Portland State University

PDXScholar

Computer Science Faculty Publications and Presentations

Computer Science

3-1-2017

Video Frame Interpolation via Adaptive Convolution

Simon Niklaus
Portland State University, sniklaus@pdx.edu

Long Mai
Portland State University, mtlong@cs.pdx.edu

Feng Liu
Portland State University, fliu@pdx.edu

Follow this and additional works at: https://pdxscholar.library.pdx.edu/compsci_fac

 Part of the Computer Engineering Commons

Let us know how access to this document benefits you.

Citation Details

Niklaus, S., Mai, L., & Liu, F. (2017). Video Frame Interpolation via Adaptive Convolution. arXiv preprint arXiv:1703.07514.

This Pre-Print is brought to you for free and open access. It has been accepted for inclusion in Computer Science Faculty Publications and Presentations by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

arXiv:1703.07514v1 [cs.CV] 22 Mar 2017

Video Frame Interpolation via Adaptive Convolution

Simon Niklaus*
Portland State University
sniklaus@pdx.edu

Long Mai*
Portland State University
mtlong@cs.pdx.edu

Feng Liu
Portland State University
fliu@cs.pdx.edu

Abstract

Video frame interpolation typically involves two steps: motion estimation and pixel synthesis. Such a two-step approach heavily depends on the quality of motion estimation. This paper presents a robust video frame interpolation method that combines these two steps into a single process. Specifically, our method considers pixel synthesis for the interpolated frame as local convolution over two input frames. The convolution kernel captures both the local motion between the input frames and the coefficients for pixel synthesis. Our method employs a deep fully convolutional neural network to estimate a spatially-adaptive convolution kernel for each pixel. This deep neural network can be directly trained end to end using widely available video data without any difficult-to-obtain ground-truth data like optical flow. Our experiments show that the formulation of video interpolation as a single convolution process allows our method to gracefully handle challenges like occlusion, blur, and abrupt brightness change and enables high-quality video frame interpolation.

1. Introduction

Frame interpolation is a classic computer vision problem and is important for applications like novel view interpolation and frame rate conversion [36]. Traditional frame interpolation methods have two steps: motion estimation, usually optical flow, and pixel synthesis [1]. Optical flow is often difficult to estimate in the regions suffering from occlusion, blur, and abrupt brightness change. Flow-based pixel synthesis cannot reliably handle the occlusion problem. Failure of any of these two steps will lead to noticeable artifacts in interpolated video frames.

This paper presents a robust video frame interpolation method that achieves frame interpolation using a deep convolutional neural network without explicitly dividing it into separate steps. Our method considers pixel interpolation as convolution over corresponding image patches in the two input video frames, and estimates the spatially-adaptive convolutional kernel using a deep fully convolutional neural network. Specifically, for a pixel (x, y) in the interpolated frame, this deep neural network takes two receptive field patches R_1 and R_2 centered at that pixel as input and estimates a convolution kernel K. This convolution kernel is used to convolve with the input patches P_1 and P_2 to synthesize the output pixel, as illustrated in Figure 1.

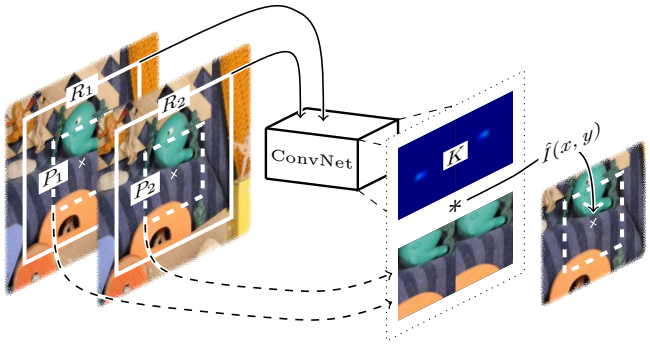


Figure 1: Pixel interpolation by convolution. For each output pixel (x, y), our method estimates a convolution kernel K and uses it to convolve with patches P_1 and P_2 centered at (x, y) in the input frames to produce its color $\hat{I}(x, y)$.

An important aspect of our method is the formulation of pixel interpolation as convolution over pixel patches instead of relying on optical flow. This convolution formulation unifies motion estimation and pixel synthesis into a single procedure. It enables us to design a deep fully convolutional neural network for video frame interpolation without dividing interpolation into separate steps. This formulation is also more flexible than those based on optical flow and can better handle challenging scenarios for frame interpolation. Furthermore, our neural network is able to estimate edge-aware convolution kernels that lead to sharp results.

The main contribution of this paper is a robust video frame interpolation method that employs a fully deep convolutional neural network to produce high-quality video interpolation results. This method has a few advantages. First, since it models video interpolation as a single process, it is able to make proper trade-offs among competing constraints and thus can provide a robust interpolation approach. Sec-

*The first two authors contributed equally to this paper.

http://graphics.cs.pdx.edu/project/adaconv

ond, this frame interpolation deep convolutional neural network can be directly trained end to end using widely available video data, without any difficult-to-obtain ground truth data like optical flow. Third, as demonstrated in our experiments, our method can generate high-quality frame interpolation results for challenging videos such as those with occlusion, blurring artifacts, and abrupt brightness change.

2. Related Work

Frame interpolation for video is one of the basic computer vision and video processing technologies. It is a special case of image-based rendering where middle frames are interpolated from temporally neighboring frames. Good surveys on image-based rendering are available [25, 44, 62]. This section focuses on research that is specific to video frame interpolation and our work.

Most existing frame interpolation methods estimate dense motion between two consecutive input frames using stereo matching or optical flow algorithms and then interpolate one or more middle frames according to the estimated dense correspondences [1, 53, 61]. Different from these methods, Mahajan *et al*. developed a moving gradient method that estimates paths in input images, copies proper gradients to each pixel in the frame to be interpolated and then synthesizes the interpolated frame via Poisson reconstruction [33]. The performance of all the above methods depends on the quality of dense correspondence estimation and special care needs to be taken to handle issues like occlusion during the late image synthesis step.

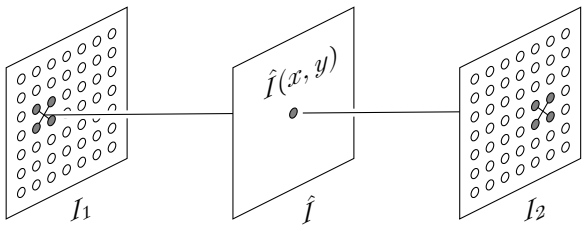
As an alternative to explicit motion estimation-based methods, phase-based methods have recently been shown promising for video processing. These methods encode motion in the phase difference between input frames and manipulate the phase information for applications like motion magnification [51] and view expansion [6]. Meyer *et al*. further extended these approaches to accommodate large motion by propagating phase information across oriented multi-scale pyramid levels using a bounded shift correction strategy [36]. This phase-based interpolation method can generate impressive video interpolation results and handle challenging scenarios gracefully; however, further improvement is still required to better preserve high-frequency detail in the video with large inter-frame changes.

Our work is inspired by the success of deep learning in solving not only difficult visual understanding problems [16, 20, 26, 28, 39, 40, 42, 45, 54, 60, 64] but also other computer vision problems like optical flow estimation [9, 14, 19, 48, 49, 52], style transfer [11, 15, 23, 30, 50], and image enhancement [3, 7, 8, 41, 43, 55, 57, 63, 66]. Our method is particularly relevant to the recent deep learning algorithms for view synthesis [10, 13, 24, 29, 47, 59, 65]. Dosovitskiy *et al*. [10], Kulkarni *et al*. [29], Yang *et al*. [59], and Tatarchenko *et al*. [47] developed deep learning algorithms that can render unseen views from input images. These algorithms work on objects, such as chairs and faces, and are not designed for frame interpolation for videos of general scenes.

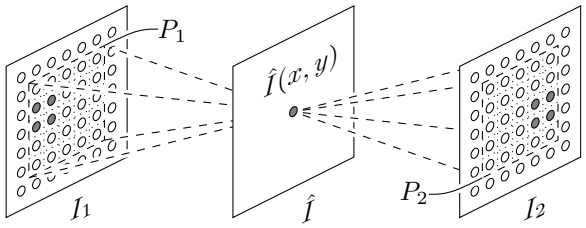
Recently, Flynn *et al*. developed a deep convolutional neural network method for synthesizing novel natural images from posed real-world input images. Their method projects input images onto multiple depth planes and combines colors at these depth planes to create a novel view [13]. Kalantari *et al*. provided a deep learning-based view synthesis algorithm for view expansion for light field imaging. They break novel synthesis into two components: disparity and color estimation, and accordingly use two sequential convolutional neural networks to model these two components. These two neural networks are trained simultaneously [24]. Long *et al*. interpolate frames as an intermediate step for image matching [31]. However, their interpolated frames tend to be blurry. Zhou *et al*. observed that the visual appearance of different views of the same instance is highly correlated, and designed a deep learning algorithm to predict appearance flows that are used to select proper pixels in the input views to synthesize a novel view [65]. Given multiple input views, their method can interpolate a novel view by warping individual input views using the corresponding appearance flows and then properly combining them together. Like these methods, our deep learning algorithm can also be trained end to end using videos directly. Compared to these methods, our method is dedicated to video frame interpolation. More importantly, our method estimates convolution kernels that capture both the motion and interpolation coefficients, and uses these kernels to directly convolve with input images to synthesize a middle video frame. Our method does not need to project input images onto multiple depth planes or explicitly estimate disparities or appearance flows to warp input images and then combine them together. Our experiments show that our formulation of frame interpolation as a single convolution step allows our method to robustly handle challenging cases. Finally, the idea of using convolution for image synthesis has also been explored in the very recent work for frame extrapolation [12, 22, 58].

3. Video Frame Interpolation

Given two video frames I_1 and I_2, our method aims to interpolate a frame $\hat{I}$ temporally in the middle of the two input frames. Traditional interpolation methods estimate the color of a pixel $\hat{I}(x, y)$ in the interpolated frame in two steps: dense motion estimation, typically through optical flow, and pixel interpolation. For instance, we can find for pixel (x, y) its corresponding pixels (x_1, y_1) in I_1 and (x_2, y_2) in I_2 and then interpolate the color from these corresponding pixels. Often this step also involves re-sampling images I_1 and I_2 to obtain the corresponding



(a) Interpolation by motion estimation and color interpolation



(b) Interpolation by convolution

Figure 2: Interpolation by convolution. (a): a two-step approach first estimates motion between two frames and then interpolates the pixel color based on the motion. (b): our method directly estimates a convolution kernel and uses it to convolve the two frames to interpolate the pixel color.

values $I_1(x_1, y_1)$ and $I_2(x_2, y_2)$ to produce a high-quality interpolation result, especially when (x_1, y_1) and (x_2, y_2) are not integer locations, as illustrated in Figure 2 (a). This two-step approach can be compromised when optical flow is not reliable due to occlusion, motion blur, and lack of texture. Also, rounding the coordinates to find the color for $I_1(x_1, y_1)$ and $I_2(x_2, y_2)$ is prone to aliasing while re-sampling with a fixed kernel sometimes cannot preserve sharp edges well. Advanced re-sampling methods exist and can be used for edge-preserving re-sampling, which, however, requires high-quality optical flow estimation.

Our solution is to combine motion estimation and pixel synthesis into a single step and formulate pixel interpolation as a local convolution over patches in the input images I_1 and I_2. As shown in Figure 2 (b), the color of pixel (x, y) in the target image to be interpolated can be obtained by convolving a proper kernel K over input patches $P_1(x, y)$ and $P_2(x, y)$ that are also centered at (x, y) in the respective input images. The convolutional kernel K captures both motion and re-sampling coefficients for pixel synthesis. This formulation of pixel interpolation as convolution has a few advantages. First of all, the combination of motion estimation and pixel synthesis into a single step provides a more robust solution than the two-step procedure. Second, the convolution kernel provides flexibility to account for and address difficult cases like occlusion. For example, optical flow estimation in an occlusion region is a fundamentally difficult problem, which makes it difficult for a typical two-step approach to proceed. Extra steps based on heuristics, such as flow interpolation, must be taken. This paper provides a data-driven approach to directly estimate the convolution kernel that can produce visually plausible interpolation results for an occluded region. Third, if properly estimated, this convolution formulation can seamlessly integrate advanced re-sampling techniques like edge-aware filtering to provide sharp interpolation results.

type	BN	ReLU	size	stride	output
input	-	-	-	-	$6 \times 79 \times 79$
conv	✓	✓	7×7	1×1	$32 \times 73 \times 73$
down-conv	-	✓	2×2	2×2	$32 \times 36 \times 36$
conv	✓	✓	5×5	1×1	$64 \times 32 \times 32$
down-conv	-	✓	2×2	2×2	$64 \times 16 \times 16$
conv	✓	✓	5×5	1×1	$128 \times 12 \times 12$
down-conv	-	✓	2×2	2×2	$128 \times 6 \times 6$
conv	✓	✓	3×3	1×1	$256 \times 4 \times 4$
conv	-	✓	4×4	1×1	$2048 \times 1 \times 1$
conv	-	-	1×1	1×1	$3362 \times 1 \times 1$
spatial softmax	-	-	-	-	$3362 \times 1 \times 1$
output	-	-	-	-	$41 \times 82 \times 1 \times 1$

Table 1: The convolutional neural network architecture. It makes use of Batch Normalization (BN) [21] as well as Rectified Linear Units (ReLU). Note that the output only reshapes the result without altering its value.

Estimating proper convolution kernels is essential for our method. Encouraged by the success of using deep learning algorithms for optical flow estimation [9, 14, 19, 48, 49, 52] and image synthesis [13, 24, 65], we develop a deep convolutional neural network method to estimate a proper convolutional kernel to synthesize each output pixel in the interpolated images. The convolutional kernels for individual pixels vary according to the local motion and image structure to provide high-quality interpolation results. Below we describe our deep neural network for kernel estimation and then discuss implementation details.

3.1. Convolution kernel estimation

We design a fully convolutional neural network to estimate the convolution kernels for individual output pixels. The architecture of our neural network is detailed in Table 1. Specifically, to estimate the convolutional kernel K for the output pixel (x, y), our neural network takes receptive field patches $R_1(x, y)$ and $R_2(x, y)$ as input. $R_1(x, y)$ and $R_2(x, y)$ are both centered at (x, y) in the respective input images. The patches P_1 and P_2 that the output kernel will convolve in order to produce the color for the output pixel (x, y) are co-centered at the same locations as these receptive fields, but with a smaller size, as illustrated in Figure 1. We use a larger receptive field than the patch to better handle the aperture problem in motion estimation. In our implementation, the default receptive field size is 79×79 pixels. The convolution patch size is 41×41 and the kernel size is 41×82 as it is used to convolve with two patches. Our method applies the same convolution kernel to each of



color loss color loss + gradient loss

Figure 3: Effect of using an additional gradient loss.

the three color channels.

As shown in Table 1, our convolutional neural network consists of several convolutional layers as well as down-convolutions as alternatives to max-pooling layers. We use Rectified Linear Units as activation functions and Batch Normalization [21] for regularization. We employ no further techniques for regularization since our neural network can be trained end to end using widely available video data, which provides a sufficiently large training dataset. We are also able to make use of data augmentation extensively, by horizontally and vertically flipping the training samples as well as reversing their order. Our neural network is fully convolutional. Therefore, it is not restricted to a fixed-size input and we are, as detailed in Section 3.3, able to use a shift-and-stitch technique [17, 32, 39] to produce kernels for multiple pixels simultaneously to speedup our method.

A critical constraint is that the coefficients of the output convolution kernel should be non-negative and sum up to one. Therefore, we connect the final convolutional layer to a spatial softmax layer to output the convolution kernel, which implicitly meets this important constraint.

3.1.1 Loss function

For clarity, we first define notations. The i^{th} training example consists of two input receptive field patches $R_{i,1}$ and $R_{i,2}$ centered at (x_i, y_i), the corresponding input patches $P_{i,1}$ and $P_{i,2}$ that are smaller than the receptive field patches and also centered at the same location, the ground-truth color $\tilde{C}_i$ and the ground-truth gradient $\tilde{G}_i$ at (x_i, y_i) in the interpolated frame. For simplicity, we omit the (x_i, y_i) in our definition of the loss functions.

One possible loss function of our deep convolutional neural network can be the difference between the interpolated pixel color and the ground-truth color as follows.

$$E_c = \sum_i \|[P_{i,1}\ P_{i,2}] * K_i - \tilde{C}_i\|_1 \qquad (1)$$

where subscript i indicates the i^{th} training example and K_i is the convolution kernel output by our neural network. Our experiments show that this color loss alone, even using ℓ_1 norm, can lead to blurry results, as shown in Figure 3. This blurriness problem was also reported in some recent work [31, 34, 38]. Mathieu *et al.* showed that this blurriness problem can be alleviated by incorporating image gradients in the loss function [34]. This is difficult within our pixel-wise interpolation approach, since the image gradient cannot be directly calculated from a single pixel. Since differentiation is also a convolution, assuming that kernels locally vary slowly, we solve this problem by using the associative property of convolution: we first compute the gradient of input patches and then perform convolution with the estimated kernel, which will result in the gradient of the interpolated image at the pixel of interest. As a pixel (x, y) has eight immediate neighboring pixels, we compute eight versions of gradients using finite difference and incorporate all of them into our gradient loss function.

$$E_g = \sum_i \sum_{k=1}^{8} \|[G^k_{i,1}\ G^k_{i,2}] * K_i - \tilde{G}^k_i\|_1 \qquad (2)$$

where k denotes one of the eight ways we compute the gradient. $G^k_{i,1}$ and $G^k_{i,2}$ are the gradients of the input patches $P_{i,1}$ and $P_{i,2}$, and $\tilde{G}^k_i$ is the ground-truth gradient. We combine the above color and gradient loss as our final loss $E_c + \lambda \cdot E_g$. We found that $\lambda = 1$ works well and used it. As shown in Figure 3, this color plus gradient loss enables our method to produce sharper interpolation results.

3.2. Training

We derived our training dataset from an online video collection, as detailed later on in this section. To train our neural network, we initialize its parameters using the Xavier initialization approach [18] and then use AdaMax [27] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, a learning rate of 0.001 and 128 samples per mini-batch to minimize the loss function.

3.2.1 Training dataset

Our loss function is purely based on the ground truth video frame and does not need any other ground truth information like optical flow. Therefore, we can make use of videos that are widely available online to train our neural network. To make it easy to reproduce our results, we use publicly available videos from Flickr with a Creative Commons license. We downloaded $3,000$ videos using keywords, such as "driving", "dancing", "surfing", "riding", and "skiing", which yield a diverse selection. We scaled the downloaded videos to a fixed size of 1280×720 pixels. We removed interlaced videos that sometimes have a lower quality than the videos with the progressive-scan format.

To generate the training samples, we group all the frames in each of the remaining videos into triple-frame groups, each containing three consecutive frames in a video. We then randomly pick a pixel in each triple-frame group and extract a triple-patch group centered at that pixel from the video frames. To facilitate data augmentation, the patches are selected to be larger than the receptive-field patches required by the neural network. The patch size in our training

dataset is 150×150 pixels. To avoid including a large number of samples with no or little motion, we estimate the optical flow between patches from the first and last frame in the triple-frame group [46] and compute the mean flow magnitude. We then sample $500,000$ triple-patch groups without replacement according to the flow magnitude: a patch group with larger motion is more likely to be chosen than the one with smaller motion. In this way, our training set includes samples with a wide range of motion while avoiding being dominated by patches with little motion. Since some videos consist of many shots, we compute the color histogram between patches to detect shot boundaries and remove the groups across the shot boundaries. Furthermore, samples with little texture are also not very useful to train our neural network. We therefore compute the entropy of patches in each sample and finally select the $250,000$ triple-patch groups with the largest entropy to form the training dataset. In this training dataset, about 10 percent of the pixels have an estimated flow magnitude of at least 20 pixels. The average magnitude of the largest five percent is approximately 25 pixels and the largest magnitude is 38 pixels.

We perform data augmentation on the fly during training. The receptive-field size required for the neural network is 79×79, which is smaller than the patch size in the training samples. Therefore, during the training, we randomly crop the receptive field patch from each training sample. We furthermore randomly flip the samples horizontally as well as vertically and randomly swap their temporal order. This forces the optical flow within the samples to be distributed symmetrically so that the neural network is not biased towards a certain direction.

3.3. Implementation details

We used Torch [5] to implemented our neural network. Below we describe some important details.

3.3.1 Shift-and-stitch implementation

A straightforward way to apply our neural network to frame interpolation is to estimate the convolution kernel and synthesize the interpolated pixel one by one. This pixel-wise application of our neural network will unnecessarily perform redundant computations when passing two neighboring pairs of patches through the neural network to estimate the convolution kernels for two corresponding pixels. Our implementation employs the shift-and-stitch approach to address this problem to speedup our system [17, 32, 39].

Specifically, as our neural network is fully convolutional and does not require a fixed-size input, it can compute kernels for more than one output pixels at once by supplying a larger input than what is required to produce one kernel. This can mitigate the issue of redundant computations. The output pixels that are obtained in this way are however not adjacent and are instead sparsely distributed. We employ the shift-and-stitch [17, 32, 39] approach in which slightly shifted versions of the same input are used. This approach returns sparse results that can be combined to form the dense representation of the interpolated frame.

Considering a frame with size 1280×720, a pixel-wise implementation of our neural network would require 921,600 forward passes through our neural network. The shift-and-stitch implementation of our neural network only requires 64 forward passes for the 64 differently shifted versions of the input to cope with the downscaling by the three down-convolutions. Compared to the pixel-wise implementation that takes 104 seconds per frame on an Nvidia Titan X, the shift-and-stitch implementation only takes 9 seconds.

3.3.2 Boundary handling

Due to the receptive field of the network as well as the size of the convolution kernel, we need to pad the input frames to synthesize boundary pixels for the interpolated frame. In our implementation, we adopt zero-padding. Our experiments show that this approach usually works well and does not introduce noticeable artifacts.

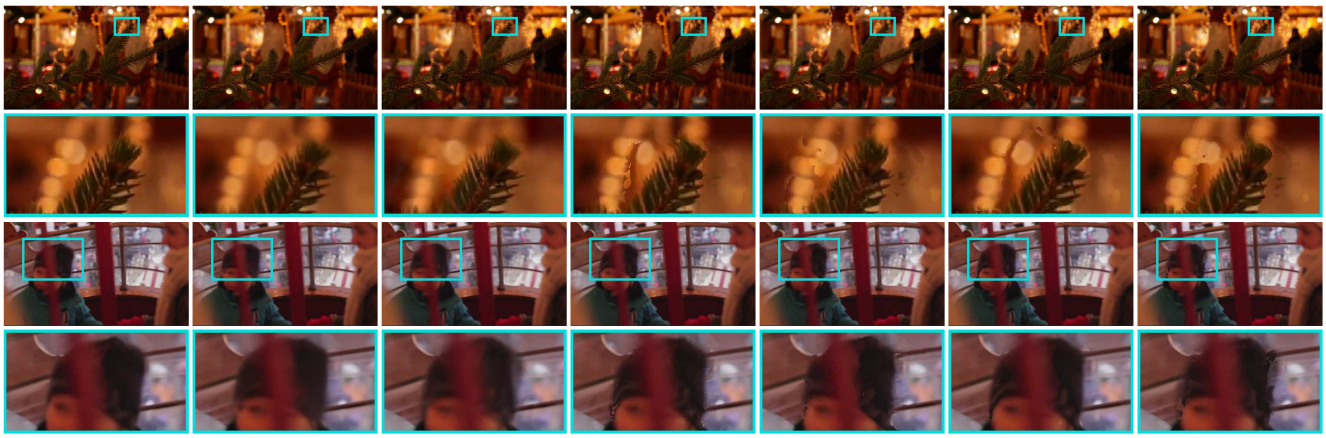
3.3.3 Hyper-parameter selection

The convolution kernel size and the receptive field size are two important hyper-parameters of our deep neural network. In theory, the convolution kernel, as shown in Figure 2, must be larger than the pixel motion between two frames in order to capture the motion (implicitly) to produce a good interpolation result. To make our neural network robust against large motion, we tend to choose a large kernel. On the other hand, a large kernel involves a large number of values to be estimated, which increases the complexity of our neural network. We choose to select a convolution kernel that is large enough to capture the largest motion in the training dataset, which is 38 pixels. Particularly, the convolution kernel size in our system is 41×82 that will be applied to two 41×41 patches as illustrated in Figure 1. We make this kernel a few pixels larger than 38 pixels to provide pixel support for re-sampling, which our method does not explicitly perform, but is captured in the kernel.

As discussed earlier, the receptive field is larger than the convolution kernel to handle the aperture problem well. However, a larger receptive field requires more computation and is less sensitive to the motion. We choose the receptive field using a validation dataset and find that 79×79 achieves a good balance.

4. Experiments

We compare our method to state-of-the-art video frame interpolation methods, including the recent phase-based interpolation method [36] and a few optical flow-based methods. The optical flow algorithms in our experiment include



Input frame 1 | Ours | Meyer *et al.* | DeepFlow2 | FlowNetS | MDP-Flow2 | Brox *et al.*

Figure 4: Qualitative evaluation on blurry videos.

MDP-Flow2 [56], which currently produces the lowest interpolation error according to the Middlebury benchmark, the method from Brox *et al.* [2], as well as two recent deep learning based approaches, namely DeepFlow2 [52] and FlowNetS [9]. Following recent frame interpolation work [36], we use the interpolation method from the Middlebury benchmark [1] to synthesize the interpolated frame using the optical flow results. Alternatively, other advanced image-based rendering algorithms [67] can also be used. For the two deep learning-based optical flow methods, we directly use the trained models from the author websites.

4.1. Comparisons

We evaluate our method quantitatively on the Middlebury optical flow benchmark [1]. As reported in Table 2, our method performs very well on the four examples with real-world scenes. Among the over 100 methods reported in the Middlebury benchmark, our method achieves the best on Evergreen and Basketball, 2nd best on Dumptruck, and 3rd best on Backyard. Our method does not work as well on the other four examples that are either synthetic or of lab scenes, partially because we train our network on videos with real-world scenes. Qualitatively, we find that our method can often create results in challenging regions that are visually more appealing than state-of-the-art methods.

Blur. Figure 4 shows two examples where the input videos suffer from out-of-focus blur (top) and motion blur (bottom). Blurry regions are often challenging for optical flow estimation; thus these regions in the interpolated results suffer from noticeable artifacts. Both our method and the phase-based method from Meyer *et al.* [36] can handle blurry regions better while our method produces sharper images, especially in regions with large motion, such as the right side of the hat in the bottom example.

Abrupt brightness change. As shown in Figure 5, abrupt brightness change violates the brightness consistency assumption and compromises optical flow estimation, causing artifacts in frame interpolation. For this example, our method and the phase-based method generate more visually appealing interpolation results than flow-based methods.

	Mequ.	Schef.	Urban	Teddy	Backy.	Baske.	Dumpt.	Everg.
Ours	3.57	4.34	5.00	6.91	**10.2**	**5.33**	**7.30**	**6.94**
DeepFlow2	2.99	3.88	**3.62**	5.38	11.0	5.83	7.60	7.82
FlowNetS	3.07	4.57	4.01	5.55	11.3	5.99	8.63	7.70
MDP-Flow2	**2.89**	**3.47**	3.66	**5.20**	**10.2**	6.13	7.36	7.75
Brox *et al.*	3.08	3.83	3.93	5.32	10.6	6.60	8.61	7.43

Table 2: Evaluation on the Middlebury testing set (average interpolation error).

Occlusion. One of the biggest challenges for optical flow estimation is occlusion. When optical flow is not reliable or unavailable in occluded regions, frame interpolation methods need to fill in holes, such as by interpolating flow from neighboring pixels [1]. Our method adopts a learning approach to obtain proper convolution kernels that lead to visually appealing pixel synthesis results for occluded regions, as shown in Figure 6.

To better understand how our method handles occlusion, we examine the convolution kernels of pixels in the occluded regions. As shown in Figure 1, a convolution kernel can be divided into two sub-kernels, each of which is used to convolve with one of the two input patches. For the ease of illustration, we compute the centroid of each sub-kernel and mark it using **x** in the corresponding input patch to indicate where the output pixel gets its color. Figure 7 shows an example where the white leaf moves up from Frame 1 to Frame 2. The occlusion can be seen in the left image that overlays two input frames. For this example, the pixel indicated by the green **x** is visible in both frames and our kernel shows that the color of this pixel is interpolated from both frames. In contrast, the pixel indicated by the red **x** is visible only in Frame 2. We find that the sum of all the coefficients in the sub-kernel for Frame 1 is almost zero, which indicates Frame 1 does not contribute to this pixel and this

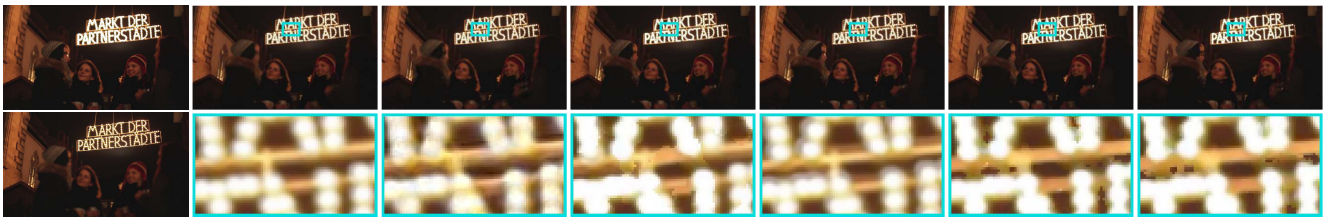


Figure 5: Qualitative evaluation on video with abrupt brightness change.

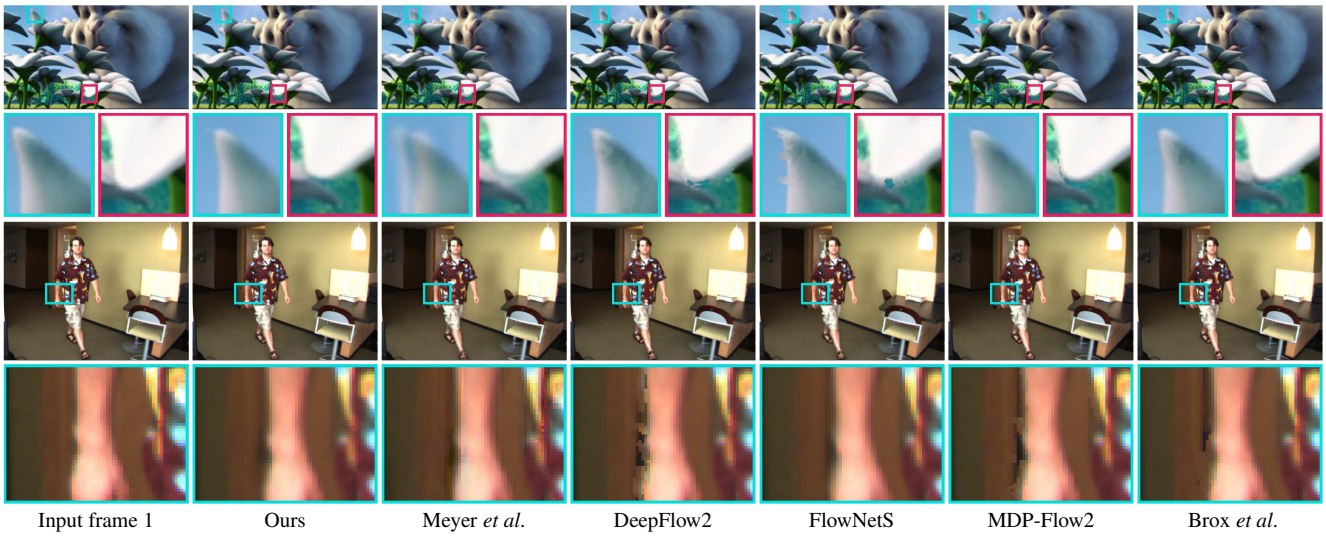


Figure 6: Qualitative evaluation with respect to occlusion.



Figure 7: Occlusion handling.

pixel gets its color only from Frame 2. Similarly, the pixel indicated by the cyan **x** is only visible in Frame 1. Our kernel correctly accounts for this occlusion and gets its color from Frame 1 only.

4.2. Edge-aware pixel interpolation

In the above, we discussed how our estimated convolution kernels appropriately handle occlusion for frame interpolation. We now examine how these kernels adapt to image features. In Figure 8, we sample three pixels in the interpolated image. We show their kernels at the bottom. The correspondence between a pixel and its convolution kernel is established by color. First, for all these kernels, only a very small number of kernel elements have non-zero values. (The use of the spatial softmax layer in our neural network already guarantees that the kernel element values are non-negative and sum up to one.) Furthermore, all these non-zero elements are spatially grouped together. This corresponds well with a typical flow-based interpolation method

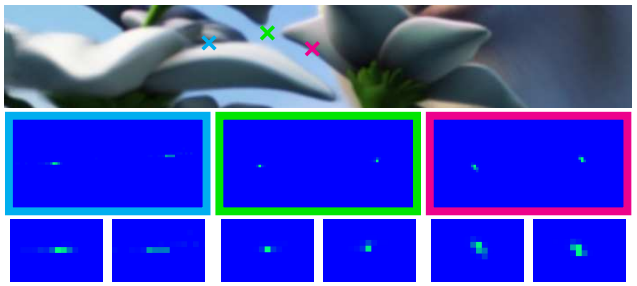
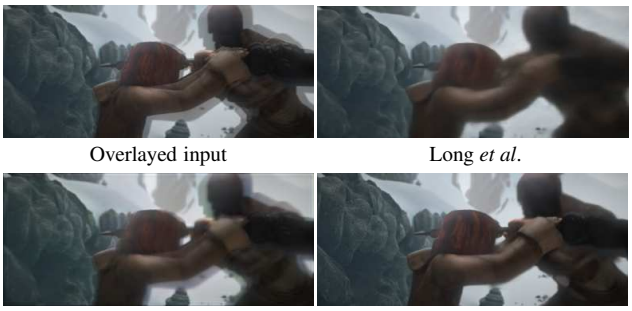


Figure 8: Convolution kernels. The third row provides magnified views into the non-zero regions in the kernels in the second row. While our neural network does not explicitly model the frame interpolation procedure, it is able to estimate convolution kernels that enable similar pixel interpolation to the flow-based interpolation methods. More importantly, our kernels are spatially adaptive and edge-aware, such as those for the pixels indicated by the red and cyan **x**.

that finds corresponding pixels or their neighborhood in two frames and then interpolate. Second, for a pixel in a flat region such as the one indicated by the green **x**, its kernel only has two elements with significant values. Each of these two kernel elements corresponds to the relevant pixel in the corresponding input frame. This is also consistent with the flow-based interpolation methods although our neural network does not explicitly model the frame in-



Overlayed input | Long *et al.*

Direct | Ours

Figure 9: Comparison with direct synthesis.

terpolation procedure. Third, more interestingly, for pixels along image edges, such as the ones indicated by the red and cyan **x**, the kernels are anisotropic and their orientations align well with the edge directions. This shows that our neural network learns to estimate convolution kernels that enable edge-aware pixel interpolation, which is critical to produce sharp interpolation results.

4.3. Discussion

Our method is scalable to large images due to its pixel-wise nature. Furthermore, the shift-and-stitch implementation of our neural network allows us to both parallel processing multiple pixels and reduce the redundancy in computing the convolution kernels for these pixels. On a single Nvidia Titan X, this implementation takes about 2.8 seconds with 3.5 gigabytes of memory for a 640×480 image, and 9.1 seconds with 4.7 gigabytes for 1280×720, and 21.6 seconds with 6.8 gigabytes for 1920×1080.

We experimented with a baseline neural network by modifying our network to directly synthesize pixels. We found that this baseline produces a blurry result for an example from the Sintel benchmark [4], as shown in Figure 9. In the same figure, we furthermore show a comparison with the method from Long *et al.* [31] that performs video frame interpolation as an intermediate step for optical flow estimation. While their result is better than our baseline, it is still not as sharp as ours.

The amount of motion that our method can handle is necessarily limited by the convolution kernel size in our neural network, which is currently 41×82. As shown in Figure 10, our method can handle motion within 41 pixels well. However, any large motion beyond 41 pixels, cannot currently be handled by our system. Figure 11 shows a pair of stereo image from the KITTI benchmark [35]. When using our method to interpolate a middle frame between the left and right view, the car is blurred due to the large disparity (over 41 pixels), as shown in (c). After downscaling the input images to half of their original size, our method interpolates well, as shown in (d). In the future, we plan to address this issue by exploring multi-scale strategies, such as those used for optical flow estimation [37].

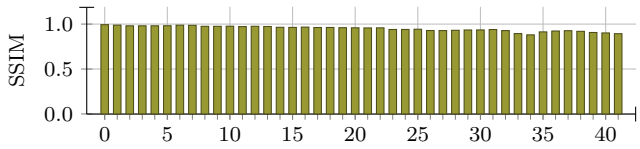
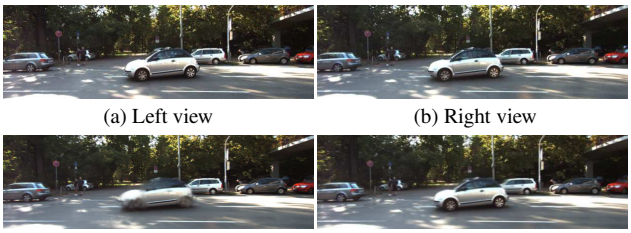


Figure 10: Interpolation quality of our method with respect to the flow magnitude (pixels).



(a) Left view | (b) Right view

(c) Ours - full resolution | (d) Ours - half resolution

Figure 11: Interpolation of a stereo image.

Unlike optical flow- or phased-based methods, our method is currently only able to interpolate a single frame between two given frames as our neural network is trained to interpolate the middle frame. While we can continue the synthesis recursively to also interpolate frames at $t = 0.25$ and $t = 0.75$ for example, our method is unable to interpolate a frame at an arbitrary time. It will be interesting to borrow from recent work for view synthesis [10, 24, 29, 47, 65] and extend our neural network such that it can take a variable as input to control the temporal step of the interpolation in order to interpolate an arbitrary number of frames like flow- or phase-based methods.

5. Conclusion

This paper presents a video frame interpolation method that combines the two steps of a frame interpolation algorithm, motion estimation and pixel interpolation, into a single step of local convolution with two input frames. The convolution kernel captures both the motion information and re-sampling coefficients for proper pixel interpolation. We develop a deep fully convolutional neural network that is able to estimate spatially-adaptive convolution kernels that allow for edge-aware pixel synthesis to produce sharp interpolation results. This neural network can be trained directly from widely available video data. Our experiments show that our method enables high-quality frame interpolation and handles challenging cases like occlusion, blur, and abrupt brightness change well.

Acknowledgments. The top image in Figure 4 is used with permission from Rafael McStan while the other images in Figures 4, 5, 6 are used under a Creative Commons license from the Blender Foundation and the city of Nuremberg. We thank Nvidia for their GPU donation. This work was supported by NSF IIS-1321119.

References

[1] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011. 1, 2, 6

[2] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision*, volume 3024, pages 25–36, 2004. 6

[3] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2392–2399, 2012. 2

[4] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision*, volume 7577, pages 611–625, 2012. 8

[5] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011. 5

[6] P. Didyk, P. Sitthi-amorn, W. T. Freeman, F. Durand, and W. Matusik. Joint view expansion and filtering for automultiscopic 3D displays. *ACM Trans. Graph.*, 32(6):221:1–221:8, 2013. 2

[7] C. Dong, Y. Deng, C. C. Loy, and X. Tang. Compression artifacts reduction by a deep convolutional network. In *ICCV*, pages 576–584, 2015. 2

[8] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016. 2

[9] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *ICCV*, pages 2758–2766, 2015. 2, 3, 6

[10] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1538–1546, 2015. 2, 8

[11] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. *arXiv/1610.07629*, 2016. 2

[12] C. Finn, I. J. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *NIPS*, pages 64–72, 2016. 2

[13] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. DeepStereo: Learning to predict new views from the world's imagery. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5515–5524, 2016. 2, 3

[14] D. Gadot and L. Wolf. PatchBatch: A batch augmented loss for optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4236–4245, 2016. 2, 3

[15] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016. 2

[16] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. 2

[17] A. Giusti, D. C. Ciresan, J. Masci, L. M. Gambardella, and J. Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. In *ICIP*, pages 4034–4038, 2013. 4, 5

[18] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256, 2010. 4

[19] F. Güney and A. Geiger. Deep discrete flow. In *Asian Conference on Computer Vision*, volume 10114, pages 207–224, 2016. 2, 3

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 2

[21] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37, pages 448–456, 2015. 3, 4

[22] X. Jia, B. D. Brabandere, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. In *NIPS*, pages 667–675, 2016. 2

[23] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, volume 9906, pages 694–711, 2016. 2

[24] N. K. Kalantari, T. Wang, and R. Ramamoorthi. Learning-based view synthesis for light field cameras. *ACM Trans. Graph.*, 35(6):193:1–193:10, 2016. 2, 3, 8

[25] S. B. Kang, Y. Li, X. Tong, and H. Shum. Image-based rendering. *Foundations and Trends in Computer Graphics and Vision*, 2(3), 2006. 2

[26] S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell, A. Hertzmann, and H. Winnemoeller. Recognizing image style. In *British Machine Vision Conference*, 2014. 2

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014. 4

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 2

[29] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. B. Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, pages 2539–2547, 2015. 2, 8

[30] C. Li and M. Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2479–2486, 2016. 2

[31] G. Long, L. Kneip, J. M. Alvarez, H. Li, X. Zhang, and Q. Yu. Learning image matching by simply watching video. In *European Conference on Computer Vision*, volume 9910, pages 434–450, 2016. 2, 4, 8

[32] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 4, 5

[33] D. Mahajan, F. Huang, W. Matusik, R. Ramamoorthi, and P. N. Belhumeur. Moving gradients: A path-based method for plausible image interpolation. *ACM Trans. Graph.*, 28(3):42:1–42:11, 2009. 2

[34] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations*, 2016. 4

[35] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3061–3070, 2015. 8

[36] S. Meyer, O. Wang, H. Zimmer, M. Grosse, and A. Sorkine-Hornung. Phase-based frame interpolation for video. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1410–1418, 2015. 1, 2, 5, 6

[37] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. *arXiv/1611.00850*, 2016. 8

[38] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv/1412.6604*, 2014. 4

[39] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations*, 2013. 2, 4, 5

[40] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv/1409.1556*, 2014. 2

[41] J. Sun, W. Cao, Z. Xu, and J. Ponce. Learning a convolutional neural network for non-uniform motion blur removal. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 769–777, 2015. 2

[42] Y. Sun, X. Wang, and X. Tang. Deeply learned face representations are sparse, selective, and robust. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2892–2900, 2015. 2

[43] P. Svoboda, M. Hradis, D. Barina, and P. Zemcík. Compression artifacts removal using convolutional neural networks. *arXiv/1605.00366*, 2016. 2

[44] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010. 2

[45] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. DeepFace: Closing the gap to human-level performance in face verification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014. 2

[46] M. W. Tao, J. Bai, P. Kohli, and S. Paris. SimpleFlow: A non-iterative, sublinear optical flow algorithm. *Computer Graphics Forum*, 31(2):345–353, 2012. 5

[47] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3D models from single images with a convolutional network. In *European Conference on Computer Vision*, volume 9911, pages 322–337, 2016. 2, 8

[48] D. Teney and M. Hebert. Learning to extract motion from videos in convolutional neural networks. *arXiv:1601.07532*, 2016. 2, 3

[49] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Deep End2End Voxel2Voxel prediction. In *CVPR Workshops*, pages 402–409, 2016. 2, 3

[50] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 48, pages 1349–1357, 2016. 2

[51] N. Wadhwa, M. Rubinstein, F. Durand, and W. T. Freeman. Phase-based video motion processing. *ACM Trans. Graph.*, 32(4):80:1–80:10, 2013. 2

[52] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. DeepFlow: Large displacement optical flow with deep matching. In *IEEE Intenational Conference on Computer Vision*, pages 1385–1392, 2013. 2, 3, 6

[53] M. Werlberger, T. Pock, M. Unger, and H. Bischof. Optical flow guided TV-L 1 video interpolation and restoration. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 6819, pages 273–286, 2011. 2

[54] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015. 2

[55] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems*, pages 350–358, 2012. 2

[56] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1744–1757, 2012. 6

[57] L. Xu, J. S. J. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In *NIPS*, pages 1790–1798, 2014. 2

[58] T. Xue, J. Wu, K. L. Bouman, and B. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NIPS*, pages 91–99, 2016. 2

[59] J. Yang, S. E. Reed, M. Yang, and H. Lee. Weakly-supervised disentangling with recurrent transformations for 3D view synthesis. In *NIPS*, pages 1099–1107, 2015. 2

[60] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, pages 3320–3328, 2014. 2

[61] Z. Yu, H. Li, Z. Wang, Z. Hu, and C. W. Chen. Multi-level video frame interpolation: Exploiting the interaction among different levels. *IEEE Trans. Circuits Syst. Video Techn.*, 23(7):1235–1248, 2013. 2

[62] C. Zhang and T. Chen. A survey on image-based rendering - representation, sampling and compression. *Signal Processing: Image Communication*, 19(1):1–28, 2004. 2

[63] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European Conference on Computer Vision*, volume 9907, pages 649–666, 2016. 2

[64] B. Zhou, À. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *NIPS*, pages 487–495, 2014. 2

[65] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros. View synthesis by appearance flow. In *ECCV*, volume 9908, pages 286–301, 2016. 2, 3, 8

[66] J. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, volume 9909, pages 597–613, 2016. 2

[67] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. A. J. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23(3):600–608, 2004. 6