Efficient Feature Extraction for High-resolution Video Frame Interpolation

Supplemental Material

Moritz Nottebaum[1]
moritz.nottebaum@stud.tu-darmstadt.de
Stefan Roth[1,2]
stefan.roth@visinf.tu-darmstadt.de
Simone Schaub-Meyer[1,2]
simone.schaub@visinf.tu-darmstadt.de

[1] Department of Computer Science
TU Darmstadt
[2] hessian.AI

A Overview

This appendix provides additional details related to training and the architectural setup for reproducibility purposes, which were omitted in the main paper due to space limitations. We further report evaluations with additional metrics and visualizations of the computed flow. The supplemental video[1] contains additional, temporal visual results and comparisons.

B Training Details

We give here the full formulas for the used loss functions. The total loss $\mathcal{L}_{total}$ in Eq. (6) consists of

$$\mathcal{L}_{recon} = \sum_{s=0}^{S} \|\hat{I}_t^s - I_t^s\|_1 \tag{7}$$

$$\mathcal{L}_{smooth} = \sum_{(i,j)\in\{(0,1),(1,0)\}} \exp\left(-e^2 \sum_c (\nabla_x I_{i_c})^2\right)^{\mathsf{T}} \cdot |\nabla_x F_{i\to j}^0| \tag{8}$$

$$\mathcal{L}_{warp} = \left(\|\overrightarrow{\omega}(I_0^0, F_{0\to1}) - I_1^0\|_1 + \|\overrightarrow{\omega}(I_1^0, F_{1\to0}) - I_0^0\|_1\right) , \tag{9}$$

where s, c, e, x, $\lambda_\cdot$, and $\overrightarrow{\omega}(\cdot,\cdot)$ define the scale, the channel index, an edge weighting factor, a spatial coordinate, the weighting factors, and the forward warping function, respectively. The values for the hyperparameters are given in Tab. 4.

As mentioned in the occlusion estimation in Sec. 4, we apply the softmax function along the last dimension of the weighting map, creating a probability distribution for every output pixel. Using the softmax function allows for temperature scaling [8], for which we divide the

[1] Available at https://github.com/visinf/fldr-vfi.

S_{train}	e	λ_{smooth}	λ_{warp}
3	150	0.125	0.5

Table 4: Overview of the used hyperparameters.

Layer name	Input	# Channels In/Out	Reuse weights
conv2d_1	fLDR^s	96/96	False
conv2d_2	conv2d_1	96/96	False
conv2d_3	conv2d_2 + fLDR^s	96/96	False
conv2d_4	conv2d_3	96/96	False
conv2d_5	conv2d_4	96/96	False
conv2d_6	conv2d_5	96/48	False
conv2d_7	conv2d_6	48/4	False

Table 5: Layers of the flow estimation network at the coarsest scale $s = S$. The column "Reuse weights" indicates if the parameters of a layer are reused from another layer. The output of the last layer is the bidirectional flow $[F^S_{0\to1}, F^S_{1\to0}]$.

estimated occlusion map M by the scalar temperature parameter T before taking the softmax. This allows to soften the distribution or make it more peaked, depending on whether T its greater or smaller than 1. After training the full pipeline with $\mathcal{L}_{total}$ and $T = 1$, we additionally end-to-end optimize for T while keeping all other parameters fixed. We finetune T with a learning rate of 10^{-3} using the mean-squared-error of the image reconstruction loss for 10 epochs.

C Network Architecture

Tabs. 5 and 6 list the details of our flow estimation network for the coarsest scale and all higher levels, respectively. After each convolutional layer, ReLU nonlinearities [45] are applied except for conv2d_7 in Tab. 5, and conv2d_3_i and conv2d_8 in Tab. 6. The kernel size of each convolutional layer is 3×3. Layer conv2d_1 and conv2d_2 in Tabs. 5 and 6 share their parameters, as well as conv2d_3_0 and conv2d_3_1 in Tab. 6. $\text{fLDR}^s = [\text{fLDR}^s_0, \text{fLDR}^s_1]$ is the result of our finetuned linear dimensionality reduction (fLDR) of image I_0 and I_1 at scale s, *i. e.* $[\tilde{I}^s_0, \tilde{I}^s_1]$.

Tab. 6 shows the architecture of our flow estimation network for $s < S$, with feat_0 and feat_1 defined as follows:

$$\text{feat}_0 : \left[(\text{fLDR}^s_0 + \text{conv2d\_2}_0), \overrightarrow{\omega}(\text{fLDR}^s_0 + \text{conv2d\_2}_0, up(F^{s+1}_{0\to1}))\right] \tag{10}$$

$$\text{feat}_1 : \left[(\text{fLDR}^s_1 + \text{conv2d\_2}_1), \overrightarrow{\omega}(\text{fLDR}^s_1 + \text{conv2d\_2}_1, up(F^{s+1}_{1\to0}))\right] , \tag{11}$$

where $F^{s+1}_{0\to1}$ and $F^{s+1}_{1\to0}$ are the estimated, bidirectional flows from the previous, coarser scale and conv2d_2_0 as well as conv2d_2_1 are the outputs of the convolutional layer conv2d_2, split along the channel dimension, such that they represent the features of input image I_0 and I_1, respectively. The flow has been upscaled ($up(\cdot)$) bilinearly.

The configuration of the occlusion estimation network is listed in Tab. 7. After each convolutional layer, ReLU nonlinearities [45] are applied except for layer dec_3.

Layer name	Input	# Channels In/Out	Reuse weights
conv2d_1	fLDR^s	96/96	conv2d_1 (Tab. 5)
conv2d_2	conv2d_1	96/96	conv2d_2 (Tab. 5)
conv2d_3_0	feat_0	96/48	False
conv2d_3_1	feat_1	96/48	conv2d_3_0
conv2d_4	$[\text{conv2d\_3}_0, \text{conv2d\_3}_1, up(F^{s+1})]$	100/96	False
conv2d_5	conv2d_4	96/96	False
conv2d_6	conv2d_5	96/48	False
conv2d_7	conv2d_6	48/48	False
conv2d_8	conv2d_7	48/4	False

Table 6: Layers of the flow estimation network at all scales except the coarsest, *i. e.* $s < S$. The column "Reuse weights" indicates if the parameters of a layer are reused from another layer. The output of the last layer is added to the upsampled flow F^{s+1} of the previous scale, which results in the final bidirectional flow $F^s = [F^s_{0\to1}, F^s_{1\to0}]$ of scale s.

Layer name	Filter Size	# Input Filters	# Output Filters
enc_0	4×4	26	16
enc_1	4×4	16	32
enc_2	4×4	32	64
dec_0	3×3	64	64
dec_1	3×3	$64+32$	32
dec_2	3×3	$32+16$	16
dec_3	3×3	16	6

Table 7: Layers of the occlusion estimation network. The output of enc_2 is additionally fed into dec_1. The output of enc_1 is additionally fed into dec_2. The outputs of dec_0, dec_1, dec_2, and dec_3 are upscaled with nearest neighbor upsampling and a scale factor of 2 before feeding them into the respective next layer.

D Quantitative Results

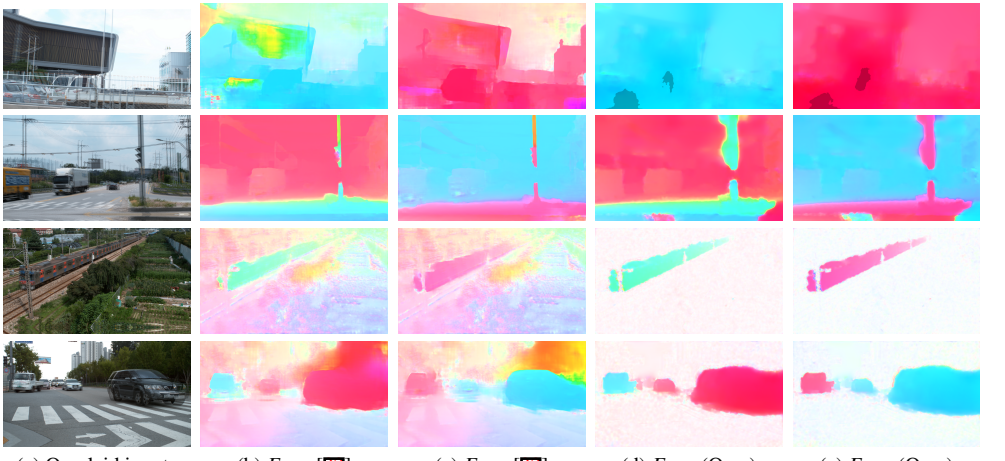
In Tab. 8 we provide, in addition to the PSNR values in Tab. 2, a quantitative analysis with SSIM [39], LPIPS [44], and inference time, where possible. We can only measure the inference time for models where we have the code and which are running on our Nvidia 3080Ti (12GB) GPU. Unfortunately, XVFI with $S = 5$ scales run out of memory on our GPU with 12 GB and the error metric computation has been computed on CPU. We, therefore, provide the inference time only for $S = 3$ (‡). We did not optimize our method for inference time. Nevertheless, we obtain inference times comparable to most of the other methods. M2M-PWC downscales the input image first by a factor of $1/16$, leading to faster inference.

E Optical Flow Visualizations

In Fig. 6 we show some visualizations of our predicted flow in comparison to the computationally more expensive pretrained PWC-Net [32] often used in frame interpolation methods [9, 21]. However, the flow shown here, is computed with the original PWC-Net without yet finetuning for the task of frame interpolation as this depends on the used frame interpolation method.

	Xiph-4K	X-Test	Inter4K-S	Inter4K-L	Inference (in s/f)
M2M-PWC [9]	**0.949**/0.219	**0.914**/0.086	**0.942/0.076**	0.883/0.145	**0.21**
RIFE$^{\diamond}_{m}$ [10]	0.910/0.171	0.793/0.227	0.894/0.117	0.826/0.196	0.40
RIFE$_{m}$ [10]	0.904/0.228	0.751/0.260	0.893/0.123	0.829/0.197	0.40
XVFI$^{\diamond}$ [28]	0.910/0.175	0.874/**0.085**	0.915/0.085	0.850/0.145	0.66‡/ N/A
Ours	0.913/**0.144**	0.871/0.099	0.917/0.084	**0.904/0.139**	0.50‡/0.51

Table 8: Extension of Tab. 2 with (SSIM [39]/LPIPS [44]) and inference time in s for a 4K image (2160×4096) on a Nvidia 3080Ti GPU.



(a) Overlaid inputs (b) $F_{0\to1}$ [32] (c) $F_{1\to0}$ [32] (d) $F_{0\to1}$ (Ours) (e) $F_{1\to0}$ (Ours)

Figure 6: **Flow visualization.** Comparison of computed flow between ours (smaller network, flow for frame interpolation) and the pretrained PWC-Net [32].

References

[45] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.