Frame Interpolation Using Generative Adversarial Networks

Mark Koren [*]
mkoren@stanford.edu

Kunal Menda[*]
kmenda@stanford.edu

Apoorva Sharma[*]
apoorva@stanford.edu

Stanford University 496 Lomita Dr. Stanford, CA 94305

Abstract

Video frame interpolation is an elusive but coveted technology with the potential to have a far reaching impact in the video streaming service industry. In this paper, we present a novel deep learning architecture for increasing the frame rate of videos. We utilize a convolutional neural network architecture and generative adversarial networks to create a model capable of taking a pair of video frames and generating the frame in-between them. By training on a combination of the ℓ_1, MS-SSIM, and GAN losses, we are capable of generating frames that far exceed the qualitative appeal of those generated by closely related works [12], and have comparable performance on the quantitative measures of SSIM and PSNR.

1. Introduction

Video services, especially streaming services, are some of the most recognizable brands in technology today. One of the hardest problems in this exciting field is that of frame interpolation.

Frame interpolation, for the purposes of this project, is the action of generating a frame for a video, given the immediate frames occurring sequentially before and after. This allows a video to have its frame rate enhanced, which is a process known as *upsampling*. In the general upsampling task, we cannot assume access to ground truth for the interpolated frames.

High fidelity upsampling can be applied to video *compression*, since could store only key frames of the video, and interpolate at playtime to fill in the missing parts. For compression tasks, the original high frame rate video exists, and thus the ground truth for the interpolated frames is available.

Inspired by the successes of end-to-end deep convolutional networks in outperforming conventional techniques for image classification, we propose an end-to-end neural architecture ("FINNiGAN") for the frame interpolation task. The input to this algorithm is a pair of sequential frames from a video. We use a convolutional neural network (CNN) architecture involving the generative adversarial network setup to generate the frame which would appear temporally between the input frames.

In the following sections, we first discuss related work, then outline our methods, introduce the dataset used for testing, and finally discuss results.

2. Related Work

There are several conventional image processing techniques for video frame interpolation. The simplest method for interpolating between two frames is *Linear Frame Interpolation*(LFI). In this technique, for each pixel location x in the interpolated frame, the value is linearly interpolated between the neighboring frames:

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

As this is a pixel to pixel method, it fails to properly account for motion of objects across pixels. This creates an effect known as "ghosting" where objects that are in motion have multiple edges in the interpolated frame.

Current state of the art frame interpolation is done using an algorithm called Motion-Compensated Frame Interpolation (MCFI), which is currently used in many HDTVs [16]. MCFI techniques work in two parts: Motion Estimation (ME) and Motion Compensation (MC). ME often involves computing the 'velocity' of each pixel in the frame, i.e. how a given pixel's content shifts between the frames [15]. At a high level, the MC step involves using these motion estimates to move each pixel halfway in the same direction. [14], but suffer from their own artifacts, such as "tears" or misplaced blocks, resulting in qualitatively unsatisfactory results described as having a "soap-opera effect."

Recent work by Guo and Lu [4] presents an improvement to MCFI called I-MCFI, and also gives a survey of

[*]These authors contributed equally.

other state-of-the-art frame interpolation techniques, such as Adaptive Vector Median Filtering (AVMF) and Motion Vector Smoothing (MVS). Our work will be compared against these algorithms as a baseline.

At its core, frame interpolation is a two image to single image translation task, which involves making sense of information from two images, and then generating a single image. Convolutional neural networks (CNN) can be applied in an encoder-decoder setup to learn implicit features in images. Previous work utilized a CNN architecture for frame interpolation task and achieved promising results [12]. The primary issue with the previous work was the blurriness and noisiness of the generated images.

Generative Adversarial Networks (GANs), have been shown to be very good at realistic image generation [3][10]. Conditional-GAN [8] and pix2pix [5] adapt the GAN framework to single-image-to-single-image translation, achieving good results in going from outlines and cartoons to photorealistic images. In this paper, we build on the work presented by Sharma et al. [12] by incorporating a GAN architecture to improve the photorealism of the result.

3. Methods

Our method performs **F**rame **I**nterpolation with a Convolutional **N**eural **N**etwork as well as **i**ncorporating **G**enerative **A**dversarial **N**etworks for image refinement, hence the name "FINNiGAN." The task is divided into two parts. The Structure Interpolation Network, covered in detail in section 3.1, takes the two adjacent frames as an input and generates the structure of the middle frame. However, the MS-SSIM loss takes a gray-scale image, so this network does not reproduce colors well. To address this, we pass the poorly colored but ostensibly structurally correct frame to the Refinement Network which corrects the colors, and cleans up some structural errors. This architecture is covered in section 3.2. We then use SSIM as an evaluative metric to compare our results to previous work, with SSIM being covered in 3.3.

3.1. Structure Interpolation Network (SIN)

3.1.1 SIN Architecture

The SIN architecture was largely inspired by the work done by Isola et. al [6], and is shown in Figure 1. Given a sequence of frames, {F1, F2, F3}, the Generator takes the image doublet {F1, F3} as a 6 channel input. Let C_K^{F-S} denote a convolution-batchnorm-lrelu-convolution-batchnorm-lrelu block with K filters, a $F \times F$ filter size, and a $S \times S$ stride for each convolution. Additionally, let T_K^{F-S} represent a resize-convolution-batchnorm-lrelu-convolution-batchnorm-lrelu block with K filters, a $F \times F$ filter size, and a $S \times S$ stride for each convolution. The resize operation is a bilinear resize. Finally, let D_K^{F-S} represent T_K^{F-S} with dropout applied on the output of the last lrelu block. The Generator architecture is C_{32}^{3-1}-C_{64}^{3-1}-C_{64}^{3-1}-C_{128}^{3-1}-D_{128}^{4-1}-D_{64}^{4-1}-D_{64}^{4-1}-D_{32}^{4-1}. All Leaky ReLUs use 0.2 for the leak slope, and the dropout probability is 0.5. The generator uses a U-net structure [11], so the output of the ith convolution block is stacked with the output of the j $-$ 1 deconvolution block as the input to the jth deconvolution block, where i and j count from the ends toward the middle. For example, the output of C_{32}^{3-1} is stacked with the output of the second D_{64}^{4-1} as the input to D_{32}^{4-1}. This allows the network to retain structural information during upsampling that may normally be lost. The output of the generator is a 3 channel image.

3.1.2 Loss Functions

The SIN stage uses a weighted combination of four different loss functions: ℓ_1 loss, MS-SSIM loss, clipping loss, and a discriminative loss.

ℓ_1 loss: The ℓ_1 loss is computed as

$$\ell_1 = \frac{1}{HWD} \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{k=1}^{C} \frac{1}{255} |p_{i,j,k}^{SIN} - p_{i,j,k}|$$

where H, W, C are the dimensions of an image and $|p_{i,j,k}^{SIN} - p_{i,j,k}|$ is the absolute difference between a specific pixel of the generated image and the target image. This difference is scaled by $\frac{1}{255}$ so the ℓ_1 loss for each pixel and the total ℓ_1 loss both scale from 0 to 1. The goal of the ℓ_1 loss is to capture the general colors and intensity of the image. In general, pixel-based losses are not sufficient for image construction tasks due to their inability to reflect the way humans process images [7]. Specifically, the ℓ_1 has difficulty capturing the high-frequency domain of an image, which results in blurry images. However, ℓ_1 loss has been shown to perform better than ℓ_2 loss for image tasks, and can be useful when used in conjunction with other losses. [19].

MS-SSIM loss: Structural Similarity is a popular choice for measuring the similarity of images as perceived by the human visual system [17]. An index used to quantitatively measure structural similarity is called the Structural Similarity Index (SSIM). However, a drawback of this index is that is is a single-scale approach, in that it is a good approximation of image quality only at a specific display resolution and viewing distance. The Multi Scale Structural Similarity Index (MS-SSIM) addresses these issues by iteratively applying low-pass filters and down-sampling the image at a variety of stages and geometrically combining the results [13].

In this work, we measure the MS-SSIM between the generated image and the ground truth, using weighting parameters suggested by the original MS-SSIM paper [13]. In

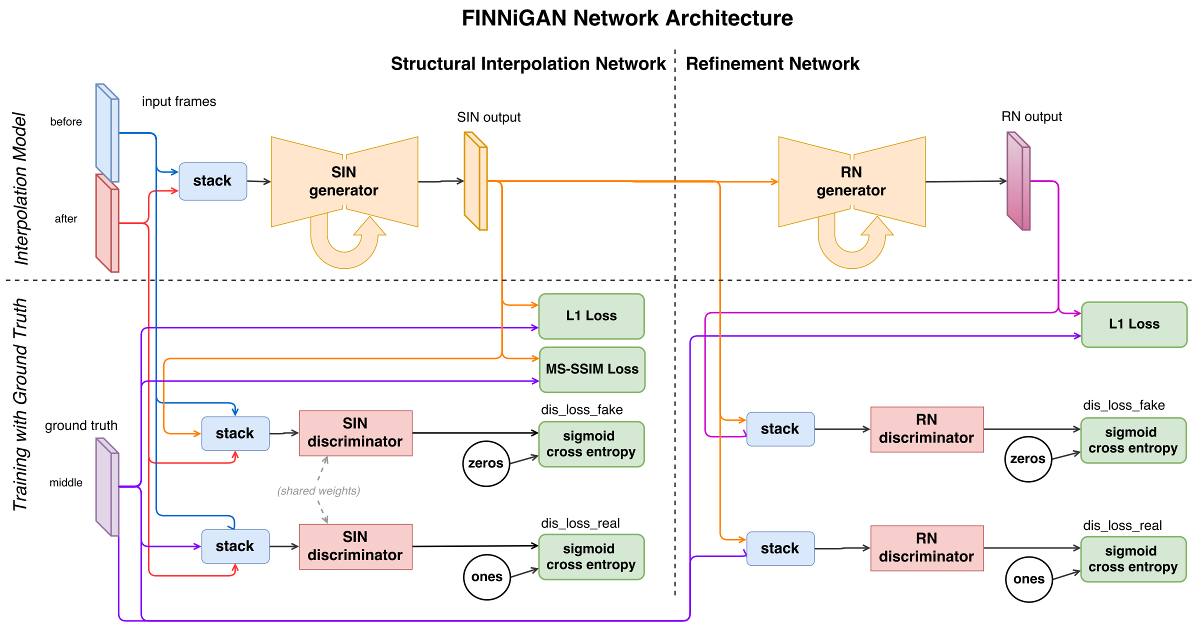


Figure 1: The FINNiGAN model has two sections, both incorporating the GAN framework with auxiliary losses. The Structural Interpolation Network weights the MS-SSIM loss heavily, while the Refinement Network uses the L1 loss alone.

order to do so, we utilize a TensorFlow implementation of MS-SSIM [1]. To pre-process the generated image and target so they are valid inputs to this implementation, we add back the mean images, clip the image to a valid range, and convert the images to gray-scale. As these operations can kill gradients and lose the ability to learn color information, we utilize the additional losses below to account for this pre-processing.

Clipping loss: As mentioned, the MS-SSIM loss is computed after clipping the generated image pixels to in the valid range, [0,1] in our case. This has the unwanted side-effect of preventing the gradient on the MS-SSIM loss to flow through pixels that are clipped. To prevent this, we add another loss to penalize the generator against outputting values outside the allowed range. The loss is calculated as follows:

$$\text{Loss}_{\text{clipping}} = \frac{1}{HWD}\sum_{i=1}^{H}\sum_{j=1}^{W}\sum_{k=1}^{C}(\bar{p}_{i,j,k}^{SIN} - p_{i,j,k}^{SIN})^2$$

where H, W, C are the dimensions of the image and $\bar{p}^{SIN}$ is the clipped generated image, and p^{SIN} is the unclipped version of the same image.

[1]https://stackoverflow.com/questions/39051451/ssim-ms-ssim-for-tensorflow (version:06-12-2017)

Discriminator Loss: One of the major changes from our previous work [12] is the addition of a discriminator, which in combination with the generator is a Generative Adversarial Network [3]. The Generator and the Discriminator play a 2-player minimax game, where the generator is trying to fool the discriminator and maximize the score from the discriminator on fake images and the Discriminator is trying to correctly score both real and fake images. Let the SIN module be defined as G_{θ_g}. Then the losses for the Generator and the Discriminator when training are as follows:

$$\text{Loss}_{\text{gen}} = -\log D_{\theta_d}(G_{\theta_g}(F1, F3)) \tag{1}$$

$$\begin{aligned}\text{Loss}_{\text{dis}} = &-\log D_{\theta_d}(F1, F2, F3) + \\ &-\log(1 - D_{\theta_d}(F1, G_{\theta_g}(F1, F3), F3)\end{aligned} \tag{2}$$

Let C_K^{F-S} denote a convolution-batchnorm-lrelu block with K filters, a $F \times F$ filter size, and a $S \times S$ stride for each convolution. The architecture for the Discriminator is C_8^{4-2}-C_{16}^{4-2}-C_{32}^{4-2}-C_{64}^{4-2}-C_1^{4-2}.

3.2. Refinement Network

The output frame by the SIN model was seen in experiment to be good at reproducing the structure of middle frame, eliminating the ghosting artifacts typically seen with the LFI method. However, the method struggled with color and texture reproduction. Section 5 shows examples of this.

To address this, the FINNiGAN pipeline applies an image-to-image translation GAN model to this SIN output to re-color and re-texture the image frame to be more realistic. Specifically, we train the pix2pix model [6] for this task. The model has been shown to successfully transform semantically segmented scenes to their photo-realistic counterparts, and thus we found the model appropriate to improve the realism of the SIN output images.

pix2pix uses a similar structure to the SIN model, however the generator only takes a single image, and the discriminator scores pairs of images rather than triplets. It augments the GAN loss with an ℓ_1 loss. Further details about the pix2pix architecture can be found in the original paper [6].

3.3. Evaluation Metrics

In order to evaluate the performance of our model on the test set, we will utilize both qualitative and quantitative metrics. We will qualitatively compare outputs from our model to the naive LFI of the two input frames, as well as compare our results to those of the Deep Frame Interpolator (DFI) [12]. Qualitative comparisons will be made in the various algorithms' abilities to avoid ghosting, preserve detail, and avoid generate jarring artifacts.

Additionally, we will compare these algorithms to state of the art Motion Compensated Frame Interpolation (MCFI) techniques [4]. We will perform these comparisons quantitatively, using single-scale SSIM as well as Peak-Signal-To-Noise-Ratio (PSNR) as our metrics. The MATLAB implementation of both metrics will be used [1]. We will present the mean statistics for the highest-scoring 50 frames, as done in related works [4].

4. Dataset

The intended application of this work is to up-sample an video from some original frame-rate to twice that frame-rate. In order to test our work's ability to achieve this task, we artificially down-sample a video to half its original frame-rate, keeping every other frame as a ground-truth comparison, which we will call the 'test-set'. We call the left-over frames the 'training-set'. In order to learn a model that can up-sample the training-set back to original frame-rate, we again perform an action of down-sampling on the training-set. We take every triplet of frames in the training-set and treat the first and third images in the triplet as the input to the model, and the middle frame as the target. Once the model is trained, we then pass every pair of images in the training set in an attempt to generate the frame in-between them which corresponds to an image in the test-set, which are images that the model has never seen in training. Figure 2 summarizes this process, demonstrating the frames used in training and their counterpart in the test-set.

For this project, a dataset of videos commonly used in frame-interpolation literature called *Xiph.org Video Test Media* [18] will be used. Particularly, we evaluate on videos

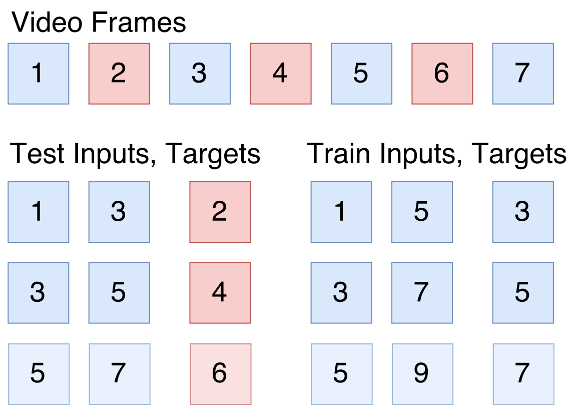


Figure 2: Example showing how a sequence of frames from a source video can be sorted to create train and test datasets. In each case, the target is the frame temporally between the two input frames.

referred to as 'Bus', 'Football', 'News', and 'Stefan'. The chosen videos represent the extrema in performance reported by related work, where Bus and News are typically the hardest and easiest videos respectively to score on by the SSIM metric, while Football and Stefan often lead to the least visually appealing results.

5. Experimental Results

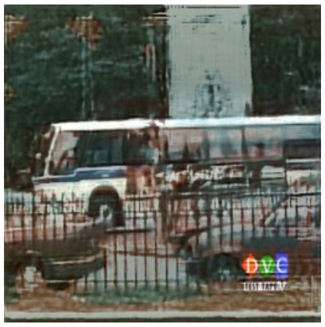
We implemented the model above using TensorFlow in Python. [2]. Significant portions of the code were adapted from the DCGAN and the Pix2Pix TensorFlow implementations. [2]

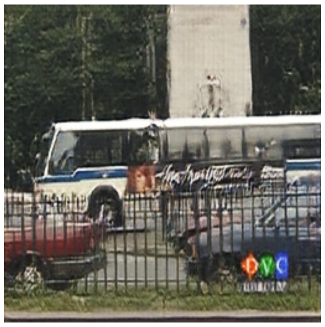
5.1. Qualitative Results

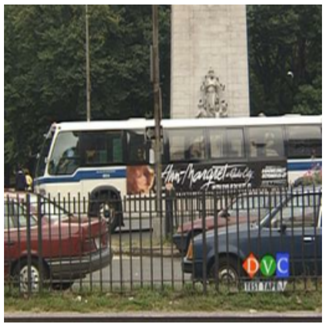
5.1.1 Comparison with Baselines

Figure 3 shows the output of both the intermediate SIN stage and the full FINNiGAN pipeline, as well as the ground truth and LFI versions as baselines. Comparing the SIN output with the LFI shows that the network does a good job of solving ghosting issues. The fence in particular is much improved, and far more visibly pleasing for a human in both the stills presented here and in video form, where the ghosting effect is particularly annoying. However, SIN does not compare favorably with either baseline in terms of color. Many parts of the image have discolorations, and even parts that seem close to correct are faded. Examples of these failures are a portion of the red car that has turned blue, or the pillar which has also taken on a blueish hue. However, many of these issues are fixed after passing through the RN. The colors and structures are very close to the ground truth, there is almost no ghosting visible, and artifacts from the

[2]https://github.com/carpedm20/DCGAN-tensorflow
https://github.com/affinelayer/pix2pix-tensorflow


(a) Output from SIN/input to Refinement Network


(b) Output from Refinement Network


(c) Ground truth

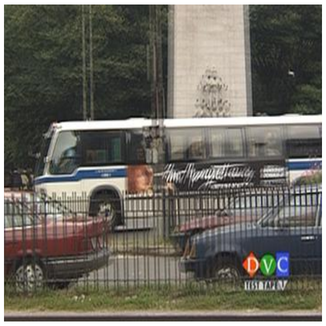
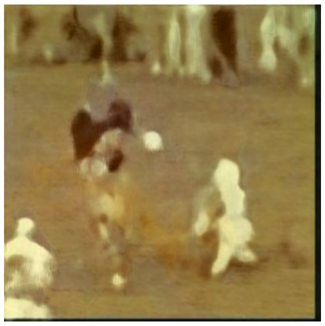
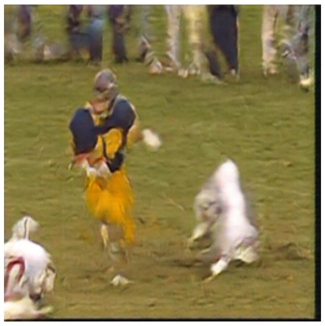
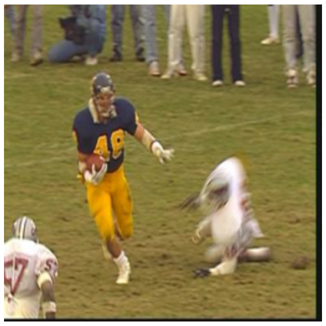

(d) Linear Frame Interpolation on inputs

Figure 3: Results from (a) just the SIN (b) the entire FINNiGAN compared to (c) the ground truth image and (d) naive linear frame interpolation. The images shown are from the Bus test set. Note that the output from the SIN already eliminates much of the ghosting in the fence seen in the LFI frame. The Refinement Network successfully reproduces the colors from the ground truth image without adding too many artifacts.


(a) Output from SIN/input to Refinement Network


(b) Output from Refinement Network


(c) Ground truth

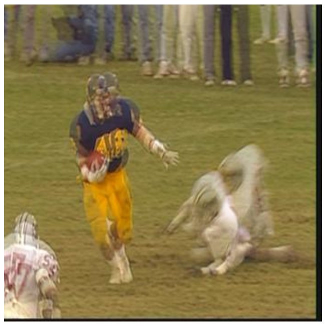
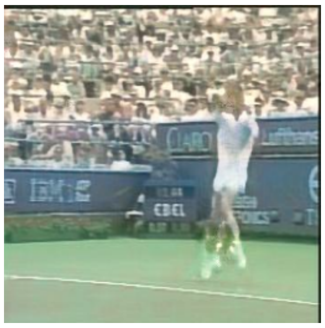
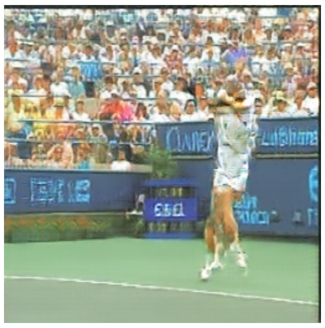
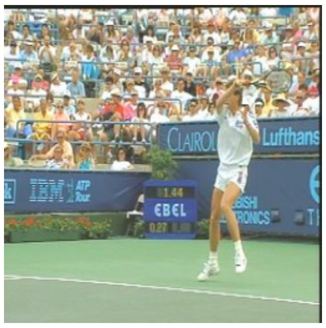

(d) Linear Frame Interpolation on inputs

Figure 4: Results from (a) just the SIN (b) the entire FINNiGAN compared to (c) the ground truth image and (d) naive linear frame interpolation. The images shown are from the Football test set. The colors and background are faithfully reproduced, but the players have some unappealing distortion.


(a) Output from SIN/input to Refinement Network


(b) Output from Refinement Network


(c) Ground truth

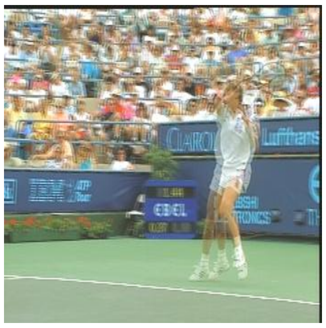
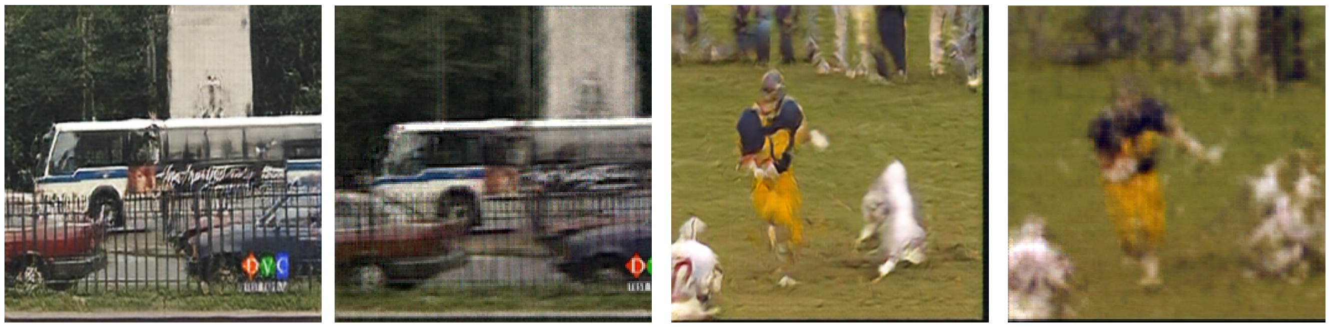

(d) Linear Frame Interpolation on inputs

Figure 5: Results from (a) just the SIN (b) the entire FINNIGAN compared to (c) the ground truth image and (d) naive linear frame interpolation. The images shown are from the Stefan test set. Although SIN and FINNIGAN capture the crowd well, the methods struggle with the head area of the player. Still, the ghosting shown in LFI is not as prevalent



(a) Output from FINNiGAN (b) Output from DFI [12] (c) Output from FINNiGAN (d) Output from DFI [12]

Figure 6: Comparison between FINNiGAN and the Deep Frame Interpolator [12] on the Bus and Football test sets. The FINNiGAN result is visually a much crisper and more realistic image. Reproduction of static areas of the video is especially improved with FINNiGAN.

Table 1: Comparison of FINNiGAN to baseline methods [4][12] on the SSIM metric.

Sequence	LFI	AVMF	MVS	I-MCFI	DFI Upsampling	FINNiGAN
Football	0.3865	0.5366	0.4972	0.5985	**0.8477**	0.8393
Bus	0.3378	0.8750	0.6568	**0.9043**	0.5806	0.5539
Stefan	0.7392	0.8870	0.8719	**0.9050**	0.7794	0.7932
News	0.9683	0.9670	0.9653	0.9676	**0.9763**	0.9551

upsampling process are small. However, the results on the Bus dataset are better than others, such as the Stefan video, which is shown in Figure 5. Here, the output from FINNiGAN is great in some areas, such as crowd recreation, or the player's legs. However, the player from the shoulders up is an indecipherable mess, and the output of SIN actually recreates the advertisements in the background more faithfully than the final FINNiGAN output. Similarly in Figure 4, which shows the Football dataset[3], The grass and the legs in the background are very good, and in fact the whole frame looks good from a distance, but closer inspection reveals distortions in the players jerseys and helmets.

5.1.2 Comparison with Deep Frame Interpolation (DFI)

A comparison of the FINNiGAN results and the DFI results for a frame from Bus and Football are shown in Figure 6. The qualitative improvement is clear. DFI was trained on the ℓ_1 loss, so it suffers from blurriness due to the incompleteness of its loss function. Additionally, the colors are slightly faded in contrast with the much sharper FINNiGAN results. However, the improvements from DFI go beyond changing the loss function. The GAN structure helps to identify failures that MS-SSIM and $\ell 1$ can't easily express. Furthermore, upsampling was done using bilinear resizing along with a convolution to eliminate common checkerboard-style artifacts [9]. These changes resulted in a sizable improvement in image quality.

5.2. Quantitative Results

Tables 1 and 2 summarize the performance of various algorithms on the single-scale SSIM and PSNR metrics respectively. The algorithms are compared on Football, Bus, Stefan, and News datasets. We observe that FINNiGAN is dominated by the state-of-the-art MCFI algorithm, called I-MCFI [4] on the the PSNR metric. This is likely because PSNR is related to the mean-squared-error metric, or the ℓ_2-loss, which is not explicitly optimized by FINNiGAN. However, we observe that FINNiGAN has similar performance to DFI on the SSIM, which is drastically better than I-MCFI on the Football dataset, comparable on the News dataset, and worse on the Bus and Stefan datasets. However, it was observed that the reported values for SSIM when using DFI may have been over-estimates, as DFI was run on down-sampled versions of the images [12], which increases the apparent SSIM scores on these datasets.

6. Conclusion

Frame interpolation is a challenging video processing task, where conventional techniques often introduce visually unpleasing artifacts. The FINNiGAN model is an effort

[3]It should be noted that the player carrying the ball here fumbles, because Cal is bad at football.

Table 2: Comparison of FINNiGAN to baseline methods [4][12] on the PSNR metric.

Sequence	LFI	AVMF	MVS	I-MCFI	DFI Upsampling	FINNiGAN
Football	19.316	20.886	20.658	21.422	**21.812**	19.2348
Bus	18.528	25.016	21.363	**26.349**	20.174	20.3076
Stefan	23.848	27.523	26.530	**28.021**	20.547	20.6814
News	38.050	37.954	37.812	**38.431**	32.252	30.2051

to apply the latest neural image processing techniques, such as Generative Adversarial Networks towards this problem. The proposed model generates interpolated frames without the unpleasant "ghosting" effect of the simple LFI method, and tends not to have the "tearing" artifact commonly seen with motion compensating frame interpolation methods. Visually, the FINNiGAN generated frames look close to the ground truth from a color and broad structure sense. As an improvement of our earlier work, the DFI method, FINNiGAN results are able to accurately reproduce details, especially in parts of the scene with little motion. FINNiGAN also attempts to fill in details in the fast-moving regions of the video, which make the result better from distance but introduces some extraneous hallucinated details. The source code for the SIN network can be found at `https://github.com/apoorva-sharma/finn`, and the source code for the RN is an out-of-the-box implementation of pix2pix found at `https://github.com/affinelayer/pix2pix-tensorflow`.

In future work, we hope to improve upon our work here by better choosing hyperparameters and network sizes to potentially combine the SIN and RN networks. Furthermore, adding recurrence to the model may help improve the quality of the result by enforcing some sort of temporal consistency. Recurrence would also allow different input-output structures, such as extrapolating the next frame given a sequence of previous frames, a use case that may have applications in the video streaming sector.

References

[1] M. (2016a). Image processing toolbox.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[4] D. Guo and Z. Lu. Motion-compensated frame interpolation with weighted motion estimation and hierarchical vector refinement. *Neurocomputing*, Mar. 2016.

[5] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[6] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[7] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[8] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[9] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.

[10] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[11] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[12] A. Sharma, K. Menda, and M. Koren. Convolutional neural networks for video frame interpolation.

[13] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 2, pages 1398–1402. IEEE, 2003.

[14] Wikipedia. Motion compensation — wikipedia, the free encyclopedia, 2016. [Online; accessed 8-November-2016].

[15] Wikipedia. Motion estimation — wikipedia, the free encyclopedia, 2016. [Online; accessed 18-November-2016].

[16] Wikipedia. Motion interpolation — wikipedia, the free encyclopedia, 2016. [Online; accessed 1-December-2016].

[17] Wikipedia. Structural similarity — wikipedia, the free encyclopedia, 2016. [Online; accessed 6-December-2016].

[18] Xiph.org. Xiph.org video test media [derf's collection]. `https://media.xiph.org/video/derf/`.

[19] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. Loss functions for neural networks for image processing. *arXiv preprint arXiv:1511.08861*, 2015.