Convolutional Neural Networks for Video Frame Interpolation

Apoorva Sharma[*] and **Kunal Menda**[†] and **Mark Koren**[‡]

Abstract

Video frame interpolation has applications in video compression as well as up-sampling to higher frame rates. However, it is a challenging task, especially when objects in the scene are moving in different ways. The current state-of-the-art methods use optical flow to account for motion in a scene, but computing optical flow robustly is still a difficult process, which can lead to artifacts in the output. This paper extends a novel approach to frame interpolation using Convolutional Neural Networks. The performance of this approach is evaluated on the task of compressing a video file, as well as upsampling a video to a higher frame rate. In both tasks, the output is free of many of the artifacts found in optical-flow based methods, but the process introduces noise on the moving parts of the video.

Introduction

The ability to generate a frame given the previous and subsequent frames of a video has many applications. It allows one to increase the quality of a given video by enhancing its frame rate, a process known as *upsampling*. Conversely, if the interpolation is good enough, it can be used for *compression*, since one can only store keyframes and interpolate the rest of the video during decompression.

Any video interpolation algorithm, including the one we present in this paper, takes a pair of images I_1 and I_2, and produces an interpolated result $I_{1/2}$.

The naive approach to frame interpolation is a basic linear interpolation of the pixel values, where the interpolated frame's value at location x is computed as

$$I_{1/2}(x) = \frac{1}{2}\left(I_0(x) + I_1(x)\right).$$

However, this breaks down when different objects are moving over a stationary background, as can be seen in Figure 1.

Current state-of-the-art algorithms attempt to estimate the motion of different pixels to improve upon this naive pixel by pixel approach. Research in this field is currently focused on improving these motion estimates, but is running into complexity limits.

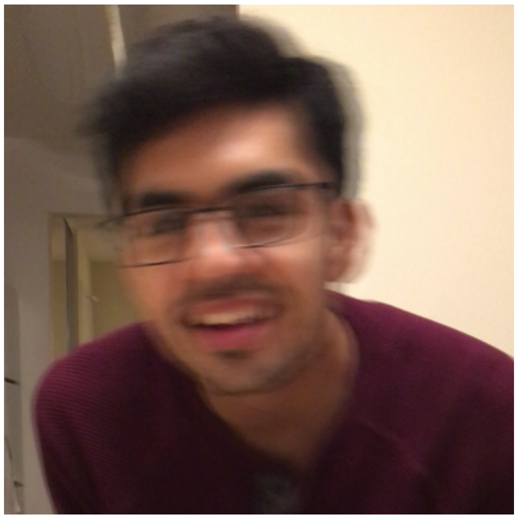

Figure 1: Linear Frame Interpolation (LFI) is simply a linear combination of the before and after frames. This interpolation method performs poorly when objects are moving, as can be seen in the LFI interpolated frame shown.

Fundamentally, however, these methods are context agnostic. In many applications in which video recordings are used, the content of the videos are quite similar. CCTV camera footage, for example, has a fixed background and predominantly only contains images of people moving across that background. Similarly, videos of soccer matches mostly contain frames containing a ball and players on the green pitch. This similarity in content suggests that a context aware approach could do better at interpolation by learning how objects in the scene move. In this work, we propose and test such a context-aware approach, the **D**eep Convolutional **N**eural Network based **I**nterpolator (DNI).

Related Work

The problem of video frame interpolation has been approached by making use of optical flow estimates [6]. This section will discuss attempts made at video frame interpolation that build on an algorithm called Motion-Compensated Frame Interpolation (MCFI), as well as an attempt to do frame interpolation using neural networks.

[*]Master's Student, Stanford Department of Aeronautics and Astronautics. apoorva@stanford.edu

[†]Master's Student, Stanford Department of Aeronautics and Astronautics. kmenda@stanford.edu

[‡]Master's Student, Stanford Department of Aeronautics and Astronautics. mkoren@stanford.edu

Linear Frame Interpolation

Linear Frame Interpolation (LFI) is the most naive method for frame interpolation, simply taking a linear combination (typically 50%-50%) of neighboring frames to generate the target frame. Though this algorithm does not introduce noise or artifacts, it makes no attempt at classifying the motion of objects within the frame. The resulting output on moving objects thus contains 'ghosts', as we can see in Figure 1. MCFI Algorithms attempt to characterize the motion of objects in an attempt to remove this ghosting effect.

Motion-Compensated Frame Interpolation

MCFI is a popular algorithm for frame interpolation, particularly for the application of upsampling videos, that is implemented in software and on hardware in many HDTVs [6]. MCFI is typically broken down into two sub-tasks: Motion Estimation (ME), and Motion Compensation (MC). ME typically involves computing the 'velocity' of every pixel in the frame [5]. MC involves using the pixel velocity estimates (typically both forward and backward) between a frame and its neighbor, in order to estimate the frame in-between them [4]. The combined process forms an algorithm that is considered a MCFI algorithm. Though a substantial improvement upon LFI, MCFI still tends to contain digital artifacts, such as tears or misplaced blocks, producing an unsatisfying output qualitatively described as having a 'soap-opera effect'.

Recent work by Guo and Lu [2] contains a detailed discussion of recent developments to both the ME and MC that tackle the ghosting and digital artifacts produced by MCFI, as well as approaches to reducing the complexity of the procedure. Their work builds off others that attempt to more accurately group motion estimates and correct for misclassified or miscalculated motion vectors that commonly occur on the boundaries of objects. The authors compare their algorithm, which we will call *Improved MCFI* (I-MCFI) to other state-of-the-art algorithms called Adaptive Vector Median Filtering (AVMF) and Motion Vector Smoothing (MVS), as well as the naive LFI algorithm. The details of these algorithms are described in Reference [2]. The algorithm presented in this paper will be compared against these four algorithms as a baseline.

Neural-Network Based Frame Interpolation

A recent Bachelor's Thesis by Haitam Ben Yahia attempts to use Neural Networks for the task of interpolating frames between 2D animations [10]. The author uses a CNN with inputs of pairs of frames, and has the network reconstruct the target frame between them, and optimizes the network on the L1-loss on each pixel. The CNN uses a common Convolutional Autoencoder architecture, but feeds forward 'residual connections' by appending previous convolutional outputs to the last deconvolutional input, and feeds the result in as input to the next deconvolutional layer. This architecture will be discussed in further detail in Section .

Though the paper presents the novel approach to frame interpolation that we hope to extend, the author is only able to demonstrate sub-par results. Particularly, the demonstrated outputs have blurry backgrounds, and do not do a convincing job of interpolating frames that contain motion. In addition, the author does not compare the outputs to the state-of-the-art MCFI algorithms on objective benchmarks.

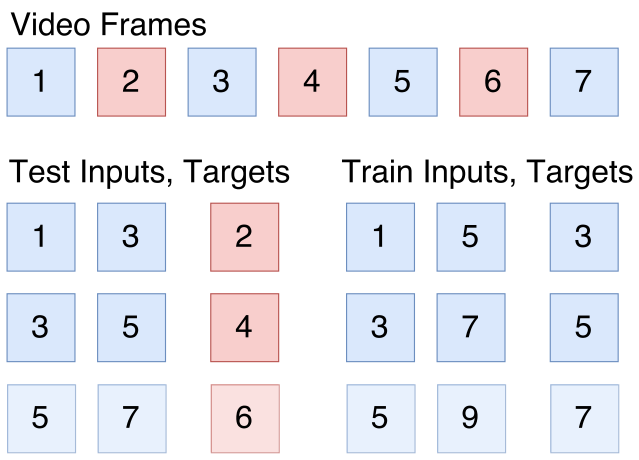

Figure 2: Example showing how a sequence of frames from a source video can be sorted to create train and test datasets. In each case, the target is the frame temporally between the two input frames.

Metrics For Comparison

In addition to assessing the outputs from Neural-Network based frame interpolation on a subjective appeal, the interpolated frames will be compared to their ground-truths using two benchmarks: Structural Similarity Index (SSIM) and Peak Signal-to-Noise Ratio (PSNR). SSIM attempts to characterize the similarity between two images based on their "perceived change in structural similarity", making use of the interdependencies between spatially close frames to map to a metric [8]. The PSNR is a log-scale ratio that is related to the Mean-Square-Error between an image and its ground-truth [7]. While the PSNR will penalize noise introduced to an image during interpolation, SSIM provides a metric closer to the qualitative notion of a good result.

Dataset and Features

Any video can be processed to create a labeled dataset for the problem of video frame interpolation. For a given target frame, the corresponding inputs are the frames that are immediately prior and immediately following the target frame.

To allow for comparisons to the conventional frame interpolation algorithms, we use the freely available uncompressed video samples available as part of Derf's collection on Xiph.org [9], a set of videos on which other frame interpolation algorithms have been evaluated.

Recall that there is a key difference between the two applications of video frame interpolation: In the *compression* task, the interpolated frames are frames that the algorithm could have seen as targets during the training phase, whereas in the *upsampling* task, the interpolated frames are frames for which no ground truth exists, so when evaluating performance on upsampling, the algorithm should not have access to these frames during training.

We generate a dataset from a video such that the training performance is a proxy for performance for compression application, while the test performance is a proxy for upsampling performance. We do this by compiling the training examples and the test examples through a process shown in

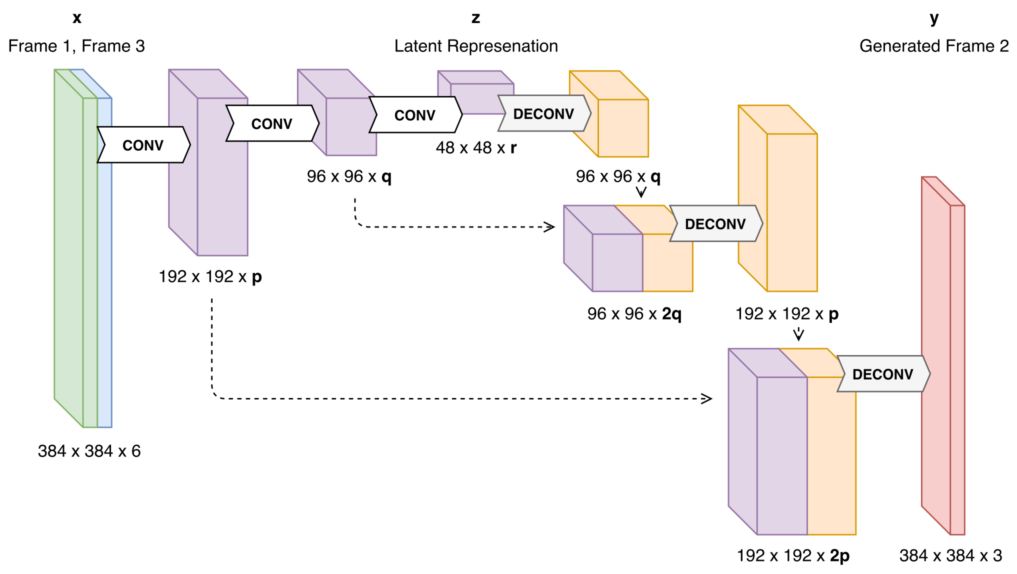

Figure 3: Illustration of the deep neural architecture used. The figure shows an architecture with three convolutional steps, with p, q, r filters in each layer.

Figure 2. Note that this method means that the test examples will have on average less motion across the frames than the training data, and thus performance between the two cannot be directly compared.

To make the task of our neural network process easier, we preprocessed all input data by subtracting off the median over all the training frames, a proxy for the static background in the video.

Methods

Building off the work done by Yahia [10], we use a deep convolutional neural network as the hypothesis model, due to the model's versatility for image processing applications.

The general pattern of the chosen architecture is illustrated in Figure 3. The input x is 6 channel array composed of stacking the RGB data from the input frames. We apply a convolutional layer on this data several times, storing the results each time, to ultimately arrive at z, the latent representation of the data. Each convolutional layer consists of a convolution step with a stride of 1 passed through a rectified linear unit and a final 2x2 max pooling step. In a convolution step, a set of 3px$\times$3px filters are convolved with the image. These 9 weights per filter are the weights that are tuned when a convolutional neural network is trained.

In the deconvolution process, we pass the data through a deconvolutional layer, then concatenate the result with the equivalently sized layer from the convolution process before applying the next deconvolution. This allows the image regeneration process to have access to details that may have been lost during the convolution process.

The model we selected had 5 convolutional layers of 20, 40, 80, 160, and 320 filters each, and 5 deconvolutional layers to fit the pattern shown in Figure 3.

We train this model by minimizing an approximate L1 loss between the generated frame and the target data. This is computed by summing a per-channel-pixel L1 loss over all three color channels and every pixel in the image. Given generated pixel channel value $\hat{y_c}$ and target pixel channel value y_c, the loss is calculated as:

$$L(\hat{y_c}, y_c) = \sqrt{(\hat{y_c} - y_c)^2 + \epsilon^2} \approx |\hat{y_c} - y_c|$$

with $\epsilon = 0.1$ as some small number.

We built this model in the Tensorflow [3] framework, and used the Adagrad optimizer to train the model. We discuss the performance of our trained model in the following section.

Results & Discussion

The DFI model was tested on 4 different videos to allow a quantitative comparison to other methods. The results of the DFI method, for both compression and upsampling, relative to LFI, AVMF, MVS, and I-MCFI are shown in Table 1 and Table 2, using SSIM and PSNR [1] as the comparative metric respectively. The potential benefits of the DNI model are apparent. The method performs the best on the Football video according to both metrics. This is true for both the compression and upsampling methods. However, the limitations of this basic implementation are apparent as well. For both the Bus and Stefan videos, the DFI model performs far below the Improved MCFI algorithm in both SSIM and PSNR. This is likely due to the static median being used instead of a rolling or instant median. Both of these videos span moving, noisy backgrounds. The Bus video features a fence with very thin rails which the method struggles to recreate. The Stefan video shows a colorful crowd, which is very noisy, as well as a large bursts of acceleration in the camera panning.

The larger the temporal difference between frames the more the DFI method struggles. This is especially present in the fact the Upsampling outperforms Compression for many videos, despite having less data available. As mentioned in the Dataset and Features section, there is a smaller temporal difference between upsampling frames than compression frames. This explains why upsampling performance is sometimes better than compression performance.

Table 1: Comparison of DFI to baseline methods [2] on the SSIM metric.

Sequence	LFI	AVMF	MVS	I-MCFI	DFI Compression	DFI Upsampling
Football	0.3865	0.5366	0.4972	0.5985	**0.8868**	0.8477
Bus	0.3378	0.8750	0.6568	**0.9043**	0.5572	0.5806
Stefan	0.7392	0.8870	0.8719	**0.9050**	0.7734	0.7794
News	0.9683	0.9670	0.9653	0.9676	**0.9757**	0.9763

Table 2: Comparison of DFI to baseline methods [2] on the PSNR metric.

Sequence	LFI	AVMF	MVS	I-MCFI	DFI Compression	DFI Upsampling
Football	19.316	20.886	20.658	21.422	**24.256**	21.812
Bus	18.528	25.016	21.363	**26.349**	20.044	20.174
Stefan	23.848	27.523	26.530	**28.021**	21.055	20.547
News	38.050	37.954	37.812	**38.431**	31.786	32.252

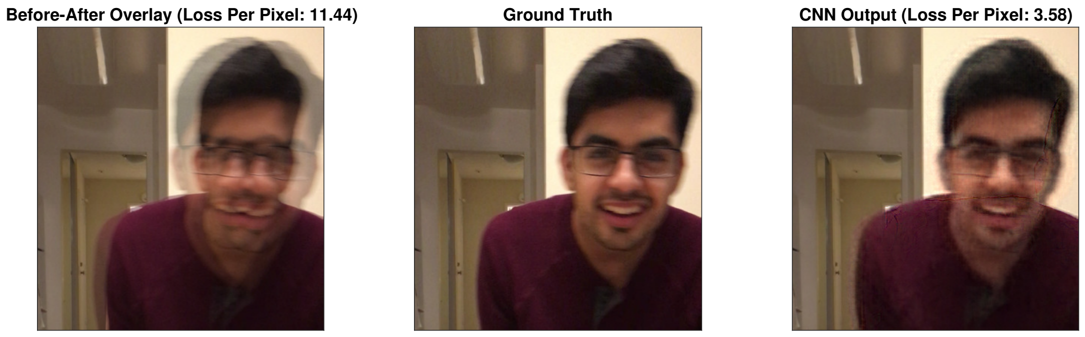
Before-After Overlay (Loss Per Pixel: 11.44)
Ground Truth
CNN Output (Loss Per Pixel: 3.58)

Figure 4: Visual comparison of the LFI and DFI methods (the left and right images respectively) when trying to estimate the ground truth image (center). Note that the CNN output eliminates the ghosting present in the LFI example, with only slightly more noise than the ground truth image

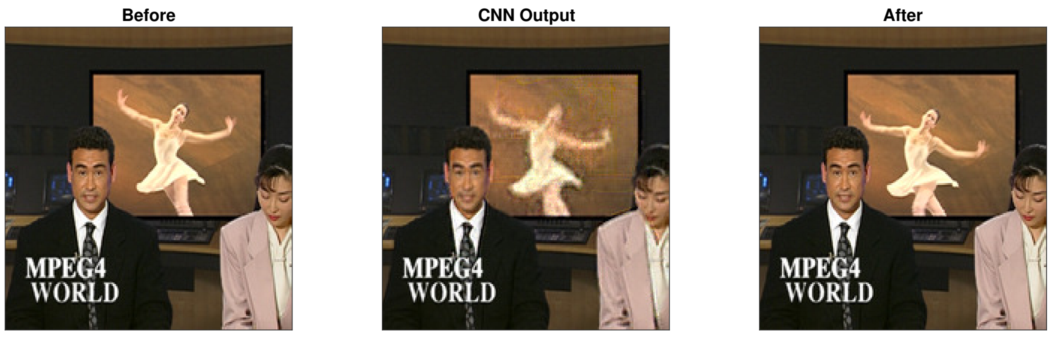

Figure 5: Visual example of the benefits of the DFI method. The left and right images are the input to the DFI algorithm, and the center image is the output from the CNN. Note that the CNN is able to transpose the front leg of the ballerina, even adding feasible separation between the legs.

A qualitative difference between LFI and DFI can be seen in Figure 4. The left image is the LFI method. There is noticeable ghosting, and the result is a poor representation of the ground truth image, shown in the center. However, the DFI method largely eliminates these issues, at the expense of some noise, especially in the detailed areas such as the eyes and hair. However, it is plain to see that the image is a far more acceptable estimate of the ground truth frame.

Another example of the advantages of the DFI method are seen in Figure 5. This figure shows the frame immediately before and after the frame generated by DFI. The result of note is the ballerina in the background. The first input shows her legs overlapping, while the second input shows them separated as she moves one away from the other. It is apparent to a human how to transpose the moving leg as a rigid body to create a decent estimate of a middle frame. However, an interpolation method requires complex pixel velocity mappings to approximate this, as mentioned earlier, and this is difficult to implement. However, the DFI method is able to identify this trend and generates a slight gap between the legs. To achieve such a difficult result with far easier implementation shows the immense potential of the DFI method.

Conclusion & Future Work

The DFI algorithm used an end-to-end convolutional neural network to take input pairs of neighboring images and interpolate a frame between them, with the only pre-processing being a simple median subtraction. The desired applications for such an algorithm were that of data compression, where the frame to be reconstructed had been seen before, as well as upsampling, where the frame being constructed has not been seen before. In both scenarios the generated images were compared to a ground truth using metric standards of SSIM and PSNR. The algorithm's performance was compared on a standard dataset to the performance of cutting-edge algorithms using Motion-Compensated Frame Interpolation. Results showed definitive promise in the DFI algorithm's ability to maintain structure where other state-of-the-algorithms demonstrate worse performance. However, this early implementation struggled with large bursts of acceleration, as well as noisy backgrounds. Still, the potential upside warrants further investigation.

One way to improve upon the implementation is to change the way medians are computed and shown. Currently, the median image is computed across all frames of a video. This is a large amount of frames, and the median loses any useful features. When combined with the output of the CNN, the resulting image is very noisy. By computing medians over smaller subsets of the video, or even for every input pair of frames, the output will be sharper, and the CNN should focus more on moving pixels. Different layer architectures can also be explored for different video types to achieve better performance.

References/Bibliography

References

[1] MATLAB (2016a). *Image Processing Toolbox*. URL: https://www.mathworks.com/help/images/index.html.

[2] Dan Guo and Zhihong Lu. "Motion-compensated Frame Interpolation with Weighted Motion Estimation and Hierarchical Vector Refinement". In: *Neurocomputing* (Mar. 2016).

[3] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[4] Wikipedia. *Motion compensation — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-November-2016]. 2016. URL: %5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Motion_compensation&oldid=748438532%7D.

[5] Wikipedia. *Motion estimation — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-November-2016]. 2016. URL: %5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Motion_estimation&oldid=750250116%7D.

[6] Wikipedia. *Motion interpolation — Wikipedia, The Free Encyclopedia*. [Online; accessed 1-December-2016]. 2016. URL: %5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Motion_interpolation&oldid=752435515%7D.

[7] Wikipedia. *Peak signal-to-noise ratio — Wikipedia, The Free Encyclopedia*. [Online; accessed 19-October-2016]. 2016. URL: %5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Peak_signal-to-noise_ratio&oldid=745106563%7D.

[8] Wikipedia. *Structural similarity — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-December-2016]. 2016. URL: %5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Structural_similarity&oldid=753322649%7D.

[9] Xiph.org. *Xiph.org Video Test Media [derf's collection]*. https://media.xiph.org/video/derf/.

[10] Haitam Ben Yahia. "Frame Interpolation using Convolutional Neural Networks on 2D animation". MA thesis. University of Amsterdam, Aug. 2016.