Motion-based frame interpolation for film and television effects

Anil Kokaram\textsuperscript{1} \emph{et al.}, Davinder Singh\textsuperscript{1}, Simon Robinson\textsuperscript{2}, Damien Kelly\textsuperscript{3}, Bill Collis\textsuperscript{4}, Kim Libreri\textsuperscript{5}

\textsuperscript{1}Trinity College Dublin, Dublin, Ireland
\textsuperscript{2}The Foundry, London, UK
\textsuperscript{3}Google Research, Mountain View, California, USA
\textsuperscript{4}Disguise, London, UK
\textsuperscript{5}EPIC Games, San Francisco, California, USA

\texttt{E-mail: anil.kokaram@tcd.ie}

Abstract: Frame interpolation is the process of synthesising a new frame in-between existing frames in an image sequence. It has emerged as a key algorithmic module in motion picture effects. In the context of this special issue, this study provides a review of the technology used to create in-between frames and presents a Bayesian framework that generalises frame interpolation algorithms using the concept of motion interpolation. Unlike existing literature in this area, the authors also compare performance using the top industrial toolkits used in the post production industry. They find that all successful techniques employ motion-based interpolation, and the commercial version of the Bayesian approach performs best. Another goal of this study is to compare the performance gains with recent convolutional neural network (CNN) algorithms against the traditional explicit model-based approaches. They find that CNNs do not clearly outperform the explicit motion-based techniques, and require significant compute resources, but provide complementary improvements in certain types of sequences.

1 Introduction

Frame interpolation is the process of synthesising a new frame in-between existing frames in an image sequence. It has emerged as a key algorithmic module in motion picture effects like slow motion (Slo-Mo) and Timeslice (http://timeslicefilms.com/). It is also a key component in television standards conversion used for instance in high frame rate TV sets to upsample the frame rate of received signals, and to convert between worldwide television standards, e.g. EU and US formats (25 and 30 fps, respectively). These processes revolve around the manipulation of time and space in a sequence of frames hence frame interpolation is implicated in them all. The field of frame interpolation has received a great deal of attention in the academic literature in the last 5 years particularly because of the success of deep neural networks (DNNs) in this application [1–10]. This paper brings an industrial context to that work and attempts to benchmark the recent rapid academic interest in the context of what has been quite a mature technology in the postproduction community.

1.1 Cinema and television context

Slo-Mo was the earliest form of time manipulation effect used to bring a kind of emphasis to fast moving objects in a scene. Two early appearances as an effect included the film The Wild Bunch and the television series in the 1970s The Six Million Dollar Man. The viewer is meant to believe that the speed and significance of events is heightened by the artificially slowed down sequence. The effect was created ‘in camera’ by shooting on set at a higher frame rate than would be used in the cinema, i.e. >24 fps. Shooting at 48 fps and playing back at 24 fps for instance yielded motion that was 2\times slower than normal. Slo-Mo generated in this way has now become popular in mobile phone apps (iPhone, Casio etc.) and YouTube (Slo-Mo guys). Today, the manipulation of time can be quite flexible as illustrated in Fig. 1.

The TimeSlice effect, invented by Tim Macmillan in the 1990s was made famous in the series of episodes of the movie The Matrix. Known thereafter as BulletTime, it has been used as a standard effect in many movies since then. In contrast to Slo-Mo, TimeSlice or BulletTime requires the instantaneous recording of multiple views of the scene in a number of different cameras. When those frames captured at the same time instant but from multiple views are played back as a temporal sequence, the effect of a ‘frozen moment in time’ is conveyed and the viewer is presented with a camera path that would be impossible with a single camera. The challenge in BulletTime is to convey a sense of a smooth camera track. That is not possible by playback direct from the onset image capture, because the camera bodies are too large to get images from viewpoints close enough together. Frames therefore have to be synthesised in-between the existing views. Used in this way, frame interpolation is known in the Computer Vision community as novel view generation or viewpoint interpolation.

In both the cases of Slo-Mo and BulletTime, the effect can be created by synthesising frames in-between the existing images in the sequence. Hence, given a 30 fps sequence, interpolating four frames in-between each existing pair, then playing back at 30 fps, yields a 4\times Slo-Mo effect. In the case of BulletTime, the choice of the number of in-between frames generated depends on an artistic
decision taken together with the viewpoint separation onset. Frame interpolation has also become important for the effect of motion blur (https://revisionfx.com/products/rsmh/) [11] or shutter angle simulation. In that effect, several frames are interpolated close to the time instant of the actual frame and then averaged to yield the effect of a longer exposure time. This is quite an important visual effect that is associated with the film look and used regularly not only in live action shots but also in animated productions.

In the motion picture effects, industry frame interpolation has become synonymous with optic flow estimation. That connection was popularised by the journalist Mike Seymour in his influential fxPhD (https://www.fxguide.com/fxfeatured/art_of_optical_flow/) online review in 2006 which contains an extremely comprehensive historical account of the evolution of the technology in moviemaking. The reason is because in all the effects discussed above it is essential that the interpolated frames do not disrupt the motion in the existing sequence. Hence, a knowledge of the existing motion in the sequence is required. However, as most researchers in this area would realise, optic flow is only a means to an end in frame interpolation. In fact to generate high-quality interpolated frames requires attention to post-processing and the robust handling of poor motion estimates.

1.2 Motion for frame interpolation

The simplest mechanism for converting between one frame or field (interlace) rate and another is to repeat (zero-order hold) or drop (subsample) frames. A variant of this idea in interlaced footage is to spatially upsample fields to create a high frame rate sequence then drop/repeat these frames accordingly. The BBC R&D unit were among the earliest to work on this kind of interpolation in the 1970s [12]. Since then there has been a huge body of work on non-motion compensated conversion both for interlaced and progressive format video [13]. These are all based on the general idea of repeating or dropping fields or frames accordingly. However, these methods do not preserve motion in the upsampled sequence well at all and lead to visible stuttering or judder artefacts. Furthermore, they do not take advantage of the high level of redundancy between frames in creating new interpolated data.

Both of these observations can be addressed by first interpolating the motion at the time instant of the missing frame. Given that motion both into the future and the past, a new frame can then be generated. The effectiveness of motion interpolation for solving this problem is actually better understood through an image sequence model as follows:

\[ I_n(x) = I_{n-1}(x + d_{n-1,n}(x)) + \epsilon(x) \]  

(1)

where the pixel intensity in frame \( n \) at site \( x \) is \( I_n(x) \) and the motion at that site from frame \( n \) into frame \( n-1 \) is \( d_{n-1,n}(x) \). This is the equation from which classic optic flow and motion estimation algorithms are derived. Given \( \epsilon \sim \mathcal{N}(0, \sigma^2) \) then the pixels in the current frame \( n \) can be created by rearranging those in the previous frame \( n-1 \). This is also illustrated in Fig. 2. Of course occlusion and uncovering plays a role, hence the need for a forward prediction model as well. Intuitively, because motion is generally smooth in space and time, we can employ motion smoothness constraints to infer motion at the missing instant in time.

Alternative strategies have proposed the notion that we can push image data from existing frames into the time instant that the data is required along contours of least gradient between the relevant images [14–16]. This results in convincing frame interpolation, but not necessarily along directions of motion, hence the motion fidelity of the conversion is not guaranteed when the sequence is viewed at the required frame rate.

1.3 Contributions of this paper

In this paper, we review motion synthesis-based approaches for frame interpolation and present a framework based on what would now be viewed as traditional image sequence modelling. In so doing we show how traditional ideas are connected with the more recent convolutional neural network (CNN) approaches. We note that previous work has not reported on the statistical significance of their findings in the context of the databases used. When taking this into account we find that CNN methods do not clearly outperform model-based approaches in terms of picture quality. In the context of this special issue, we focus on the ability to produce pictures of the kind suitable for cinema applications, hence we make comparisons against industrial techniques which are well known for reproduction quality. None of the previous work published has attempted to do this presumably because the academic community is largely unaware of industrial developments in this area. We show that the motion-based method presented here has a significant advantage when dealing with challenging camera motion. Finally, we examine the performance of frame interpolation techniques as a function of original frame rate. Our results show that as frame rate increases it becomes significantly easier to interpolate in-between frames in all the classes of techniques. We first present our general formulation of the problem in a probabilistic sense.

2 General formulation

Fig. 3 shows four existing input frames, and one (missing) frame to be synthesised at location \( t + \Delta \) in time. Proceeding in a probabilistic fashion, we require to manipulate the p.d.f. \( p(U_{t+\Delta} | D_{t-\Delta}, D_{t+\Delta}, \Delta) \) to generate an estimate \( U_{t+\Delta} \) of the missing frame. Here, \( D_{t+\Delta} \) denotes the missing motion fields and \( D_{t-\Delta} \) is used to denote motion fields between the existing frames. In this example, it includes the motion field mapping frame \( t \) into \( t - 1 (D_{t-1,t}), t \) into \( t + \Delta (D_{t+\Delta,t}), t + 1 \) into \( t (D_{t+1,t}) \) and \( t + 1 \) into \( t + 2 (D_{t+2,t+1}) \). We assume that \( D_{t+\Delta} \) has been estimated between existing frames in a pre-process using any existing motion estimation technique. For the moment, we ignore the effect of occlusion and uncovering for simplicity.

Jointly solving for motion and image data is possible but more efficient algorithms result by first factorising the posterior as follows:

\[ p(U_{t+\Delta} | D_{t-\Delta}, D_{t+\Delta}, \Delta, \epsilon) = p(U_{t+\Delta} | D_{t-\Delta}, D_{t+\Delta}, \epsilon) p(D_{t-\Delta}, D_{t+\Delta}, \epsilon) \]  

(2)
Maximising the two factors in turn then leads to an estimate of the missing motion, followed by the interpolated image using that estimated motion.

Considering the motion factor, and proceeding in a Bayesian fashion, we can write

\[ p(D_{t \Delta t}, D_{t \Delta t+1} \mid \cdot) \propto p_{LD}(\cdot | D_{t \Delta t}, D_{t \Delta t+1}) p_{PD}(D_{t \Delta t}, D_{t \Delta t+1}) \]

where \( p_{LD}(\cdot) \) is the motion likelihood and \( p_{PD}(\cdot) \) the motion prior. \( p_{LD}(\cdot) \) ensures that the interpolated motion explains the observed image frames and the observed motion \( D_{t \Delta t} \) in the rest of the sequence. \( p_{PD}(\cdot) \) inserts information about the spatial and temporal smoothness of motion. Liu et al. [17] and Wang et al. [18] both also formulate the problem in a similar Bayesian fashion but go on to include only spatial priors and operate only on a block basis.

The missing image p.d.f. follows a similar pattern

\[ p(I_{t \Delta t} | D_{t \Delta t}, D_{t \Delta t+1}, \cdot) \propto p_{LD}(D_{t \Delta t}, D_{t \Delta t+1}, \cdot | I_{t \Delta t}) p_{PD}(I_{t \Delta t}) \]

where again \( p_{LD}(\cdot), p_{PD}(\cdot) \) are, respectively, the likelihood and prior distributions for the missing image data.

The different approaches to motion-based interpolation proposed in the past can be explained as resulting from particular choices for the various likelihood and prior terms above. We use this framework to review the previous work next.

3 Previous work

We may divide the historical evolution of this area into the development of model-based algorithms and learning algorithms. Learning-based algorithms have evolved rapidly since about 2010 but before this model-based algorithms were the most common.

3.1 Motion interpolation

Deinterlacing and field rate conversion work in the early 1980s [13, 19, 20] all employed more than two fields (usually 3 or 4) to generate the missing motion field. Deinterlacing will not be discussed further here, but it is important to note that the idea of motion interpolation originated there. Most of the early work can be derived by specifying the motion likelihood as a function of observed or measured motion between existing frames as follows:

\[ p_{LD}(\cdot | d_{t \Delta t}(x), d_{t \Delta t+1}(x)) \propto \exp \left( -\frac{1}{2\sigma^2} \left( \| d_{t \Delta t+1} - d_{t} \| + \| d_{t \Delta t+1} - d_{t} \| \right) \right) \]

where \( d_{t}, d_{t+1} \) are estimated from the observed motion between frame \( t \) and \( t+1 \) and \( \sigma^2 \) measures the confidence in the model to some extent. The main body of work uses a maximum likelihood (ML) approach for motion interpolation in the sense that no explicit prior for the interpolation is employed, i.e. \( p_{PD}(\cdot) \) is constant. In detail, however, since \( d_{t}, d_{t+1} \) are derived from \( d_{t+1}, d_{t} \), and/or \( d_{t+1}, d_{t} \), there is an implicit notion of temporal motion smoothness. Assuming no acceleration across the frames it is typical [15, 18, 20–28] to state \((1 - \Delta)d_{t \Delta t} = \Delta d_{t \Delta t+1}\). Hence, the ML estimates for the interpolated motion become \( d_{t \Delta t} = d_{t} \) and \( d_{t \Delta t+1} = d_{t+1} \). Note that the complication of motion between the frames is not treated consistently in the literature. In fact, the observed motion should be compensated for motion itself before being used as a constraint in the motion interpolation process. This is not always the case in previous work. For example in [5], the initial motion interpolant is not derived from compensated motion.

By far, the majority of work after about 2000 uses a symmetric motion constraint to estimate the forward and backward interpolated motion fields directly from the observed image data [17, 18, 29–38]. The first example of its use is probably as far back as 1996 [39]. This is variously referred to in the literature as bidirectional, bilateral or symmetric motion estimation. The likelihood in this case is expressed as a function of observed image frames as follows: (see (6)) , where the forward motion is expressed as a function of the backward motion, i.e. \( \Delta d_{t \Delta t+1}(x) = (1 - \Delta) d_{t \Delta t}(x) \). Hence, estimation of the interpolated motion becomes a problem of motion estimation or optic flow estimation from a location \( x \) situated at the interpolated time instant \( t + \Delta \). Most of the previous works use a block matching scheme of some description to estimate this motion. This entails searching for matching blocks in frame \( t, t+1 \) using typically the sum absolute error as the objective function \( E_{SAD} \) defined as follows:

\[ E_{SAD}(d_{t \Delta t}(x)) = \sum_{x \in \mathcal{B}} | f(x + d_{t \Delta t}(x)) - f(x - \Delta d_{t \Delta t}(x)) | \]

where \( \beta \) denotes the sites in a block of pixels. Despite the wide success of these techniques leading to use in real-time applications like frame rate upconversion in television sets, they do not explicitly incorporate motion smoothness in time.

3.2 Motion compensated frame interpolation

Having estimated an interpolated motion field it remains to actually synthesise the interpolated frame. All previous works at this stage acknowledge that the motion interpolated in the previous step will contain errors and so some kind of robust interpolation process is necessary. Furthermore, occlusion and uncovering tend to be taken

![Fig. 4. Four existing frames at \( t - 1, t, t + 1, t + 2 \) are shown, as well as the missing frame at \( t + \Delta \) (dashed). The missing frame is offset from \( t \) by \( \Delta \) of a time interval and hence from \( t + 1 \) by \( 1 - \Delta \). An object is shown translating between the frames, and the correct manifestation for that object at \( t + \Delta \) is shown in the dashed rectangle. The motion \( d_{t \Delta t+1} \), between frames \( t \) and \( t + 1 \), is shown using one example motion vector as the forward motion \( d_{t \Delta t+1} \). Motion \( d_{t \Delta t+1}, d_{t \Delta t+1+1} \) is also shown using a vector in the background of frame \( t + 1 \) as an example. The correct interpolated motion at any position \( t + \Delta \) is shown as dashed arrows. The association of each position in frame \( t + \Delta \) with an occlusion state \( s \) is also indicated in the strip representing \( t + \Delta \). For instance, in the region \( s = 10 \), the interpolated image data does not exist in frame \( t \) but does in frame \( t + 1 \). The other states \( s \) are illustrated similarly.](image-url)
into account in this step rather than in the motion interpolation step.

Taking a ML approach to image interpolation, ignoring occlusion, and using the simple sequence model in (1) leads to the simple weighted temporal averaging operation proposed as early as 1989 [40]. Hence, the interpolated pixel at \( \hat{I}_{t+\Delta}(x) \) is as follows:

\[
\hat{I}_{t+\Delta}(x) = w_t I_t(x) + w_{t+\Delta} I_{t+\Delta}(x) + w_{t+1} I_{t+1}(x)
\]

where \( w_t = (1 - \Delta), w_{t+\Delta} = \Delta \).

For robustness to erroneous motion estimates, or occlusion and uncovering, the two main approaches taken were either (i) generate a number of candidate image interpolants and filter them with a non-linear filter [23, 29, 40] or (ii) use overlapped block motion compensation (OBMC) [30, 31]. Candidate interpolated pixels included typically the non-motion compensated frame average, the estimate as shown previously and pixels in a support neighbourhood in the previous and next frames suitably compensated for motion. OBMC is a robust scheme for implicitly dealing for many kinds of motion compensation errors by combining blocks in the previous and next frames based on some kind of motion reliability weight. For example, using four possible motion candidates (perturbed from the ML estimate above), we can measure the block similarities between \( I_t \) and \( I_{t+1} \) (typically using structural similarity index measure or mean square error). The similarity scores can be normalised and then used as filter weights together with spatial window functions for generating the interpolated block in the new frame. In a sense, the work by Wang et al. [26, 27, 41] falls into this category because they employ bilateral filters in space and time (trilateral filtering).

Thoma and Bierling [20] were perhaps the first to explicitly deal with occlusion and uncovering in a frame interpolation context. In their work, however, they were not strictly dealing with creating a new interpolated frame but instead they were predicting existing frames in a sequence for the purposes of video compression. Expressing their idea in the context of this paper, we can introduce an occlusion state variable \( s = \{00, 01, 10\} \) to indicate no-occlusion and occlusion in the next and previous frames, respectively (see Fig. 3). Their interpolation process can then be described as follows:

\[
\hat{I}_{t+\Delta}(x) = \begin{cases} (1 - \Delta) I_t(x) + \Delta I_{t+\Delta}(x) & : s = 00 \\ \Delta I_t(x) + \Delta I_{t+\Delta}(x) & : s = 01 \\ I_{t+1}(x) + \Delta I_{t+1}(x) & : s = 10 \end{cases}
\]

(9)

where \( \Delta = 0.5 \) in their work. As they were predicting existing frames, they were able to use motion compensated frame differences to configure \( s \) by thresholding the forward and backward motion compensated frame differences. Since then several authors have established Bayesian approaches for estimating these occlusion indicators [42–45] in the context of motion estimation and missing data interpolation.

Phase-based interpolation deserves special mention [46]. That work is based on the idea of expressing the model in (1) in the Fourier frequency space by taking the Fourier transform of both sides as follows:

\[
\mathcal{F}[\omega, \omega] = \exp \{-j[d(\omega, \omega) + \Delta \omega] \mathcal{F}[\omega, \omega] + \mathcal{F}[\omega, \omega] \}
\]

where \( \Delta = 0.5 \) in their work. As they were predicting existing frames, they were able to use motion compensated frame differences to configure \( s \) by thresholding the forward and backward motion compensated frame differences. Since then several authors have established Bayesian approaches for estimating these occlusion indicators [42–45] in the context of motion estimation and missing data interpolation.

\[
\mathcal{F}[\omega, \omega] = \exp \{-j[d(\omega, \omega) + \Delta \omega] \mathcal{F}[\omega, \omega] + \mathcal{F}[\omega, \omega] \}
\]

(10)

Since the early 1990s, this was used to express the motion estimation problem in the phase space [22, 47, 48]. As can be seen the problem of motion estimation now becomes that of estimating the phase shift \( d(\omega, \omega) + \Delta \omega \) between the Fourier transform of image patches in different frames. Meyer et al. [46] recognised that for small motion magnitudes between frames, the problem of frame interpolation using motion compensation can be reposed as the interpolation of the Fourier phase between the frames. This leads to elegant and robust frame interpolation results when motion is relatively small. We use this as a baseline algorithm in comparative results at the end of this paper.

### 3.3 Neural network approaches

The most exciting development in recent times has been the deployment of deep learning for frame interpolation [1–3, 5, 7, 9, 49]. Using a database of high frame rate sequences, a DNN is configured to correctly interpolate the high frame rate sequences from different subsampled sequences. The DNN acts as a powerful non-linear adaptive filter in this case. Much of this work builds on the use of DNNs for motion or optical flow estimation [50–52]. In all approaches using DNNs thus far, only two frames are used at \( t+1 \) to interpolate the intermediate frames, and the architectures have to be retrained for use with interpolated frames at different \( \Delta \).

In fact by far the majority of techniques can only interpolate multiple frames recursively by subdivision by two [6, 7, 9, 49, 53]. That makes them difficult to use in a special effects application which typically requires a continuous accelerate/decelerate option.

Two broad trends for the image synthesis have emerged. The first trend is exemplified by Niklaus et al. [3, 4] who design their DNN to estimate two convolution kernels \( H_t(x), H_{t+\Delta}(x) \) that change from site to site for the estimation of the intermediate frame as follows:

\[
\hat{I}_{t+\Delta} = H_t(x) \otimes I_t + H_{t+\Delta}(x) \otimes I_{t+1}
\]

(11)

This is a generalisation of the motion compensated weighted frame averaging filter in (8) and in many ways related to the use of spatio-temporal linear predictors [32, 43, 54] for frame synthesis. Occlusion is implied because the kernel changes at every site and for forward and backward frames.

The other trend is to follow the existing motion-based pipeline in the sense that first motion is interpolated using a DNN and then the frames are synthesised with that motion [1, 5–9, 53, 54]. Zhang et al. [1] actually use bilateral motion estimation and weighted frame averaging to generate a first estimate for the intermediate frame. Their DNN acts as a post-processor to clean up artefacts. Bao et al. and Tseng et al. [9, 53] use auxiliary motion generating networks. Jiang et al. were the first to [5] directly use a DNN to estimate \( w_t, w_{t+\Delta} \) in (8) as well as motion in a two-step refinement process. In the first step, a DNN is used to estimate motion between the frames at \( t, t+1 \) in both directions. That motion information is copied into the intermediate frame location (without motion compensation) to generate an initial interpolated motion field \( d_t, d_{t+\Delta}, d_{t+\Delta+1} \) and associated interpolated image \( \hat{I}_t, \hat{I}_{t+\Delta} \) using weighted frame averaging. This initial estimate is input to a DNN to generate refinements to the motion field as well as visibility maps \( w_t, w_{t+\Delta} \). These maps act as soft occlusion indicators. This updated data is used in (8) to generate the final interpolated frame.

The reader is directed to the papers above for a detailed description of the network architectures. It is sufficient to note here that in all cases, the standard U-Net convolutional architecture is deployed. In addition, both L1 loss and perceptual loss functions have been investigated for training and authors report that the perceptual loss functions appear to yield networks that generate sharper results. In addition, Jiang et al. [5] introduce smoothness loss functions on the motion interpolated and a temporal image smoothness loss between the motion compensated frames \( I_t, I_{t+\Delta} \). Results show the motion refinement strategy [5] edging ahead of the kernel estimation strategy [3, 6]. However, the kernel estimation strategy is certainly more general even though the size of the kernels limits the amount of motion it can handle.

### 4 Bayesian overview

Recall that we require to first interpolate motion and then perform reconstruction. In our proposed process, we wish to explicitly incorporate temporal smoothness of motion and image data as well as occlusion. This interpolation process is a pull process rather than push, since given the motion at \( t, t+\Delta \) we can pull pixels from \( t+1 \) to create the image \( I_{t+\Delta} \). We assume that some existing...
motion estimation process has been used to estimate motion between the four existing frames shown in Fig. 3. The use of explicit modelling of the retiming instant $\Delta$ is important as it allows us to reconstruct a frame at any point on the timeline without retraining any model.

Following from (2), we proceed in a Bayesian fashion by manipulating the posterior probability distribution,

$$p(d_{t-\Delta},d_{t+\Delta}) D, i)$$

where $D, i$ contain all the existing (known) motion estimates and image data as follows:

$$p(d_{t-\Delta},x), d_{t+\Delta}, x)(s(x)) \propto \frac{\exp \left\{ e_f(x) \right\}}{2\sigma_f^2} : s(x) = 00$$

$$e_f = \left[ d_{t-\Delta} + (1 - \Delta) d_{t+\Delta}, (x + d_{t+\Delta}, x) \right]$$

$$e_s = [d_{t+\Delta} - \Delta d_{t-\Delta}, (x + d_{t+\Delta}, x)]$$

$$e_{fb} = [d_{t-\Delta} + (1 - \Delta) d_{t+\Delta}, (x + d_{t+\Delta}, x)]$$

$$e_{fb} = [d_{t+\Delta} + \Delta d_{t-\Delta}, (x + d_{t+\Delta}, x)]$$

$$e_s = \left[ d_{t-\Delta} + d_{t+\Delta}, \frac{1}{1 - \Delta} \right]$$

where $d_{t\pm\Delta}$ collects the motion in the interpolated frame in the neighbourhood of the current site. The estimate for $d_{t\pm\Delta}$ used as the interpolated motion, is therefore that which maximises the posterior in expression above. To continue, we need to define the likelihood and priors in that expression.

4.1 Image likelihood

This expression is derived directly from the image model given in (1). We define $e_f(x) = I(x + d_{t+\Delta}, x) - l_{t+\Delta}(x + d_{t+\Delta}, x)$, which is the motion compensated pixel difference between the pixel in the next frame and the pixel in the previous. For colour images $e_f$ is a vector of three differences corresponding to the three colour planes. If the interpolated motion is correct this difference should be small, unless occlusion occurs. It is more robust to use motion to explicitly incorporate $s(\cdot)$ rather than the image data, since the data at $t + \Delta$ is clearly not known a-priori. We still incorporate $s(\cdot)$ explicitly in this likelihood as follows:

$$p(d_{t-\Delta},x), d_{t+\Delta}, x)(s(x)) \propto \frac{\exp \left\{ e_f(x) \right\}}{2\sigma_f^2} : s(x) = 00$$

$$\exp - k_i : otherwise$$

where $k_i = 10 \times 2.7^2$ to allow for a strong bias away from occlusion in the image data. In colour images $e_f$ is the scaled vector magnitude, i.e. the average of the square of the three difference components. $e_f$ can be measured from the pixel data but we set it to 1.0.

4.2 Motion likelihood

The true interpolated motion should provide smooth trajectories when compared with the motion already estimated between the existing frames. We enforce this constraint by encouraging various motion compensated motion differences to be small. We define these motion compensated motion differences as follows (dropping the $x$ argument in $d_{t\pm\Delta}$ for clarity):

$$e_f = \left[ d_{t-\Delta} + (1 - \Delta) d_{t+\Delta}, (x + d_{t+\Delta}, x) \right]$$

$$e_s = [d_{t+\Delta} - \Delta d_{t-\Delta}, (x + d_{t+\Delta}, x)]$$

$$e_{fb} = [d_{t-\Delta} + (1 - \Delta) d_{t+\Delta}, (x + d_{t+\Delta}, x)]$$

$$e_{fb} = [d_{t+\Delta} + \Delta d_{t-\Delta}, (x + d_{t+\Delta}, x)]$$

$$e_s = \left[ d_{t-\Delta} + d_{t+\Delta}, \frac{1}{1 - \Delta} \right]$$

In these expressions, $e_f$ and $e_{fb}$ are energies that penalise deviation from the mirror constraint for motion. $e_s$ is a direct manifestation of the constraint used in symmetric or bilateral motion [31, 39]. Finally, $e_f, e_s$ constrain the estimated forward and backward motion to provide low acceleration into the future and past frames at $t + 1, t - 1$, respectively.

4.3 Motion priors

We assume that the motion fields are all Markov random fields. The motion prior consists of two factors, $p_k(\cdot)$ which enforce spatial smoothness of the estimated motion field in both forward and backward directions. Hence, we employ the usual Gibbs energy prior as follows:

$$p_k(d_{t\pm\Delta}) \propto \exp - \lambda_k \left( \sum_{k=0}^{K-1} \lambda_k h(s(\cdot)) \right)$$

and similarly for motion in the opposite direction. In this expression, $\lambda_k$ controls the strength of the smoothness ($\lambda_k = 2.0$ typically), and $\lambda_k$ weights the contribution from each of the clique terms inversely with their distance from $x$, i.e. $\lambda_k = 1/|k|$. We use $K = 8$, to index the eight nearest neighbours of the current site with $v_k$. These offset vectors all have unit values in the horizontal and vertical directions. This relationship is shown in Fig. 4. In addition, $f(\cdot)$ is a robust function defined as follows:

$$f(a) = \begin{cases} a : |a| < 10.0 \\ 10.0 : otherwise \end{cases}$$

4.4 Occlusion priors

Finally, the prior for occlusion $p(s(\cdot))$ encourages spatial smoothness in the estimated states

$$p(s(x)) \propto \exp - \lambda_s \left( \sum_{k=0}^{K-1} \lambda_h h(s(x + v_k)) \right)$$

where $h(s, s)$ is an energy function that assigns energies according to the state pairs $(s, s)$ as follows:

$$h(s, s) = \begin{cases} 0 : s = s \\ 1 : s = 00 \\ 2 : otherwise \end{cases}$$

This heavily discourages states 01, 10 from sharing a boundary, while encouraging the states to be the same in local neighbourhoods. We use $\lambda_s = 10.0$ and $\lambda_h$ is as defined previously.
5. Optimisation

Using the expressions above we can solve for the unknown motion fields \(d_{t,t+1}\) by manipulating (12) using optimisation techniques such as graph cuts, belief propagation or any other local update scheme. In order to reduce the computational load of the final system, we precompute a list of possible temporal motion candidates. These candidates are estimated based on the observation that we can predict the motion at the interpolated locations by copying the motion between existing frames along their motion directions into the pixel locations at \(t + \Delta\). This is motion prediction in time. Hence, each motion vector between frames \(t, t+1\) is used to predict possible vectors (candidates) for the interpolated field \(d_{t,t+1}\). Similarly, \(d_{t+1,t}\) is used to predict possible vectors for \(d_{t,t+1}\). Fig. 5 illustrates the mechanism. The process is as follows:

i. Scan every vector \(d_{t+1,t}(x)\) for all \(x\) in frame \(t+1\).

ii. At each site \(x + \Delta d_{t+1,t}(x)\) in frame \(t + \Delta\), keep a record of \(d_{t+1,t}(x)\) since it has caused a ‘hit’ at that site. These are forward hits.

This process is repeated for the reverse direction, to yield backward hits, starting from frame \(t + 1\) using \(x + (1 - \Delta) d_{t+1,t}(x)\) instead. The result is two co-located lists of candidate temporal vectors (pointing in the forward and backward temporal directions) for every site in the interpolated frame at \(t + \Delta\). In practice, because of inaccuracies in the pre-computed motion fields and the difficulty in handling occlusion, there will be several sites at which there is more than one hit in each list, or no hits in some lists. A no hit case can be seen in Fig. 5.

5.3 Generating an initial estimate

It is possible to initialise the interpolated motion and occlusion information by random assignment. However, given the temporal list above it is preferable to generate a quick initial estimate of the interpolated motion field using the following rules derived from a consideration of the ideal case shown in Fig. 5. Here \(N_f(x,t)\) indicates the number of temporal candidates (hits) in the backward direction and \(N_f(x)\) for the forward direction.

i. Scan all sites in \(t + \Delta\).

ii. If \((N_f(x,t) == 1) && (N_f(x) == 1)\) assign the motion in the lists to the interpolated motion and set \(s = 00\).

iii. If \((N_f(x,t) >= 1) && (N_f(x) == 0)\) assign the first motion hit in the backward direction to both directions of interpolated motion and set \(s = 10\).

iv. If \((N_f(x,t) == 0) && (N_f(x) >= 1)\) assign the first motion hit in the forward direction to both directions of interpolated motion and set \(s = 01\).

v. Otherwise set the interpolated motion to 0 and \(s = 00\).
Table 1 Mean PSNR (dB) for upsampling × 2 starting from 30, 60 and 120 fps with the Adobe 240 fps dataset. All methods find it easier to interpolate successfully when starting from a higher frame rate, and KRONOS yields the highest PSNR of the methods tested. However, tests of significance reveal that the difference in means between FFmpeg, ACKMRF, SNN, KRONOS and TWIXTOR are not significant at the 95% level.

| Method  | Method  | Method  |
|---------|---------|---------|
| PHASE   | 28.94   | 32.89   |
| DNN     | 30.18   | 33.09   |
| FFmpeg  | 30.18   | 33.82   |
| ACKMRF  | 30.58   | 34.31   |
| SNN     | 30.56   | 34.46   |
| KRONOS  | **30.85** | **34.67** |
| TWIXTOR | 30.75   | 34.02   |

Bold values indicate the best performing system in each column.

6 Algorithm

The final algorithm is a local pixel update scheme derived from a maximum-a-posteriori (MAP) estimate of the missing motion and video frame. First the following pre-computation steps are performed:

i. Generate motion for existing frames (see Section 5.1).

ii. Generate temporal hit list (see Section 5.2).

iii. Generate initial estimate (see Section 5.3).

Then at each pixel site $x$, the following local updates are executed:

i. Fetch temporal hit candidates in forward and backward directions using the hit list compiled previously. If there are none then this list is empty.

ii. Fetch the motion at the eight nearest neighbours of the current site and use them as eight motion candidates for forward and backward directions.

iii. Include the current motion information at the current site as a final candidate.

iv. Reduce the length of the forward and backward motion candidate lists using a primitive vector quantiser: remove vectors which are within $±0.25$ pixels. Denote this new candidate list of vectors as $d_i^f, d_i^b$ for the $k$th forward and backward candidates. Assume that this yields $K$ candidate pairs.

v. For each pair of motion candidates, generate three possible motion/occlusion candidates by augmenting each pair with the three possible states $s = 00, 01, 10$. Denote this new candidate set as $m_i^0 = [d_i^f, d_i^b, s = 00]$, $m_i^1 = [d_i^f, d_i^b, s = 01]$, $m_i^2 = [d_i^f, d_i^b, s = 10]$. There are now $3K$ motion candidates.

vi. For each of the $3K$ motion candidates, calculate the following energies:

$$E_{ij} = A_1 \sum_{s=0}^{S-1} h(d_i^{f}(x) - d_{i+s},(x + v_i))$$

$$E_{i} = \sum_{j=0}^{K-1} h(d_i^{b}(x) - d_{i-s},(x + v_i))$$

$$E_i = (I_i(x + d_i^{f}(x))) - I_{i+s},(x + d_{i+s}(x)))$$

$$E_{i} = E_{ij} + E_{ij} + (2e_{ij} + 2e_{ij})$$

$$E_{ij} = \sum_{s=0}^{S-1} h(00, s(x + v_i))$$

where $e_{ij}, e_{ij}, e_{ij}, e_{ij}, e_{ij}, e_{ij}$ are defined in (18). We use $A_1 = 10.0, A_2 = 2.0$ and $A_3$ is defined previously.

vii. Assign the motion candidate pair having the lowest total energy to the interpolated motion field, replacing the values currently in that field. For that candidate the state value $s$ is indicated by the energy which is minimum, i.e. if $E_{ij}$ is minimum, then $s = 00$ and so on.

viii. Continue to the next site unless all sites have been visited.

ix. To reduce the occurrence of impulsive single site artefacts as the algorithm proceeds, detect all occurrences of sites at which $s(x) \neq s(v_i + x)$ and $s(v_i + x)$ are all the same. Then replace $s(\cdot)$ with the value of the neighbours. In addition, replace the motion at the site with the average motion of its neighbours.

x. Terminate iterations when (i) there has been no change in any estimated motion or (ii) there have been $R$ iterations (e.g. $R = 5$) of passes over the sites.

xi. Build the interpolated picture with the estimated motion and occlusion states, using (9).

6.1 Multiresolution

The process above alone is not robust to large motion. Hence, a multiresolution scheme is sensible. In such a scheme, the algorithm as outlined above is applied to a coarse block-based motion field instead. Thus, each site becomes a block of $B \times B$ pixels ($B = 4$ for 1080p images in our case). Site image differences then become the average pixel intensity difference. The interpolated block motion field at the coarse level is then used to initialise iterations at the next finer level. We use four levels of decimation in our pyramid. At the highest scale, when the iterations are complete, we use the block-based vector field as the final interpolated motion field.

6.2 Post-processing

Due to difficulty in estimating motion when that motion is fast, or the recording was taken in low light, a post-processing step is employed to reduce the appearance of image artefacts. A post-processing step like this is inevitable in a production environment because of the huge variety of material that will have to be dealt with. Artefacts typically appear as holes in the image as outlined above is applied to a coarse block-based motion field instead.

At the highest scale, when the iterations are complete, we use the block-based vector field as the final interpolated motion field.
The SloFlow dataset contains 13 × 2 datasets for experiments on sequence databases to evaluate frame interpolation techniques. For instance, the Middlebury dataset frame resolution is just 480p resolution which is not even as good as mobile phone video quality today. The SloFlow dataset contains only 8 frames each, and all 13,000 sequences in SloFlow sets). Using UCF101 and Middlebury datasets do give some idea of relative performance for picture processing applications, but cinema applications require motion consistency over time as well as consideration of high density pixel sampling. It is also not quite clear how many frames in each example sequence have been processed for the results in various papers, for example results reported in [5, 49] use interpolations over 2 frames in (about) 379 sequences from UCF101 instead of the full length of those sequences, ≃ 200 frames each.

We do report results using these databases; however, we believe results from Adobe240fps are more meaningful in this context. The Adobe240fps dataset (first used in [5]) consists of 133 clips of varying durations (155–11,297 frames), all at 720p resolution. There are 148,815 frames in total. The clips contain consumer derived content from iPhones or GoPros. While they do not represent high-quality cinema material, they are of a better quality and resolution and represent a useful baseline of performance for post-production.

### 8 Results

In the results that follow the algorithm presented in this paper is labelled ACKMRF. Three very important and industrially popular retimers are also compared: FFMPEG, Twixtor (revisionfx.com) and Kronos/Nuke (thefoundry.com). Both Twixtor and Kronos won Academy Awards (Science and Engineering Oscars) in 2007 and have been improved significantly since that time. FFPMEG is open source and used as substrate infrastructure in all the streaming media services available today (Netflix, YouTube, Vimeo, Facebook to name a few). That frame interpolator uses bidirectional motion estimation followed by OBMC [31, 59]. Twixtor and Kronos represent two of the most important post-production tools in use by professional post-production houses today. Kronos is a commercial implementation of ACKMRF using a robust initial estimate of motion. Twixtor is also a variant of motion-based interpolation taken as a bundle (FFPMPEG, Twixtor,

#### Table 2: PSNR (dB) using the Middlebury and UCF 101 datasets

| Method | Middlebury | UCF101 |
|--------|------------|--------|
| DNN    | 35.2       | 31.40  |
| FFMPEG | 34.83      | 32.47  |
| ACKMRF | 37.06      | 32.37  |
| SNK    | 37.12      | 32.61  |
| KRONOS | 36.94      | 32.43  |
| TWIXTOR| 35.26      | —      |

Bold values indicate the best performing system in each column.

---

Fig. 6 Cumulative density w.r.t. PSNR for the Adobe240fps dataset. Left to right 30 → 60 fps, 60 → 120 fps, 120 → 240 fps

---

7 Datasets for experiments

There has been a trend in the motion interpolation literature in the Computer Vision community, to use a wide range of video sequence databases to evaluate frame interpolation techniques. For instance, [4–6, 8, 9, 53–55] have converged on the use of the Middlebury [56] (http://vision.middlebury.edu/flow/), UCF101 [57] (www.crcv.ucf.edu/data/UCF101.php), SloFlow [58] and Adobe240fps [5] datasets. However, UCF101 and Middlebury datasets do not capture the kind of resolutions or picture quality for the kind of postproduction application discussed in this paper. For instance, the Middlebury dataset frame resolution is just 480p containing only 8 frames each, and all 13,000 sequences in UCF101 are 240p resolution which is not even as good as mobile phone video quality today. The SloFlow dataset contains 13 × 2-frame sequences of content shot at very high frame rate using a non-standard aspect ratio/resolution 1280 × 1024 and without the colour balancing for dynamic range that would normally be applied in a post-production application.

Most of these databases are excellent for work in semantic content-based analysis (the purpose of the UCF101 dataset) and in optic flow measurement (the purpose of the Middlebury and SloFlow sets). Using UCF101 and Middlebury datasets do give an idea of relative performance for picture processing applications, but cinema applications require motion consistency over time as well as consideration of high density pixel sampling. It is also not quite clear how many frames in each example sequence have been processed for the results in various papers, for example results reported in [5, 49] use interpolations over 2 frames in (about) 379 sequences from UCF101 instead of the full length of those sequences, ≃ 200 frames each.

We do report results using these databases; however, we believe results from Adobe240fps are more meaningful in this context. The Adobe240fps dataset (first used in [5]) consists of 133 clips of varying durations (155–11,297 frames), all at 720p resolution. There are 148,815 frames in total. The clips contain consumer derived content from iPhones or GoPros. While they do not represent high-quality cinema material, they are of a better quality and resolution and represent a useful baseline of performance for post-production.

---

8 Results

In the results that follow the algorithm presented in this paper is labelled ACKMRF. Three very important and industrially popular retimers are also compared: FFPMEG, Twixtor (revisionfx.com) and Kronos/Nuke (thefoundry.com). Both Twixtor and Kronos won Academy Awards (Science and Engineering Oscars) in 2007 and have been improved significantly since that time. FFPMEG is open source and used as substrate infrastructure in all the streaming media services available today (Netflix, YouTube, Vimeo, Facebook to name a few). That frame interpolator uses bidirectional motion estimation followed by OBMC [31, 59]. Twixtor and Kronos represent two of the most important post-production tools in use by professional post-production houses today. Kronos is a commercial implementation of ACKMRF using a robust initial estimate of motion. Twixtor is also a variant of motion-based interpolation taken as a bundle (FFPMPEG, Twixtor,
Kronos) these would represent the best that industrial strength
motion-based synthesis can provide now.

Using the open source software provided by Niklaus et al. [3]
and Jiang et al. [5], we were also able to compare against two very
effective DNN-based approaches to date. Niklaus et al. provide
their DNN model at [github.com/sniklaus/pytorch-
seqconv], while the model from Jiang et al. was a third-party
implementation at [github.com/avinashtalpaliwal/Super-Glow].
In the results that follow these implementations are labelled as SNN,
DNN, respectively.

8.1 High resolution dataset
Table 1 shows the mean PSNR for each method used to upsample
all 133 sequences (Adobe240fps) by a factor of 2, starting from 30,
60 and 120 fps respectively. We create the sequences by
downsampling the original 240 fps sequences by 2, 4, 8
respectively. As expected, the performance of all methods used to
upsample from 30 fps is worse than upsampling from 120 fps. That
is because the sequence becomes significantly temporally aliased at
30 fps and motion is larger. KRONOS yields the best result overall,
taking from 30.85 to 35.83 dB at 120 fps. However, tests of significance (t-test) reveals the surprising observation that across all
the upsampling experiments none of the differences between
ACKMRF, SNN, KRONOS and TWIXTOR are significant at the
95% level across any of the experiments. At 30 → 60 fps DNN and
FFmpeg join that group in the sense that the differences between
the means are also not significant at the 95% level. The PHASE
method is the worst performer except at 120 fps when it swaps
places with DNN. DNN is otherwise the worst performer.

To try to tease out further what is happening, Fig. 6 shows the
cumulative distribution of peak signal-to-noise ratios (PSNRs) across all the sequences for each method. The increase in quality
is illustrated by a drift of all the curves to the right as the frame rate
of the original sequence increases. Note how closely grouped all
the methods are except PHASE, DNN, FFmpeg at 30 fps. It
shows that the temporal aliasing of the sequence is a challenge to
all the methods. It also shows that the implicit modelling of motion
in SNN with large kernel sizes is almost keeping up with the
explicit motion modelling of the other methods. The fact that DNN
is the worst at 120 fps though indicates something about
motion handling there since that is an explicit motion handling
method.

Fig. 7 ranks the performance of each method per sequence and
provides a bar chart to summarise that performance. This analysis
approach tries to separate the performance of the techniques based
on content. For every method we measure how many times it is
ranked 1st, 2nd, 3rd... 7th. Then we histogram that performance.
The ideal performer would show zeros for bars 1–6 and 100% for a
single red bar indicating that it was always ranked first and never
any other rank. The worst performer would show a single blue bar
at 100% indicating that it was always last. The top chart shows 30
fps, middle 60 fps and bottom 120 fps. Now we can see that
KRONOS is always ranked first the most number of times, SNN
and ACKMRF are ranked second and third the most number of
times, while TWIXTOR has the best performance at 30 fps.
Overall, though KRONOS, TWIXTOR, SNN are the only three
that appear to rank first for at least 15% of the sequences. Taken
together, these numerical results show that the CNN methods are incorporating information that is complementary to the explicit
motion modelling of the other methods.

8.2 Low resolution datasets
For completeness we also evaluated the algorithms using the
UCF101 dataset and the Middlebury (OTHER) set (12 sequences).
Unlike previous work, we use all of the data in the UCF101 dataset
sequences, i.e. more than 10,000 clips totalling more than 1.8
million frames. We cannot report on the performance of
TWIXTOR for the sequences in UCF101, because it does not
function at such low resolution. Table 2 shows the mean PSNR for
both these datasets as well as results reported using the UCF101
(379 sequences instead of 10,000) in [5] (UCF101*), for
comparison. Analysis on the Middlebury dataset shows that the
SNN and ACKMRF are the best performers (only 0.06 dB
difference between them). However, in fact, none of the differences
between any of the methods are statistically significant at the 95 or
99% level (p-values all >0.9).

The UCF101 result shows just 0.23 dB difference between SNN
and ACKMRF and the performance of NUKE, ACKMRF, FFmpeg, EG, SNN and roughly within the same range. In fact,
ACKMRF and KRONOS are statistically the same (p-value 0.52),
but DNN (31.4 dB) is the worst performer (p-value <0.0001). On
this dataset, SNN (32.61 dB) is the best (and significant p-value 0.001) at 0.13 dB better than FFmpeg. It is quite an interesting
result that FFmpeg performs very well on this dataset but not
significantly different from ACKMRF. We note [5] reports DNN
mean PSNR as 33.14 dB on their subsampled UCF1010 dataset
(1600 frames). This is 1.6 dB more than our measurement and
shows the importance not only of using a large dataset, but also
processing all the frames in a sequence. Hence, our results give
a better idea of the performance over motion varying imagery,
affecting the ranking significantly.

The much improved performance of SNN here (32.61 dB) when
compared to the Adobe240fps database (30.56 dB) is due to the
influence of picture size on evaluating video frame interpolation.
SNN [3, 6] uses separable kernels with 51 taps for interpolation.
Therefore as the picture size reduces, the relative size of the picture
material used for prediction is substantially increased. For a frame
size of H × K, the percentage of the surface being used to predict a
single pixel in the output frame is (51/H) × (51/K). On the 1280p
Adobe240fps database, this is 0.2%. That increases by an order of
magnitude to 3% at 240p in the UCF101 sequences. To gain the
same performance at 1280p as we see at 240p, the filters used by
that neural network must therefore also increase in size, up to about
166 taps. That would amount to an increase in computational
complexity of over ten times.

8.3 Visual inspection
Figs. 8–11 enable visual inspection for a few frames to illustrate
some important observations especially about motion. Fig. 8 shows
that large amounts of material appearing and disappearing off the
edges of the frame are an issue for the CNN-based methods. The
PHASE method simply does not perform well when the motion is
large but no doubt a more robust motion compensated version of
that algorithm should perform well. The explicit motion-based
methods perform the best especially in this case of global/ perspectice motion. This is emphasised by the energy in the
difference images. As can be seen, ACKMRF is the best on this
sequence when the motion is large and the texture complicated.
Neither of the NN approaches handle this well.

Fig. 9 shows that many of the differences between the methods
disappear when the motion is small, i.e. at high original frame
rates. This makes sense because at that frame rate the temporal
aliasing is substantially reduced. Nevertheless, PHASE still
struggles even at high frame rate presumably because of poor
motion handling.

Fig. 10 shows that all the methods struggle with complicated
object interaction. The thin top of the bollard poses a challenge for
all the techniques as it is not attached to either the moving
background or the stationary foreground with confidence. In the
visual effects workflow this kind of problem would be dealt with
by segmenting the sequence into different moving layers and then
retiming each layer separately.

In Fig. 11, SNN outperforms the other techniques (by 1.0 dB in
this sequence) because it captures the motion of the thin bicycle
wheel very well. It is unclear why it can perform in this way in this
scenario but fails in others. One possibility is the behaviour of the
background which in this case is static and uniform.

9 Computational load
The complexity of the DNN methods is rather low in the sense that
the control structure needed for execution of the networks is
simple. They are cascades of filters and non-linear pointwise

This is an open access article published by the IET under the Creative Commons Attribution -NonCommercial License
(http://creativecommons.org/licenses/by-nc/3.0/)}
Fig. 7 Ranking the performance of each method over all 133 clips. Top to bottom: 30 → 60 fps, 60 → 120 fps, 120 → 240 fps. There are eight groups of eight bars each. Each group of bars indicates the number of times each method is ranked seventh, sixth, fifth and so on from left to right in each group (as a fraction out of 133). Kronos ranks first the most number of times, followed by SNN (second the most number of times) and ACKMRF (third the most number of times). In general, performance improves as the original frame rate is higher as expected.
functions. However, the computational load is massive. For instance, SNN requires 21.6 million parameters [9] and more recent networks (taking a post-processing approach) require as much as 70 million parameters [54]. Despite this, execution times

---

**Fig. 8** Performance of all methods on one sequence with high texture and large camera motion (IMG0188 in the Adobe Dataset). Top row: original 60 fps sequence, frames 17–20. The other rows show the performance of each method in reconstructing frames 18, 20. Left to right in each row: difference between interpolated frame 18 and actual frame 18, interpolated frame 18, difference between interpolated frame 20 and actual frame 20, interpolated frame 20. Note that both SNN and DNN struggle with the large amount of picture material disappearing off the left hand edge of the frame. Video illustrates this better than these stills. This issue is worse for all methods in frame 20 than in frame 18. However, the motion-based techniques still cope better overall.
are impressive once the computational resource of a GPU with enough memory is available. Execution times are reported [5, 6] at

![Image](https://i.imgur.com/90K.png)

**Fig. 9** Performance improves as temporal aliasing reduces. Top 4 rows: 30 → 60 fps. Bottom: 120 → 240 fps. Top row shows frame 77–79 from GOPRO9637b in the Adobe dataset at 60 fps. Subsequent rows show left to right: error between interpolated frame 78 and the actual, interpolated frame 78 (upsampled using 77, 79) and the non-motion compensated difference between frames 79 and 78 (to give an idea of the motion in the sequence). The second set of four rows shows frames 308–310 but at 240 fps. The time instant at frame 309 at 240 fps corresponds to frame 78 at 30 fps. At 30 fps the motion-based techniques perform much better than PHASE or SNN.
7 fps (DNN) and our measurement of unoptimised SNN execution times or 0.3 fps (presumably due to the large set-up cost of the network).

Fig. 10  All methods struggle with complicated occlusion. Top row shows the original sequence at 60 fps (IMG014 from the Adobe dataset). Subsequent rows show the upsampled sequence 30 → 60 fps using various methods. The first and third panes in each of these rows show the frame difference between the reconstructed frames in the second and fourth panes with the original sequence. As the car passes behind the orange bollard, the top part is distorted in all the reconstructions except the PHASE method. The PHASE method performs well here because the bollard is stationary.

The traditional techniques in contrast require orders of magnitude less parameters (10s) and subsequently an order of magnitude less memory. In fact in each case (FFMPEG, ACKMRF,
NUKE), the largest memory requirement is for at most a buffer of four image frames, or about 3.6 million pixels. That is about 10% of the parameter size of the convolutional network approaches, quite aside from the fact that those algorithms still need at least two image buffers as well. ACKMRF is not an optimised implementation and that runs at 3 fps at 720p, while NUKE, TWIXTOR and FFmpeg execute at better than 10 fps.

Finally, note that the technique presented in this paper does not have to be retrained to operate at different retiming factors. This is in contrast to most of the neural network approaches to date [6, 8, 9, 49, 53, 60] (building from SNN above) excepting [5] (DNN above). The explicit motion-based techniques are therefore cheaper per frame w.r.t. computational resource load than the NN techniques and provide a practical tradeoff between resources and quality.

10 Final comments

Retiming tools are an important part of cinema production and television applications. Most of the genres of techniques in the literature currently all use explicit motion information in some way to perform interpolation. We have presented a Bayesian approach for the interpolation of video frames using motion without any elaborate refinement step. Unlike previous work, we have incorporated temporal motion smoothness constraints across more frames and strengthened the link between motion and occlusion in interpolation. A fast scheme has been presented to initialise the final motion flow field before optimisation. Extensive tests are performed over more than 14,000 frames of 720p video using high frame rate (240 fps) to provide ground truth for interpolation. The Bayesian technique performs at the state-of-the-art level (on average) compared to more recent CNN techniques but using orders of magnitude less compute resources. However, in cases where the camera motion is large and there are textural regions (i.e. quite relevant for cinema applications), the Bayesian approach is ranked top.

We have shown the importance of framerate in the interpolation problem, demonstrating that interpolation from 120 to 240 fps is much easier than 30–60 fps. We also demonstrate the significance of picture real-estate in interpolation showing that the performance of systems in the UCF101 240p dataset is significantly different from the more high-quality Adobe dataset.

We have also compared with industrial strength retimers for the first time and introduced them as other comparison points for the academic community. Surprisingly, the CNN methods do not necessarily perform as well as the retimers used in post-production now. While we do not expect any recent research to outperform existing production tools that have been tried and tested in the wild, the relatively small advantage (≃<1 dB) of the CNN methods compared here shows that hybrid schemes are the future. This is particularly important given the massive computational resource required for these algorithms. Hybrid schemes may perhaps be derived by considering a CNN as a post-processor for an explicit motion-based technique. The rapid pace of CNN research will certainly edge performance further ahead in the years to come. While this paper has been in review, this kind of...
exploration is beginning to emerge in the literature [54, 60, 61] with various CNN-derived post-processing strategies able to contribute another 1.3 dB [61] on to existing systems.

11 Acknowledgments

This work was partially funded by donations from Google/YouTube. Many thanks go to Damien Kelly, Hugh Dennnan, Andrew Crawford and Francis Kelly for important discussions about motion estimation and motion interpolation over the years at GreenParrtPictures.

12 References

[1] Zhang, Y., Chen, L., Yan, C., et al.: ‘Weighted convolutional motion-compensated frame rate up-conversion using deep residual network’, IEEE Trans. Circuits Syst. Video Technol., 2018, 12, pp. 1-11

[2] Kappeler, A., Voo, S., Dai, Q., et al.: ‘Video super-resolution with convolutional neural networks’, IEEE Trans. Comput. Imag., 2016, 2, (2), pp. 109-122

[3] Niklaus, S., Mai, L., Liu, F.: ‘Video frame interpolation via adaptive separable convolution’, 2017 IEEE Int. Conf. on Computer Vision (ICCV), Venice, Italy, 2017, pp. 261-270

[4] Niklaus, S., Mai, L., Liu, F.: ‘Video frame interpolation via adaptive convolution’, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Honolulu, Hawaii, 2017, pp. 2270-2279

[5] Jiang, H., Sun, D., Jampanti, V., et al.: ‘Super slomo: high quality estimation of multiple video input frames’, IEEE Conf. on Computer Vision and Pattern Recognition, Salt Lake City, Utah, 2018, pp. 9060-9068

[6] Yuan, H., Ye, J., et al.: ‘Context-aware synthesis for video frame interpolation’, 2018 IEEE/CVF Conf. on Computer Vision and Pattern Recognition, Salt Lake City, Utah, 2018, pp. 1701-1710

[7] Lu, C., Hirsch, I., Scholz, B.: ‘Flexible spatio-temporal networks for video prediction’, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Honolulu, Hawaii, 2017, vol. 7, pp. 2137-2145

[8] Li, Y.-L., Liao, Y.-T., Lin, Y.-Y., et al.: ‘Deep video frame interpolation using cyclic convolution’, 2017 IEEE Int. Conf. on Artificial Intelligence (AAAI 2019), Honolulu, Hawaii, 2019, pp. 8794-8802

[9] Bao, W., Lai, W., Ma, S., Gao, W.: ‘Frame rate up-conversion via Bayesian motion estimation and hierarchical vector refinement’, Proc. of the 8th int. Conf. on Energy Efficient 3D Processing for Multimedia Systems, 2007, pp. 178-183

[10] Zhang, Y., Chai, Y., et al.: ‘Video motion compensation frame interpolation with motion vector refinement’, 2012 IEEE Int. Symp. on Circuits and Systems (ISCAS), Seoul, Korea, 2012, pp. 500-503

[11] Zhang, Y., Chen, L., Yan, C.: ‘Video frame interpolation via weighted robust principal component analysis’, 2013 IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vancouver, Canada, 2013, pp. 1404-1408

[12] Lee, K., Jeong, J.: ‘Bilateral frame rate up-conversion algorithm based on the comparison of texture contents’, Electron. Lett., 2016, 52, (5), pp. 354-355

[13] Guo, D., Lu, Z.: ‘Motion compensated frame interpolation with weighted motion estimation and hierarchical vector refinement’, Neurocomputing, 2016, 181, pp. 76-84

[14] Castagno, R., Haavisto, P., Ramponi, G.: ‘A method for motion adaptive frame rate up-conversion’, IEEE Trans. Circuits Syst. Video Technol., 1996, 6, pp. 436-438

[15] Haavisto, P., Juhola, N., Neuvo, Y.: ‘Fractional frame rate up-conversion using weighted median filters’, IEEE Trans. Consum. Electron., 1989, 35, (3), pp. 272-276

[16] Wang, C., Zhang, L., He, Y., et al.: ‘Frame rate up-conversion using trilateral frame interpolation’ , IEEE Trans. Circuits Syst. Video Technol., 2010, 20, (6), pp. 886-893

[17] Kokaram, A.: ‘Joint detection, reconstruction of severely degraded image sequences’. Image Analysis and Processing, 9th Int. Conf. on Image Analysis and Processing (ICIAP '97), Florence, Italy, 1997, pp. 773–780

[18] Kokaram, A.: ‘Motion picture restoration: digital algorithms for artifact suppression in archived images and video’ (Springer Verlag, Berlin, 1998), ISBN 3-540-76040-7

[19] Konrad, J., Dubois, E.: ‘Bayesian estimation of motion vector fields’, IEEE Trans. Pattern Anal. Mach. Intell., 1992, 14, (9), pp. 910-927

[20] Yalin, H., Black, M.J., Fablet, R.: ‘The dense estimation of motion and appearance in layers’. 2004 Conf. on Computer Vision and Pattern Recognition Workshop, Washington DC, USA, 2004, pp. 165-165

[21] Meyer, S., Wang, O., Zimmer, H., et al.: ‘Phase-based frame interpolation for video’, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston, 2015, pp. 1410-1418

[22] Dahbou, S.C.: ‘Real time motion measurement hardware: Phase correlation unit’. BBC Research and Development Report, BBC RD 1990/11, 1990

[23] Biswas, M., Nguyen, T.: ‘A novel motion estimation algorithm using phase plane correlation for frame rate conversion’, Thirty-Sixth Asilomar Conf. on Signals, Systems and Computers, Pacific Grove, CA, USA, 2002, vol. 11, pp. 492-496

[24] Liu, Z., Yeh, R.A., Tang, X., et al.: ‘Video frame synthesis using deep voxel flow’. 2017 IEEE Int. Conf. on Computer Vision (ICCV), Venice, Italy, 2017, pp. 4473-4481

[25] Igel, E., Nemenyi, N., Sakaia, T., et al.: ‘Flownet 2.0: evolution of optical flow estimation with deep networks’. IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR), Honolulu, Hawaii, 2017, pp. 1647–1655

[26] Donovitsky, A., Fischer, P., Igel, E., et al.: ‘Flownet: learning optical flow with convolutional networks’. IEEE Int. Conf. on Computer Vision (ICCV), Santiago, Chile, 2015, pp. 2758-2766

[27] Ranjan, A., Black, M.J.: ‘Optical flow estimation using a spatial pyramid network’. IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR), Honolulu, Hawaii, 2017, pp. 1720-1729

[28] Tseng, R., Liu, Y., Chen, J., et al.: ‘Adaptive frame interpolation using an end-to-end deep net with high quality flow estimation’. 2019 Int. Conf. on Technologies and Applications of Artificial Intelligence (TAAI), Kaohsiung, Taiwan, 2019, pp. 1-4

[29] Bao, W., Lai, W., Zhang, X., et al.: ‘MEMC-Net: motion estimation and motion compensation driven neural network for video interpolation and enhancement’, IEEE Trans. Pattern Anal. Mach. Intell., 2019, to appear, pp. 1-11, doi: 10.1109/TPAMI.2019.2941941

[30] Li, D., Wang, Z.: ‘Video superresolution via motion compensation and deep residual learning’, IEEE Trans. Comput. Imag., 2017, 3, (4), pp. 749-755

[31] Baker, S., Scharstein, D., Lewis, J.P., et al.: ‘A database and evaluation methodology for optical flow’, Int. J. Comput. Vis., 2011, 92, (1), pp. 1-31
[57] Soomro, K., Zamir, A.R., Shah, M.: ‘Ucf101: A dataset of 101 human action classes from videos in the wild’. University of Central Florida, Centre for Research in Computer Vision Technical Report, CRCV-TR-12-01, 2012

[58] Janai, J., Güney, F., Wulff, J., et al.: ‘Slow flow: exploiting high-speed cameras for accurate and diverse optical flow reference data’. 2017 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Honolulu, Hawaii, 2017, pp. 1406–1416

[59] Zhai, J., Yu, K., Li, J., et al.: ‘A low complexity motion compensated frame interpolation method’. 2005 IEEE Int. Symp. on Circuits and Systems (ISCAS), Kobe, Japan, 2005, vol. 5, pp. 4927–4930

[60] Niklaus, S., Liu, F.: ‘Softmax splatting for video frame interpolation’, ArXiv (CVPR 2020), 2020, (arXiv:2003.05534)

[61] Park, M., Kim, H.G., Lee, S., et al.: ‘Robust video frame interpolation with exceptional motion map’, IEEE Trans. Circuits Syst. Video Technol., 2020, To appear, (3), pp. 1–1, doi: 10.1109/TCSVT.2020.2981964