Frame Interpolation for Animation

Corlene Rhoades
University of Wisconsin Madison

I. Abstract

Animation is a costly and time-consuming form of art, as a single second often includes 24 original drawings of the characters onscreen. This becomes expensive, requiring many man-hours for animators, and as such a framework for speeding up the need to draw in-between frames is desirable. Animated footage has qualities unique to live-action video interpolation, with sharp edges between regions of color and smooth blocks of consistent color. Other methods for frame interpolation fall short, often introducing artifacts similar to motion blur in live action footage. To combat this problem, I aim to create a framework to estimate in-between frames for animation through changing the loss function of other video interpolation methods and fine-tuning training on animated data.

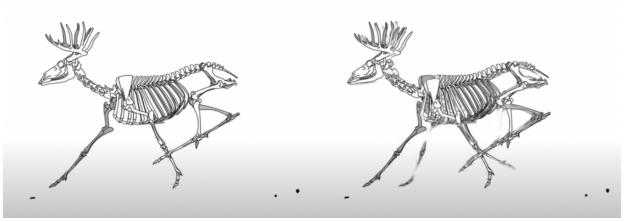
See this project's github at https://github.com/corlene-r/img_processing_proj.

II. Introduction

Most work for the task of frame interpolation has been largely focused on live action video. When extended to animation, many interpolation techniques have artifacts that would look fine in live action footage, but off for 2D animated footage.

Some 60-fps frame interpolation videos are publicly available on youtube, especially those using DAIN to interpolate video. This video[1] showcases a simple example of frame interpolation. DAIN specifically uses depth estimation to sense different components of the input frame to better get a sense as to the movement of components, but for the application of animation has some noticeable issues. Due to training on live action, the generated footage often has motion blur; 2D animation, however, is typically crisp without blurs removing detailed lines. Additionally, some artifacts can be seen where objects seem to phase into existence rather than smoothly moving across the screen, as seen here:

[1]Full link: https://www.youtube.com/watch?v=KbuiyYBdH6I



Convolution between two input frames has also been used to estimate movement [1]. This paper produces many sensible results, and is even shown to work fairly well on 3D animation, however performance on 3D animation does not generalize to performance on 2D animation. 3D animation more closely mimics the pixel-level semantics of live-action footage, as 3D animation can have blur as well as shadows similar to live action. These shadows have gradients that more closely mimic live-action shadows.

Other methods of frame interpolation exist which have similar issues to those mentioned above. Super SloMo attempts to focus on defining an optical flow for objects, and process afterwards to obtain more pixel level semantics [2]. Softsplat attempts to predict the future flow of objects in frame, allowing for multiple source pixels to land at the same output pixel [3]. [4] RIFE focuses on intermediate frame estimation, and does not rely on pre-trained optical flow models.

AnimeInterp was created specifically for the task of animation frame interpolation. This training results in many interpolated frames that are semantically better for 2D animation than other methods discussed. However, its outputs often include motion blur, which is out of place for 2D animations.

Figure 1 compares frame interpolation for animation for some of the methods discussed above (RIFE excluded). AnimeInterp noticeably has motion blur, with blurred eyes and many outlines blurred out, even though we would expect animation frames to include sharp, crisp edges. I attribute this to AnimeInterp's loss function, which I will discuss further in my proposed methodology.

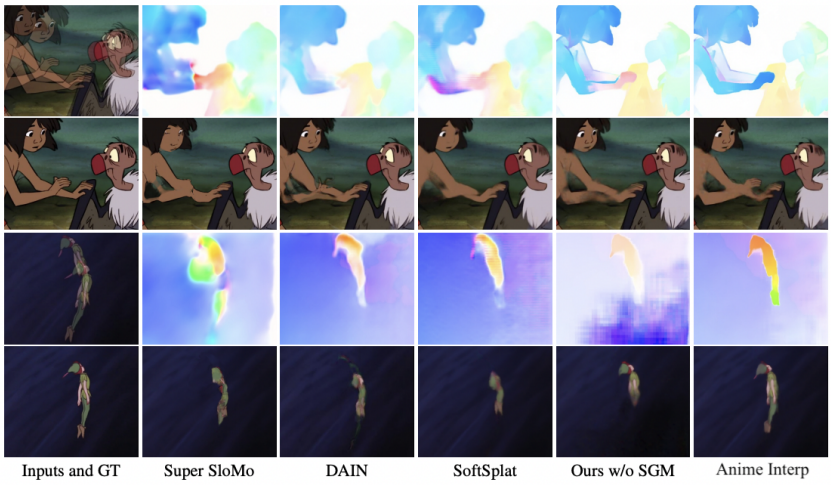


Fig. 1. Performance of frame interpolation on animation data. The first row shows the overlapped input frames and the predicted optical flow. The second shot shows the estimated truth. Figure from [5]

III. Dataset

For this task, a dataset involving a large collection of animated frames to interpolate is desirable. The dataset I plan to use can be found here. This dataset contains 12,000 frame triplets of resolution 1920×1080 or 1280×720 from over 30 movie series from different producers [5]. The dataset also removes any scenes from animated movies where interpolation would not be viable, such as cuts between shots where no sensible interpolation is possible.

IV. Methodology

A. Loss Function

I would like to focus specifically on the task of reducing motion blur in animated footage, attempting to maintain the crisp edges between colored regions found in animation footage. I hypothesize that this is caused by the use of loss functions that minimize the difference between pixel values.

The loss function used for RIFE is as follows:

$$L_{RIFE} = \sigma_x \sqrt{(\hat{I}_t(x) - I_t^{GroundTruth}(x))^2 + \epsilon^2}$$

This loss is similar to an L_1 loss, with an error factor [4]. With this loss function, we see a fair ammount of motion blur in estimated output images. AnimeInterp also directly uses L_1 loss as [5]:

$$||\hat{I}_{1/2} - I_{1/2}||_1$$

I believe that this is part of what contributes to a large amount of blur. For live action, this blur is acceptable and understood as a part of the filming process, but blur looks strange and out of place in animated frames. It would make sense to use a pixel that smudges values between adjacent pixels in live action, but in animated footage, colors are typically separated into distinct blocks without a blur between color boundaries.

As such, I would like to redesign the training loss to focus on a classification loss. [6] proposes the following loss function based on color classes; though note that this loss is created with the mindset of image colorization:

$$L_{CL}(\hat{Z}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_{q} Z_{h,w,q} \log(\hat{Z}_{h,w,q})$$

In frame interpolation, we can sample the input frame to generate expected color classes for the input pixels. Given data dependent classes for each pixel, we would want the following:

- The distribution of color classes should be similar to the distribution in the frames interpolated.
- The movement of color classes should match the movement of the underlying pixels.
- We would like to ensure that class pixels are locally clustered.

Thus, I would like to propose a weighting to the loss function as the following. I is the ground-truth image, $\hat{I}$ is the interpolated image, h, w sum across

the width and height of an image, and q sums across the generated classes:

$$L_{CL}(\hat{I}, I) = -\sum_{h,w,q} I_{h,w,q} \log(\hat{I}_{h,w,q})$$

A weighted combination of this loss function and that used by RIFE is used in the final model. This results in the loss function of the final project being the sum:

$$\mathcal{L} = L_{RIFE} + \beta L_{CL}$$

I will call the parameter β the loss weighting parameter; this is used to determine how much classification loss weights into the total loss generated by the model.

B. Code Implementation

I have built my code from the RIFE codebase, and modified its loss function as well as other portions of its code for this project [4]. First, I needed to make the code suitable for programming on my machine (a mac). This involved configuring environment variables and changing lines of the code to use the GPUs on my mac. Next, I processed all data found from AimeInterp to be put into the format required by the RIFE model, allowing me to train the base RIFE model. After this model was trained, I edited the file *model/laplacian.py* to add a categorical loss function and placed this in the file `model/crossentropy.py`. This altered loss function was configured to allow for input of the loss weighting parameter β. This led to many training trials using different parameters for β, which shall be detailed further in the results section.

A more detailed log of the details of my implementation can be found in the file `corlene_notes.md`, which logs the files edited and created as well as a chronological runthrough. This also details problems encountered in setting up the environment; theoretically `requirements.txt` should install all necessary packagees, except for potentially the nightly torchvision build.

C. Training

The unedited RIFE model, $\beta = 0.3$, and $\beta = 3$ models were each trained for 300 epochs. The models for $\beta = 30$ and $\beta = 300$ were trained by taking the model for $\beta = 3$ at epoch 255 and fine-tuning for the remaining 45 epochs (this was due to the time required to train from epoch 0 taking around 100 hours).

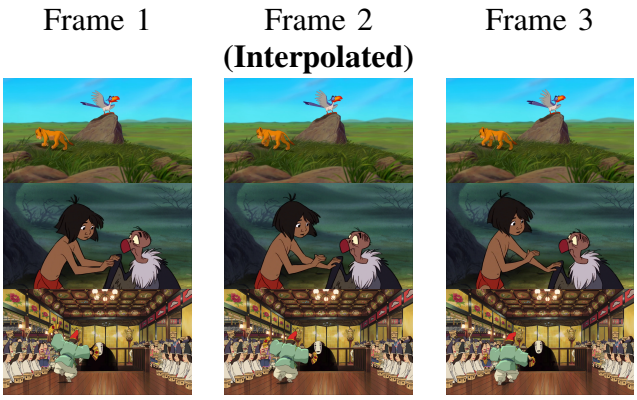


Fig. 2. Input images for model. Note that frame 2 is the desired output result in these cases, with frames 1 and 3 as input.

You may run `sh run_training.sh` to replicate the training on your machine. Note that training is batched into epochs of 15 to save intermediate results in case if training crashed.

V. Results and Analysis

Figure 3 shows the output of my trained model with various weightings for the loss weighting parameter β. Note that a higher β corresponds to more weighting to the categorical loss. The performance of $\beta = 0.3$ and $\beta = 3$ seem to result in semantically the best output, as they have more well-blocked colors with less motion blur.

We can analyze these details more closely by looking at Simba's leg in 4. Note that simba's leg in the RIFE model lacking my categorical loss ($\beta = 0$) blurs in notably more with the background than $\beta = 0.3, 3,$ and 30. This could be due to the distance-based loss metric averaging the color of the background and simba's leg, resulting in a more blurred out coloration than would be desirable for animation. The leg in the image with $\beta = 300$ shows a quite high level of blurring as well; and generally the $\beta = 300$ results compare quite poorly to every other interpolation method, showing that an the distance-based loss metric does provide some improvement to results over a purely categorical loss metric. In my subjective opinion, the frames generated by $\beta = 3$ look the best amongst the results, as they seem to have a strong coloration as compared to the background, maintain some sense

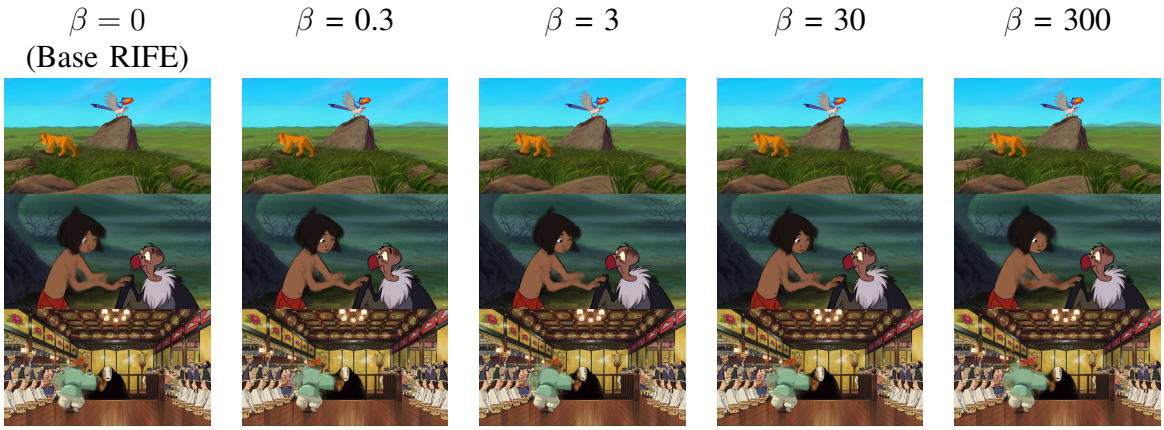


Fig. 3. Output images generated by unedited RIFE model and my model trained using different values for its loss weighting parameter.

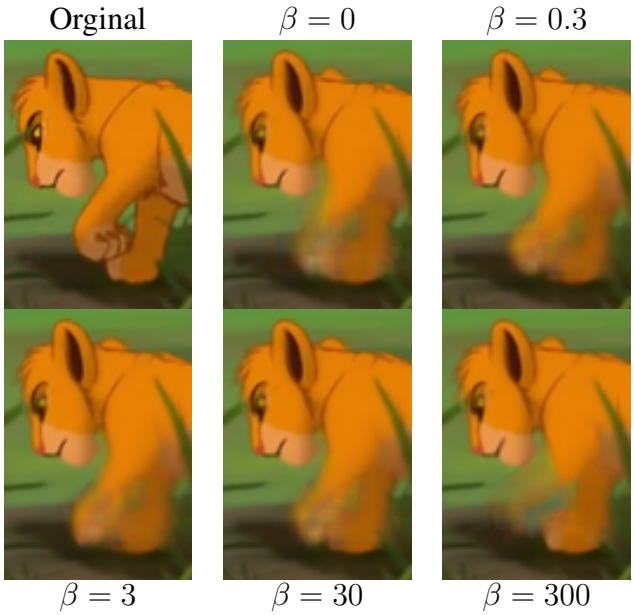


Fig. 4. Detailed look at the characteristics of Simba's leg, and how the parameter of β influences interpolation.

of the outline of the images, and be in roughly the right place. $\beta = 0.3$ performs similarly well to $\beta = 3$. It is possible that training the $\beta = 30$ and $\beta = 300$ models from epoch 0, rather than finetuning another model from epoch 255, could result in better results.

VI. Conclusion

Though this work shows progress, there is also a notable skew to the dataset which could impact the extendability of this model to other forms of animation. This dataset primarily features animation from disney and ghibli movies, which is representative of especially high quality, large frame count, detailed animation and does not showcase animation of different styles, choppy animation, or animation with more static posing (such as that which is predominantly in television). Curating a dataset more representative of varied animated styles would result in performance more applicable to more types of animation. Also, analyzing the performance of this model on data of different animation styles, such as televisions shows like the simpsons that are restricted to more static animation or sketchier animation styles, could show further areas in which this model succeeds and fails artistically.

This work could additionally be extended by playing with more forms of loss function, and comparing their results on animated versus live action data. The characteristics of movement of animation do vary quite a bit from live action, and thus it would be desirable to improve these further. Trying different forms of distance-weighted loss, changing the weighting of adjacent pixels, and other methods could improve the visual performance of this model.

This project shows considerable improvement of frame interpolation for the application of animation over RIFE's model. Blurriness in these models are still very visible, but notably less when trained with categorical loss, showing that this methodology shows improvement for the task of animation. Further work could be done to improve this performance further.

References

[1] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive separable convolution. *CoRR*, abs/1708.01692, 2017.

[2] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation, 2018.

[3] Simon Niklaus and Feng Liu. Softmax splatting for video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[4] Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. Real-time intermediate flow estimation for video frame interpolation, 2022.

[5] Li Siyao, Shiyu Zhao, Weijiang Yu, Wenxiu Sun, Dimitris N. Metaxas, Chen Change Loy, and Ziwei Liu. Deep animation video interpolation in the wild, 2021.

[6] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. *CoRR*, abs/1603.08511, 2016.

[7] Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. RIFE: real-time intermediate flow estimation for video frame interpolation. *CoRR*, abs/2011.06294, 2020.