Learning to Synthesize Motion Blur

Tim Brooks Jonathan T. Barron
Google Research

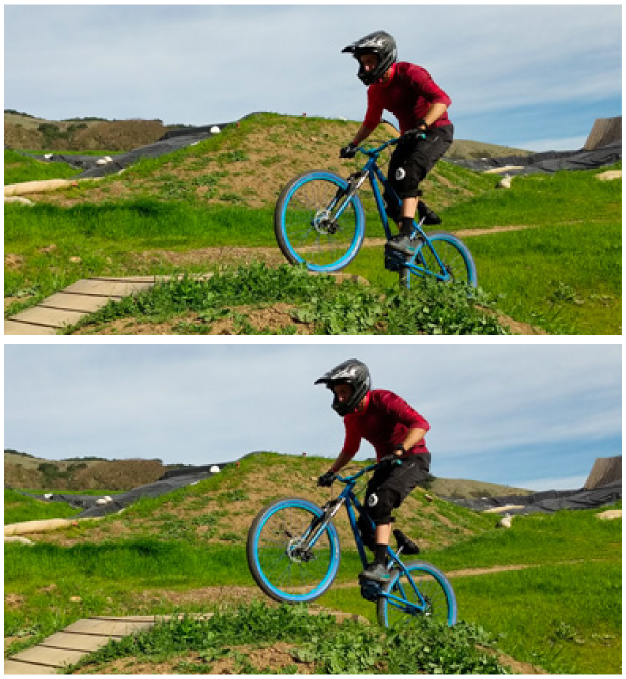
Abstract

We present a technique for synthesizing a motion blurred image from a pair of unblurred images captured in succession. To build this system we motivate and design a differentiable "line prediction" layer to be used as part of a neural network architecture, with which we can learn a system to regress from image pairs to motion blurred images that span the capture time of the input image pair. Training this model requires an abundance of data, and so we design and execute a strategy for using frame interpolation techniques to generate a large-scale synthetic dataset of motion blurred images and their respective inputs. We additionally capture a high quality test set of real motion blurred images, synthesized from slow motion videos, with which we evaluate our model against several baseline techniques that can be used to synthesize motion blur. Our model produces higher accuracy output than our baselines, and is several orders of magnitude faster than baselines with competitive accuracy.

1. Introduction

Though images are often thought of as capturing a single moment in time, all images in fact capture a duration of time: an image begins when a camera starts collecting light, and ends when that camera stops collecting light. If the camera or the scene move while light is being collected, the resulting image will exhibit motion blur. That blur may indicate the speed of a subject or may serve to separate a subject from the background, depending on the relative motion of the camera and the subject (see Figure 1(b)).

Motion blur is a valuable cue for image understanding. Given a single image containing motion blur, one can estimate the relative direction and magnitude of scene motion that resulted in the observed blur [7, 8]. This motion estimate may be semantically meaningful [33], or may be used by a deblurring algorithm to synthesize a sharp image [5, 9, 17, 23]. Recent work has relied on deep learning for removing motion blur and inferring the underlying motion of the scene [6, 11, 31]. Deep learning techniques


(a) A pair of input images.

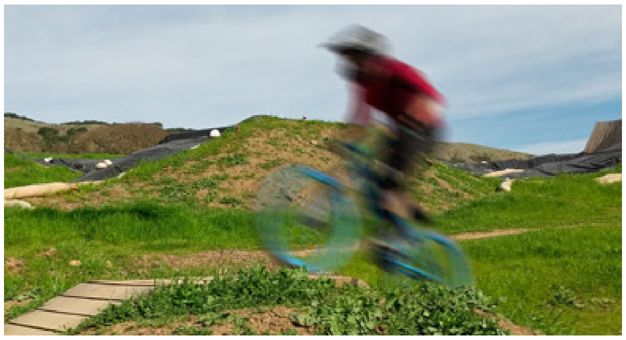

(b) Our model's output.

Figure 1. In (a) we present two images of a subject moving across the image plane. Our system uses these images to synthesize the motion blurred image in (b), which conveys a sense of motion and separates the subject from the background.

tend to need an abundance of training data to work well, and so to train these techniques one must generate large amounts of synthetic training data by synthetically blurring sharp images. These techniques also tend to use synthetic data (usually sharp images convolved by real or synthetic "camera shake" kernels) for quantitative evaluation, using real motion-blurred images only to produce qualitative visualizations. Naturally, the ability of these learned models to generalize to real images depends on the realism of their synthetic training data. In this paper, we treat the *inverse* of this well-studied blur removal task as a first class problem. We present a fast and effective way to synthesize the training data necessary to train a motion deblurring algorithm, and we quantitatively demonstrate that our technique generalizes from our synthetic training data to real motion-blurred imagery.

Talented photographers sometimes use motion blur for artistic effect (Figure 2(a)). But composing an artful motion-blurred photograph is a difficult process, typically requiring a tripod, manual camera settings, perfect timing, expert skill, and much trial and error. As a result, for casual photographers motion blur is likely to manifest as an unwanted artifact (Figure 2(b)). Because of the difficulty in using motion blur effectively, most consumer cameras are designed to take images with as little motion blur as possible — though if noise is a concern *some* motion blur is unavoidable, especially in low-light environments or in scenes with significant motion [12]. Artistic control over motion blur is therefore out of reach for most casual photographers. By allowing motion blurred images to be synthesized from the conventional unblurred images that are captured by standard consumer cameras, our technique allows non-experts to create motion blurred images in a post-capture setting. This is analogous to how recent progress in depth estimation has enabled post-capture on-device depth-of-field manipulation, also known as "Portrait Mode" [2, 4, 32].

Motion blur is also an important tool in cinematography, where filmmakers will carefully adjust the shutter angle of their camera to create a particular "film look". As in photography, this requires expert domain knowledge and skillful execution. Our system (or indeed any system that operates on pairs of frames) can be used to manipulate the motion blur of video sequences after the fact, by independently processing all pairs of adjacent frames in the input video.

Motion blur synthesis has been extensively studied in the rendering community [22], though these methods typically require perfect knowledge of scene velocities and depths as inputs. We instead target the most general form of this problem, and assume the only inputs available to our system are unblurred input images, as is the case in most general vision and imaging contexts.

To enable the varied image understanding and image manipulation tasks that require a method for creating motion blur, we present an algorithm that takes two sharp images taken one after the other, as shown in Figure 1(a), and synthesizes a corresponding motion blurred image, such as in Figure 1(b). The synthesized image resembles an image captured over the time spanned by the input images — the image "starts" at the first input image, and "ends" at the second input image. To achieve this, we adapt recent advances in machine learning to the task of predicting line kernels for motion blurring image pairs.

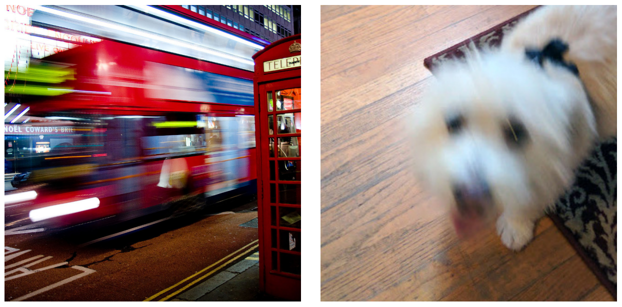

(a) Artful motion blur. (b) Unwanted motion blur.

Figure 2. Capable photographers can use motion blur to produce striking photographs, as in (a). But for most casual photographers, motion blur is more likely to manifest as an unwanted artifact in an image that was intended to be completely sharp, as in (b).

We build upon the recent success of convolutional neural networks [16] and end-to-end training on tasks similar to ours, such as optical flow [13, 30, 34] and frame interpolation [14, 24, 25, 26]. We use state-of-the-art frame interpolation to synthesize training data for our motion blur model, and demonstrate that our model, trained directly on the task of synthesizing motion blur, produces improved results on real images over baselines derived from optical flow and frame interpolation techniques. Though frame interpolation achieve only slightly decreased accuracy, our technique is many orders of magnitude faster, and is thereby better suited to the online synthesis of training data in a deep learning context, and is easier to deploy in a consumer-facing rendering or smartphone-photography setting.

The remainder of this paper is structured as follows: In Section 2 we discuss the nature of motion blur as a function of linear motion and motivate our novel line prediction layer. In Section 3 we define a deep neural network architecture based on our line prediction layer. In Section 4 we construct a synthetic dataset that is used for training, and a real-world dataset that is used for evaluation. In Section 5 we evaluate the performance of our model compared to its ablations and variants, and to techniques in the literature that can be adapted to the task of synthesizing motion blur.

2. Problem Formulation

We aim to take two adjacent images from a camera, say from a video or from a "burst" of photos [12], and from them synthesize a motion blurred image that spans the duration between the input images. That is, letting I_1 be the image exposed for the duration $[s_1, t_1]$ and I_2 be the image exposed for the duration $[s_2, t_2]$ (where $s_1 < t_1 < s_2 < t_2$), we synthesize the long exposure photograph $I_{1\to 2}$, which spans the duration $[s_1, t_2]$.

Similar to the assumptions of optical flow, which describes motion between two frames in terms of per-pixel velocity vectors, we assume locally linear motion between the two input images. We further assume that each pixel in the motion blurred image can be linearly interpolated from pixels lying on lines drawn from the corresponding pixel in each of the input images. While these assumptions are not always valid—for example, in the case of objects that are rotating or oscillating—we will demonstrate that this simple linear model is sufficiently expressive to produce high quality results.

Our neural network architecture uses a novel "line prediction" layer, which we define here. For each pixel in our images I_i ($i \in \{1, 2\}$) we predict a line, where one endpoint of that line is at the pixel's location (x, y) and the other endpoint is at $(x + \Delta_i^x(x, y), y + \Delta_i^y(x, y))$—the pixel's location when advected by some predicted offset Δ_i. The line is composed of N evenly-spaced discrete samples, for which we also predict $W_i(x, y, n)$, a weighting for each sample. Our final predicted image $I_{1\to 2}$ is defined as the weighted average of the two input images according to the discrete samples along all lines:

$$I_{1\to 2}(x,y) = \sum_{i\in\{1,2\}} \sum_{n=0}^{N-1} W_i(x,y,n) \times \tag{1}$$
$$I_i\left(x + \left(\frac{n}{N-1}\right)\Delta_i^x(x,y), y + \left(\frac{n}{N-1}\right)\Delta_i^y(x,y)\right),$$

where $I_i(x, y)$ is the result of bilinear interpolation of I_i at any continuous location (x, y).

We refer to this approach as "line prediction", analogously to the "kernel prediction" literature [3, 21, 25]. Our model can be thought of as a form of kernel prediction, as the weighted average in Equation 1 can be rasterized into a per-pixel convolution with a discrete kernel composed of the sum of the weighted bilinear interpolation kernels used in line prediction—though reformulating the blur in this way makes it significantly more expensive to compute.

For our line prediction technique to work properly, we must reason about the relationship between our line offsets Δ_i and our sampling density. Since the standard deep learning techniques we use for estimating the parameters of our line prediction layer have difficulty producing variable-length outputs, the number of estimated line samples N is fixed. However, if the motion estimated at a given pixel is significantly greater than the number of samples available to reconstruct our predicted line, then our resulting motion blurred image will be *temporally undersampled*, and will therefore contain artifacts from these "gaps" when synthesizing motion blur. See Figure 3 for a visualization of this sampling issue. For this reason, when determining a value for N, we must impose a bound on the magnitude of our line endpoint displacements $(\Delta_i^x(x, y), \Delta_i^y(x, y))$. We only address the task of synthesizing motion blurred images whose maximal displacement is 32 pixels in length, and we set $N = 17$. We found that we are able to use half as many samples as our maximum displacement because the kernel used by bilinear interpolation effectively prefilters the convolution induced by our line prediction. This limit on pixel displacement and sampling density is analogous to the similar limits of kernel prediction-based video frame interpolation techniques with regard to their kernel sizes.

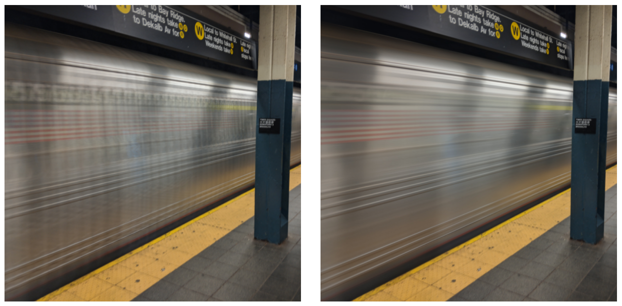

(a) Temporal undersampling (b) Temporal supersampling

Figure 3. Temporal sampling is critical to the construction of our model and our training data. If a motion blurred image is synthesized using significantly fewer samples than the maximum displacement of any pixel across those samples, then that synthesized image may be temporally undersampled. This results in discontinuous artifacts along the direction of the motion, as in (a). If the sampling density is sufficiently large with respect to image resolution and object motion then the synthesized images will not exhibit any such artifacts, as in (b).

Our decision to have our network predict a set of sampling weights $W_i(x, y, n)$ may seem unusual, as techniques from the graphics literature tend to assign uniform weights to pixels when rendering motion blur [20]. These learned weights allow our algorithm to handle complex motions and occlusions, and to hedge against certain failure modes. For example, by emitting a weight of 0, our model can ignore certain pixels during integration, which may be necessary if the pixel of interest moves behind an occluder on its path towards its location in the other frame. Because our synthesis happens simultaneously in both the "forward" and "backward" direction, our model can use these weights to smoothly transition across images or to selectively draw from one image but not the other, further improving its

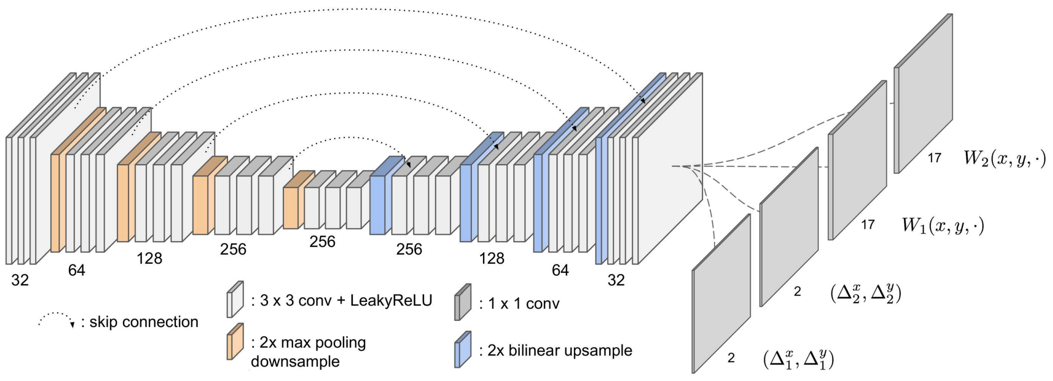


Figure 4. A visualization of our architecture, which takes as input a concatenation of our two input images and uses a U-Net convolutional neural network to predict the parameters for our line prediction layer.

ability to reason about occlusion. Though our model is constrained to linear motion, these weights can be used to model an object as moving at a non-constant speed along its line. For example, if an object accelerates towards its destination, our model can synthesize a more accurate motion blur (without introducing any temporal undersampling issues) by giving early samples higher weights than later samples.

3. Model Architecture

Our model is built around the U-Net architecture of [28], which feeds into our line prediction layer whose output is used to synthesize a motion blurred image. The input to our model is simply the concatenation of our two input images. See Figure 4 for a visualization of our architecture.

The U-Net architecture, which has been used successfully for the related task of frame interpolation [14, 26], is a fully-convolutional encoder/decoder model with skip connections from each encoder to its corresponding decoder of the same spatial resolution. Our encoder consists of five hierarchies (sets of layers operating at the same scale) each containing three 'conv' layers, and where all but the last hierarchy are followed by a max pooling layer that downsamples the spatial resolution by a factor of $2\times$. Our decoder consists of four hierarchies, each with three conv layers that are followed by a bilinear upsampling layer that increases spatial resolution by a factor of $2\times$. Each conv layer uses 3×3 kernels and is followed by leaky ReLU activation [19].

We train our model end-to-end by minimizing the L1 loss between our model's predicted motion blurred image and our ground-truth motion blurred images. Our data augmentation and training procedure will be described in more detail in Section 5. We experimented with pretraining our line prediction model using optical flow training data, as prescribed in [34], but this did not appear to improve performance or significantly speed up convergence. Our model is implemented using TensorFlow [1].

4. Dataset

Training or evaluating our model requires that we produce ground truth data of the following form: two input images, and an output image wherein the camera has integrated light from the start of the first image to the end of the second image. Because large neural networks require an abundance of data, for training we present our own synthetic data generation technique based around video frame interpolation, which we use to synthesize motion blurred images from conventional, abundantly available video sequences (Sec 4.1). We take sets of adjacent video frames, synthesizing many intermediate images between those frames, and average all resulting frames to make a single synthetic motion blurred image (where the original two frames can then be used as input to our algorithm). These synthesized motion blurred images look reasonable and are easy to generate in large quantities, but they may contain artifacts due to mistakes in the underlying video frame interpolation technique and so have questionable value as a "test set". Therefore, for evaluation, where data fidelity is valued more highly than quantity, we use a small number of real slow-motion video sequences. The first and last frames of each sequence are used as input to our algorithm, and the sum of all frames in the sequence is used as the "ground-truth" motion blurred image (Sec 4.2).

4.1. Synthetic Training Data

We manually created our own dataset directly from publicly available videos, as this gives us precise control over things like downsampling and the amount of motion present in the scene, while allowing us to select for interesting, high-frequency scene content. To construct this dataset, we first extract sets of adjacent triplets from carefully chosen video sequences, and then use those triplets to train a video frame interpolation algorithm. This video frame interpolation algorithm is then applied recursively to all triplets, which allows us to synthesize a 33 frame interpolated sequence from each triplet that can then be averaged to produce a synthetically motion blurred image. These images are then treated as "ground truth" when training our model.

We downloaded ∼30,000 Creative Commons licensed 1080p videos from YouTube in categories that tend to have significant amounts of motion, such as "Wildlife," "Extreme Sports," and "Performing Arts." We then downsampled each video by a factor of 4× using bicubic interpolation to remove compression artifacts, and then center-cropped each sequence to a resolution of 270×270. From these video sequences, we extracted triplets of adjacent frames that satisfy the following properties:

1. **High frequency image content**: Focusing training on images with interesting gradient information tends to improve training for image synthesis tasks such as our own, as shown in [10]. We therefore rejected any triplet whose average gradient magnitude (computed using Sobel filters) over all pixels was less than 13 (assuming images are in $[0, 255]$).
2. **Sufficient motion**: Scenes without motion are unlikely to provide much signal during training. Therefore, for each triplet we estimated per-pixel motion across adjacent frames (using the fast optical flow technique of [18]) and only accepted triplets where at least 10% of each pixel's flow had a magnitude (∞-norm) of at least 8 pixels.
3. **Limited motion**: Our learned model and many of the baseline models we compare against have outputs with limited spatial support, and we would like our training data to lie entirely within the receptive field of our models. We therefore discarded any triplet that contained a flow estimate with a magnitude (∞-norm) of more than 16.
4. **No abrupt changes**: Significant and rapid changes across adjacent frames in our video data are often due to cuts or other kinds of video editing, or global changes in brightness or illumination. To address this, we warp each frame in each triplet according to its estimated motion and discard triplets with an average L1 distance of more than 13 (assuming images are in $[0, 255]$).
5. **Approximately linear motion**: Our model architecture is only capable of estimating and applying a linear motion blur. Images that are not expressible using linear blurs will therefore likely not contribute much signal during training. We therefore compare the "forward" flow between the second and third frame to the negative of the "backward" flow from the first and second frame, and discard any triplets with a mean disagreement of >0.8 pixel widths.

Note that (5) represents a kind of "co-design" of our algorithm and our training data, in that we craft our dataset to complement the assumptions of our model. To evaluate a broader generalization of our model, we do not impose this constraint on our "real" testing dataset.

To ensure diversity, we extract no more than 50 triplets from each video, and no more than a single triplet from a given scene within each video. This process resulted in >300,000 unique triplets, of which 5% are set aside for validation with the remaining 95% used for training. This training/validation split is carefully constructed such that all triplets generated from any given video are assigned to either the training or validation split — no video's triplets are present in both the training or validation splits.

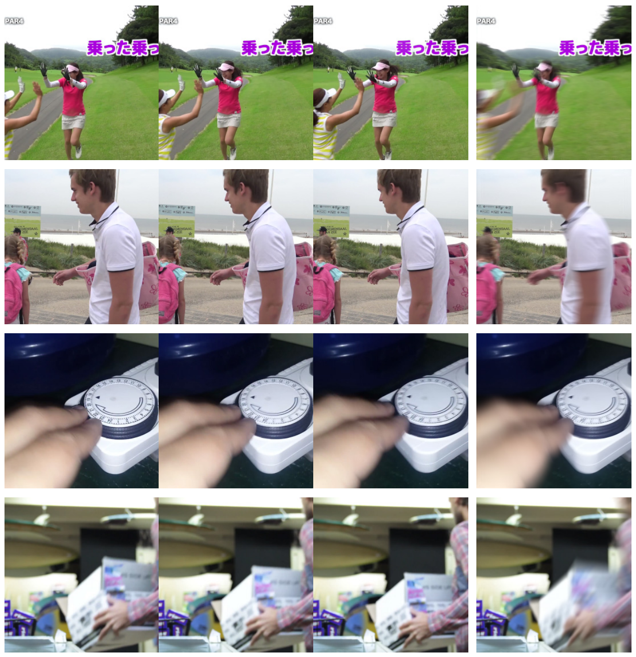


Figure 5. Here we show randomly chosen input/output pairs from our synthetic training dataset. To generate this data we identify triplets (shown in the first three columns) of adjacent frames that satisfy our criteria for motion and image content, use those triplets to train a video frame interpolation model, and apply that model recursively on each triplet to generate intermediate frames which are then averaged to synthesize a single motion blurred image (shown in the last column). When training our motion blur model, we use the first and last images of each triplet as input and the averaged image as ground-truth.

With this dataset we then train a video frame interpolation network based on [26], which will shortly be used to produce the final motion blur training data we are pursuing. Our frame interpolation network is the same model as described in Section 3, but using a separable kernel prediction layer of 33×33 learned kernels instead of our line prediction layer. Our training procedure is described in more detail in Section 5. The need to train this frame interpolation model is why we chose to extract triplets from our video sequences as opposed to just two frames, as the middle frame of each triplet can be used as ground-truth during this training stage (but will be ignored when training our motion blur model). After training, this frame interpolation model takes two frames as input, and from them synthesizes an output frame that should lie exactly in between the two input frames. We apply this network to our triplet of video frames, first using the first and second frames of the triplet as input to the network to synthesize an in-between frame, then using the second and third frames to synthesize another in-between frame. We then apply this same process recursively using the real and newly-interpolated frames as input. This is done 4 times, resulting in a 33 frame sequence of interpolated frames. These frames are all then averaged to produce a synthetically motion blurred image. Note that our recursive interpolation process yields 15 frames between each image in our triplet. Because our previously-described data collection procedure omitted adjacent frames with a motion of more than 16 pixels, this means that we should expect our interpolated images to have a motion of less than one pixel width per frame. This means that our resulting motion blurred images should not suffer from temporal undersampling. See Figure 5 for some examples of our synthetic training data.

4.2. Real Test Data

For evaluation purposes we would like a small, high-quality dataset that is not vulnerable to the artifacts that may be introduced by frame interpolation algorithms, and is as close as possible to a real in-camera motion blurred image. Although it is easy to acquire motion blurred images by themselves, acquiring the two input images alongside that motion blurred image is not possible with conventional camera sensors. We therefore capture a series of short slow motion videos, where the first and last frames of each video are used as input to our system, and the per-pixel mean of all frames is used as the ground-truth motion blurred image. Our dataset was gathered by a photographer using the Panasonic LUMIX GH5s, which records videos at 240fps. The photographer was instructed to photograph subjects that are well-suited to an artistic use of motion blur: people walking or running, vehicles moving, falling water, etc. Images were bicubicly downsampled by $2\times$ to help remove demosaicing and compression artifacts, and center-cropped to 512×512 pixels. From each video we selected a span of frames such that the total motion across the span is no more than 32 pixels. Any sequences that exhibited any temporal undersampling were removed. For each sequence we generated a single motion blurred image by simply averaging the frames, and we set aside the first and last frame of each sequence for use as input to our model. Each sequence has a variable length of frames, as we saw no need to omit frames from each sequence if they happened to be temporally super-sampled. Our final dataset consists of 21 diverse sequences. See Figure 6 and the supplement for examples.

5. Experiments

Our motion blur models, as well as our frame interpolation model used to generate our synthetic data, were trained distributedly over 8 NVIDIA Tesla P100 GPUs for $3.5M$ iterations on batches of size 16 using the Adam optimization algorithm [15] with a learning rate of $\alpha = 0.00002$ and momentum decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.998$. During training we performed data augmentation by randomly extracting a 256×256 crop from each image, and then randomly applying a horizontal flip, vertical flip, and a 90° rotation. Training to convergence took $\sim$2.5 days.

We evaluate our model against five baseline algorithms: A "naive" baseline that is simply the mean of the two input images (see Figure 7(a)), the non-learned and non-deep optical flow algorithm of [27], the state-of-the-art learned flow method of [30], the video frame interpolation work of [26] (which improves upon [25]), and the state-of-the-art video interpolation work of [14]. We additionally evaluate against three ablated versions of our model:

1. **Direct Prediction**: instead of using line prediction our network directly estimates the motion blurred image, by replacing our line prediction model with a single 1×1 conv layer that produces a 3 channel output.
2. **Uniform Weight**: we use uniform weights for each sample along lines rather than learning weights (i.e., all $W_i(x, y, n) = 1/2N$).
3. **Kernel Prediction**: instead of using line prediction we use the separable kernel prediction of [26], by replacing our line prediction layer with a single 1×1 conv layer at the end of our network that produces a 65×65 separable kernel (represented as a 65×1 and 1×65 kernel) at each pixel.

Our "kernel prediction" model has an inherent limitation, as separable kernels are limited in their ability to represent angled blur kernels. For example, the matrix corresponding to a blur kernel of a diagonal line is full-rank and cannot be represented well as a rank-1 matrix, so equivalently, the kernel cannot be represented well by a separable kernel. This limitation can be addressed by using non-separable kernels as in [25], however, the large kernels needed for our application require extreme amounts of memory that far exceeded the limits of our GPUs when we attempted to use this approach for training.

To generate motion blurred comparisons from our optical flow baselines, we employed the same line blurring scheme as our "uniform weight" model, and bilinearly sample N evenly-spaced values from the input images along lines corresponding to the optical flow fields. These sampled images are then averaged to produce a motion blurred image. We found that both flow algorithms benefited significantly (a PSNR improvement of $\sim$5) from using the negative backward flow instead of the forward flow to produce motion blur, so we adopted that strategy when evaluating our baseline flow techniques. More sophisticated strategies for gathering and scattering in forward and backward directions of object velocities have been used to synthesize motion blur in the graphics literature [20, 22], but these techniques assume that perfect scene geometry is known and so

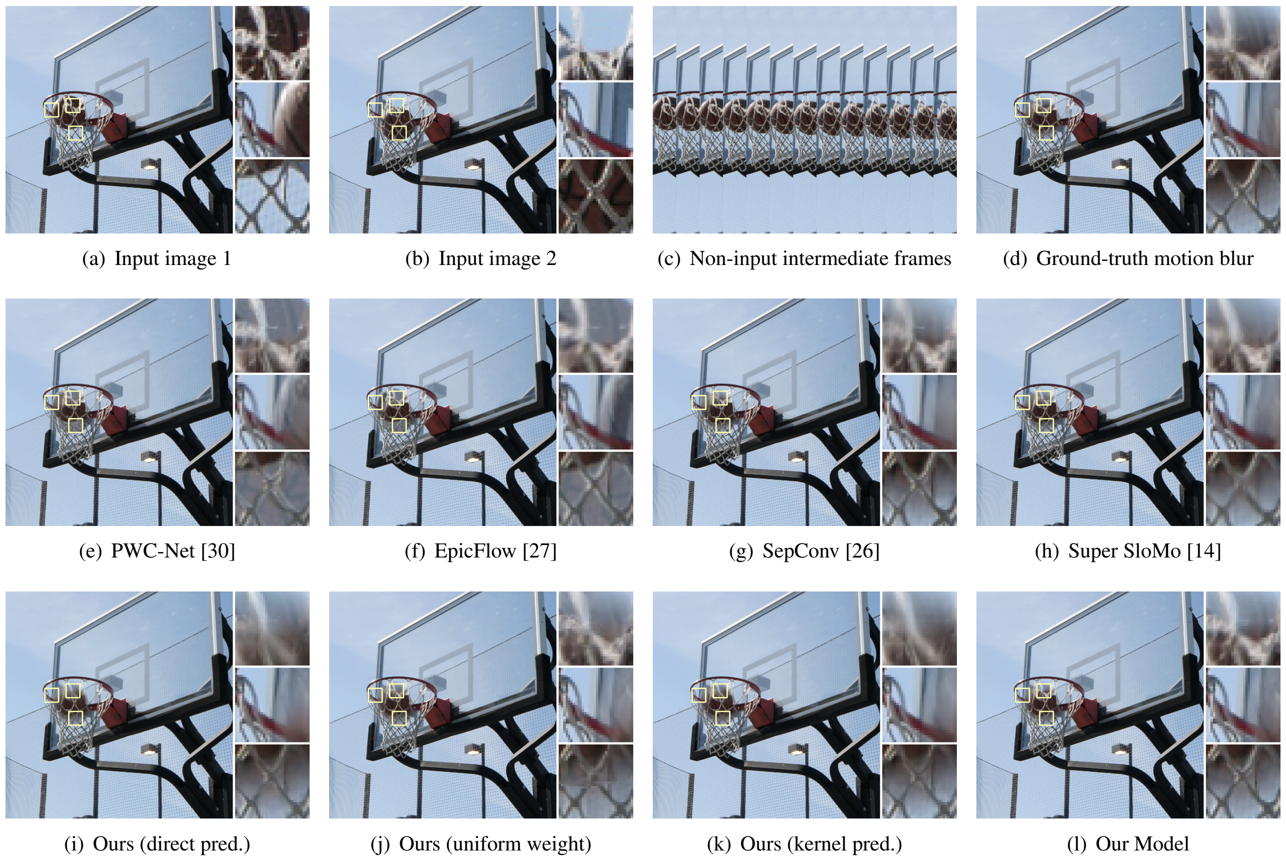


Figure 6. Results for one scene from our test dataset. The ground truth image (d) is the sum of the input images (a) & (b) and of the frames between those two images (c). We programmatically select the three non-overlapping 32×32 sub-images with maximal variance across all frames in (c) and present crops of those regions, rendered with nearest-neighbor interpolation and sorted by their y-coordinates. We compare our model (l) against four baselines (e)-(h), and three ablations (i)-(k). See the supplement for additional results.

cannot be used for our task.

Comparisons against frame interpolation baselines were conducted by recursively running frame interpolation on the input image pair for 5 iterations, which results in a 33-frame sequence — a sufficiently dense sampling given the limit of 32-pixel displacements in our real test set. The resulting synthetic slow motion sequences were then averaged to produce a motion blurred image.

Algorithm	PSNR	SSIM	Runtime (ms)
Naive Baseline	28.06 ± 4.05	0.888 ± 0.087	-
PWC-Net [30]	29.93 ± 3.47	0.938 ± 0.057	39.5
EpicFlow [27]	30.07 ± 3.49	0.940 ± 0.057	96.3×10^6
SepConv [26]	32.91 ± 4.60	0.954 ± 0.054	10.9×10^4
Super SloMo [14]	33.64 ± 4.66	0.958 ± 0.048	13.7×10^6
Ours (direct pred.)	33.97 ± 4.53	0.961 ± 0.044	34.7
Ours (uniform weight)	33.88 ± 4.68	0.959 ± 0.050	42.8
Ours (kernel pred.)	33.73 ± 4.31	0.961 ± 0.045	65.5
Our Model	34.14 ± 4.65	0.963 ± 0.045	43.7

Table 1. Performance on our real test dataset, in which we compare our model to three of its ablated variants and five baseline algorithms.

We primarily evaluate our model on the real test dataset described in Section 4.2, shown in Table 1. We report the mean PSNR and SSIM for the dataset, and note that our model produces the highest value of both out of all baselines and ablations. Though at first glance the difference between models may appear small, the unusually high PSNR of the "naive" baseline serves to anchor these scores and suggests that small variations in scores are meaningful. The two optical flow baselines are the lowest-performing techniques, with the two video frame interpolation techniques performing nearly as well as ours. However, the gap in runtime between our model and the baseline techniques is quite substantial, as our model is $\sim$300,000$\times$ faster. This is partially due to our compact architecture and the fact that line prediction is amenable to a fast implementation, but is also because video frame interpolation techniques must predict a 33 frame sequence that is then averaged to produce a single image, and so necessarily suffer a 33$\times$ speed decrease.

The reported runtimes of our model, its ablations, and the technique of SepConv [26] are the mean of 1000 runs on a GeForce GTX 1080 Ti, at our test set image resolution of 512×512. The runtimes of PWC-Net [30] and Super SloMo [14] were reported by the authors of those papers, who graciously ran their code on our data using a NVIDIA Pascal TitanX (a faster GPU than the one used for our model). The runtime for EpicFlow [27] was extrapolated from the numbers cited in the paper, which were produced on a 3.6Ghz CPU. Reported times for the optical flow methods are underestimates of their true runtimes, as we only measure the time taken to generate their flow fields, and do not include the time taken to render images from those flow fields.

The reduced performance of our "uniform weight" ablation appears to be due to its difficulty in handling occlusions and motion boundaries, which appear to particularly benefit from the learned sample weights. This can be seen in Figure 6(l), where our model appears to use its learned weights to blur around the occlusions of the basketball net webbing.

The output of our model superficially resembles an optical flow algorithm, in that the line endpoint $\Delta_i^x(x, y)$ predicted at each pixel can be treated as a flow vector. Though this is an oversimplification (our model actually predicts a weighting for a set of points along this line and those weights may be zero, effectively shortening or shifting this line) it is illustrative to visualize our output as a flow field and compare it to optical flow algorithms, as in Figure 7. Because our model is trained solely for the task of synthesizing motion blur, its "flow" often looks irregular and inaccurate compared to optical flow algorithms, which are trained or designed to minimized end point error of with respect to scene motion. This difference manifests itself in a number of ways: our model assigns a near-zero "flow" to pixels in large flat regions of the image, because blurring a flat region looks identical to not blurring a flat region and so our training loss is agnostic in these flat regions. Also, our model attempts to model the motion of things like shadows, which optical flow algorithms are trained to ignore as they do not represent motion of the underlying physical object. This disconnect between apparent motion in an image and true motion in world geometry may explain why our optical flow baselines perform poorly on our task. This difference between our model's learned "flow" and explicit optical flow techniques is analogous to prior work on learning monocular depth cues using defocus blur as a supervisory cue [29].

As our test-set performance demonstrates, our model performs well on diverse cases, including a variety of scene content, types of motion, duration of blurs, and amounts of blur in input frames. However, our model is limited in its inability to handle motions larger than those in the training dataset (32 pixels) and (similarly to other techniques) its inability to render nonlinear motion blur.

In the supplemental video we present results in which our system has been used to add motion blur to video sequences, by running on all pairs of adjacent video frames.

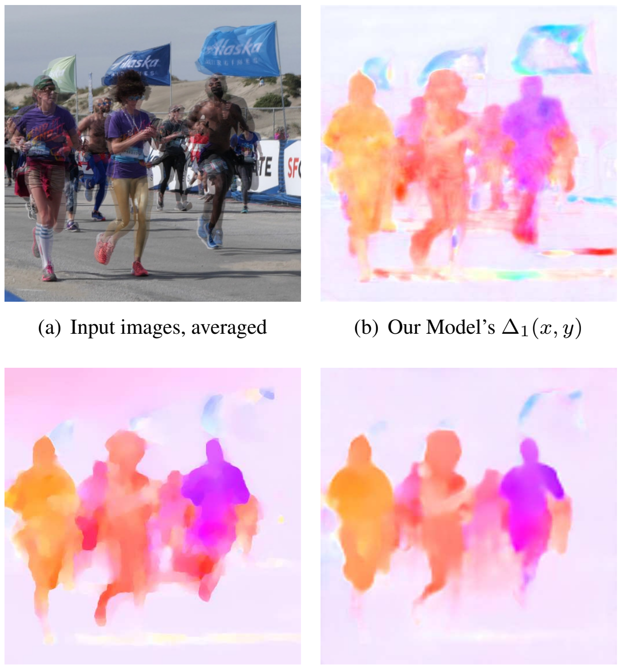

(a) Input images, averaged (b) Our Model's $\Delta_1(x, y)$ (c) EpicFlow [27] (d) PWC-Net [30]

Figure 7. A subset of our model's output can be visualized by using the endpoint of each pixel's predicted line as a flow vector. Here we render our model's "flow field" alongside two optical flow algorithms. Our "flow fields" tend to look irregular, highlighting the difference between training for accurate motion blur synthesis and training for accurate motion estimation.

6. Conclusion

We have presented a technique for synthesizing motion blurred images from pairs of unblurred images. As part of our neural network architecture we have proposed a novel line prediction layer, which is motivated by the optical properties of motion blur, and which is capable of producing accurate motion blur even when faced with occlusion and complex motion. We have described a strategy for using frame interpolation techniques to generate a large-scale synthetic dataset for use in training our motion blur synthesis model. We additionally captured a ground truth test set of real motion blurred images with their corresponding input images, and with that we have demonstrated that our proposed model outperforms prior work in terms of accuracy and speed. Our approach is fast, accurate, and uses readily available imagery from videos or "bursts" as input, and so provides a path for enabling motion blur manipulation in consumer photography applications, and for synthesizing the realistic training data needed by deblurring or motion estimation algorithms.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. *OSDI*, 2016.

[2] Apple. Use portrait mode on your iphone. `https://support.apple.com/en-us/HT208118`, 2017.

[3] Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *SIGGRAPH*, 2017.

[4] Jonathan T. Barron, Andrew Adams, YiChang Shih, and Carlos Hernández. Fast bilateral-space stereo for synthetic defocus. *CVPR*, 2015.

[5] Benedicte Bascle, Andrew Blake, and Andrew Zisserman. Motion deblurring and super-resolution from an image sequence. *ECCV*, 1996.

[6] Ayan Chakrabarti. A neural approach to blind motion deblurring. *ECCV*, 2016.

[7] Ayan Chakrabarti, Todd E. Zickler, and William T. Freeman. Analyzing spatially-varying blur. *CVPR*, 2010.

[8] Shengyang Dai and Ying Wu. Motion from blur. *CVPR*, 2008.

[9] Rob Fergus, Barun Singh, Aaron Hertzmann, Sam T. Roweis, and William T. Freeman. Removing camera shake from a single photograph. *SIGGRAPH*, 2006.

[10] Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. Deep joint demosaicking and denoising. *SIGGRAPH Asia*, 2016.

[11] Dong Gong, Jie Yang, Lingqiao Liu, Yanning Zhang, Ian Reid, Chunhua Shen, Anton van den Hengel, and Qinfeng Shi. From motion blur to motion flow: A deep learning solution for removing heterogeneous motion blur. *CVPR*, 2017.

[12] Samuel W. Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T. Barron, Florian Kainz, Jiawen Chen, and Marc Levoy. Burst photography for high dynamic range and low-light imaging on mobile cameras. *SIGGRAPH Asia*, 2016.

[13] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CVPR*, 2017.

[14] 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. *CVPR*, 2018.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[16] Yann Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[17] Anat Levin. Blind motion deblurring using image statistics. *NIPS*, 2006.

[18] Ce Liu. *Beyond Pixels: Exploring New Representations and Applications for Motion Analysis*. PhD thesis, MIT, 2009.

[19] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *ICML*, 2013.

[20] Morgan McGuire, Padraic Hennessy, Michael Bukowski, and Brian Osman. A reconstruction filter for plausible motion blur. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2012.

[21] Ben Mildenhall, Jonathan T. Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. Burst denoising with kernel prediction networks. *CVPR*, 2018.

[22] Fernando Navarro, Francisco J. Sern, and Diego Gutierrez. Motion Blur Rendering: State of the Art. *Computer Graphics Forum*, 2011.

[23] Shree K Nayar and Moshe Ben-Ezra. Motion-based motion deblurring. *TPAMI*, 2004.

[24] Simon Niklaus and Feng Liu. Context-aware synthesis for video frame interpolation. *CVPR*, 2018.

[25] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. *CVPR*, 2017.

[26] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive separable convolution. *ICCV*, 2017.

[27] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. *CVPR*, 2015.

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *MICCAI*, 2015.

[29] Pratul P. Srinivasan, Rahul Garg, Neal Wadhwa, Ren Ng, and Jonathan T. Barron. Aperture supervision for monocular depth estimation. *CVPR*, 2018.

[30] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. *CVPR*, 2018.

[31] Jian Sun, Wenfei Cao, Zongben Xu, and Jean Ponce. Learning a convolutional neural network for non-uniform motion blur removal. *CVPR*, 2015.

[32] Neal Wadhwa, Rahul Garg, David E. Jacobs, Bryan E. Feldman, Nori Kanazawa, Robert Carroll, Yair Movshovitz-Attias, Jonathan T. Barron, Yael Pritch, and Marc Levoy. Synthetic depth-of-field with a single-camera mobile phone. *SIGGRAPH*, 2018.

[33] Jacob Walker, Abhinav Gupta, and Martial Hebert. Dense optical flow prediction from a static image. *ICCV*, 2015.

[34] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *arXiv*, 2017.