

# Video Frame Interpolation via Cyclic Fine-Tuning and Asymmetric Reverse Flow

Morten Hannemose<sup>1</sup>, Janus Nørtoft Jensen<sup>1</sup>, Guðmundur Einarsson<sup>1</sup>, Jakob Wilm<sup>2</sup>, Anders Bjarholm Dahl<sup>1</sup>, and Jeppe Revall Frisvad<sup>1</sup>

<sup>1</sup> DTU Compute, Technical University of Denmark

<sup>2</sup> SDU Robotics, University of Southern Denmark

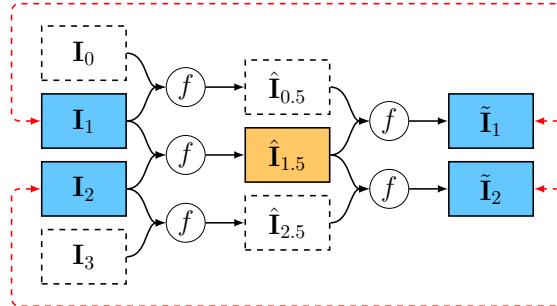
**Abstract.** The objective in video frame interpolation is to predict additional in-between frames in a video while retaining natural motion and good visual quality. In this work, we use a convolutional neural network (CNN) that takes two frames as input and predicts two optical flows with pixelwise weights. The flows are from an unknown in-between frame to the input frames. The input frames are warped with the predicted flows, multiplied by the predicted weights, and added to form the in-between frame. We also propose a new strategy to improve the performance of video frame interpolation models: we reconstruct the original frames using the learned model by reusing the predicted frames as input for the model. This is used during inference to fine-tune the model so that it predicts the best possible frames. Our model outperforms the publicly available state-of-the-art methods on multiple datasets.

**Keywords:** slow motion · video frame interpolation · convolutional neural networks.

## 1 Introduction

Video frame interpolation, also known as inbetweening, is the process of generating intermediate frames between two consecutive frames in a video sequence. This is an important technique in computer animation [19], where artists draw keyframes and let software interpolate between them. With the advent of high frame rate displays that need to display videos recorded at lower frame rates, inbetweening has become important in order to perform frame rate up-conversion [2]. Computer animation research [9, 19] indicates that good inbetweening cannot be obtained based on linear motion, as objects often deform and follow nonlinear paths between frames. In an early paper, Catmull [3] interestingly argues that inbetweening is “akin to difficult artificial intelligence problems” in that it must be able understand the content of the images in order to accurately handle e.g. occlusions. Applying learning-based methods to the problem of inbetweening thus seems an interesting line of investigation.

Some of the first work on video frame interpolation using CNNs was presented by Niklaus et al. [17, 18]. Their approach relies on estimating kernels to jointly represent motion and interpolate intermediate frames. Concurrently, Liu



**Fig. 1.** Diagram illustrating the cyclic fine-tuning process when predicting frame  $\hat{\mathbf{I}}_{1.5}$ . The model is first applied in a pairwise manner on the four input frames  $\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2$ , and  $\mathbf{I}_3$ , then on the results  $\hat{\mathbf{I}}_{0.5}, \hat{\mathbf{I}}_{1.5}$ , and  $\hat{\mathbf{I}}_{2.5}$ . The results of the second iteration,  $\tilde{\mathbf{I}}_1$  and  $\tilde{\mathbf{I}}_2$ , are then compared with the input frames and the weights of the network are updated. This process optimizes our model specifically to be good at interpolating frame  $\hat{\mathbf{I}}_{1.5}$ .

et al. [11] and Jiang et al. [6] used neural networks to predict optical flow and used it to warp the input images followed by a linear blending.

Our contribution is twofold. Firstly, we propose a CNN architecture that directly estimates asymmetric optical flows and weights from an unknown intermediate frame to two input frames. We use this to interpolate the frame in-between. Existing techniques either assume that this flow is symmetric or use a symmetric approximation followed by a refinement step [6, 11, 16]. For non-linear motion, this assumption does not hold, and we document the effect of relaxing it. Secondly, we propose a new strategy for fine-tuning a network for each specific frame in a video. We rely on the fact that interpolated frames can be used to estimate the original frames by applying the method again with the in-between frames as input. The similarity of reconstructed and original frames can be considered a proxy for the quality of the interpolated frames. For each frame we predict, the model is fine-tuned in this manner using the surrounding frames in the video, see Figure 1. This concept is not restricted to our method and could be applied to other methods as well.

## 2 Related work

Video frame interpolation is usually done in two steps: motion estimation followed by frame synthesis. Motion estimation is often performed using optical flow [1, 4, 25], and optical flow algorithms have used interpolation error as an error metric [1, 12, 23]. Frame synthesis can then be done via e.g. bilinear interpolation and occlusion reasoning using simple hole filling. Other methods use phase decompositions of the input frames to predict the phase decomposition of the intermediate frame and invert this for frame generation [14, 15], or they use local per pixel convolution kernels on the input frames to both represent motion and synthesize new frames [17, 18]. Mahajan et al. [13] determine where each pixel in an intermediate frame comes from in the surrounding input frames by

solving an expensive optimization problem. Our method is similar but replaces the optimization step with a learned neural network.

The advent of CNNs has prompted several new learning based approaches. Liu et al. [11] train a CNN to predict a symmetrical optical flow from the intermediate frame to the surrounding frames. They synthesize the target frame by interpolating the values in the input frames. Niklaus et al. [17] train a network to output local 38x38 convolution kernels for each pixel to be applied on the input images. In [18], they are able to improve this to 51x51 kernels. However, their representation is still limited to motions within this range. Jiang et al. [6] first predict bidirectional optical flows between two input frames. They combine these to get a symmetric approximation of the flows from an intermediate frame to the input frames, which is then refined in a separate step. Our method, in contrast, directly predicts the final flows to the input frames without the need for an intermediate step. Niklaus et al. [16] also initially predict bidirectional flows between the input frames and extract context maps for the images. They warp the input images and context maps to the intermediate time step using the predicted flows. Another network blends these to get the intermediate frame.

Liu et al. [10] propose a new loss term, which they call cycle consistency loss. This is a loss based on how well the output frames of a model can reconstruct the input frames. They retrain the model from [11] with this and show state-of-the-art results. We use this loss term and show how it can be used during inference to improve results. Meyer et al. [14] estimate the phase of an intermediate frame from the phases of two input frames represented by steerable pyramid filters. They invert the decomposition to reconstruct the image. This method alleviates some of the limitations of optical flow, which are also limitations of our method: sudden light changes, transparency and motion blur, for example. However, their results have a lower level of detail.

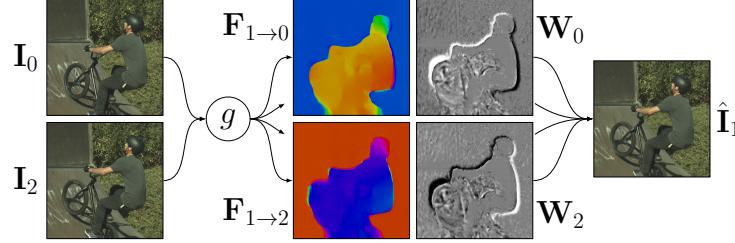
### 3 Method

Given a video containing the image sequence  $\mathbf{I}_0, \mathbf{I}_1, \dots, \mathbf{I}_n$ , we are interested in computing additional images that can be inserted in the original sequence to increase the frame rate, while keeping good visual quality in the video. Our method doubles the frame rate, which allows for the retrieval of approximately any in-between frame by recursive application of the method. This means that we need to compute estimates of  $\mathbf{I}_{0.5}, \mathbf{I}_{1.5}, \dots, \mathbf{I}_{n-0.5}$ , such that the final sequence would be:

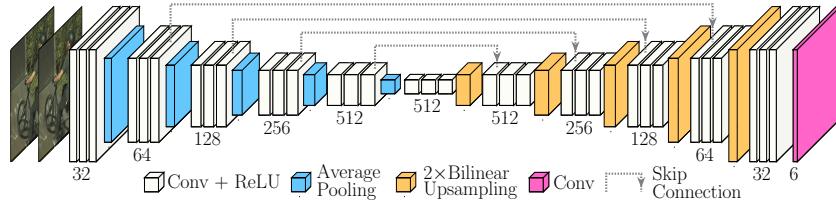
$$\mathbf{I}_0, \mathbf{I}_{0.5}, \mathbf{I}_1, \dots, \mathbf{I}_{n-0.5}, \mathbf{I}_n.$$

We simplify the problem by only looking at interpolating a single frame  $\mathbf{I}_1$ , that is located temporally between two neighboring frames  $\mathbf{I}_0$  and  $\mathbf{I}_2$ . If we know the optical flows from the missing frame to each of these and denote them as  $\mathbf{F}_{1 \rightarrow 0}$  and  $\mathbf{F}_{1 \rightarrow 2}$ , we can compute an estimate of the missing frame by

$$\hat{\mathbf{I}}_1 = \mathbf{W}_0 \mathcal{W}(\mathbf{F}_{1 \rightarrow 0}, \mathbf{I}_0) + \mathbf{W}_2 \mathcal{W}(\mathbf{F}_{1 \rightarrow 2}, \mathbf{I}_2), \quad (1)$$



**Fig. 2.** Illustration of the frame interpolation process with  $g$  from Equation (2). From left to right: Input frames, predicted flows, weights and final interpolated frame.



**Fig. 3.** The architecture of our network. Input is two color images  $I_0$  and  $I_2$  and output is optical flows  $F_{1\rightarrow 0}, F_{1\rightarrow 2}$ , and weights  $W_0, W_2$ . Convolutions are  $3\times 3$  and average pooling is  $2\times 2$  with a stride of 2. Skip connections are implemented by adding the output of the layer that arrows emerge from to the output of the layers they point to.

where  $\mathcal{W}(\cdot, \cdot)$  is the backward warping function that follows the vector to the input frame and samples a value with bilinear interpolation.  $\mathbf{W}_0$  and  $\mathbf{W}_2$  are weights for each pixel describing how much of each of the neighboring frames should contribute to the middle frame. The weights are used for handling occlusions. Examples of flows and weights can be seen in Figure 2. We train a CNN  $g$  with a U-Net [20] style architecture, illustrated in Figure 3. The network takes two images as input and predicts the flows and pixel-wise weights

$$g(\mathbf{I}_0, \mathbf{I}_2) \rightarrow \mathbf{F}_{1\rightarrow 0}, \mathbf{F}_{1\rightarrow 2}, \mathbf{W}_0, \mathbf{W}_2. \quad (2)$$

Our architecture uses five  $2\times 2$  average pooling layers with stride 2 for the encoding and five bilinear upsampling layers to upscale the layers with a factor 2 in the decoding. We use four skip connections (addition) between layers in the encoder and decoder. It should be noted that our network is fully convolutional, which implies that it works on images of any size, where both dimensions are a multiple of 32. If this is not the case, we pad the image with boundary reflections.

Our model for frame interpolation is obtained by combining Equations (1) and (2) into

$$f(\mathbf{I}_0, \mathbf{I}_2) = \hat{\mathbf{I}}_1, \quad (3)$$

where  $\hat{\mathbf{I}}_1$  is the estimated image. The model is depicted in Figure 2. All components of  $f$  are differentiable, which means that our model is end-to-end trainable. It is easy to get data in the form of triplets  $(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2)$  by taking frames from videos that we use as training data for our model.

### 3.1 Loss-functions

We employ a number of loss functions to train our network. All of our loss functions are given for a single triplet  $(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2)$ , and the loss for a minibatch of triplets is simply the mean of the loss for each triplet. In the following paragraphs, we define the different loss functions that we employ.

*Reconstruction loss* models how well the network has reconstructed the missing frame:

$$\mathcal{L}_1 = \left\| \mathbf{I}_1 - \hat{\mathbf{I}}_1 \right\|_1. \quad (4)$$

*Bidirectional reconstruction loss* models how well each of our predicted optical flows is able to reconstruct the missing frame on its own:

$$\mathcal{L}_b = \left\| \mathbf{I}_1 - \mathcal{W}(\mathbf{F}_{1 \rightarrow 0}, \mathbf{I}_0) \right\|_1 + \left\| \mathbf{I}_1 - \mathcal{W}(\mathbf{F}_{1 \rightarrow 2}, \mathbf{I}_2) \right\|_1. \quad (5)$$

This has similarities to the work of Jiang et al. [6] but differs since the flow is estimated from the missing frame to the existing frames, and not between the existing frames.

*Feature loss* is introduced as an approximation of the perceptual similarity by comparing feature representation of the images from a pre-trained deep neural network [7]. Let  $\phi$  be the output of `relu4_4` from VGG19 [21], then

$$\mathcal{L}_f = \left\| \phi(\mathbf{I}_1) - \phi(\hat{\mathbf{I}}_1) \right\|_2^2. \quad (6)$$

*Smoothness loss* is a penalty on the absolute difference between neighboring pixels in the flow field. This encourages a smoother optical flow [6, 11]:

$$\mathcal{L}_s = \left\| \nabla \mathbf{F}_{1 \rightarrow 0} \right\|_1 + \left\| \nabla \mathbf{F}_{1 \rightarrow 2} \right\|_1, \quad (7)$$

where  $\left\| \nabla \mathbf{F} \right\|_1$  is the sum of the anisotropic total variation for each  $(x, y)$  component in the optical flow  $\mathbf{F}$ . For ease of notation, we introduce a linear combination of Equations (4) to (7):

$$\mathcal{L}_r(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2) = \lambda_1 \mathcal{L}_1 + \lambda_b \mathcal{L}_b + \lambda_f \mathcal{L}_f + \lambda_s \mathcal{L}_s. \quad (8)$$

Note that we explicitly express this as a function of a triplet. When this triplet is the three input images, we define

$$\mathcal{L}_\alpha = \mathcal{L}_r(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2). \quad (9)$$

Similarly, for ease of notation, let the bidirectional loss from Equation (5) be a function

$$\mathcal{L}_B(\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2, \mathbf{F}_{1 \rightarrow 0}, \mathbf{F}_{1 \rightarrow 2}) = \mathcal{L}_b \quad (10)$$

where, in this case,  $\mathbf{F}_{1 \rightarrow 0}$  and  $\mathbf{F}_{1 \rightarrow 2}$  are the flows predicted by the network.

*Pyramid loss* is a sum of bidirectional losses for downsampled versions of images and flow maps:

$$\mathcal{L}_p = \sum_{l=1}^{l=4} 4^l \mathcal{L}_B(A_l(\mathbf{I}_0), A_l(\mathbf{I}_1), A_l(\mathbf{I}_2), A_l(\mathbf{F}_{1 \rightarrow 0}), A_l(\mathbf{F}_{1 \rightarrow 2})), \quad (11)$$

where  $A_l$  is the  $2^l \times 2^l$  average pooling operator with stride  $2^l$ .

**Cyclic loss functions.** We can apply our model recursively to get another estimate of  $\mathbf{I}_1$ , namely

$$\tilde{\mathbf{I}}_1 = f\left(\hat{\mathbf{I}}_{0.5}, \hat{\mathbf{I}}_{1.5}\right) = f\left(f\left(\mathbf{I}_0, \mathbf{I}_1\right), f\left(\mathbf{I}_1, \mathbf{I}_2\right)\right). \quad (12)$$

*Cyclic loss* is introduced to ensure that outputs from the model work well as inputs to the model [10]. It is defined by

$$\mathcal{L}_c = \mathcal{L}_r\left(\hat{\mathbf{I}}_{0.5}, \mathbf{I}_1, \hat{\mathbf{I}}_{1.5}\right). \quad (13)$$

*Motion loss* is introduced in order to get extra supervision on the optical flow and utilizes the recursive nature of our network.

$$\mathcal{L}_m = \|\mathbf{F}_{1 \rightarrow 0} - 2\mathbf{F}_{1 \rightarrow 0.5}\|_2^2 + \|\mathbf{F}_{1 \rightarrow 2} - 2\mathbf{F}_{1 \rightarrow 1.5}\|_2^2 \quad (14)$$

This is introduced as self-supervision of the optical flow, under the assumption that the flow  $\mathbf{F}_{1 \rightarrow 0}$  is approximately twice that of  $\mathbf{F}_{1 \rightarrow 0.5}$  and similarly for  $\mathbf{F}_{1 \rightarrow 2}$  and  $\mathbf{F}_{1 \rightarrow 1.5}$ , and assuming that the flow is easier to learn for shorter time steps.

### 3.2 Training

We train our network using the assembled loss function

$$\mathcal{L} = \mathcal{L}_\alpha + \mathcal{L}_c + \lambda_m \mathcal{L}_m, \quad (15)$$

where  $\mathcal{L}_\alpha$ ,  $\mathcal{L}_c$  and  $\mathcal{L}_m$  are as defined in Equations (9), (13) and (14) with  $\lambda_r = 1$ ,  $\lambda_b = 1$ ,  $\lambda_f = 8/3$ ,  $\lambda_s = 10/3$  and  $\lambda_m = 1/192$ . The values have been selected based on the performance on a validation set.

We train our network using the Adam optimizer [8] with default values  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$  and with a minibatch size of 64.

Inspired by Liu et al. [10], we first train the network using only  $\mathcal{L}_\alpha$ . This is done on patches of size  $128 \times 128$  for 150 epochs with a learning rate of  $10^{-5}$ , followed by 50 epochs with a learning rate of  $10^{-6}$ . We then train with the full loss function  $\mathcal{L}$  on patches of size  $256 \times 256$  for 35 epochs with a learning rate of  $10^{-5}$  followed by 30 epochs with a learning rate of  $10^{-6}$ . We did not use batch-normalization, as it decreased performance on our validation set. Due to the presence of the cyclic loss functions, four forward and backward passes are needed for each minibatch during the training with the full loss function.

**Training data.** We train our network on triplets of patches extracted from consecutive video frames. For our training data, we downloaded 1500 videos in 4k from youtube.com/4k and resized them to  $1920 \times 1080$ . For every four frames in the video not containing a scene cut, we chose a random  $320 \times 320$  patch and cropped it from the first three frames. If any of these patches were too similar or if the mean absolute differences from the middle patch to the previous and following patches were too big, or too dissimilar, they were discarded to avoid patches that either had little motion or did not include the same object. Our final training set consists of 476,160 triplets.

**Data augmentation.** The data is augmented while we train the network by cropping a random patch from the  $320 \times 320$  data with size as specified in the training details in [Section 3.2](#). In this way, we use the training data more effectively. We also add a random translation to the flow between the patches, by offsetting the crop to the first and third patch while not moving the center patch [18]. This offset is  $\pm 5$  pixels in each direction. Furthermore, we also performed random horizontal flips and swapped the temporal order of the triplet.

### 3.3 Cyclic fine-tuning (CFT)

We introduce the concept of fine-tuning the model during inference for each frame that we want to interpolate and refer to this as cyclic fine-tuning (CFT). Recall that the cyclic loss  $\mathcal{L}_c$  measures how well the predicted frames are able to reconstruct the original frames. This gives an indication of the quality of the interpolated frames. The idea of CFT is to exploit this property at inference time to improve interpolation quality. We do this by extending the cyclic loss to  $\pm 2$  frames around the desired frame and fine-tuning the network using these images only.

When interpolating frame  $\mathbf{I}_{1.5}$ , we would use surrounding frames  $\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2$ , and  $\mathbf{I}_3$  to compute  $\hat{\mathbf{I}}_{0.5}, \hat{\mathbf{I}}_{1.5}$ , and  $\hat{\mathbf{I}}_{2.5}$ , which are then used to compute  $\tilde{\mathbf{I}}_1$  and  $\tilde{\mathbf{I}}_2$  as illustrated in [Figure 1](#) on page 2. Note that the desired interpolated frame  $\hat{\mathbf{I}}_{1.5}$  is used in the computation of both of the original frames. Therefore by fine-tuning of the network to improve the quality of the reconstructed original frames, we are improving the quality of the desired intermediate frame indirectly.

Specifically, we minimize the loss for each of the two triplets  $(\hat{\mathbf{I}}_{0.5}, \mathbf{I}_1, \hat{\mathbf{I}}_{1.5})$  and  $(\hat{\mathbf{I}}_{1.5}, \mathbf{I}_2, \hat{\mathbf{I}}_{2.5})$  with the loss for each triplet given by

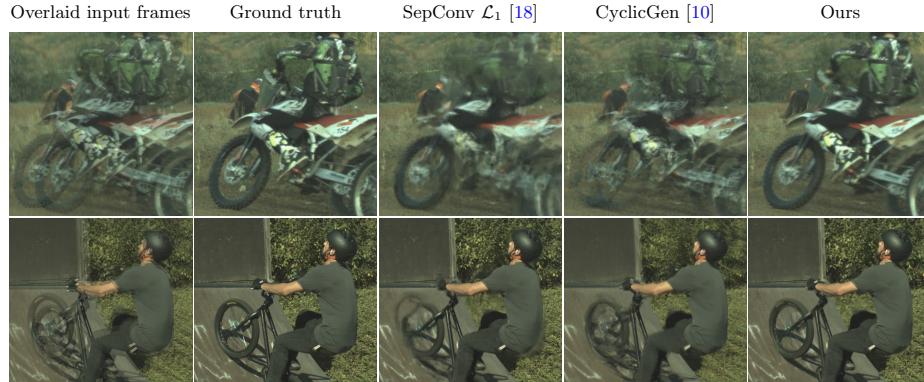
$$\mathcal{L}_{CFT} = \mathcal{L}_c + \lambda_p \mathcal{L}_p, \quad (16)$$

where  $\lambda_p = 10$ , and  $\mathcal{L}_p$  is the pyramid loss described in [Section 3.1](#). In order for the model to be presented with slightly different samples, we only do this fine-tuning on patches of  $256 \times 256$  with flow augmentation as described in the previous section. For computational efficiency, we only do this for 50 patch-triplets for each interpolated frame.

More than  $\pm 2$  frames can be applied for fine-tuning, however, we found that this did not increase performance. This fine-tuning process is not limited to our model and can be applied to any frame interpolation model taking two images as input and outputting one image.

## 4 Experiments

We evaluate variations of our method on three diverse datasets: UCF101 [22], SlowFlow [5] and See You Again [18]. These have previously been used for frame interpolation [6, 11, 18]. UCF101 contains different image sequences of a variety of actions, and we evaluate our method on the same frames as Liu et al. [11], but



**Fig. 4.** Qualitative examples on the SlowFlow dataset. Images shown are representative crops taken from the full images. Note that our method performs much better for large motions. The motion of the dirt bike is approximately 53 pixels, and the bike tire has a motion of 34 pixels.

**Table 1.** Overview of the datasets we used for evaluation.

Dataset	Number of sequences	Resolution	Number of frames		Avg. sequence length
			Interpolated	Total	
SlowFlow [5]	34	$1280 \times \{1024, 720\}$	17,871	20,458	602
See You Again	117	$1920 \times 1080$	2,503	5,355	46
UCF101 [22]	379	$256 \times 256$	379	1,137	3

did not use any motion masks as we are interested in performing equally well over the entire frame. SlowFlow is a high-fps dataset that we include to showcase our performance when predicting multiple in-between frames. For this dataset, we have only used every eighth frame as input and predicted the remaining seven in-between in a recursive manner. All frames in the dataset have been debayered, resized to 1280 pixels on the long edge and gamma corrected with a gamma value of 2.2. See You Again is a high-resolution music video, where we predict the even-numbered frames using the odd-numbered frames. Furthermore, we have divided it into sequences by removing scene changes. A summary of the datasets is shown in Table 1.

We have compared our method with multiple state-of-the-art methods [6, 10, 11, 18] which either have publicly available code and/or published their predicted frames. For the comparison with SepConv [18], we use the  $\mathcal{L}_1$  version of their network for which they report their best quantitative results. For each evaluation, we report the Peak Signal to Noise Ratio (PSNR) and the Structural Similarity Index (SSIM) [24].

**Table 2.** Interpolation results on SlowFlow, See You Again and UCF101. PSNR, SSIM: higher is better. Our baseline is our model trained only with  $\mathcal{L}_1 + \mathcal{L}_b + \mathcal{L}_s$  and constrained to symmetric flow. Elements are added to the model cumulatively. Larger patches means training on  $256 \times 256$  patches. Bold numbers signify that a method performs significantly better than the rest on that task with  $p < 0.02$ .

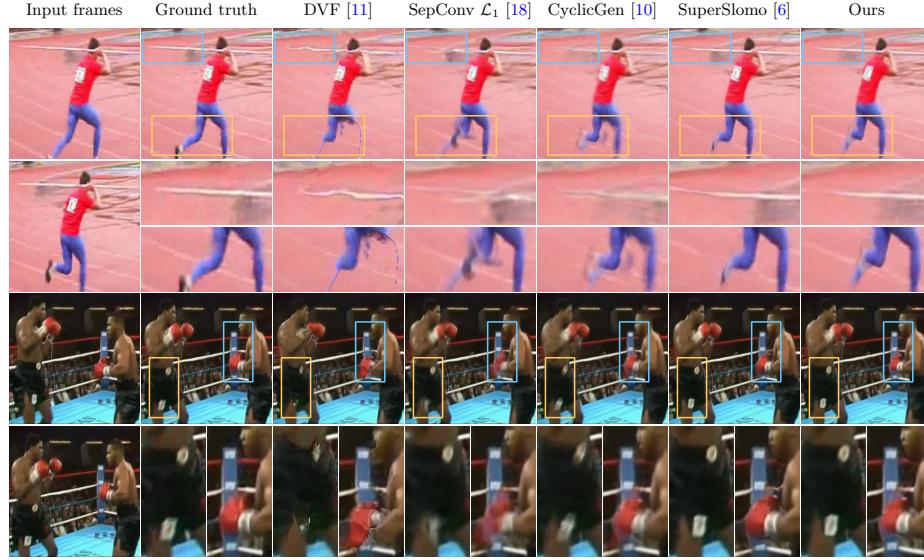
Method	SlowFlow		See You Again		UCF101	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
DVF [11]	-	-	-	-	34.12	0.941
SuperSloMo [6]	-	-	-	-	34.75	0.947
SepConv $\mathcal{L}_1$ [18]	34.03	0.899	42.49	0.983	34.78	0.947
CyclicGen [10]	31.33	0.839	41.28	0.975	<b>35.11</b>	<b>0.949</b>
Our baseline	34.28	0.903	42.50	0.984	34.39	0.946
+ asymmetric flow	34.33	0.904	42.54	0.985	34.40	0.946
+ feature loss	34.29	0.901	42.62	0.984	34.60	0.948
+ cyclic loss	34.33	0.900	42.73	0.984	34.62	0.947
+ motion loss	34.31	0.900	42.74	0.984	34.61	0.948
+ larger patches	34.60	0.907	43.14	0.986	34.69	0.948
+ CFT	<b>34.91</b>	<b>0.912</b>	<b>43.21</b>	<b>0.986</b>	34.94	<b>0.949</b>

**Comparison with state-of-the-art.** Table 2 shows that our best method, with or without CFT, clearly outperforms the other methods on SlowFlow and See you Again, which is also reflected in Figure 4. On UCF101 our best method performs better than all other methods except CyclicGen, where our best method has the same SSIM but lower PSNR. We suspect this is partly due to the fact that our CFT does not have access to  $\pm 2$  frames in all sequences. For some of the sequences, we had to use  $-1, +3$  as the intermediate frame was at the beginning of the sequence. Visually, our method produces better results as seen in Figure 5. We note that CyclicGen is trained on UCF101, and their much worse performance on the two other datasets could indicate overfitting.

**Effect of various model configurations.** Table 2 reveals that an asymmetric flow around the interpolated frame slightly improves performance on all three datasets as compared with enforcing a symmetric flow. There is no clear change in performance when we add feature loss, cyclic loss and motion loss.

For all three datasets, performance improves when we train on larger image patches. Using larger patches allows for the network to learn larger flows and the performance improvement is correspondingly seen most clearly in SlowFlow and See You Again which, as compared with UCF101, have much larger images with larger motions. We see a big performance improvement when cyclic fine-tuning is added, which is also clearly visible in Figure 6.

**Discussion.** Adding CFT to our model increases the run-time of our method by approximately 6.5 seconds per frame pair. This is not dependent on image



**Fig. 5.** Qualitative comparison for two sequences from UCF101. Top: Our method produces the least distorted javelin and retains the detailed lines on the track. All methods perform inaccurately on the leg, however, SuperSlomo and our method are the most visually plausible. Bottom: Our method and SuperSlomo create accurate white squares on the shorts (left box). Our method also produces the least distorted ropes and white squares on the corner post, while creating foreground similar to the ground truth (right box).

size, as we only fine-tune on  $256 \times 256$  patches for 50 iterations per frame pair. For reference, our method takes 0.08 seconds without CFT to interpolate a  $1920 \times 1080$  image on an NVIDIA GTX 1080 TI. It should be noted that CFT is only necessary to do once per frame pair in the original video, and thus there is no extra overhead when computing multiple in-between frames.

Training for more than 50 iterations does not necessarily ensure better results, as we can only optimize a proxy of the interpolation quality. The best number of iterations remains to be determined, but it is certainly dependent on the quality of the pre-training, the training parameters, and the specific video.

As of now, CFT should only be used if the target is purely interpolation quality. Improving the speed of CFT is a topic worthy of further investigation. Possible solutions of achieving similar results include training a network to learn the result of CFT, or training a network to predict the necessary weight changes.

## 5 Conclusion

We have proposed a CNN for video frame interpolation that predicts two optical flows with pixelwise weights from an unknown intermediate frame to the frames



**Fig. 6.** Representative example of how the cyclic fine-tuning improves the interpolated frame. It can be seen that the small misalignment of the tire and yellow “41” is corrected by the cyclic fine-tuning.

before and after. The flows are used to warp the input frames to the intermediate time step. These warped frames are then linearly combined using the weights to obtain the intermediate frame. We have trained our CNN using 1500 high-quality videos and shown that it performs better than or comparably to state-of-the-art methods across three different datasets. Furthermore, we have proposed a new strategy for fine-tuning frame interpolation methods for each specific frame at evaluation time. When used with our model, we have shown that it improves both the quantitative and visual results.

**Acknowledgements.** We would like to thank Joel Janai for providing us with the SlowFlow data [5].

## References

1. Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J., Szeliski, R.: A database and evaluation methodology for optical flow. *International Journal of Computer Vision* **92**(1), 1–31 (2011)
2. Castagno, R., Haavisto, P., Ramponi, G.: A method for motion adaptive frame rate up-conversion. *IEEE Transactions on Circuits and Systems for Video Technology* **6**(5), 436–446 (October 1996)
3. Catmull, E.: The problems of computer-assisted animation. *Computer Graphics (SIGGRAPH ’78)* **12**(3), 348–353 (August 1978)
4. Herbst, E., Seitz, S., Baker, S.: Occlusion reasoning for temporal interpolation using optical flow. Tech. rep., Microsoft Research (August 2009)
5. Janai, J., Güney, F., Wulff, J., Black, M., Geiger, A.: Slow Flow: Exploiting high-speed cameras for accurate and diverse optical flow reference data. In: *IEEE Conference on Computer Vision and Pattern Recognition*. pp. 1406–1416 (2017)
6. Jiang, H., Sun, D., Jampani, V., Yang, M.H., Learned-Miller, E., Kautz, J.: Super SloMo: High quality estimation of multiple intermediate frames for video interpolation. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 9000–9008 (2018)

7. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European Conference on Computer Vision (ECCV). pp. 694–711 (2016)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR [abs/1412.6980](https://arxiv.org/abs/1412.6980) (2014)
9. Lasseter, J.: Principles of traditional animation applied to 3D computer animation. Compututer Graphics (SIGGRAPH '87) **21**(4), 35–44 (August 1987)
10. Liu, Y.L., Liao, Y.T., Lin, Y.Y., Chuang, Y.Y.: Deep video frame interpolation using cyclic frame generation. In: AAAI Conference on Artificial Intelligence (2019)
11. Liu, Z., Yeh, R.A., Tang, X., Liu, Y., Agarwala, A.: Video frame synthesis using deep voxel flow. In: International Conference on Computer Vision. pp. 4463–4471 (2017)
12. Long, G., Kneip, L., Alvarez, J.M., Li, H., Zhang, X., Yu, Q.: Learning image matching by simply watching video. In: ECCV. pp. 434–450. Springer, Cham (2016)
13. Mahajan, D., Huang, F.C., Matusik, W., Ramamoorthi, R., Belhumeur, P.: Moving gradients: A path-based method for plausible image interpolation. ACM Transactions on Graphics **28**(3), 42:1–42:11 (July 2009)
14. Meyer, S., Djelouah, A., McWilliams, B., Sorkine-Hornung, A., Gross, M., Schroers, C.: Phasenet for video frame interpolation. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 498–507 (2018)
15. Meyer, S., Wang, O., Zimmer, H., Grosse, M., Sorkine-Hornung, A.: Phase-based frame interpolation for video. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1410–1418 (2015)
16. Niklaus, S., Liu, F.: Context-aware synthesis for video frame interpolation. In: Conference on Computer Vision and Pattern Recognition. pp. 1701–1710 (2018)
17. Niklaus, S., Mai, L., Liu, F.: Video frame interpolation via adaptive convolution. In: Conference on Computer Vision and Pattern Recognition. pp. 670–679 (2017)
18. Niklaus, S., Mai, L., Liu, F.: Video frame interpolation via adaptive separable convolution. In: International Conference on Computer Vision. pp. 261–270 (2017)
19. Reeves, W.T.: Inbetweening for computer animation utilizing moving point constraints. Computer Graphics (SIGGRAPH '81) **15**(3), 263–269 (August 1981)
20. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
21. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR [abs/1409.1556](https://arxiv.org/abs/1409.1556) (2014)
22. Soomro, K., Zamir, A.R., Shah, M., Soomro, K., Zamir, A.R., Shah, M.: UCF101: A dataset of 101 human actions classes from videos in the wild. CoRR [abs/1212.0402](https://arxiv.org/abs/1212.0402) (2012)
23. Szeliski, R.: Prediction error as a quality metric for motion and stereo. In: IEEE International Conference on Computer Vision (ICCV). vol. 2, pp. 781–788 (1999)
24. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing **13**(4), 600–612 (2004)
25. Werlberger, M., Pock, T., Unger, M., Bischof, H.: Optical flow guided TV-L<sup>1</sup> video interpolation and restoration. In: Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR). pp. 273–286. Springer (2011)