Efficient Feature Extraction for High-resolution Video Frame Interpolation

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

Abstract

Most deep learning methods for video frame interpolation consist of three main components: feature extraction, motion estimation, and image synthesis. Existing approaches are mainly distinguishable in terms of how these modules are designed. However, when interpolating high-resolution images, *e.g.* at 4K, the design choices for achieving high accuracy within reasonable memory requirements are limited. The feature extraction layers help to compress the input and extract relevant information for the latter stages, such as motion estimation. However, these layers are often costly in parameters, computation time, and memory. We show how ideas from dimensionality reduction combined with a lightweight optimization can be used to compress the input representation while keeping the extracted information suitable for frame interpolation. Further, we require neither a pretrained flow network nor a synthesis network, additionally reducing the number of trainable parameters and required memory. When evaluating on three 4K benchmarks, we achieve state-of-the-art image quality among the methods without pretrained flow while having the lowest network complexity and memory requirements overall.

1 Introduction

Video frame interpolation (VFI) is one of the classic problems in video processing. Generating intermediate frames in a video sequence can be used for various applications, *e.g.*, video compression [6, 41], video editing [18], animation [3, 14], and event cameras [36, 37].

Deep learning-based approaches for VFI [11, 15, 25, 28] commonly consist of three modules: feature extraction, motion estimation, and image synthesis, which can be realized and combined in various forms. The higher the input image resolution is, however, the more important an efficient feature representation becomes to allow the subsequent modules to handle the large input space within reasonable memory constraints. Convolutional neural networks (CNNs) can reduce the spatial dimensions with an encoder network. This still requires initially applying the convolutional filters at the full image resolution, however. We suggest instead to first compress the images with a block-wise transformation inspired by

© 2022. The copyright of this document resides with its authors.
It may be distributed unchanged freely in print or electronic forms.

Code and additional resources (test set and supplemental material) are available at https://github.com/visinf/fldr-vfi

linear dimensionality reduction methods such as principal component analysis (PCA) [7]. PCA is known to efficiently represent image data, but is not directly suitable as a representation for neural networks as it projects each input image to a different low-dimensional space. We propose a method that finetunes an initial block-based PCA basis end-to-end for video frame interpolation. This has two advantages: First, it results in one general projection space for all images and resolutions. Second, it allows us to optimize the representation for the targeted task of video frame interpolation.

Our main contribution is a deep learning architecture for video frame interpolation that first applies linear dimensionality reduction to efficiently compress and represent images. Our approach is carefully designed to learn from the projected data as well as avoids a memory and computation intensive synthesis network in favor of a learned weighting of the (warped) input images. As a result, the overall complexity of the network architecture can be significantly reduced, which makes it particularly suitable for high-resolution videos and limited hardware resources. To evaluate the generalization of our approach to various video scenes, we introduce a novel 4K test set, with an order of magnitude more video scenes than existing ones. We achieve highly competitive results and even outperform existing methods for larger motions across various benchmarks with only a fraction of the network complexity.

2 Related Work

Feature extraction. Most deep learning-based VFI methods encode the input with CNNs due to their general suitability for feature representation [13, 29]. However, for compressing images and videos, algorithms used in standard codecs [33, 38, 40] still rely on mathematical transformations like the discrete cosine transform (DCT). The advantage of well defined mathematical transformations like DCT, PCA, and DFT (discrete Fourier transformation) is that they are comparably cheap to compute while also entailing a good compression ratio. Some exemplary work showed their use in neural networks in different scenarios, *e.g.*, DFT [19, 27] and PCA [2]. The advantage of CNNs is their task-specific adaptability, while PCA dimensionality reduction only adapts to the input data, not the task. Here, we show how we can still use the benefits of dimensionality reduction in a neural network architecture.

Video frame interpolation. Generating intermediate frames between two frames requires some understanding of the temporal relationship between the input frames. We can generally categorize VFI methods in terms of whether they estimate the motion explicitly, *e.g.* optical flow, or implicitly. VFI methods representing the motion implicitly either directly estimate the intermediate frame [16], use phase-based [17, 19] or a kernel-based [4, 22, 23] representation, or apply PixelShuffle [5]. However, most of them are limited to generating a single intermediate frame. Kernel-based methods are further limited, due to the kernel size, in the amount of pixel displacement they can handle. This makes them unsuitable for high-resolution imagery.

Traditionally, explicit methods for VFI combine optical flow estimation [1, 31] and correspondence-based image warping [1]. As a result, they heavily depend on the quality of the optical flow. Thus, deep learning-based methods using flow estimation often not only estimate the bi-directional flow between the input images but further refine the intermediate flow vectors [11, 28] or the output with a synthesis network [20] to address inaccuracies. Two recent advances have enabled the current state-of-the-art in frame interpolation: On the one hand, pretrained flow estimation modules [32, 35, 42] became more powerful and can be used as a starting point; on the other hand, using forward warping instead of backward warping

allows to directly compute the intermediate flow vectors without approximating them [9, 21]. However, their performance heavily depends on the pretrained flow network used as well as on the image resolution of the input [9]. M2M-PWC [9], for example, requires bilinear downscaling of 4K images for optimal performance. Furthermore, such models have many trainable parameters and require a lot of memory.

Applying VFI techniques successfully and efficiently to high-resolution data remains a challenging task. Most successful high-resolution models have in common that they do not compute the flow via an expensive cost volume [*e.g.*, 10] and use a multi-scale approach, incorporating some kind of weight sharing between the different scales of the input images [*e.g.*, 25, 26, 28]. We take these ideas a step further and combine the shared multi-resolution approach with a preceding image compression based on linear dimensionality reduction.

3 Dimensionality Reduction for Feature Extraction

As our approach to obtaining a compact image representation is inspired by principal component analysis (PCA) [7], let us first recap its basics. PCA describes the process of computing principal components from the data. The principal components represent an orthonormal basis of the zero-centered data. PCA can be used to perform a change of basis of the data, where the first basis vector maximizes the variance of the projected data, the second accounts for the largest remaining variability, and so on. Because of this characteristic, the PCA basis is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components. As a result, the lower-dimensional, projected data preserves as much of the data's variation as possible with a linear transformation.

Instead of extracting a compact feature representation with a neural network directly from the high-resolution input image, we here argue that dimensionality reduction can be used to first compress the image data. This reduces the memory footprint of the input data significantly without resorting to downscaling. There are two main advantages of combining explicit dimensionality reduction with neural networks: *(i)* Using a compressed representation of the input reduces the memory requirements of the subsequent neural network modules. *(ii)* Due to the compressed representation also the model capacity of the subsequent neural network modules can be reduced, decreasing the number of trainable parameters.

Block-based PCA. Similar to [34], we use block-based PCA to compress a single image. We split an image $I \in \mathbb{R}^{H\times W\times 3}$ into $d\times d$ blocks, where each block, vectorized, represents a data point $x_i \in \mathbb{R}^{d^2}$, which are concatenated into a matrix as $X = [x_1,\dots,x_N] \in \mathbb{R}^{d^2\times N}$. The number of blocks N depends on the image resolution, *i. e.*, $N = 3HW/d^2$. After subtracting the mean $\bar{x} = {}^1\!/_N \sum_i x_i$ from X, resulting in $\bar{X}$, we compute the covariance matrix $C \in \mathbb{R}^{d^2\times d^2}$ of $\bar{X}$ as $C = {}^1\!/_{N-1}\cdot \bar{X}\bar{X}^T$. We perform an eigendecomposition of C and take the first k eigenvectors u_i as our projection axes, *i. e.*, $U_k = [u_1,\dots,u_k] \in \mathbb{R}^{d^2\times k}$. We compute the projected data as $\tilde{X} = U_k^T \bar{X}$, $\tilde{X} \in \mathbb{R}^{k\times N}$. Reshaping $\tilde{X}$ back yields the compressed image representation $\tilde{I} \in \mathbb{R}^{H/d\times W/d\times 3k}$ with a compression ratio of $r = k/d^2$ *w. r. t.* the original image I.

Simply applying block-based PCA to each image is computationally not ideal as the principal components have to be recomputed every time. Further, the projected data is not directly suitable as input to a neural network. First, the data range can be very large and second, representing each image in a different projected space makes it difficult for the network to learn from it. We address these challenges explicitly in our framework design.

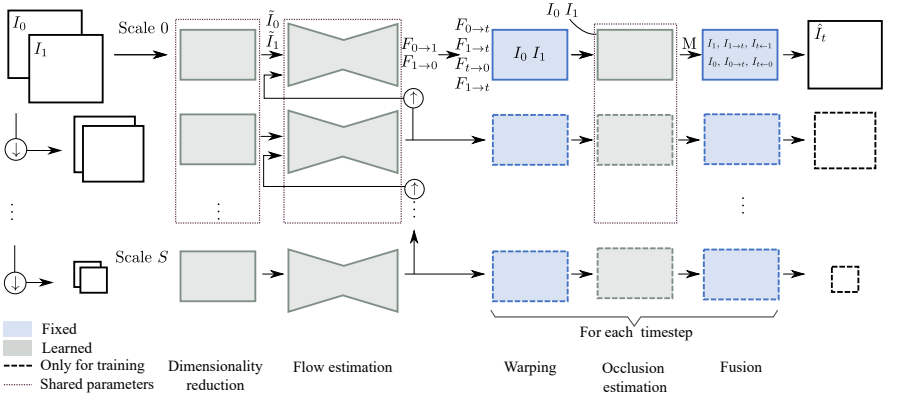


Figure 1: **fLDR-Net architecture.** Given two images, I_0 and I_1, we generate the intermediate image $\hat{I}_t$ at time t by applying our finetuned dimensionality reduction followed by flow estimation, image warping (forward and backward), and estimation of the weighting map M for fusing the warped images, giving the final result. Our method is very lightweight in terms of trainable parameters as the parameters are shared across scales; warping and fusion are fixed, non-learnable computations. The synthesis of $\hat{I}_t^s$ for $s > 0$ is only used during training.

4 fLDR-Net for Video Frame Interpolation

Given two images, I_0 and I_1, the goal of video frame interpolation is to generate several intermediate images I_t for $t \in (0,1)$. The focus of our method is to propose a lightweight framework in terms of memory and the number of trainable parameters that is especially suitable for high-resolution images, in our case 4K, and limited hardware resources. We achieve this by compressing the input images using ideas from dimensionality reduction and projecting the data into a low-dimensional space. However, it is not trivial how to use this different representation efficiently in a neural network. Following the recap of the basics of principal component analysis PCA in Sec. 3, we now explain how to use a block-based linear representation successfully in a neural network, as well as our overall framework design to further reduce the computational overhead.

Overview. Fig. 1 shows our proposed framework. It consists of three trainable modules, namely the finetuned linear dimensionality reduction (fLDR), flow estimation, and occlusion estimation. The image synthesis part at each time step t is very lightweight in terms of trainable network modules and memory. We do not perform any learned refinement of the estimated flow vectors or image synthesis besides the occlusion estimation. In the following, we describe each part in detail. To simplify the notation, we focus on scale level $s = 0$. If not mentioned otherwise, the other scales are carried out in the same way except that the input image is first bilinearly downscaled by a factor of 2 from the previous level.

Finetuned linear dimensionality reduction (fLDR). We apply dimensionality reduction of the input images inspired by block-based PCA as described in Sec. 3. However, we cannot apply this to each image I separately as this would lead to different, data-dependent low-dimensional projection spaces, unfavorable for neural networks. Additionally, the projection spaces should be consistent among the scales, in order to enable parameter sharing. This

requires a single set of projection vectors suitable for all images at all scales. We achieve this by using the k first block-based PCA projection vectors U_k of one random sample image as initialization and treating the eigenvectors and the mean vector of the computed initial PCA decomposition as trainable parameters, optimized end-to-end for the task of frame interpolation. We call this procedure finetuned block-based linear dimensionality reduction, short fLDR, yielding the compressed image $\tilde{I}$. Before further processing $\tilde{I}$, we divide each projected data point by the mean of its absolute values along the principal direction dimensions and normalize the whole matrix to values in the range of $[-1,1]$ suitable for neural networks.

Flow estimation. Existing flow estimation modules often estimate flow only at a downsampled input resolution, *e.g.* $1/4$ ([28, 32]) or $1/8$ ([35]), for computational reasons. Because the input to the flow estimation is in our case already a compressed image representation obtained from the fLDR module, we estimate the flow at the same resolution as the compressed input representation, *i. e.*, at $1/d$ resolution. We use the high-level idea of XVFI [28] of resharing the parameters across scales for efficient multi-scale optical flow estimation without a pretrained flow network. We also estimate the flow at different scales $s = 0,\dots,S$ with shared parameters, except for the lowest level $s = S$. This allows to increase the number of scales at test time, when the input image has a significantly larger resolution than at training time (*e.g.*, 4K *vs.* training patch size). Contrary to XVFI [28], we can further significantly simplify the flow estimation module and only need one encoder-decoder network instead of two to predict the intermediate flow fields. The reasons are our compressed image representation as input as well as changes in the pipeline, mainly using forward warping instead of backward warping.

As illustrated in Fig. 1, we predict $F_{0\to1}$ and $F_{1\to0}$ at each scale . The flow is estimated from the compressed image representation $\tilde{I}$, which first gets passed through two convolutional layers. On all levels except $s = S$, the feature representation is concatenated with the bilinearly upscaled flow estimation from the previous level. Architectural network details are provided in the supplemental material.

To compute the intermediate image I_t, we need the flow from and to the intermediate time step t. To compute the forward flow, we can simply scale it accordingly:

$$F_{0\to t} = tF_{0\to1} \quad \text{and} \quad F_{1\to t} = (1-t)F_{1\to0}\,. \tag{1}$$

Computing the backward flow, $F_{t\to0}$ and $F_{t\to1}$, is only possible in an approximated manner [11]. XVFI [28] uses a second neural network for this, while we opt for a direct, non-trainable approximation. In order to obtain $F_{t\to0}$, we can take the flow $F_{1\to0}$, adjusting the magnitude by multiplying it with t and warping it to the position from which it points at time t. We do the latter by backward warping $tF_{1\to0}$ with the flow $F_{0\to1}$, scaled with $1-t$, approximating the starting point of the flow vector at time t. $F_{t\to1}$ is approximated analogously as

$$F_{t\to0} \approx \overleftarrow{\omega}(tF_{1\to0}, (1-t)F_{0\to1}) \quad \text{and} \quad F_{t\to1} \approx \overleftarrow{\omega}((1-t)F_{0\to1}, tF_{1\to0})\,, \tag{2}$$

where $\overleftarrow{\omega}(\cdot,\cdot)$ describes the backward warping function.

Image warping. Before we can use the computed flow vectors for warping, we upscale them bilinearly to the non-compressed image resolution at each scale. We can then warp the input images, I_0 and I_1, to the intermediate time step t. We do this by forward warping them with softmax splatting [21], noted as $\overrightarrow{\sigma}$, with occlusion estimation [1] as importance metric:

$$I_{0\to t} = \overrightarrow{\sigma}(I_0, F_{0\to t}) \quad \text{and} \quad I_{1\to t} = \overrightarrow{\sigma}(I_1, F_{1\to t})\,. \tag{3}$$

The advantage of forward splatting is that we can directly use the intermediate flow vectors $F_{0\to t}$ and $F_{1\to t}$. However, forward splatting can lead to small holes due to occlusions and divergent flow vectors. Backward warping, on the other hand, would give a dense result, but requires an approximation of the backward flow. Video frame interpolation methods that use backward warping often use an additional network to refine $F_{t\to 0}$ and $F_{t\to 1}$, *e.g.*, XVFI [28] for better accuracy. In our case, we only use the backward images as additional input to compute the occlusion mask. Therefore, the directly computed flow vectors from Eq. (2) are enough and no further learned refinement is needed. The backward warping gives us two additional images at time instant t:

$$I_{t\leftarrow 0} = \overleftarrow{\omega}(I_0, F_{t\to 0}) \quad \text{and} \quad I_{t\leftarrow 1} = \overleftarrow{\omega}(I_1, F_{t\to 1}) \,. \tag{4}$$

Occlusion estimation. In order to accurately merge the different warped images at time step t, we propose a simple occlusion estimation network to estimate a weighting map $M \in \mathbb{R}^{H\times W\times c}$, where c corresponds to the number of images to merge. The input to the occlusion estimation are the warped images as well as the input images. In the last layer of the estimation network, we apply a softmax along the last dimension, yielding a probability distribution for every output pixel. We include temperature scaling [8] of the softmax function to better calibrate the distribution. The details of the temperature scaling and the network architecture can be found in the supplemental material.

Image synthesis. We omit a dedicated image synthesis network to predict $\hat{I}_t$. Instead, we directly fuse the warped images by weighting them with the predicted occlusion map M (by pixelwise multiplication $\odot$) as well as the temporal distance from the input images:

$$\hat{I}_t = \frac{\sum_{i\in 0,1}\left[(i\cdot t)+(1-i)(1-t)\right]\cdot\left(M_{t\leftarrow i}\odot I_{t\leftarrow i} + M_{i\to t}\odot I_{i\to t} + M_i \odot I_i\right)}{(1-t)\cdot(M_{t\leftarrow 0}+M_{0\to t}+M_0)+t\cdot(M_{t\leftarrow 1}+M_{1\to t}+M_1)} \,. \tag{5}$$

Loss functions. We train our model with a similar loss function as in XVFI [28], consisting of an image reconstruction loss, $\mathcal{L}_{recon}$, at all levels and an edge-aware smoothness loss, $\mathcal{L}_{smooth}$, on the estimated flow vectors $F_{0\to 1}$ and $F_{1\to 0}$ at the finest scale $s = 0$. We add an additional term to supervise the flow estimation at the highest resolution by computing the warping error, $\mathcal{L}_{warp}$, yielding the total loss

$$\mathcal{L}_{total} = \mathcal{L}_{recon} + \lambda_{smooth}\cdot\mathcal{L}_{smooth} + \lambda_{warp}\cdot\mathcal{L}_{warp} \,, \tag{6}$$

where $\lambda_{(\cdot)}$ define the weighting factors. Details are given in the supplemental.

5 Experiments

5.1 Setup

Training. We train our method on X-Train [28], mainly following the training procedure of XVFI [28]. X-Train consists of 4,408 clips of 768×768 images cropped from the original 4K frames. Each clip consists of 65 consecutive frames. Following [28], we also select random triplets from the sequences, at most 32 frames apart, and randomly select training patches of 512×512. We train on one Nvidia 3080Ti (12GB) GPU with a batch size of 8 for 200 epochs.

Dataset	Interpolation factor	# clips	# eval. frames / clip	# total eval. frames	PWC-Net [32]		
					25_{th}	50_{th}	75_{th}
Xiph-4K	2×	8	49	392	9.1	14.1	25.0
X-Test [28]	8×	15	7	105	23.9	81.9	138.5
Inter4K-S	8×	144	7	1008	7.6	30.9	102.0
Inter4K-L	8×	133	7	931	13.4	56.7	162.5

Table 1: **Overview of used datasets.** Comparison of our newly proposed test sets based on Inter4K [30] to existing datasets. Inter4K-S and Inter4K-L cover more scenes and Inter4K-L has statistically larger motion. The percentiles of the flow magnitudes (last 3 columns) are computed from the estimated flow [32] between the input frames.

We use the Adam [12] optimizer and an initial learning rate of 10^{-4}, which is decreased by 0.25 after each 50 epochs, starting after 70 epochs. The optimization of the temperature parameter T [8] is done with a learning rate of 10^{-3}. Because the projection vectors are very sensitive to changes, we use a learning rate of 10^{-5} for the linear dimensionality reduction layer. We use $S = 3$ for training. For evaluation, we take the best checkpoint based on the validation set. The full list of hyperparameters is given in the supplemental.

Testing. We use $S = 5$ during testing, *i. e.* 2 scale levels more than during training to handle the high-resolution images. For evaluation, we focus on datasets consisting of 4K video frames. The two commonly used 4K datasets in frame interpolation are the videos from Xiph[1] as described in [21], as well as X-Test from [28]. However, both of them have limited variability. X-Test only has 15 image pairs, each with 7 intermediate frames. The selected clips have a lot of camera motion leading to large, dominating flow magnitudes. Xiph, on the other hand, has only 8 videos, with much less camera motion and mainly object motion, resulting in most of the flow vectors having a small magnitude. This is summarized in Tab. 1. To have more variability, we selected additional evaluation scenes from the Inter4K [30] test set. The original test set consists of 100 videos with up to 60 fps and 300 frames per video. We only take the videos with 60 fps and split them into their different scenes by thresholding the pixel difference between consecutive frames. We create two different versions to evaluate different degrees of motion. For Inter4K-S, we take the first 9 frames of each scene, and for Inter4K-L the first 17, respectively. Both test sets analyze 8× temporal interpolation with 7 intermediate frames, meaning for Inter4K-L every second intermediate frame is skipped to simulate larger motion.

5.2 Comparison with the state of the art

We compare to SoftSplat [21] and M2M [9], which both rely on pretrained flow and forward splatting, as well as to ABME [25], RIFE [10], and XVFI [28], which do not use a pretrained flow network. All methods support interpolation at arbitrary time steps.

Whenever possible, we used the original code and checkpoints from the authors. The performance of M2M [9] heavily depends on the input image resolution to the flow network, *e.g.*, M2M-PWC [9] performs best for 4K input when images are first downscaled by a factor of 8 or 16, while M2M-DIS is less sensitive, and performs best at the original input. For M2M-DIS, we report the numbers from the original paper [9] as the code is not online. For

[1] from xiph.org

	Pretrained flow	# Param. (Mill.)	Memory (for 4K)	Training dataset	Xiph-4K	X-Test	Inter4K-S	Inter4K-L	Inference (in s/f)
SoftSplat [21]	✓	7.7	–	Vimeo90K	34.2*[24]	25.48*[9]	OOM	OOM	–
M2M-PWC [9]	✓	7.6	10 GB	Vimeo90K	**34.88**	**30.81**	29.22	24.87	**0.21**
M2M-DIS [9]	✓	–	–	Vimeo90K	–	30.18*[9]	–	–	–
ABME [25]	✗	18.1*[25]	–	Vimeo90K	OOM	30.16*[25]	OOM	OOM	–
RIFE$_m$ [10]	✗	9.8	6.8 GB	Vimeo90K	34.80	26.80	28.37	24.40	0.40
RIFE$_m^\diamond$ [10]	✗	9.8	6.8 GB	X-Train	34.00	28.06	28.36	24.47	0.40
XVFI$^\diamond$ [28]	✗	5.5	>12 GB	X-Train	34.04	30.34	28.82	24.62	–
Ours	✗	**0.9**	**4.6** GB	X-Train	34.16	30.45	**29.29**	**25.16**	0.51

Table 2: **Quantitative results** on 4K images in terms of PSNR. Numbers with $*$ are taken from the given references; $\diamond$ denotes retraining. The methods have been trained on either Vimeo90K [43] or X-Train [28]. The inference time has been measured on a Nvidia 3080Ti (12GB) GPU. The best and 2nd best values are in **bold** and underlined, respectively. An evaluation with SSIM [39] and LPIPS [44] is given in the supplemental.

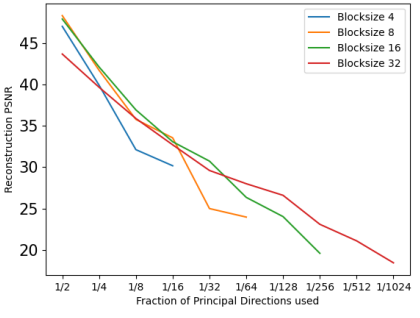
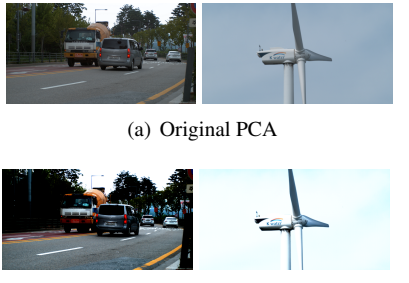


Figure 2: Reconstruction acc. for different block sizes d *w. r. t.* the compression ratio $1/r$.



(a) Original PCA

(b) With finetuning

Figure 3: Image reconstruction with a $1/4$ of the projection directions, with and without finetuneing them.

M2M-PWC, we test different scale factors for each dataset, reporting the results for the best option. For RIFE [10], we use the RIFE$_m$ model, which supports arbitrary interpolation. We also retrain it on X-Train with an increased patch size of 512×512, as we observed this to slightly improve the results. We evaluate the models on a Nvidia 3080 Ti (12GB) GPU. With the available memory, we were not able the evaluate SoftSplat [21] and ABME [25] on 4K images and hence report numbers from the referenced papers instead.

Tab. 2 gives a quantitative comparison. We achieve state-of-the-art accuracy for most of the datasets among the methods without pretrained flow. A powerful pretrained flow network can occasionally help, but the results are heavily dependent on the type of flow network and amount of motion. Our approach yields consistently high PSNR values, especially for the datasets with larger motion, despite having by far the fewest trainable parameters as well as the lowest memory requirements. In comparison to XVFI [28], we can see how our conceptual contributions lead to a significant reduction of memory and parameters, yet an increase in PSNR of up to 0.54dB. Without specifically optimizing for inference time, we can achieve competitive runtimes comparable to most of the other methods (M2M-PWC downscales the input image first by a factor of $1/16$, leading to faster inference) without the need for large, high-end GPUs. Qualitative visual results are shown in Fig. 4 and in the supplemental video.

	fDLR	Flow estim.	Warping	Occl. estim.	Fusion	Full model
Memory (GB)	3.5	2.7	4.6	4.0	4.6	4.6
# Parameters	1k	733k	1	194k	0	0.9M
Inference time (ms)	30	22	410	44	3.1	510

Table 3: **Analysis of the computational resources.** Breakdown of the individual modules of our framework for performing inference on a 4K frame.

	# Param. (in Mill.)	Memory (for 4K)	Xiph-4K	X-Test	Inter4K-S	Inter4K-L
Ours (full)	0.9	4.6GB	34.16	30.45	29.29	25.16
w/o finetuning projection vectors	0.9	4.6GB	33.92	29.46	28.34	24.43
w/o backward flow ($F_{t\to0}$ & $F_{t\to1}$)	0.9	4.6GB	33.95	30.13	28.81	24.77
w/o $\mathcal{L}_{warp}$	0.9	4.6GB	33.99	30.25	28.97	24.96
w/o T scaling	0.9	4.6GB	34.13	30.41	29.20	25.09
with synthesis (from [28])	2.6	9.5GB	34.20	30.59	29.12	25.07

Table 4: **Ablation study.** Results from individually adding or removing one of the options from our full method. The largest benefit is obtained by finetuning the projection vectors.

5.3 Method analysis

Efficiency. The focus of our method is to reduce the complexity of the network and the required memory. To select an ideal block size d and the number of projection dimensions k, we analyze the general compression capability of block-based PCA by computing the basis from a single image of 512×512, taking a fraction of it to reconstruct the images in X-Test [28], and averaging the reconstruction PSNR. For a targeted compression rate of $1/r = k/d^2 = 1/4$, Fig. 2 shows that a block size of $d = 8$ or 16 has the best reconstruction accuracy. Because the compressed image representation is directly used for the flow computation and, therefore, a higher image resolution is preferable, we choose $d = 8$ and thus $k = d^2/r = 8^2/4 = 16$. Fig. 3 shows the comparison of reconstructed images when using the original k PCA vectors and after finetuning with our fLDR module. The finetuned vectors increase the contrast, highlighting features that intuitively seem suitable for flow estimation in the context of frame interpolation. The efficiency in feature representation using fLDR becomes evident when looking at the remaining trainable parameters of our framework. In Tab. 3 we report the memory usage, number of trainable parameters, and inference time *w. r. t.* the individual modules. Our flow network only has 0.73 million (M) parameters, while XVFI [28] needs 2.5M for flow estimation plus an additional 1M for flow refinement. By replacing the synthesis network of XVFI [28] with weighted averaging (our fusion module), we save another 1.7M parameters. Estimating optical flow with a pretrained PWC-Net [32] requires 90ms at half the image resolution (to fit on our GPU), 9.4M trainable parameters, and 5.1GB memory. Visualizations of the optical flow are provided in the supplemental. Our inference time is dominated by our warping module, designed to reduce the number of trainable parameters.[2]

Ablation design choices. In Tab. 4 we analyze the effect of our design choices. Finetuning the projection vectors is crucial and leads to a PSNR improvement of up to 1dB. The variance of the final image quality (PSNR) for selecting different images for initialization of the projection dimensions is for Xiph-4K and X-Test only 0.02dB and 0.05dB, respectively. The

[2] One trainable parameter is needed to compute the importance metric for forward warping as described in [21].

(a) Overlaid inputs (b) RIFE$_m$ [10] (c) XVFI [28] (d) M2M-PWC [9] (e) Ours

Figure 4: **Qualitative examples.** Our method can handle large motion in 4K images better. From top to bottom, the images are taken from X-Test [28] (*top*), Inter4K-S [30] (*middle*, two examples), and Inter4k-L [30] (*bottom*). The last row shows a cropped part of the original result. For Rife$_m$, we take the model trained on X-Train, giving better results on these datasets.

Inter4K dataset has a larger variability in images, leading to a slighter higher variance of around 0.14dB. The powerful synthesis network only gives a slight positive effect in accuracy on one dataset, which is not proportional to the overhead it adds. The remaining design choices have a positive effect on the PSNR value while not adding any notable overhead.

Limitations. Our framework has been explicitly and carefully designed for high-resolution video. Before flow computation, we compress the frames using linear dimensionality reduction on 8×8 blocks. For this reason, flow can only be computed using $1/8$ of the original image resolution as input. This makes our approach perform comparatively worse on smaller images such as the common Vimeo90K [43] test set, as the input to the flow estimation is not fine-grained enough *w. r. t.* to the full image resolution. Evaluating our method (trained on X-Train) directly on Vimeo-90k, we obtain a PSNR of 33.03dB using 1.7GB of memory. For small images, where memory constraints are less of an issue, better accuracy can be obtained with more memory intensive approaches; *e.g.*, SoftSplat[21] achieves a PSNR of 36.10dB.

6 Conclusion

We present a method for video frame interpolation in high-resolution videos, which uses linear dimensionality reduction and a lightweight neural network architecture to boost the efficiency. By compressing images first with our finetuned projection vectors, we can significantly reduce the number of trainable parameters and the overall memory requirements in the subsequent network modules. We show that we can achieve highly competitive results on various 4K datasets with only a fraction of trainable parameters and low memory footprint.

Acknowledgments and disclosure of funding

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 866008). The project has also been supported in part by the State of Hesse through the cluster project "The Third Wave of Artificial Intelligence (3AI)".

References

[1] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[2] Mohammed Kamel Benkaddour and Abdennacer Bounoua. Feature extraction and classification using deep convolutional neural networks, PCA and SVC for face recognition. *Traitement du Signal*, 34(1-2):77–91, 2017.

[3] Karlis Martins Briedis, Abdelaziz Djelouah, Mark Meyer, Ian McGonigal, Markus H. Gross, and Christopher Schroers. Neural frame interpolation for rendered content. *ACM Transactions on Graphics*, 40(6):239:1–239:13, 2021.

[4] Xianhang Cheng and Zhenzhong Chen. Video frame interpolation via deformable separable convolution. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2020.

[5] Myungsub Choi, Heewon Kim, Bohyung Han, Ning Xu, and Kyoung Mu Lee. Channel attention is all you need for video frame interpolation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2020.

[6] Abdelaziz Djelouah, Joaquim Campos, Simone Schaub-Meyer, and Christopher Schroers. Neural inter-frame compression for video coding. In *Proceedings of the Seventeenth IEEE International Conference on Computer Vision*, 2019.

[7] Karl Pearson F.R.S. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2 (11):559–572, 1901.

[8] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34rd International Conference on Machine Learning*, 2017.

[9] Ping Hu, Simon Niklaus, Stan Sclaroff, and Kate Saenko. Many-to-many splatting for efficient video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[10] Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. Real-time intermediate flow estimation for video frame interpolation. In *Proceedings of the 17th European Conference on Computer Vision*, Lecture Notes in Computer Science. Springer, 2022.

[11] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik G. Learned-Miller, and Jan Kautz. Super SloMo: High quality estimation of multiple intermediate frames for video interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, 2012.

[14] Siyao Li, Shiyu Zhao, Weijiang Yu, Wenxiu Sun, Dimitris N. Metaxas, Chen Change Loy, and Ziwei Liu. Deep animation video interpolation in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[15] Ziwei Liu, Raymond A. Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *Proceedings of the Sixteenth IEEE International Conference on Computer Vision*, 2017.

[16] Gucan Long, Laurent Kneip, Jose M. Alvarez, Hongdong Li, Xiaohu Zhang, and Qifeng Yu. Learning image matching by simply watching video. In *Proceedings of the 14th European Conference on Computer Vision*, Lecture Notes in Computer Science. Springer, 2016.

[17] Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. Phase-based frame interpolation for video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[18] Simone Meyer, Victor Cornillère, Abdelaziz Djelouah, Christopher Schroers, and Markus H. Gross. Deep video color propagation. In *Proceedings of the British Machine Vision Conference*, 2018.

[19] Simone Meyer, Abdelaziz Djelouah, Brian McWilliams, Alexander Sorkine-Hornung, Markus H. Gross, and Christopher Schroers. PhaseNet for video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[20] Simon Niklaus and Feng Liu. Context-aware synthesis for video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[21] Simon Niklaus and Feng Liu. Softmax splatting for video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[22] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[23] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive separable convolution. In *Proceedings of the Sixteenth IEEE International Conference on Computer Vision*, 2017.

[24] Simon Niklaus, Ping Hu, and Jiawen Chen. Splatting-based synthesis for video frame interpolation. *arXiv:2201.10075 [cs.CV]*, 2022.

[25] Junheum Park, Chul Lee, and Chang-Su Kim. Asymmetric bilateral motion estimation for video frame interpolation. In *Proceedings of the Eightteenth IEEE International Conference on Computer Vision*, 2021.

[26] Fitsum A. Reda, Janne Kontkanen, Eric Tabellion, Deqing Sun, Caroline Pantofaru, and Brian Curless. FILM: Frame interpolation for large motion. In *Proceedings of the 17th European Conference on Computer Vision*, Lecture Notes in Computer Science. Springer, 2022.

[27] Rachid Riad, Olivier Teboul, David Grangier, and Neil Zeghidour. Learning strides in convolutional neural networks. In *Proceedings of the Tenth International Conference on Learning Representations*, 2022.

[28] Hyeonjun Sim, Jihyong Oh, and Munchurl Kim. XVFI: eXtreme video frame interpolation. In *Proceedings of the Eightteenth IEEE International Conference on Computer Vision*, 2021.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the International Conference on Learning Representations*, 2015.

[30] Alexandros Stergiou and Ronald Poppe. Adapool: Exponential adaptive pooling for information-retaining downsampling. *arXiv:2111.00772 [cs.CV]*, 2021.

[31] Deqing Sun, Stefan Roth, and Michael J. Black. Secrets of optical flow estimation and their principles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[32] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[33] David S. Taubman and Michael W. Marcellin. JPEG2000: standard for interactive imaging. *Proceedings of the IEEE*, 90(8):1336–1357, 2002.

[34] Jin-Shiuh Taur and Chin-Wang Tao. Medical image compression using principal component analysis. In *Proceedings of the IEEE International Conference on Image Processing*, 1996.

[35] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *Proceedings of the 16th European Conference on Computer Vision*, Lecture Notes in Computer Science. Springer, 2020.

[36] Stepan Tulyakov, Alfredo Bochicchio, Daniel Gehrig, Stamatios Georgoulis, Yuanyou Li, and Davide Scaramuzza. Time Lens++: Event-based frame interpolation with parametric non-linear flow and multi-scale fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[37] Stepan Tulyakov, Daniel Gehrig, Stamatios Georgoulis, Julius Erbach, Mathias Gehrig, Yuanyou Li, and Davide Scaramuzza. Time Lens: Event-based video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[38] Gregory K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

[39] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[40] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[41] Chao-Yuan Wu, Nayan Singhal, and Philipp Krähenbühl. Video compression through image interpolation. In *Proceedings of the 15th European Conference on Computer Vision*, Lecture Notes in Computer Science. Springer, 2018.

[42] Haofei Xu, Jing Zhang, Jianfei Cai, Hamid Rezatofighi, and Dacheng Tao. Gmflow: Learning optical flow via global matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[43] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T. Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision*, 127(8): 1106–1125, 2019.

[44] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.