Frame Interpolation via AdaConv with PyTorch

Zhiyun Ling
zhiyunling@ufl.edu
University of Florida
Gainesville, Florida, USA

Figure 1. Interpolated frames on Middlebury dataset.

Abstract

Generally, video frame interpolation is composed of two individual steps: motion estimation and pixel synthesis. So, the quality of motion estimation is vital to the perfromance of the whole system. In this project, we revisited a paper by Niklaus et.al named "Video Frame Interpolation via Adaptive Convolution", which presents a robust video frame interpolation method that uses only one process to do the task. We implemented the final network structure in PyTorch and trained on Vimeo90k dataset, which contains 73,171 triple frame sequences extracted from 15k selected videos. The resulting model is evaluated on Middlebury benchmark dataset for visual examination. Finally, we generated interpolated videos out of low-fps input for comparison with ground-truth video using DAVIS2017 dataset. The resulting model is still able to provide visibly pleasing videos, although achieving lower evaluation scores.

Keywords: frame interpolation, neural networks

ACM Reference Format:
Zhiyun Ling. 2020. Frame Interpolation via AdaConv with PyTorch. *ACM Trans. Graph.* 1, 1 (December 2020), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Author's address: Zhiyun Ling, zhiyunling@ufl.edu, University of Florida, Gainesville, Florida, USA, 32608.

0730-0301/2020/12-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The video frame interpolation, also known as motion compensated frame interpolation, is a classical computer graphics problem and is important for slow motion interpolation and frame rate conversion. It can be viewed as a special case of image-based rendering, in which middle frames are estimated from neighboring frames using interpolation techniques. The goal is to make the video visually smooth, meaning increasing frame rate or reducing motion blur. So, the task is to generate interpolated frames with high quality, meaning less visible distortion or abrupt changes.

Traditionally, it involves two steps: motion estimation and pixel synthesis. Motion estimation, as the first step, is very important in deciding the quality of the interpolation result. Usually, it uses optical flow to model the motion between two frames. Then, the interpolated frame can be synthesized using the estimation. But optical flow can be difficult to estimate when encountering frames with abrupt brightness change and blur. Besides, flow-based pixel synthesis method won't work on occlusion regions. Hence, videos with above features can produce significant artifacts during interpolation.[6]

In deep learning, we can try to solve the problem with convolutional neural networks. Following the paper by [7], we can combine motion estimation and pixel synthesis into one step, by treating the synthesis task as a convolution over two input frames. We know that convolution network can extracts the relationship, in this case, local motion, between two inputs. At the same time, the coefficients for pixel synthesis are obtained. Since we don't rely on the optical flow, a wider range of videos can be used to train.

The network described in this paper is aiming at generating interpolated frames given the immediately preceding and following frames. We implemented the adaptive convolution network described in paper by Niklaus et.al, as shown in Figure 2.

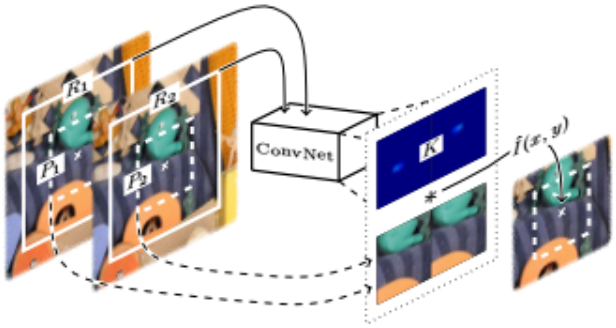


Figure 2. Pixel interpolation by convolution. Image from [6]

Without explicitly dividing it into separate steps, we use a deep convolutional network to achieve a robust video frame interpolation method. First, we use convolution to represent the interpolation task, by convolving over paired image patches extracted from two input frames in a video. So, the remaining task is to estimate a kernel for later interpolating tasks. In this project, the pixel-adaptive kernel is estimated with a fully deep fully convolutional neural network. As is illustrated in Figure 2, for each interpolated pixel $\hat{I}(x, y)$, we take two receptive field patches P_1 and P_2 from input frames to generate the kernel K.[6]

In our implementation and training, this model is implemented using PyTorch [9] and trained with whole Vimeo90K [12] dataset for temporal interpolation tasks. The evaluation is performed on Middlebury benchmark dataset [1], which is widely used for benchmarking frame interpolators. Also, we apply the best model to real-life applications by generating videos of doubled frame rate on DAVIS2017 challenge dataset [10]. And, through comparing with the ground-truth video, we are able to conclude that the model is an end-to-end solution for generating interpolated videos for almost any given videos.

2 Previous Work

Recently, CNN have shown great power in image recognition. Also, CNN is widely used in generate image and videos directly, for instance, GAN [2]. And, optical flow has been learned by CNN to used in object motion rendering. [13]

As a common technique for compression and video quality enhancement, video interpolation is one of the fundamental computer vision and digital image processing techniques. Conventionally, we can use motion estimation-based method to interpolate the middle frame, where optical flow is estimated between two consecutive frames in a video. [1] But, these methods require later image synthesis to complete the task, which incorporates some extent of hand-crafted adjustment. One phase-based method utilizes phase information and propagates it through pyramid levels to represent large motion between two frames. [5]

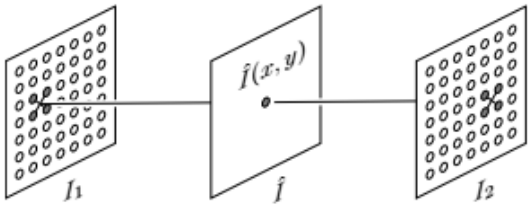
Using deep learning, some uses Mean Squared Error loss as well as multi-scale network model to deal with the blurring effects when interpolating. [4] Liu et al. tackles the problem using optical-flow-based architecture. [3]

In the paper by Niklaus et al., a fully convolutional neural network is used to estimate the spatially-adaptive convolution kernel for later pixel synthesis. Besides, the latter step is also embedded into the neural net, so the task is done in one process.[6, 7]

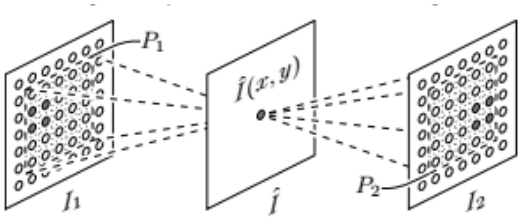
3 Frame Interpolation

3.1 Overview

The goal is to interpolate the middle frame $\hat{I}$ out of two frames I_1 and I_2 in a video. For the estimation of the color value of a pixel $\hat{I}(x, y)$ in the interpolated frame, traditionally we first find the corresponding value $I_1(x, y)$ and $I_2(x, y)$. Then, interpolate with the two value, using, for example, linear interpolation. Problems could occur when these corresponding location indexes are not integers, as in Figure 3a. In this condition, we would have to re-sample the images I_1, I_2 to produce a high quality output. This re-sampling action could render aliasing. Besides, as we mentioned before, this strategy relies on optical-flow which can be heavily impacted by motion blur and lack of texture.[6]



(a) traditional interpolation



(b) interpolation by convolution

Figure 3. Interpolation. Image from [6]

So, we combine two steps into one and use local convolution to represent pixel interpolation. As is shown in Figure 3b, we first extract patches P_1 and P_2 out of I_1 and I_2. Then, the interpolated $\hat{I}(x, y)$ can be computed by convolving a kernel K over P_1 and P_2. Thus, the interpolated value is directly computed by:

$$\hat{I}(x, y) = K_1(x, y) * P_1(x, y) + K_2(x, y) * P_2(x, y) \quad (1)$$

where K_1 and K_2 are 2D kernels for convolution, $P_1(x, y)$ and $P_2(x, y)$ are patches centered at (x, y) in I_1 and I_2.

3.2 Dataset

The dataset we use for training the interpolation net is Vimeo90k-Triplet dataset. [12] It consists of 73,171 3-frame sequences with a fixed resolution of 448 x 256, extracted from 15K selected video clips from Vimeo-90K. This dataset is designed for temporal frame interpolation. The size of both training and test set is 33GB. Considering the volume of dataset, we use a small portion of it to debug the architecture and demo. In the end, we use the whole dataset to train our best model.

Also, Middlebury - optical flow dataset is used for evaluating the interpolation result.[1] It's a widely used benchmarking dataset for interpolation tasks. It provides both color and gray images for evaluation. We use the color-two-frame images as input and compare with the ground-truth it provides. This dataset has 4 catagories : 1) hidden texture 2) synthetic 3) stereo 4) high-speed camera. These catagories indicate the area of focus on the evaluation.

The last dataset we use is DAVIS2017-unsupervised challenge dataset (480p), for generating doubled fps videos.[10] It consists of multiple sequences of frames. Each sequence consists around 90 frames.

3.3 Model

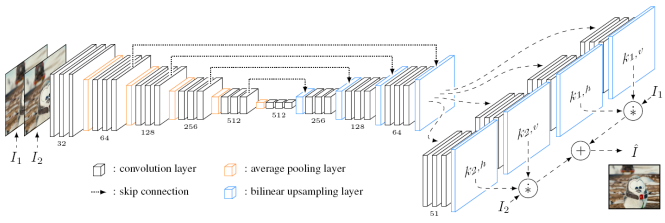


Figure 4. The detailed model architecture. Image from [7]

The detailed architecture for this project is shown in Figure 4.

3.3.1 Autoencoder. The first part network has a u-shape structure, containing multiple base modules with skip connections. In the left-half encoder, each base module is composed of three convolutional layer and one pooling layer. In the right-half decoder, upsampling layer is used in place of pooling.

The input is two 128x128 color patches extracted from random locations within the two input frames. So, we format a data point as a triplet $[I_1, I_2, I_{gt}]$. Then, we stack the color patches I_1 and I_2 into 6 channel tensor and send it into the autoencoder subnet. The feature units in intermediate layers are clearly shown in Figure 4. It follows a scheme of 6-32-64-128-256-512-256-128-64-kernelsize(51). All the convolutional layers in the modules use a stride=1, padding=1and kernel size of 3x3, followed by ReLU layer.

3.3.2 Local Convolution. The last part of the network is divided into 4 local convolutions between 1d kernel and input frames I_1 or I_2. Then, convolving I_1 with $< k_{1,h}, k_{1,v} >$, convolving I_2 with $< k_{2,h}, k_{2,v} >$, averaging two results to predict the $\hat{I}$. Here, we actually using patches P_1, P_2 for computation.

$$\hat{I}(x, y) =< k_{1,h}, k_{1,v} > *P_1(x, y) + < k_{2,h}, k_{2,v} > *P_2(x, y) \quad (2)$$

As is argued in another paper by Niklaus, replacing the 2d kernel convolution with separable local convolution with 1d kernel can drastically reducing the memory consumption and computation time. [7] Here, local means different kernels are used for every pixel of the inputs. The predicted kernel therefore represents the local motion between two frames. As convolution with 1d horizontal kernel and 1d vertical kernel can represent convlution with a 2d kernel, this strategy completes the task and saves computation resources.

3.3.3 Loss Functions. The widely used loss function in Deep Learning is MSEloss, known as L_2. We calculate the MSEloss based on per-pixel color difference, as defined below:

$$L_2 = ||\hat{I} - I_{gt}||_2^2 \quad (3)$$

Also, Niklaus et al. reports that sum of absolute differences, known as L_1, outperforms MSEloss in dealing with the blurring effect in interpolation. [6]

$$L_1 = ||\hat{I} - I_{gt}||_1 \quad (4)$$

We compared the two loss criterions in the experiments.

3.4 Implementation

The project is implemented using PyTorch, which has a rapidly growing community around it. The network model contains multiple convolutional layers, pooling layers and up-sampling layers. The last operation – local convolution, previously implemented using CUDA by [7], is embedded into python with CuPy[8]. This is worth noticing because the code **only** works on Nvidia GPU enabled platforms because of the CUDA part.

3.5 Training

The training was taken place on a platform with below configuration:

Table 1. Training platform configuration

OS	CPU	GPU	Memory
Ubuntu	Intel i7-8700K	Nvidia RTX2080	32GB

The whole dataset Vimeo90k is divided into training:testing = 14:1. Training (with L_1 loss) for one epoch takes around

20 minutes on whole vimeo90k (51313 samples for training). To boosting training process, we start by training with first 4096 data of the training samples for 1000 epochs. Then, with the pre-trained network, we do hyper-parameter selection and debugging. Also, the final model is trained on top of the pre-trained network. The final model is trained for 100 epochs on whole dataset.

The optimizer we use is Adam, with learning rate=0.01 for pre-training (first 1000 epochs) and 0.001 for post-training. As in the latter case, the model is almost near the converging point. Considering the memory capacity of GPU, we choose batch size as 16.

Given the complexity of the model and volume of the data, we only perform random horizontal and vertical flips for data augmentation.

3.6 Hyper-parameter Selection

The size of convolutional kernel size are vital for CNNs. First, we have to make sure the kernel is large enough for the model the capture the large motion between two frames. Thus, the model can accommodate most video interpolation tasks. But a larger kernel means more values to be estimated. That means increasing the burden of the auto-encoder. It's a trade-off between model representation ability and complexity.

Niklaus et al. originally chooses 41 as the 2d kernel size, arguing that the largest motion in the dataset is 38 pixel. [6] However, on the benchmark dataset, we observe some larger motions that cannot be dealt with, as in Figure 5. Considering the implementation of the separable 1d kernel version, the computation burden is eased a lot. We can increase the kernel size to 51, in order to capture larger motions.

4 Result and Analysis

4.1 How to Run The Code

All source codes, saved model and example output/datasets are under code folder.

The link for Vimeo90K dataset is: link1. Unzip and put vimeo_triplet under code folder.

The link for DAVIS2017 dataset is: link. Unzip only JPEGImages subfolder, and put it under code/DAVIS2017.

We only included some samples from above datasets in submission. The Middlebury benchmark dataset is all included.

The packages we used are cupy, sklearn, pytorch.

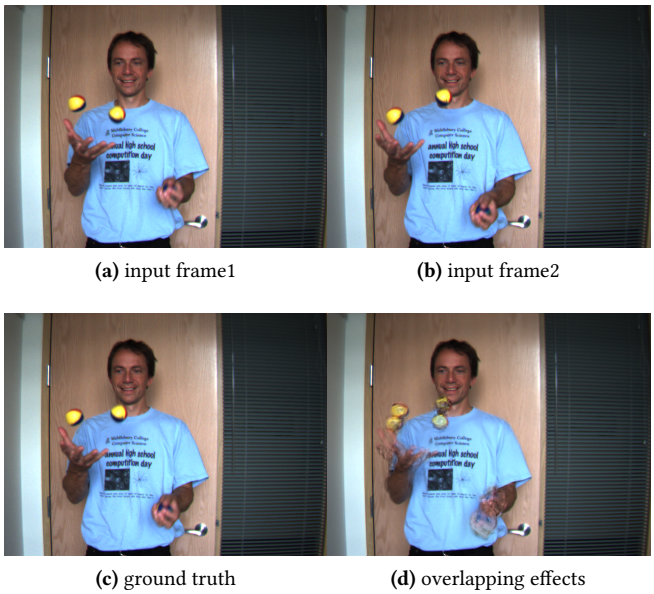
Run *python train.py* to train the model. Change the arguments on the top for the epoachs, batch size, etc.

Run *python test.py* to evaluate on validation set.

Run *python demo.py* to generate doubled frame videos on DAVIS2017/bear data.

4.2 Loss Functions

The comparison between L1 and L2 loss are in Figure 6. We can easily tell that results generated by L2-trained model present a lot more blurring effects, compared to L1-trained model. The blurring are especially obvious on the images with abundant textures, such as the last two images, the fur of the dog and the flower. Based on that, our model chooses L1 loss as the main loss criterion during training.



(a) input frame1 **(b)** input frame2

(c) ground truth **(d)** overlapping effects

Figure 5. Motion of the balls are not captured, becoming overlapping balls.

4.3 Training

The whole training process uses the training and validating loss as the metric. With lively printing losses, we are able to monitor closely on the train progress. Also, for every 2 epochs, Middlebury benchmark images will be used to examine the interpolation result, which provides a more accurate view of the performance. Below Figure 7 shows the training loss curve obtained.
On different epoaches, we can see the model starts to learn the details and becomes less blurring, as in Figure 8.

4.4 Evaluation

We use two metrics for evaluation: Peak signal-to-noise ratio (PSNR) and structural similarity (SSIM), which are two widely used quantity metrics in frame interpolation evaluation.

$$PSNR = 10log_{10}(\frac{(M)^2}{MSE}) = 20log_{10}(\frac{M}{RMSE}) \tag{5}$$

,where M is the maximum possible pixel value of the image and RMSE is the root of Mean Squared Error.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{6}$$

(a) Interpolation result trained with L1 Loss

(b) Interpolation result trained with L2 Loss

Figure 6. Comparison of the interpolation results trained with different loss functions.

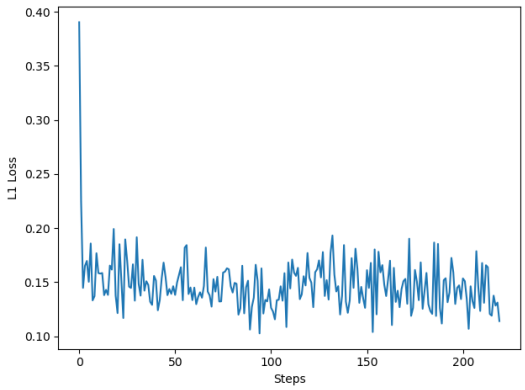


Figure 7. Training loss curve

(a) epoch 2 **(b)** epoch 4 **(c)** epoch 6

(d) epoch 8 **(e)** epoch 10 **(f)** epoch 12

Figure 8. Outputs from intermediate models for every 2 epochs.

, where μ_x, σ_x^2 are the average and variance of x, σ_{xy} is the covariance of x and y, c_1, c_2 are two variables to stabilize the division with weak denominator.[11]

Our scores are calculated on the Validation set from Vimeo90K. We compare the results with MDP-Flow2 [], DeepFlow2[] and phase-based method by Meyer et al.[5] as well as the AdaConv by [6] The results are shown in Table 2. As we can see from the table, the scores for our model is near the original AdaConv model. But, there is still gap between them.

4.5 Doubling Frame rate

The demo video for generating a 30fps videos out of 15fps videos can be found here. The upper left is the 15fps input videos from DAVIS2017 dataset. The upper right is the ground truth 30fps videos. The lower right is the interpolated 30fps videos using our model. We can see that the interpolated videos look smoother than 15fps videos, and look similar to the ground-truth 30fps video.

Table 2. Evaluation on the Vimeo90K testing set

	RMSE	PSNR	SSIM
Ours	11.54	27.40	0.88
MDPFlow2	7.40	32.47	0.940
DeepFlow2	7.82	32.16	0.935
Phase-based	17.16	26.05	0.705
AdaConv	10.14	30.16	0.885

5 Conclusion

In this project, we implemented a temporal frame interpolation model proposed by [6] using PyTorch. The model uses autoencoder with skip-connections to estimate the kernel and adaptive convolution to generate the interpolated frames. We trained the model with Vimeo90K dataset, and evaluated it on Middlebury and DAVIS2017 datasets. The resulting model can achieves some good evaluation scores, but still of some distance from the state-of-art model.

The future work for this project could focus on trying different loss functions, and applying more data augmentation techniques. As we haven't tried many loss functions in this project. And the data augmentation in this project is limited in horizontal or vertical flips.

6 Acknowledgements

We want to thank Dr. Corey Toler-Franklin for giving such an amazing course! And HyperGators for kindly providing computing resources.

References

[1] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. 2011. A database and evaluation methodology for optical flow. *International journal of computer vision* 92, 1 (2011), 1–31.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. (2014). arXiv:stat.ML/1406.2661

[3] Ziwei Liu, Raymond A Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. 2017. Video frame synthesis using deep voxel flow. In *Proceedings of the IEEE International Conference on Computer Vision*. 4463–4471.

[4] Michael Mathieu, Camille Couprie, and Yann LeCun. 2015. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440* (2015).

[5] Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. 2015. Phase-based frame interpolation for video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1410–1418.

[6] Simon Niklaus, Long Mai, and Feng Liu. 2017. Video frame interpolation via adaptive convolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 670–679.

[7] Simon Niklaus, Long Mai, and Feng Liu. 2017. Video Frame Interpolation via Adaptive Separable Convolution. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[8] ROYUD Nishino and Shohei Hido Crissman Loomis. 2017. Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems* (2017), 151.

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

[10] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. 2016. A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation. In *Computer Vision and Pattern Recognition*.

[11] Structural Similarity. 2010. Structural Similarity — Wikipedia, The Free Encyclopedia. (2010). https://en.wikipedia.org/wiki/Structural_similarity [Online; accessed 8-December-2020].

[12] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. 2019. Video enhancement with task-oriented flow. *International Journal of Computer Vision* 127, 8 (2019), 1106–1125.

[13] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. 2016. View synthesis by appearance flow. In *European conference on computer vision*. Springer, 286–301.