CS 763 / 764: Computer Vision

Real-Time Video Frame Interpolation via Adaptive Separable Convolution

Debasish Das (203050103)
Sagar Biswas (213050079)
Anandu A S (213074001)

Abstract

Existing paper[4], although has reduced the memory requirements of [3] for high-resolution (720p) videos, it still depends on the traditional encoder-decoder architecture based convolutional neural-network (similar to SegNet[2]) for extracting the features. This makes it difficult to be deployed in real-time scenarios because of the amount of computations needed to be performed to extract the features. We propose substituting this encoder-decoder architecture with another one which is specifically meant for real-time tasks (for ex., ENet[1]) and try to quantify the performance improvement while trading-off the quality of the output on a (spatially scaled-down) subset of `X4K1000FPS` dataset, used in XVFI [6]. The code is publicly made available as open-source.[1]

1 Introduction

Video frame interpolation is the task of synthesizing new video frames from the existing frames so as to increase the frame-rate of the corresponding video while ensuring that the motion of the objects captured in the video remains fluid. Typical way to achieve this is to compute the *optical flow* between two consecutive frames f_{t-1} and f_t and then interpolate a new frame *between* f_{t-1} and f_t based on the obtained motion parameters. Although this works, this approach heavily depends on the quality of motion estimation [3]. What [3] did was to compose both these operations into a single local convolution over the frames f_{t-1} and f_t, i.e. the interpolated frame is obtained by running a convolution over the (obtained features of the) two frames and then adding the intermediate results together. The idea was that the convolution kernel captured all the motion information as well as the coefficients required for synthesis. Although it worked, it was still constrained by large memory requirements due to large kernels required to capture large motion between the frames. What [4] did was to output two 1D convolution kernels, which will act as a proxy for the 2D convolution kernel output produced by [3]. This reduced the memory requirements from $O(n^2)$ to $O(n)$. Fig.(1) describes the architecture proposed by [4], where the encoder-decoder network is the main component, which uses traditional 2D convolutions. For down-sampling, the authors used traditional max-pool operations and for up-sampling, they used *bilinear* interpolation, followed by 2D convolution operations.

Because in real-life, we typically are interested in producing high-resolution videos, the given encoder-decoder network will serve as the bottle-neck, especially in real-time scenarios, due to the extremely large number of computations that would need to be done because of high-resolution

[1] GitHub repository: `https://github.com/0xd3ba/deep-video-interpolation`

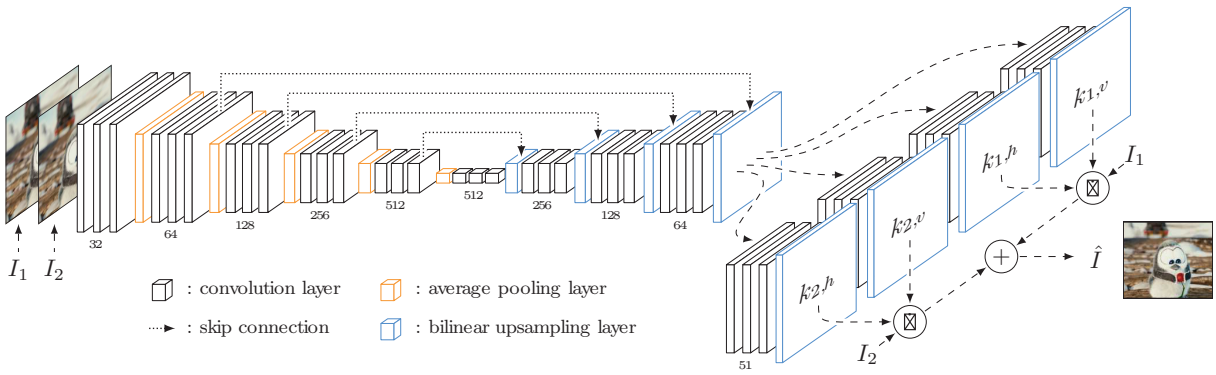

Figure 1: Network architecture of [4] for video frame interpolation. The encoder-decoder network extracts the features, which are then given as input to four different networks, each responsible for outputting one 1D kernel, which are then used to convolve with I_1 and I_2 to estimate the interpolated frame..

video frames, i.e. images. To be more specific, the encoder-decoder architecture used in [4] and [3] is *not* meant to *efficiently* work with high-resolution images and/or in real-time scenarios. What our project tries to do is to *check* the impact in computational performance and the quality of outputs produced, if an encoder-decoder network architecture meant specifically for real-time and/or high-resolution images is used, for ex. ENet[1] or ICNet [5], instead.

2 Dataset Used

The dataset that we used for training/testing is a subset of `XV4K1000FPS`, which is a dataset containing high-resolution (4K) and high-FPS (1000 FPS) videos (1 second each), used in [6]. Because of lack of computation resources on our part, we scaled down the videos spatially by a factor of 8, i.e. the resolution is (1/8)'th of the actual resolution and used the testing set (approx. 110 MB) instead of the actual training set (approx. 240GB) for the training and evaluation of the model.

3 Experiments and Results

We implemented the entire model proposed in [4] (as shown in fig.(1)) from scratch in PyTorch[2]. We decided to implement ENet[1] due to its simplicity, which was also implemented from scratch. Each model was trained on a GPU for around 100-300 epochs (4-5 hours per model) on Google Colab, using `Adam` optimizer with a learning rate of 10^{-3}. As mentioned before, due to lack of dedicated computation resources, we did not employ any other sophisticated training approaches, for ex., using learning-rate schedulers etc. because they would require hyper-parameter tuning, which would take even more time for training.
Using `ptflops` library, (available here), we tracked the number of computations per-input size and the total number of model parameters for each of the two implemented models. Table(1) shows

[2]One exception is the final separable convolution layer, defined in `sepconv.py` here or here, which we couldn't implement due to authors using CUDA for implementing it and due to some discrepancy we found (based on our limited understanding) in what they claimed in the paper and what was actually implemented, as explained in later sections. We borrowed that entire part of source code and have given proper credit wherever necessary.

Model	No. of Parameters (in Millions)
Default Encoder-Decoder	21.68
ENet	1.65

Table 1: Number of model parameters in the implemented models

Frame Size	No. of Multiply-Add Operations (in billions)		% Reduction
	Default Encoder-Decoder	ENet	
(3, 128, 128)	19.16	3.78	80.27%
(3, 256, 256)	25.02	15.12	39.56%
(3, 512, 512)	100.09	60.46	39.57%
(3, 1024, 1024)	400.37	241.85	39.59%
(3, 270, 512)	56.30	34.01	39.59%

Table 2: No. of computations (in billions) for each of the two models and the reduction in the number of computations when ENet is used for feature extraction. The highlighted cell indicates the frame-size that was used for training while as the rest are just for reference.

the number of parameters obtained and table(2) shows the amount of computations done by both the networks as well as the reduction in the number of computations when ENet is chosen as the network for feature extraction.

#TODO: Insert images of some the frames for reference by both the models + links to the output videos.

4 Observations

#TODO: Add comments regarding the quality of output

One discrepancy (based on our limited understanding) that we found out was the author's claim of outputting **four 1D kernels** from the four independent networks (after the feature extraction). But when we looked through their provided source code, we found out that the outputs were actually 2D, i.e. $(K_2, h), (K_2, v), (K_1, h)$ and (K_1, v) in fig.(1) were actually 2D, not 1D. Although this looks obvious just by looking at the fig.(1), we still do not understand how 1D kernels are obtained. Those outputs are then fed to separable convolution layer (implemented in `sepconv.py`, which was borrowed, as mentioned in the footnote previously) along with the corresponding input frames. Since it was implemented using CUDA, we could not make sense out of it due to limited time and hence we might have missed out on the crux of the paper, i.e. how the 1D kernels were obtained. Whether or not this is true, it is left as a future work to find out.

5 Conclusion

In this project, we have tackled the problem of video frame interpolation via deep-learning. Although existing solutions to the problem, for ex. [3] and [4] directly learn to output the interpolated frame without needing to compute the optical flow using convolutions, what they lack is the ability to operate in real-time scenarios because of the naive encoder-decoder based feature extraction module. We quantitatively analyzed the performance gain that can be obtained if an encoder-

decoder architecture specifically meant for real-time tasks can be used instead of the one that is currently being used in [3] and [4]. We also qualitatively analyzed the difference in quality of the outputs obtained by both the models. As for how it would actually perform in real-world scenarios and how practical it is, is currently left as a future work.

References

[1] Paszke, A., Chaurasia, A., Kim, S. and Culurciello, E., 2016. Enet: A deep neural network architecture for real-time semantic segmentation. arXiv preprint arXiv:1606.02147.

[2] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 12, pp. 2481-2495, 1 Dec. 2017, doi: 10.1109/TPAMI.2016.2644615.

[3] S. Niklaus, L. Mai and F. Liu, "Video Frame Interpolation via Adaptive Convolution," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2270-2279, doi: 10.1109/CVPR.2017.244.

[4] Niklaus, S., Mai, L. and Liu, F., 2017. Video frame interpolation via adaptive separable convolution. In Proceedings of the IEEE International Conference on Computer Vision (pp. 261-270).

[5] Zhao, Hengshuang et al. "ICNet for Real-Time Semantic Segmentation on High-Resolution Images." ArXiv abs/1704.08545 (2018): n. pag.

[6] H. Sim, J. Oh and M. Kim, "XVFI: eXtreme Video Frame Interpolation," 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 14469-14478, doi: 10.1109/ICCV48922.2021.01422.