fixed $\times 4$ VSR

When training our method for fixed $\times 4$ scaling we start from an existing VSR method (the VSR backbone) and train our LRRF module on top of it. During training all parameters of the VSR backbone are kept frozen and only parameters in the LRRF module are trained. The parameters of the optical flow method required by the HRIP and the correspondence estimation module required by the RA module are also kept frozen during training. The remaining parameters in the LRRF module are trainable.

Training Data. To train our method we generate training examples consisting of a LR video clip, a number of reference images, and a corresponding HR video clip. We start by randomly selecting a HR video clip of length 24 from our training dataset and refer to this clip as C^{ref}. From C^{ref} we randomly select a subclip of length 12 and refer to this as C^{gt}. We then apply bicubic downscaling by $\times 4$ to C^{gt} to generate C^{lr}. To generate reference images we first select between 1 and 3 frames in C^{lr} randomly and designate them as key-frames. For each key-frame we then generate between 1 and 5 reference images. Each reference image is generated by selecting a random frame in C^{ref} and two random scaling factor $s_1 \in [1, 4]$, $s_2 \in [1, s_1]$. The selected frame is then first downscaled by s_1 and then upscaled by s_2 to generate the reference image.

Training Parameters. For this task we train our method on the REDS dataset using ground truth crops of size 256×256. We use the AdamW optimizer and we use cosine annealing to decay the learning rate from $2 \cdot 10^{-4}$ to 10^{-6}. During training we use the L1 loss to get a training signal. When using BasicVSR++ as a backbone we perform 100k training step using a batch size of 2. Training for this version is performed on 2 RTX4090 GPUs and took 1 day and 18 hours. When using the IART backbone we perform 150k training steps using a batch size of 4. Training for this version is performed on 4 RTX4090 GPUs and took 3 days and 15 hours. For comparison, IART was trained on 8 V100 GPUs for 4 weeks.

8.2. Arbitrary Scaling VSR

Training our method for arbitrary scaling VSR is performed in two steps.

Initial AS-VSR training. Starting from the backbone VSR method we first replace the upscaling layer and image decoder. These new modules are then trained on REDS using clips of length 12. During training we keep the size of the LR video fixed to 64×64 and adjust the GT video size according to the chosen scaling factor. The scaling factor is chosen randomly between 1.5 and 4.5. We use the AdamW optimizer and we use cosine annealing to decay the learning rate from $2 \cdot 10^{-4}$ to 10^{-6}. During training we use the L1 loss to get a training signal. Independent of the backbone VSR method used we perform 100k training steps. When

Resolution	2 References	4 References	8 References
640×360	0.2s / 1.0GB	0.3s / 1.6GB	0.6s / 2.8GB
1280×720	0.7s / 3.6GB	1.1s / 5.8GB	1.9s / 10.7GB
1920×1080	1.5s / 8.0GB	2.4s / 12.9GB	4.2s / 23.8GB
2560×1440	2.6s / 14.1GB	4.3s / 22.8GB	7.5s / 42.2GB

Table 6. LRRF module Runtime and peak GPU memory consumption with respect to image resolutions and number of references (on an RTX A6000).

using BasicVSR++ as a backbone we use a batch size of 2 and the model is trained on 2 RTX4090 GPUs for 14 hours. When using IART as a backbone we use a batch size of 4 and the model is trained on 4 RTX4090 GPUs for 22 hours.

LRRF training. Starting from the trained arbitrary scaling version of BasicVSR++ and IART we train our LRRF module on top. Training is performed identically to the fixed $\times 4$ VSR case but with varying scaling factors. As for the initial AS-VSR training we choose random scaling factors between 1.5 and 4.5. Here, however, we keep the ground truth crop size fixed to 256×256 and vary the low-resolution size according to the scaling factor. Independent of the backbone used we train our method for 100k steps using a batch size of 2 on 2 RTX4090 GPUs. Training using the BasicVSR++ backbone takes 1 day and 20 hours, while training with IART as a backbone takes 3 days and 10 hours.

LRRF Computational Cost In Table 6 we present an evaluation of our proposed LRRF modules computational cost across different resolutions and number of reference images.