Video-Specific Autoencoders for Exploring, Editing and Transmitting Videos

Kevin Wang Deva Ramanan Aayush Bansal
Carnegie Mellon University

Abstract

We study video-specific autoencoders that allow a human user to explore, edit, and efficiently transmit videos. Prior work has independently looked at these problems (and sub-problems) and proposed different formulations. In this work, we train a simple autoencoder (from scratch) on multiple frames of a specific video. We observe: (1) latent codes learned by a video-specific autoencoder capture spatial and temporal properties of that video; and (2) autoencoders can project out-of-sample inputs onto the video-specific manifold. These two properties allow us to explore, edit, and efficiently transmit a video using one learned representation. For e.g., linear operations on latent codes allow users to visualize the contents of a video. Associating latent codes of a video and manifold projection enables users to make desired edits. Interpolating latent codes and manifold projection allows the transmission of sparse low-res frames over a network.

1. Introduction

In this work, we demonstrate that simple video-specific autoencoders learn meaningful representations that enable a multitude of video processing tasks, without being optimized for any specific task. An autoencoder trained using individual frames (without any temporal information) of a specific video via a simple reconstruction loss can learn both spatial and temporal aspects of the video. Simple operations on its latent code, encoder, and decoder enables a wide variety of tasks including video exploration, video editing and video transmission. To the best of our knowledge, we are the first to explore such diverse video processing tasks using a single representation not optimized for any specific task.

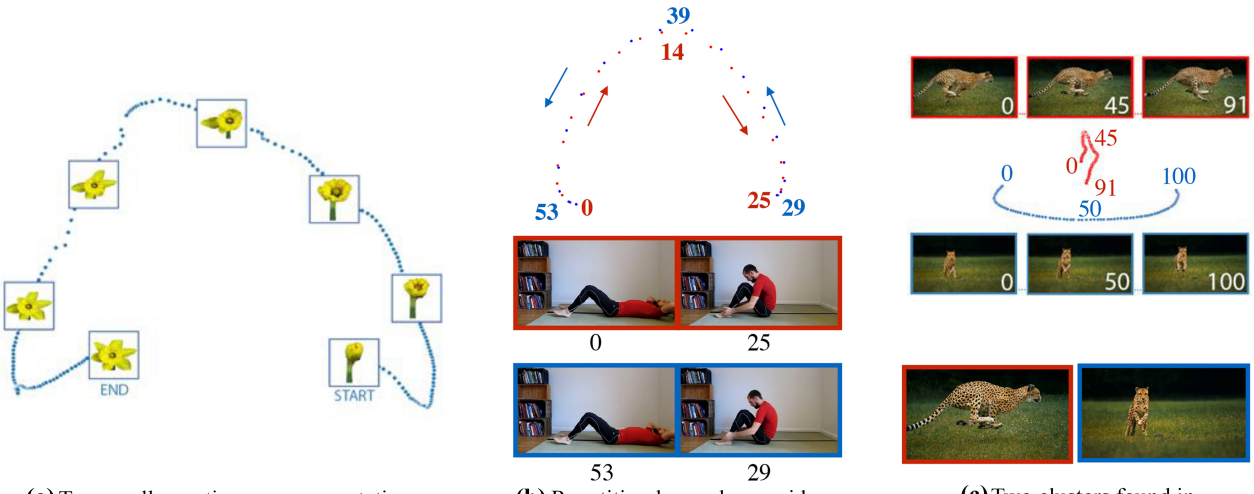
Contributions: (1) We introduce a simple, unsupervised approach for learning video-specific exemplar representations without needing large training data. This representation enables us to do a wide variety of the aforementioned video processing tasks that generally require a dedicated approach. (2) Our approach allows for intuitive user interaction, via a low dimensional visualization of latent codes that allow for video exploration and editing; and (3) finally, we demonstrate that the latent codes and manifold created using a single autoencoder allows for the transmission of sparse, low-resolution frames over a network with the ability to reconstruct the hi-res video using the autoencoder.

2. Related Work

There is a large body of work on specialized video processing tasks such as video completion [16], video enhancement [57], video inpainting [11, 22, 56], video editing [10, 36], temporal super-resolution [25, 34, 62], spatial super-resolution [20, 55], space-time super-resolution [44, 45], removing obstructions [30], varying speed of a video [8] or the humans in it [33], video textures [1, 29, 42], finding unintentional events in a video [14], video prediction [50, 51], generative modeling for associating two videos [6] discriminative modeling for association [13, 40, 52], associating multi-view videos [49], or pixel-level correspondences in a video [4, 58]. In this work, rather than exploring specialized architectures, we learn a single video-specific representation that enables many of these tasks.

Learning from a Single Instance: There is plethora of work that has explored representation learned using a single image for various tasks [2, 17, 35, 47, 48, 54]. Recent approaches [43, 46] have also explored image-specific representation that enables a wide variety of image editing tasks. Often there exists repetitive structure in a signal, such as patch-recurrence in an image [2], that allows one to learn meaningful representations for that signal without any additional information. We extend these observations to videos. Our goal is to learn a representation for a video without any additional information. Because we have more informative data, we could learn a simple autoencoder that optimizes the reconstruction of individual video frames. Prior work on video processing [6, 16, 25] often encodes spatial-temporal information explicitly. In this work, we considered frames from a video as independent images. Despite this, our video-specific autoencoder learns a continuous representation as shown in Figure 1-(a).

Human-Controllable Representation: Usually the front-end of an application is designed around the task of interest. For example, prior work [3, 15, 61] on user-control are limited to a task. In this work, we hope to provide users with a simple representation that they can easily work with to design new



(a) Temporally continuous representation learned on a blooming flower

(b) Repetition learned on a video of a man doing situps

(c) Two clusters found in a video of a running cheetah

Figure 1. **Representations learned without temporal information using autoencoders:** In each figure, a dot represents the latent code of a frame from a video that the autoencoder is trained on. The latent code is reduced from a high dimension to a 2-dimensional visualisation using PCA. **(a)** We find that video-specific autoencoders learn a temporally continuous representation without any explicit temporal information, for example on a video of a blooming flower. **(b)** We observe that latent spaces are able to learn the repetitive motion without using any temporal information, such as the repetitive motion of a situp. **(c)** Finally, distinct modes emerge when the autoencoder encounters different visual concepts in a video. For this cheetah example, we see that the two modes represent two different running poses in the video. We can also show the average image of each mode, as seen below the example.

application without much overhead (such as simple algebraic operations on the latent codes). Our approach also allows a user to see the contents of a video in a glance. Figure 1-(c) shows how a user can explore the contents of a video and see two prominent aspects of this video. Our work also aims at reducing the requirement of application-dependent modules. For e.g., object removal approaches [16] usually require a user to provide extensive object-level mask across the video or use an off-the-shelf segmentation module trained for a particular object. Our learned representation allows us to both: (1) track the object in the video when marked in a single frame by a user and (2) edit the content.

3. Video-Specific Autoencoders

In this section, we review basic properties of autoencoders that we will exploit for various video editing tasks. Recall that an autoencoder [18] compresses the information in a signal via an encoding function. The compressed signal or latent codes are represented using a few bits of information. Given a set of frames from a video $x \in V$, video-specific autoencoders learn to encode each frame into a low-dimensional latent code $f(x)$ that can be decoded (via a function g) back into the high-dimensional input space, so as to minimize the reconstruction error:

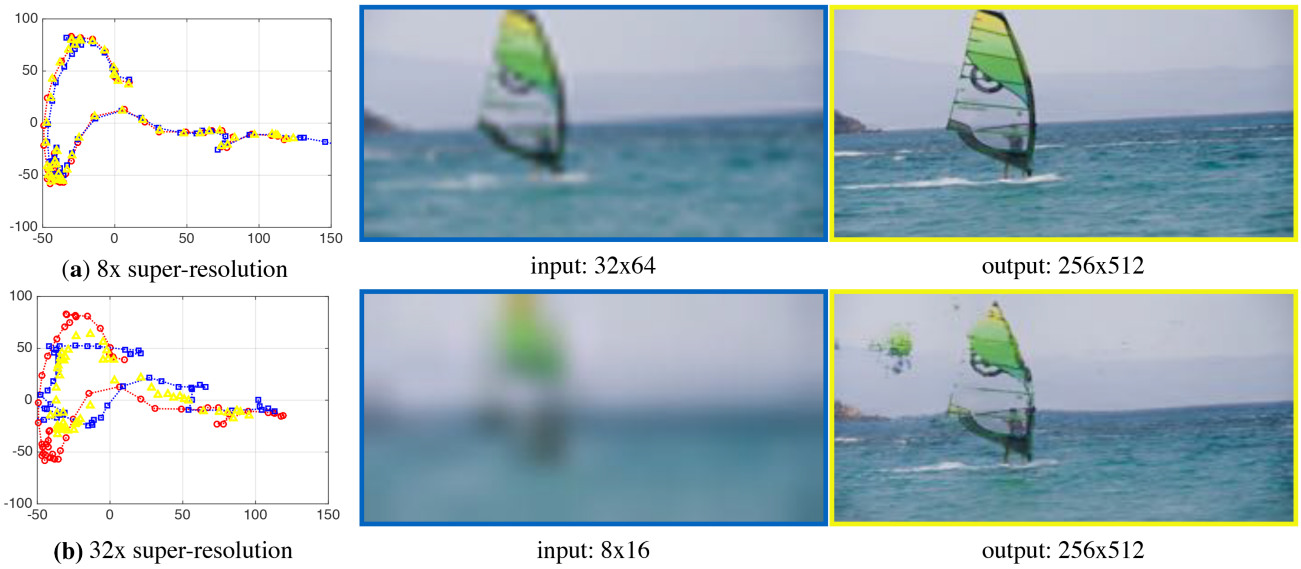
$$\min_{f,g} \sum_{x \in V} ||x - g(f(x))||^2 \quad (1)$$

We use a convolutional feed-forward model that inputs an image and reconstructs it. Importantly, we ensure all operations are convolutional, implying that the size of the latent code scales with the resolution of the video V. To ensure that the latent code contains all the information needed to reconstruct a video frame, we do not use skip connections.

Convolutional Encoder (f): The encoder consists of six 2D convolutional layers. We use 5×5 kernels for first four layers and a stride of 2 that downsamples the input by 0.5 after each convolution. The last two layers have 5×5 kernels without any downsampling. The output of each of these layers is max-pooled in a 2×2 region with a stride of 2. Each conv-layer is followed by batch-normalization [23] and a ReLU activation function [27]. The output of last layer of the encoder is used as a latent representation (or also termed as latent code) in this work.

Convolutional Decoder (g): The decoder inputs the latent code to reconstruct the output. It consists of six up-sampling conv-layers with a 4×4 kernels and a stride of 2 that up-samples the input by 2. Each conv-layer is followed by batch-normalization and a ReLu activation function.

Latent Codes ($f(x)$): Given an input image x with shape h$\times$w, the encoder outputs an encoding with the shape $(k * 12) \times \frac{h}{64} \times \frac{w}{64}$, where k is the number of filters in the first layer of encoder. We later show that such codes can be intuitively visualized with low-dimensional projections (via 2D PCA). For most experiments, we fix $k = 64$. This implies that for



(a) 8x super-resolution

(b) 32x super-resolution

Figure 2. **Manifold Projection with autoencoders**: We train a video-specific autoencoder on 55 high-res 256×512 frames. We visualize the latent codes for the original frames using the red points on the left side. At test time, we input low-res versions of *held-out* frames of varying resolution. We visualize the latent codes for the low-res input using the blue points. We also visualize the latent codes for the output image using the yellow points. We observe that red points, blue points, and yellow points for **(a)** 8X super-resolution. We observe perfect reconstruction for this input. The results, however, degrade as we further reduce the resolution of images, as seen in **(b)** 32X super-resolution.

input: 8x16 **iteration**: 2 **iteration**: 5

Figure 3. **Iterative Improvement via Reprojection Property**: Given a 8×16 image, we iteratively improve the quality of the output. The reprojection property allows us to move towards a good solution with every iteration. At the end of the fifth iteration, we get a sharp hi-res (256×512) output despite beginning with an extremely low-res input. We show that iterative projection can lead to quantitatively better exploration and transmission of videos.

an input frame of size $h = 256$ by $w = 512$, latent codes compress inputs by a factor of 16.

Continuous Temporal Spaces: We study the space of latent codes $f(x)$ to examine the impact of temporal variation of input frames x_t, since the autoencoder is learned without any explicit temporal input. We visualize the latent code space via multidimensional scaling with PCA [9], as seen in examples in Figure 1-(a). An autoencoder trained on a specific video implicitly learns the correlations in the various frames and a continuous temporal space emerges. This property allows us to slow-down or speed-up a video (through latent code resampling). Latent codes can also capture repetitive motion as shown in Figure 1-(b). This property allows us to temporally edit the video.

Video-Specific Manifold (M): We define the *manifold* of an autoencoder to be the set of all possible output reconstructions obtainable with *any* input:

$$M = \{g(f(x)) : \forall x\}. \tag{2}$$

where f, g are the "argmin" encoder and decoder learned from (1). In our setting, M corresponds to be a video-specific manifold of potential image/frame reconstructions. It is well-known that feedforward autoencoders, when properly trained, act as projection operators that project out-of-sample inputs x into the manifold set M [18].

$$||g(f(x)) - x||^2 = \min_{m \in M} ||m - x||^2, \quad \forall x \tag{3}$$

$$= \min_{x'} ||g(f(x')) - x||^2, \quad \forall x. \tag{4}$$

(a) Per-pixel average **(b)** Latent code average **(c)** Latent code average (five iterations)

Figure 4. **Video Averages**: We compare a per-pixel average of video frames **(a)** with the decoded average of the latent codes of individual video frames **(b)**, which is far less blurry. We use the iterative reprojection property to sharpen the average image **(c)**.

One can build intuition for above by appealing to linear autoencoders, which can be learned with PCA. In this case, the above equations point out the well-known fact that PCA projects out-of-sample inputs into the closet point in the linear subspace spanned by the training data [9]. We make use of this property to "denoise" noisy input images $x \notin V$ into manifold M (where noisy inputs can consist of e.g., blurry frames).

Manifold Reprojection: The reprojection property enables the model to map noisy inputs x onto the video-specific manifold M spanned by the training set. We write this as:

$$\text{Project}_0(x) = g(f(x)). \tag{5}$$

Figure 2 shows that 8X-downsampled inputs can be effectively upsampled by reprojecting a blurry bilinearly-upsampled input x onto the video manifold. While results are nearly perfect for 8X upsampling, reprojected outputs contain visual artifacts for more agressive downsampled inputs.

Iterative Reprojection: One can iteratively reproject outputs with an autoencoder:

$$\text{Project}_n(x) = g(f(\text{Project}_{n-1}(x))). \tag{6}$$

For linear autoencoders, one can show that this iteration converges after one step: $\text{Project}_n(x) = \text{Project}_0(x), \forall n$ because a single linear projection ensures the output falls within the linear subspace of the training data [9]. For nonlinear autoenoders, convergence may take longer and is not necessarily guaranteed [18]. Figure 3 shows that one can iteratively improve the quality of results for 32X super-resolution when inputting a 8×16 image. Because it is not guaranteed to converge, we use a fixed $n = 5$ in our experiments unless otherwise noted.

Multi-Video Manifolds: Because our autoencoders are learned with collections of frames, they can easily be trained on frames from multiple videos. We show in our applications that such shared latent representations can be used to compare, cluster, and process collections of videos.

Pixel Codes ($f_i(x)$): Finally, we can use our encoder to extract *pixel*-level representations, similar to past work that extracts such representations from classification networks [5, 19]. Conceptually, one can resize each of the six convolutional layers of the encoder f back to the original image input size, and then extract out the "hyper"-column of features aligned with pixel i. In practice, one can extract features from the image-level encoder $f(x)$ without resizing through judicious bookkeeping. Our final pixel representation, written as $f_i(x)$ is 2176 dimensional.

Training Details: We train a video-specific autoenocder from scratch using the Adam solver [26] with a batch-size of 6. The learning rate is kept constant to 0.0002 for first 100 epochs and then linearly decayed to zero over the next 100 epochs. For larger videos (i.e., more than $3,000$ frames), we reduce the number of epochs to 40.

We now study various applications of autoencoders to explore (Section 4), edit (Section 5), and efficiently transmit videos (Section 6).

4. Exploring Videos

Video Averages: Arguably the simplest way to summarize a video may be an pixelwise average of its frames [61]. Figure 4 compares this with a reconstructed average of latent codes from a DAVIS video [39]. We contrast it with a simple per-pixel mean of the frames, and observe sharper results using our approach. We quantitatively show these improvements in Table 1.

Video clustering: Latent codes can be used to discover different visual modes in a video by clustering (e.g., with k-means). Figure 1-(c) shows that two distinctive modes naturally appear for videos with multiple shots. We also use this property to cluster collections of videos by clustering latent codes learned from multi-video manifolds (i.e., from an autoencoder trained on frames from multiple videos; see Appendix A).

Video exploration allow users to quickly peruse large amounts of video (e.g., consider an analyst who must process large amounts of surveillance video). A user can select a region in the PCA-based 2D visualization that allows for them to quickly visualize summaries by averaging arbitrary

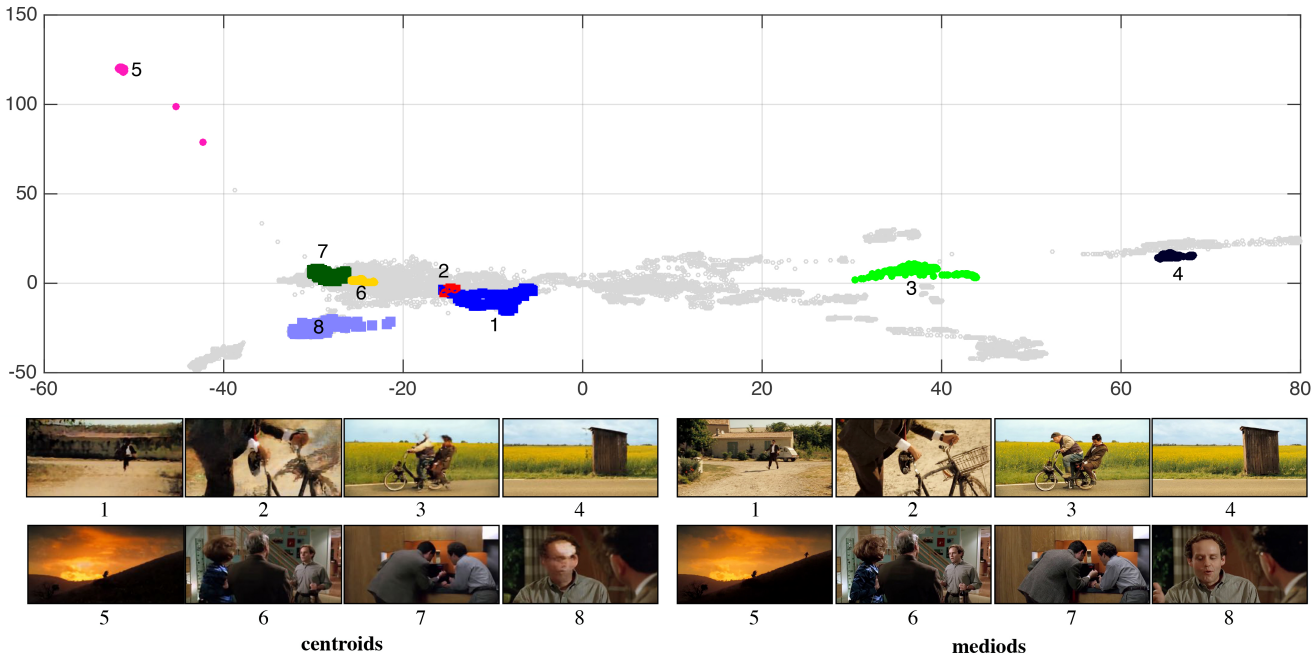


Figure 5. **Video Exploration**: We embed an 11-minute video with a video-specific autoencoder. Our approach allows us to quickly visualize the contents of the video by providing an average summary of the selected region. A user can select the region on the 2D visualization. Our approach generates the centroid of the points in selected region. We also show the mediod corresponding to each centroid. A user can quickly glance by browsing over the points or selecting the region, and can also stitch representative/selected frames in the video.

	FID [21] ↓
DAVIS [39]	
Per-pixel Average	306.01
Latent Code Average	280.15
Latent Code Average + iterative	**205.96**

Table 1. **Video Averages:** We compare approaches for computing an average frame from a video, using 50 videos from the DAVIS dataset. We measure perceptual quality using the FID metric [21], which measures the frechet distance between the average frame and all frames of the corresponding video. We see that a simple pixelwise average is not perceptually faithful to the video, while averaging latent codes is more effective. Iteratively reprojecting the latent reconstruction results in an even more faithful average.

subsets of video frames $V_{\text{subset}} \subseteq V$:

$$g\left(\frac{1}{|V_{\text{subset}}|}\sum_{x \in V_{\text{subset}}} f(x)\right). \tag{7}$$

We give an example of how to explore a video on an 11-minute video in Figure 5.

5. Editing Videos

Video Textures: We can create infinite-loop video textures [42] from a short sequence, as seen in Figure 6. We can find corresponding frames via a cosine similarity on latent codes $f(x_1), f(x_2)$. We emphasize two important distinctions from prior work: (1) PCA visualization enables a user to *interactively* define loops and even paths for the target video; and (2) we can close loops without requiring an exact match by interpolating nearby latent codes, which we see in Figure 7. Table 2 shows the benefits of our interpolating nearby latent codes.

Object Removal and Insertion: The reprojection property of autoencoder also enables us to learn patch-level statistics in a frame. To remove an object, we copy a patch from surroundings to fill the bounding box. Despite discontinuities, the autoencoder generates a continuous spatial image as shown in Figure 8. We can also use the video-specific autoencoder to insert the known content from the video in a frame. The video-specific autoencoder also allows us to stitch spread-out video frames as shown in Figure 9. We naively concatenate the different frames and feed it through the video-specific autoencoder, and the learned model can generate a seamless output. Using the same property, we can also stretch frames and do spatial extrapolation. Examples in Appendix B.

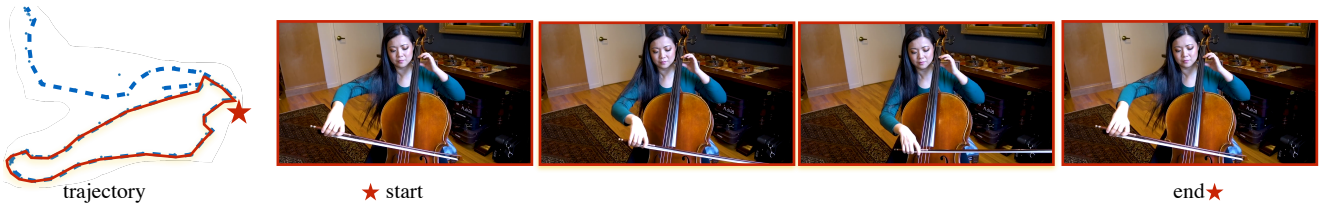


Figure 6. **Video Textures** : We create arbitrary length video (also known as video textures) from an existing short sequence by making continuous loops. Our ability to associate frames within a video using the latent codes enable us to create infinite loops for repetitive motion. We show many examples in the supp material.

Figure 7. **User-Controllable Trajectories**: A user can create trajectories they want for various videos using PCA visualization. Prior approaches [42] find close loops to do video texture. It is not a strict requirement for our work as we are able to interpolate new frames between x_1 and x_2 to smoothly transition between these frames.

	LPIPS [59] ↓
PENN [60]	
Frame Difference	0.136
Texture Difference	**0.100**

Table 2. **Video Textures:** We compare approaches for connecting two similar frames from a video, using 50 videos from the PENN dataset. We evaluate these approaches using LPIPS [59], which measures the perceptual similarity between two images. Given two similar frames x_1 and x_2, prior works would simply connect these frames directly. However, these frames may not necessarily have a smooth transition, while we can create arbitrarily long, smooth video textures by continuously connecting frames with interpolation. For every video, we take two frames x_1 and x_2 and calculate the LPIPS between them. Our method interpolates a new frame between these two frames, and we calculate the LPIPS between this new frame and x_2. **A lower score is better**.

Pixel Correspondences and Mask Propagation: We use our pixel codes to establish a pixel-wise correspondences in the adjacent video frames via a cosine similarity measure. The pixel correspondences allow us to propagate the instance labels from one frame to the rest of video. The mask propagation is especially crucial from user perspective as: (1) it is a tedious task to label entire video; and (2) it helps us to use one less segmentation module in our system. Crucially, we do not need perfect labels for spatial editing as the video-specific autoencoder corrects the imperfections due to its ability to generate continuous spatial outputs. We show an example of mask propagation in Figure 10, and how we can use masks to edit videos.

	Task	**PSNR**↑	**SSIM**↑
PENN [60]			
SuperSlowMo [25]	✓	31.060 ± 2.324	0.951 ± 0.018
Ours	✗	33.643 ± 4.315	0.969 ± 0.020
Random Web Videos			
SuperSlowMo [25]	✓	34.156 ± 1.853	0.721 ± 0.100
Ours	✗	34.274 ± 1.935	0.723 ± 0.100

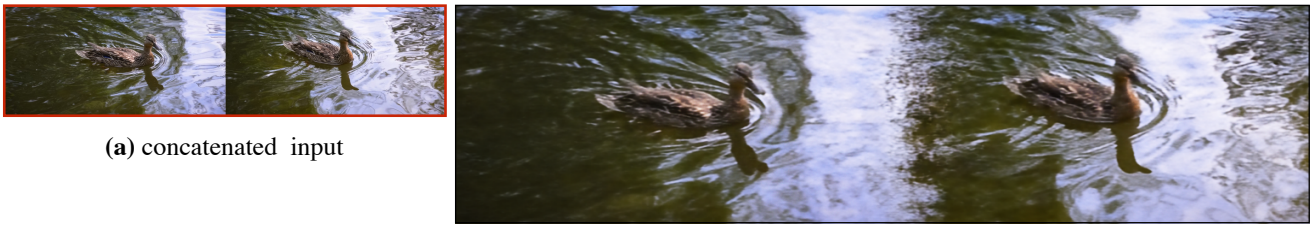
Table 3. **Temporal Super-Resolution:** We contrast our approach with an off-the-shelf SuperSlowMo [25] model. We use 150 videos from Penn Action dataset. We show interpolation between every other frames. The original frames (more than $10,000$ frames in total) are used as a ground-truth for evaluation and not used for training the video-specific autoencoders. We compute PSNR and SSIM scores between the original frames and interpolated frames (**Higher is Better**). Our approach achieves competitive performance. We also compare our approach with SuperSloMo using random web videos in the same manner.

6. Transmitting Videos

Finally, we examine applications of autoencoders motivated by real-time, low bit-rate video transmission (e.g., video conferencing). We envision a setting where one can transmit network parameters off-line along with online transmission of aggressively-subsampled frames, both temporally and spatially. Such samples can then be decoded at the

(a) Original Image **(b)** Replacing player with patches **(c)** Output

Figure 8. **Spatial Editing:** Given an image, a user can edit it by copy-pasting a patch from the surroundings to the target location and feed it to the video-specific autoencoder. We show an example of spatial editing on a frame in the tennis video from the **DAVIS** [39] dataset. We train an autoencoder on the tennis video. Given a frame from the video, we are able to remove the tennis player from the image by replacing the player with surrounding patches in the image and passing it through the autoencoder.



(a) concatenated input

(b) seamless output

Figure 9. **Seamlessly Stitching Frames:** We naively concatenate the spread-out frames in a video and feed it through the video-specific autoencoder. The learned model generates a seamless output and can capture reflection and ripples in water.

receiver to produce hi-res, hi-frame rate video.

Transmitting Sparse Temporal Frames: We use simple operation on latent-codes to slow-down (known as temporal super-resolution) and speed-up a video. Given a frame x_t and x_{t+1}, we can insert an arbitrary number of frames between by linearly interpolating their latent codes:

$$g\Big(\alpha f(x_t) + (1-\alpha) f(x_{t+1})\Big), \quad \alpha \in [0,1] \qquad (8)$$

We quantitatively compare our approach with Super Slowmo [25][1] in Table 3. We use an off-the-shelf SuperSlowMo [25] model trained on a large dataset in a supervised manner specifically for the task of temporal super-resolution. We use 150 videos (10 videos from each action) from PennAction dataset [60]. We do 2X interpolation, i.e., interpolate between every other frame. The original frames that are used as a ground-truth for evaluation and are not used for training the video-specific autoencoder. We compute PSNR and SSIM scores between the original frames and interpolated frames. **Higher is Better**. Table 3 shows the performance of our approach with SuperSlowMo [25]. We also contrast the performance on random web videos in Table 3 and observe competitive performance to SuperSlowMo.

We also study our approach using 90 videos of DAVIS dataset [39] in Table 4. Sequences in DAVIS dataset consists of $50-80$ frames (roughly sampled at 5fps) with substantial dynamics. Training a video-specific autoencoder with alternate frames is challenging as it becomes even more sparse. Since we do not use temporal information and optimize for temporal super-res, we contrast the performance of our model when trained with all frames (ALL) and alternate frames (ALT). We observe that performance of a video-specific autoencoder degrades when trained on extremely sparse frames from a video with large camera and object motion. We, however, observe that performance can be improved by using iterative reprojection. This means we can sparsely transmit frames over the network and do temporal super-res with iterative reprojection property to get dense outputs. Finally, video-specific autoencoders enable arbitrary temporal resampling of a video, in contrast to previous methods that are often trained for a fixed resampling factor without being able to generalize to others [31, 32, 37]. Our approach can also benefit from advances in temporal super-resolution approaches [28, 38] using optical flow. However, in this work we wanted to limit the use of an application-dependent module.

Transmitting Low-Res Frames: Our goal is to transmit minimal bits over network and yet be able to get an original quality original video output at the reception. Spatial super-resolution is enabled by the reprojection property of the autoencoder (discussed in Section 3). The convolutional autoencoder allows us to use videos of varying resolution.

[1] We use the publicly available model from https://github.com/avinashpaliwal/Super-SloMo.

first frame (labeled) middle frame last frame

Figure 10. **Instance Mask Propagation:** Given the first labeled frame, we propagate the instance labels using the pixel codes. We show here the labels in the first frame on the left. We show the propagated labels from the first frame to the middle and last frame of the video. Using these propagated masks, and the method demonstrated in Fig. 8, we can then perform object removal on the video.

	Task	PSNR↑	SSIM↑
DAVIS [39]			
Ours (ALT)	✗	22.065 ± 5.205	0.728 ± 0.171
Ours (ALL)	✗	24.272 ± 6.092	0.802 ± 0.154
Ours (ALL+iterative)	✗	28.938 ± 7.356	0.883 ± 0.151
SuperSlowMo [25]	✓	27.077 ± 5.950	0.871 ± 0.134

Table 4. **Studying Influence of Sparse Samples on Video-Specific Autoencoders via Temporal Super-Resolution:** We study our approach with sparse frames from 90 videos of DAVIS dataset [39]. We show interpolation between every other frames. The original frames are used as a ground-truth for evaluation. We consider two scenarios: (1) using alternate (**ALT**) frames from a video for training; and (2) using **ALL** frames from a video for training the video-specific autoencoders (since we do not use temporal information and optimize for this task). We compute PSNR and SSIM scores between the original frames and interpolated frames (**Higher is Better**). We observe that performance of video-specific autoencoders degrade when making samples extremely sparse, especially for the video with large dynamics. We, however, observe that performance can be improved by using iterative reprojection property. This means we can sparsely transmit frames over the network and do temporal super-res with iterative reprojection property to get dense outputs at the reception. For reference, we also provide the performance of an off-the-shelf SuperSlowMo [25] model.

We get temporally smooth outputs without using any temporal information for all the experiments. We quantitatively compare our approach with an off-the-shelf ESR-GAN [53] model for 4X, 16X, and 64X super-resolution on 90 videos (roughly $6,000$ frames) from DAVIS dataset [39]. ESR-GAN is trained for 4X super-resolution. We use it twice for 16X super-resolution, and thrice for 64X super-resolution. Table 5 contrast our approach with ESR-GAN for these three settings. Our approach achieves better performance without being optimized for the task. We show examples in Appendix C.

	Task	PSNR↑	SSIM↑
4X Super-Res			
ESR-GAN [53]	✓	26.886 ± 3.821	0.847 ± 0.087
Ours	✗	31.493 ± 3.681	0.940 ± 0.053
16X Super-Res			
ESR-GAN [53]	✓	19.324 ± 3.163	0.605 ± 0.162
Ours	✗	25.737 ± 3.511	0.856 ± 0.086
64X Super-Res			
ESR-GAN [53]	✓	15.441 ± 2.752	0.490 ± 0.170
Ours	✗	17.862 ± 4.789	0.622 ± 0.196

Table 5. **4X, 16X, and 64X Spatial Super-Resolution:** We contrast our approach with an off-the-shelf ESR-GAN model [53] trained for 4X super-resolution. For 16X super-resolution, we use the ESR-GAN model twice. For 64X super-resolution, we use it three times iteratively. We use 90 videos from DAVIS dataset [39] for this evaluation. The original frames (256×512) are used as a ground-truth for evaluation. We compute PSNR and SSIM scores between the original frames and the outputs from two approaches (**Higher is Better**). Our approach achieves better performance without being optimized for the task.

7. Discussion

An autoencoder trained using individual frames of a specific video without any temporal information can be used for

a wide variety of tasks in video analytics without even optimizing for any of those tasks. This feat could be possible by careful analysis of spatial and temporal properties of a video-specific autoencoder. We hope that an interface based on this simple representation can enable users to easily design new applications (not even explored in this work) without much overhead such as simple algebraic operations on the latent codes. We also hope that every video uploaded on web get its autoencoder. It may not take a long time to train a model but it can drastically reduce the amount of resources required for video transmission, exploration, and processing because once trained, the model can be used for fairly large number of applications without much constraints.

8. Limitations and Societal Impact

400 hours of video data is uploaded to YouTube every minute. The rich video data opens up enormous opportunities for exploring the vast visual content. Creating ways that allows a user to explore the contents of a video, edit them, and efficiently transmit the content would provide a comprehensive platform to the users. We take a small step towards this grand goal. Our work requires training a video-specific autoencoder. Training a new model at test time is not instantaneous and it depends on the length of the videos. We need to come up with faster ways to train a new model and use insights from transfer learning such as a simple fine-tuning. In this work, we also observe that training a reliable model is challenging if the frames of a video are temporally sparse. Such a situation is observed when the cameras are fast moving like a hand-held camera. Our current evaluation is conducted using standard video benchmarks and regular web videos that are properly curated. We plan to collect a wide variety of videos from hand-held devices for better analysis. Finally, tools for editing videos can be misused for spreading misinformation. While videos from any source are not considered as an evidence in the court of law, they are still important in forming public opinion on social media like Twitter, Facebook, YouTube. Every time a viewer see a generated video on these platforms, there needs to be a banner displaying "This is a generated content." so that they do not make any opinions out of it.

A. Exploring a Video

Video-specific autoencoders work well on exemplar data distribution. One may wonder their applicability for long and diverse videos. We observe that latent codes of a trained autoencoder can also separate different visual concepts. This allows us to cluster a long video into multiple short videos. In this section, we study this property using the task of video classification for 90 videos in DAVIS dataset [39], 862 videos of JHMDB dataset [24], and 2326 videos in Penn dataset [60]. We learn an autoencoder using all the frames from each of these datasets. Once trained, we do k-means clustering (where k is the number of videos in each dataset) using the latent codes obtained from the trained autoencoder. Figure 11-(a)-(c) demonstrates our ability to separate different visual concepts. Here, we also contrast with *fc-7* features of an AlexNet model [27] trained on the labeled ImageNet dataset [41]. We observe competitive results. We also study the ability of a *randomly-initialized* autoencoder. We use the latent codes from an *untrained* autoencoder. We observe that even the randomly initialized autoencoder can reliably separate different visual concepts. Finally, we observe that both JHMDB and Penn-Action have visually similar videos but with different ids. We, therefore, computed the performance considering the "action" class of each cluster instead of their video-id. The respective plots are shown in Figure 11-(d)-(e).

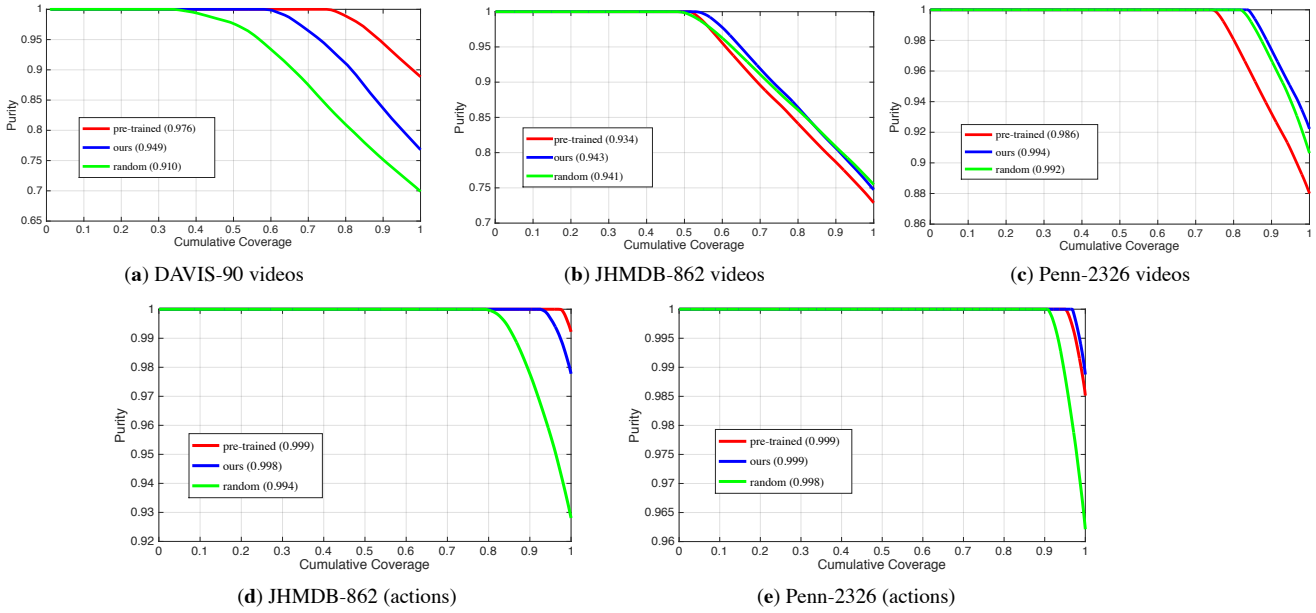
We visualize various visual clusters (with purity of 1) from DAVIS dataset in Figure 12. We show average images for these clusters using the single autoencoder trained on all frames. We can further sharpen these average images by fine-tuning the autoencoder only on the examples in the cluster for 5 iterations. We also visualize various impure clusters in Figure 13 along with 2D visualization of latent codes for the examples in each cluster. We observe that different concepts within an impure cluster can further be separated. We also visualize the clusters of Penn-Action dataset in Figure 14. We observe that both JHMDB and Penn-Action dataset consists of different videos that look similar because of different reasons like same person and background but different orientation, different camera location etc. We visualize these interesting clusters in Figure 15. Finally, we visualize impure clusters in Figure 16. Most of these clusters belong to same action or have a similar background.

With these analysis, we posit that one may also use a simple autoencoder to first cluster the different videos and then use a video-specific autoencoder for each cluster, or fine-tune the model for the cluster. These analysis also shows the potential of autoencoders for average image exploration and unsupervised learning of exemplar visual concepts. We, however, leave them for the future work.

B. Editing a Video

Stitching and Stretching Video Frames: The ability of the video-specific autoencoder to generate continuous spatial imaging allows us to stitch spread-out video frames. Figure 17 shows various examples where we stitch random frames from arbitrary videos with varying texture and content. We naively concatenate the different frames and feed it through the video-specific autoencoder. The learned model generates a seamless output. For e.g., the reflections in water, and circular patterns formed by motorbike. More examples in Figure 18.

The iterative reprojection property of autoencoder and its



(**a**) DAVIS-90 videos (**b**) JHMDB-862 videos (**c**) Penn-2326 videos

(**d**) JHMDB-862 (actions) (**e**) Penn-2326 (actions)

Figure 11. **Visual concept discovery** on three datasets: **(a)** DAVIS-90 videos [39]; **(b)** JHMDB-862 videos [24]; and **(c)** Penn-Action 2326 videos [60]. We naively concatenate all frames from each dataset and study various methods for recovering the original videos via clustering with k-means (where K is the number of videos in each dataset), measuring purity vs. cumulative coverage. We compare AlexNet [27] *fc-7* features pre-trained on ImageNet [41], a randomly initialized autoencoder (i.e., no training), and trained autoencoder. We observe that even random weights of the autoencoder (without any training) can reliably separate visual concepts. Finally, we observe that both JHMDB and Penn-Action have visually similar videos but with different ids. We, therefore, computed the performance considering the "action" class of each cluster instead of their video-id. The respective plots are shown in **(d)** and **(e)**. We also show **area-under-curve** for each method in the legends of each plot.

ability to generate continuous spatial imaging allows us to arbitrarily stretch a video frame. We show various examples of stretching a 256×512 video frame to 256×2048 in Figure 19.

Spatial Extrapolation: Our approach allows us to do a spatial extrapolation as shown in Figure 20. Given a 256×512 image, we spatially extrapolate on its edges to create a 512×1024 image. To do this, we mirror-pad the image to the target size and feed it to the video-specific autoencoder. We use the iterative reprojection property of the autoencoder. After a few iterations, the video-specific autoencoder generates continuous spatial outputs. We also show the results when zero padding the input image. We observe that mirror-padded input preserves the spatial structural of the central part, i.e., the actual input, whereas zero-padded input leads to a different spatial structure. Instead of mirror-padding, one may also place known content from the videos and yet be able to get a valid output.

User-Controlled Editing: The ability of video-specific autoencoder to generate continuous outputs from a noisy input allows avenues for user-controlled editing. We show this aspect in Figure 21 on stretched-out video frames as it allows us to make multiple edits. Given an image, a user can do the editing by copy-pasting a patch from surroundings to the target location and feed it to the video-specific autoencoder. We can naively replace a forgeround object with a background patch, and the video-specific autoencoder will automatically correct the imperfections. We get consistent results due to the reprojection property of the autoencoder and its ability to generate continuous spatial image.

Insertion: We can also use the video-specific autoencoder to insert the known content from the video in a frame as shown in Figure 22. A user inserts patches from the far-apart frames and feed it to the video-specific autoencoder. We use the iterative reprojection property here. The autoencoder generates a continuous and seamless spatial output. We, however, observe that there are no guarantees if the video-specific autoencoder (trained on a single scale) will preserve the input edits all the times when foreground/moving objects are inserted. It is very likely that it may generate a completely different output at that location. In the bottom row of Figure 22, we inserted a number of small fish. The video-specific autoencoder, however, chose to generate different outputs. This behaviour would require more analysis of the reprojection property and we leave it to the future work.

Pixel Correspondences: Finally, we show the results of

Figure 12. **Leaning a single autoencoder on all the frames of DAVIS dataset:** We show different visual clusters (with purity of 1) when doing k-means on the latent codes for all the frames of DAVIS dataset. We show the average image for each of these clusters using the single model. We can further sharpen the average image by fine-tuning the model for a few iterations on the examples of specific cluster.

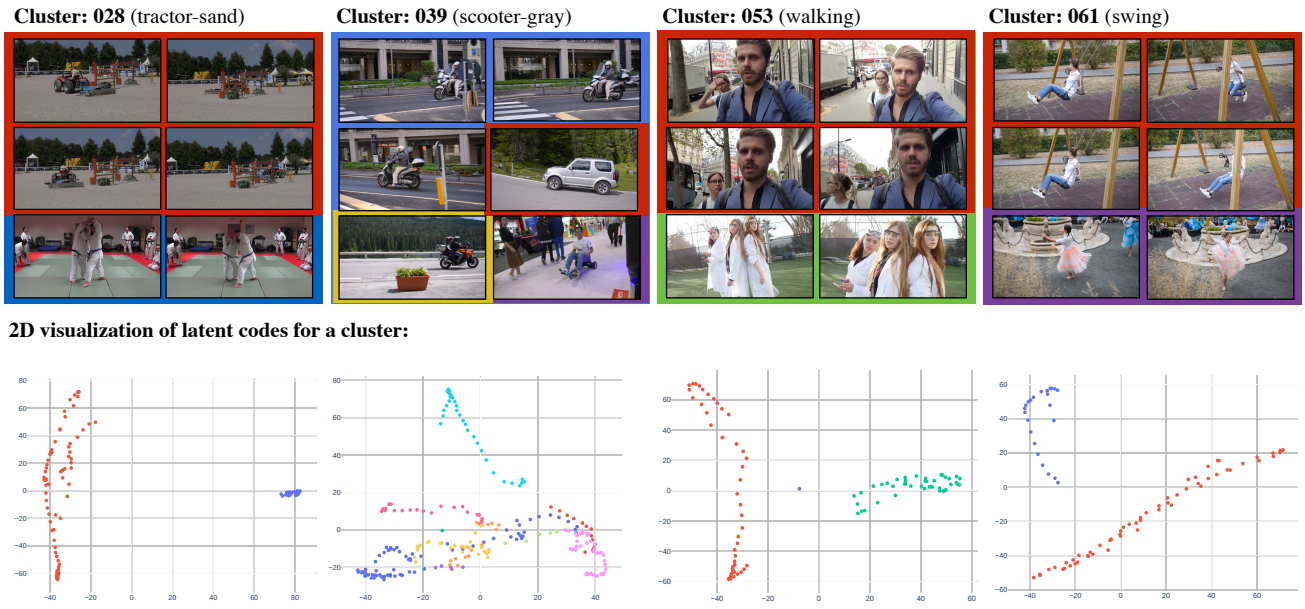


Figure 13. **Impure clusters of DAVIS dataset can be further separated**: We show different clusters with lower purity value. On a closer observation, we find that concepts within most of these clusters can further separate out as shown by the 2D visualization of latent codes.

Figure 14. **Pure clusters from the Penn Action dataset** We visualize different clusters (purity of 1) obtained by simple k-means clustering on the latent codes via an autoenoder trained on all the frames on Penn Dataset.

Figure 15. **Interesting clusters from the Penn Action dataset** We show different visual concepts that come from different videos but looks visually similar, i.e., these are the clusters with video purity of less than 1 but action purity of 1. This happens because we have various videos in Penn-Action dataset that has same person and background with slight variation. We show these variations with each cluster.

pixel correspondences across adjacent frames in Figure 23. We show examples of instance mask propagation in Figure 24 and Figure 25.

C. Efficiently Transmitting a Video

We show examples of 10X super-resolution in Figure 26-27 using this property for hi-res videos. The convolutional autoencoder allows us to use videos of varying resolution. Crucially, we get temporally smooth outputs without using any temporal information for all the experiments.

Robust Space for Low-Res Inputs: One method to deal with extremely low-res inputs is to use the reprojection property iteratively. We also observe that one can *robustify* the space of video-specific autoencoder by varying the resolution of input images at training time (the resolution of output does not change). This added noise makes the model robust to low-res noise and allows us to get hi-res outputs for a extremely low-res input without iterative reprojection. Figure 28 shows an example of 32X super-resolution in one iteration. We show example where we input 32×32 Barack Obama frames and yet be able to get 1024×1024 hi-res output in a single iteration. We have not use this aspect in any other example/evaluation shown in this paper. We leave it to the future work for more exploration of this property.

Summary: We summarize different operations that we can perform on a video using a single representation in Figure 29. Finally, we show different application on various videos in Figure 30.

D. More Discussion on Properties

Exploring the Manifold: Video-specific manifold allows us to move in the 2D space and visualize the characterstics of a video. We explore the video-specific manifolds in Figure 31 via two video instance specific autoencoders on: **(I)** 82 individual frames from a bear sequence; and **(II)** 55 individual frames from a surfing event. The latent codes of

Figure 16. **Impure clusters from the Penn Action dataset:** We now show different clusters with lower purity value. Most of them either capture the same action or are captured in similar background.

original points on a 2D plot for two sequences are visualized using the bold squares. Each square represents an original frame in the video and is shown using a different color. The red line connecting the squares show the temporal sequence. We show **original** images and **reconstructed** images for four points: **(a)**, **(b)**, **(c)**, and **(d)**. We observe sharp reconstruction. We then show various points on this manifold. The color represents the closest original frame. We also show ten random points around original points in the latent space. We do not see artifacts for the bear sequence as we move away from the original points due to highly correlated frames. We see minor artifacts in the surfing event as we move away from the original points as the frames are sparse and spread-out.

Perturbation via Unknown Data-Distribution: Prior work on audio conversion via exemplar autoencoders [12] showed that one can input an unknown voice sample from a different person (other than training sample) as input and yet be able to get a consistent output. We study if this property holds for the video-specific autoencoders. This property could potentially allow us to establish correspondences across the frames of two videos and do video retargeting [6]. We study this behaviour via three controlled experiments: (1) **multi-view videos**: training a video-specific autoencoder on one stationary camera from a multi-view sequence [7], and test it on other cameras. The multi-view sequences via stationary cameras allow us to study the role of slight perturbation. Figure 32 shows the analysis of perturbing data distribution via multi-views. We observe that the points are farther away from the original points (sequence used for training) as we move away from them. This means we cannot naively use a video-specific autoencoder for the inputs that largely vary from original points; (2) **semantically similar videos**: training a video-specific autoencoder on one baseball game and test it on other baseball games [60]. The game videos allow us to study the role of semantic perturbation. Figure 33 shows the reprojection of various semanitically similar events. We observe that the points move farther as the input becomes less similar. We also observe that we can iteratively bring the points close to the original points by iterative reprojection. After a few iterations, we observe that two semantically similar events align with each other as shown in Figure 34. We also get temporally coherent outputs showing alignment between two videos. This property allows us to establish correspondences between two semantically similar videos and do video retargeting; and (3) finally

Figure 17. **Stitching Far-Apart Frames in a Video:** We naively concatenate the spread-out frames in a video and feed it through the video-specific autoencoder. The learned model generates a seamless output. The top row in each example shows concatenated frames. The bottom row shows seamless output of video-specific autoencoders.

using **completely different videos** from DAVIS dataset [39] (for e.g., a model trained on bear sequence and tested it with a surfing event). The completely different videos allow us to see if the autoencoder can learn a reasonable pattern between two videos (for e.g., movement of objects in a similar direction) or leads to indecipherable random projections. We show two examples in Figure 35. In the first example, we train a video-specific autoencoder using a cow sequence. We input the frames of a surfing event to this trained model. The first iteration yields a noisy outputs and far-away from the

Figure 18. **More examples of stitching far-apart frames in a video:** We show more examples of stitching for different videos and observe continuous seamless outputs.

original points in the video-specific manifold. We reproject the input iteratively multiple times, thereby bringing it close to the original points. We show the results of 51^{st} iteration. We also show a few examples showing the mapping from the frames of surfing event and the corresponding reconstructed frames. The outputs are noisy and does not have a temporal coherence. We observe similar behavior in other example.

Negative Influence of Data Augmentation: We observe an interesting phenomenon in the surfing example shown in Figure 38. The direction of final surfing output is horizontally flipped. We realize that we train the video-specific autoencoder using random horizontal flip that is a standard data augmentation strategy for training deep neural networks. However, when training a model using random horizontal flips creates distinct video-specific manifolds for both original and flipped samples as shown in the top-row of Figure 39. We observe that a model trained with data augmentation confuses a noisy input as to which direction it should move to obtain a hi-res output (shown by the direction of two points in the plot). However, we are able to overcome this issue when a model is trained without random horizontal flips. We also see sharper results when using the model trained without horizontal flips. We observe that autoencoder maps noisy input to the manifold spanned by flipped samples. This is a reason why we see output similar to flipped samples. Additionally, the output of the model trained with horizontal flips suffer averaging artifacts whereas we get good sharp results when not using it. This behaviour specifically holds when the input is quite noisy (e.g. 4×8 resolution).

References

[1] Aseem Agarwala, Ke Colin Zheng, Chris Pal, Maneesh Agrawala, Michael Cohen, Brian Curless, David Salesin, and Richard Szeliski. Panoramic video textures. *ACM Trans. Graph.*, 2005. 1

[2] Yuval Bahat and Michal Irani. Blind dehazing using internal patch recurrence. In *IEEE ICCP*, 2016. 1

[3] Yuval Bahat and Tomer Michaeli. Explorable super resolution. In *CVPR*, 2020. 1

[4] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *IJCV*, 2011. 1

[5] Aayush Bansal, Xinlei Chen, Bryan Russell, Abhinav Gupta, and Deva Ramanan. PixelNet: Representation of the pixels, by the pixels, and for the pixels. *arXiv:1702.06506*, 2017. 4

[6] Aayush Bansal, Shugao Ma, Deva Ramanan, and Yaser Sheikh. Recycle-gan: Unsupervised video retargeting. In *ECCV*, 2018. 1, 14

[7] Aayush Bansal, Minh Vo, Yaser Sheikh, Deva Ramanan, and Srinivasa Narasimhan. 4d visualization of dynamic events from unconstrained multi-view videos. In *CVPR*, 2020. 14

[8] Sagie Benaim, Ariel Ephrat, Oran Lang, Inbar Mosseri, William T Freeman, Michael Rubinstein, Michal Irani, and Tali Dekel. Speednet: Learning the speediness in videos. In *CVPR*, 2020. 1

[9] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006. 3, 4

[10] Nicolas Bonneel, Kalyan Sunkavalli, James Tompkin, Deqing Sun, Sylvain Paris, and Hanspeter Pfister. Interactive intrinsic video editing. *ACM Trans. Graph.*, 2014. 1

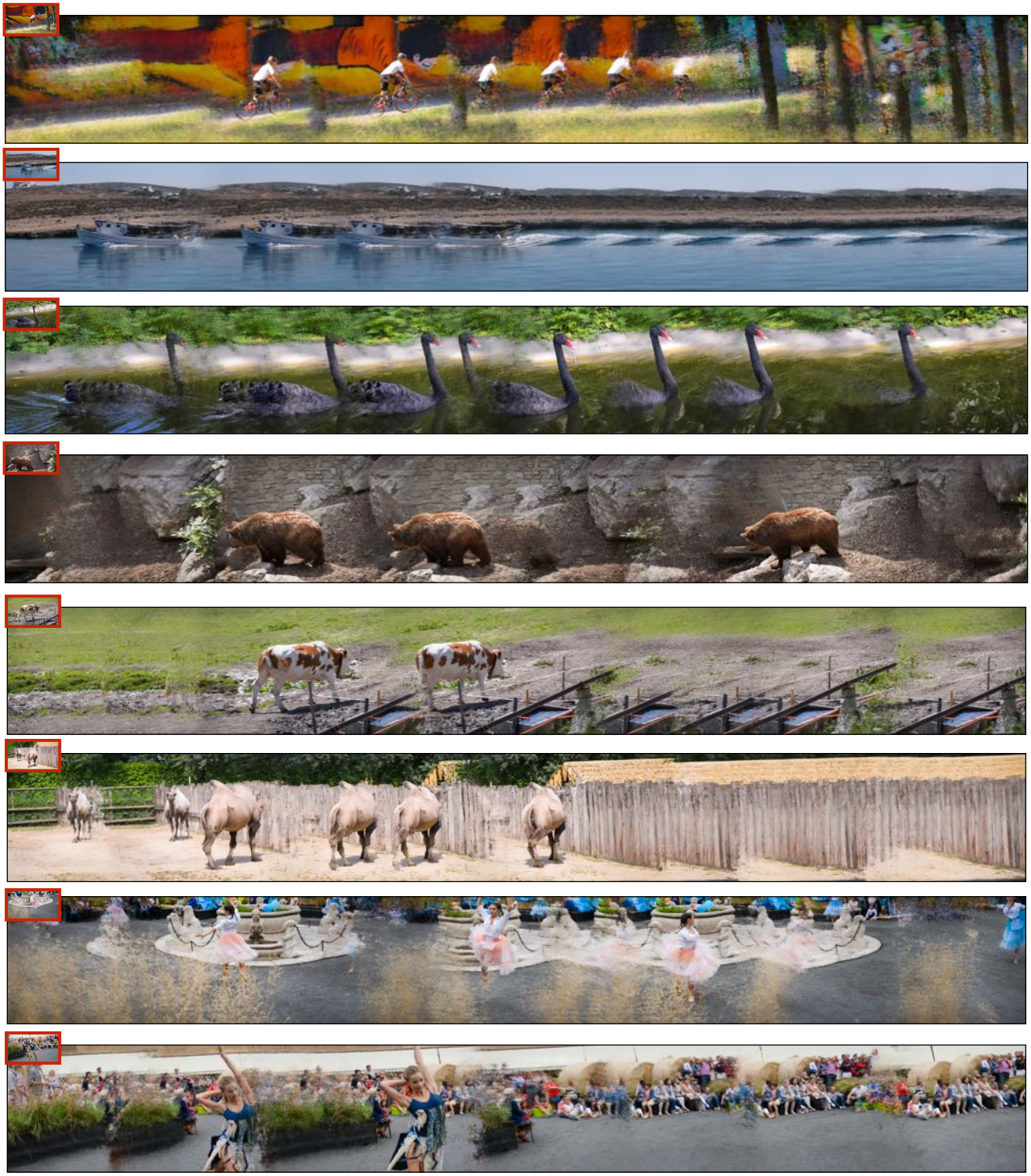


Figure 19. **Stretching a Frame:** Given a 256×512 frame of a video (shown in the red box on the left), we stretch it horizontally to 256×2048 and feed it to a learned video-specific autoencoder. We iterate 30 times and get a consistent panorama-type output. We can further use our editing tools to create desirable panoramas as shown in Figure 21.

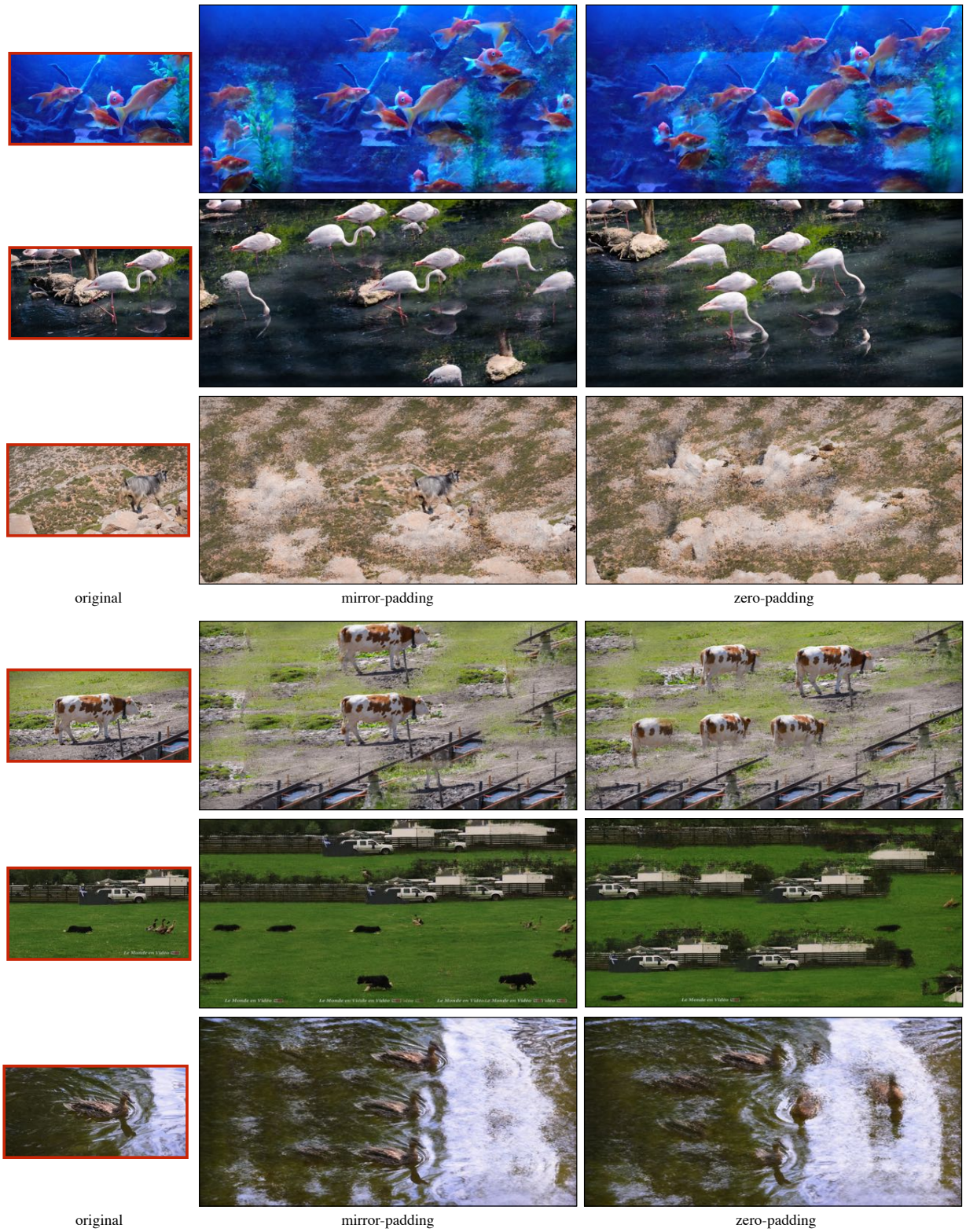


Figure 20. **Spatial Extrapolation:** Given a 256×512 image, we spatially extrapolate on its edges to create a 512×1024 image. To do this, we **mirror-pad** the image to the target size and feed it to the video-specific autoencoder. We also show the results of **zero-padding** here. While mirror-padded input preserves the spatial structure of the central part, zero-padded input leads to a different spatial structure.

Figure 21. **Spatial Editing (Removal):** Given a image, a user can do editing by copy-pasting a patch from surroundings to the target location and feed it to the video-specific autoencoder. We show examples (input-output pairs) of spatial editing for stretched out images. The video-specific autoencoder yields a continuous and consistent spatial outputs as shown in the various examples here.

[11] Ya-Liang Chang, Zhe Yu Liu, Kuan-Ying Lee, and Winston Hsu. Learnable gated temporal shift module for deep video inpainting. In *BMVC*, 2019. 1

[12] Kangle Deng, Aayush Bansal, and Deva Ramanan. Unsupervised audiovisual synthesis via exemplar autoencoders. In *ICLR*, 2021. 14

[13] Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. Temporal cycle-consistency learning. In *CVPR*, 2019. 1

[14] Dave Epstein, Boyuan Chen, and Carl Vondrick. Oops! predicting unintentional action in video. In *CVPR*, 2020. 1

[15] Ohad Fried, Ayush Tewari, Michael Zollhöfer, Adam Finkelstein, Eli Shechtman, Dan B Goldman, Kyle Genova, Zeyu Jin, Christian Theobalt, and Maneesh Agrawala. Text-based editing of talking-head video. *ACM Trans. Graph.* 1

[16] Chen Gao, Ayush Saraf, Jia-Bin Huang, and Johannes Kopf. Flow-edge guided video completion. In *ECCV*, 2020. 1, 2

[17] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *IEEE ICCV*, 2009. 1

[18] Ian Goodfellow, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT Press Cambridge, 2016. 2, 3, 4

[19] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015. 4

[20] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. Recurrent back-projection network for video super-resolution. In *CVPR*, 2019. 1

[21] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fer-

Figure 22. **Spatial Editing (Insertion):** We show more examples (input-output pairs) of spatial editing for stretched out images. Here a user inserts patches from far-apart frames and feed it to the video-specific autoencoder. The autoencoder generates a continuous spatial output. However, there are no guarantees if the video-specific autoencoder will preserve the input edits. It may generate a completely different output at that location. For e.g., we placed small fishes in the bottom example but the video-specific autoencoder chose to generate different outputs.

gus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 5

[22] Jia-Bin Huang, Sing Bing Kang, Narendra Ahuja, and Johannes Kopf. Temporally coherent completion of dynamic video. *ACM Trans. Graph.*, 2016. 1

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 2

[24] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black. Towards understanding action recognition. In *ICCV*, 2013. 9, 10

[25] 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. In *CVPR*, 2018. 1, 6, 7, 8

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *NerIPS*, 2012. 2, 9, 10

[28] Hyeongmin Lee, Taeoh Kim, Tae-young Chung, Daehyun Pak, Yuseok Ban, and Sangyoun Lee. Adacof: Adaptive collaboration of flows for video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 7

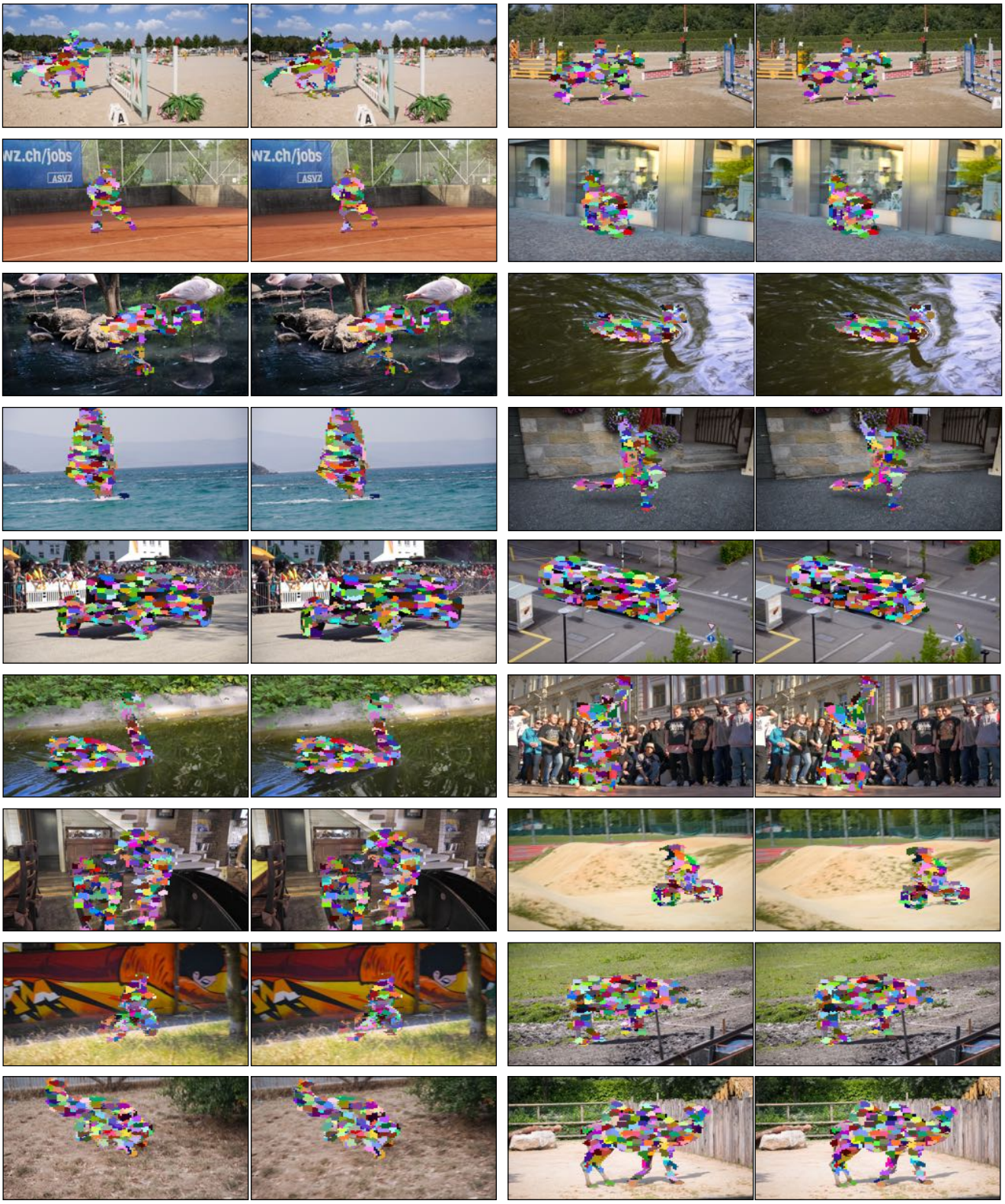
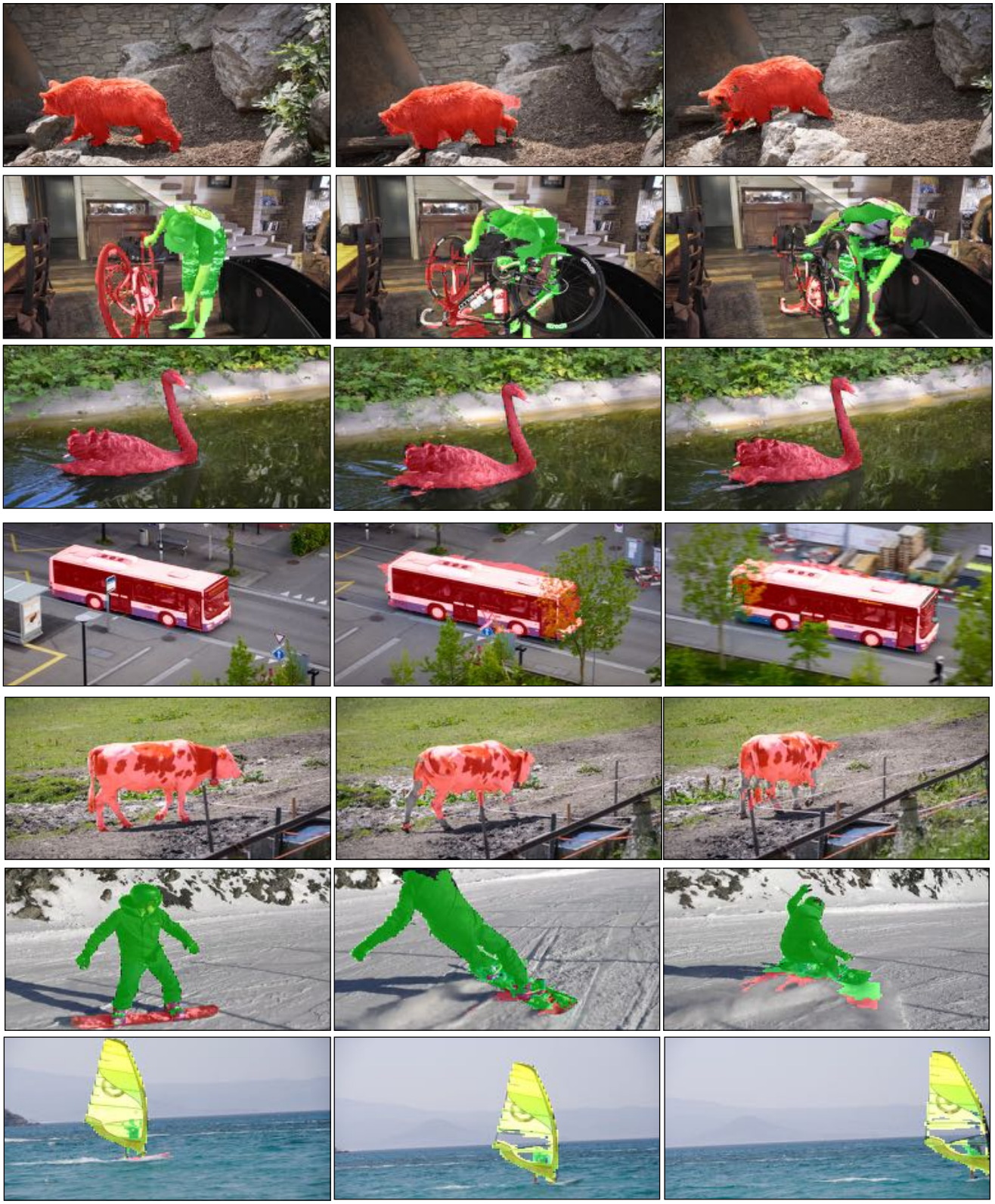
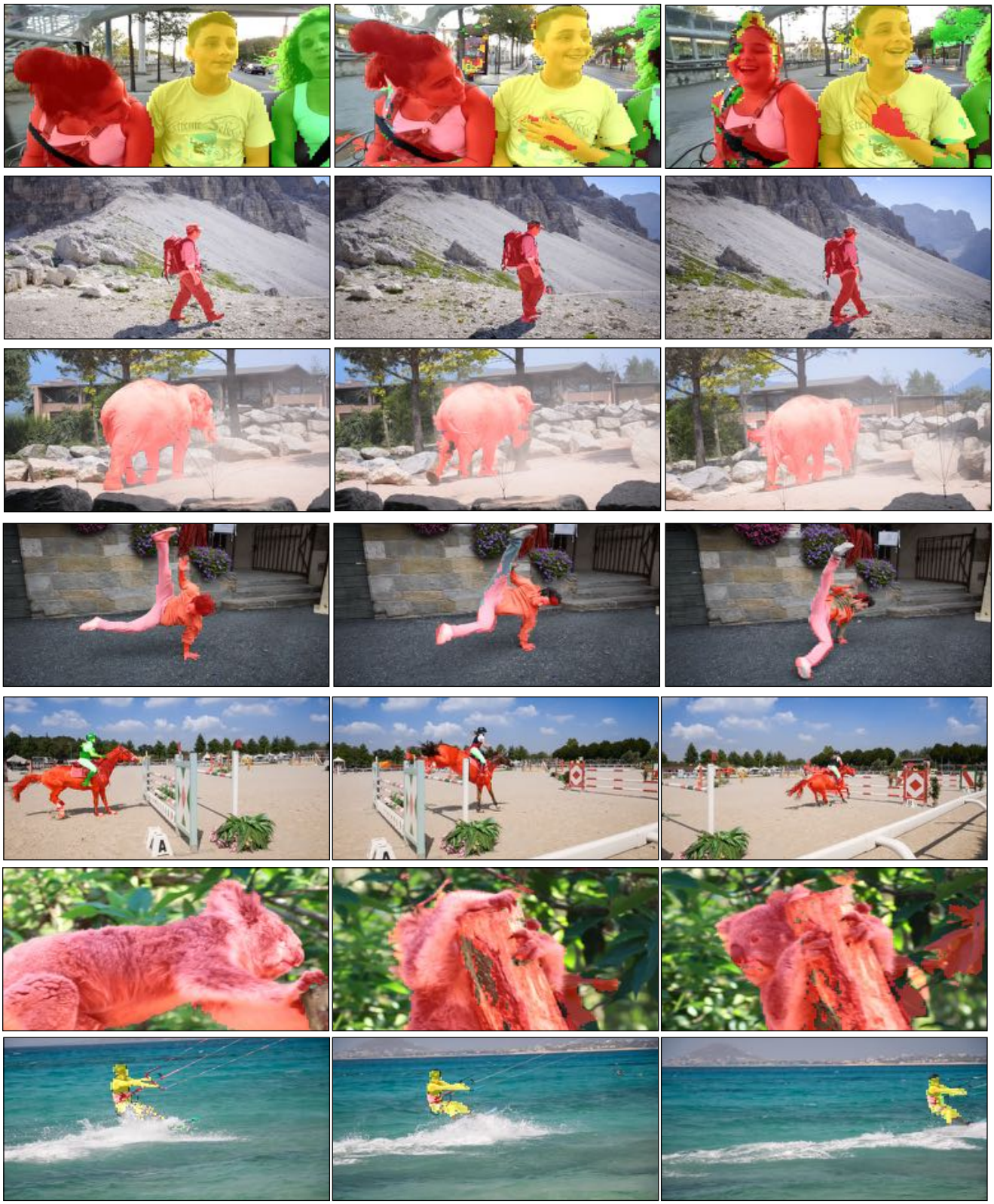


Figure 23. **Correspondences across Adjacent Frames** We show correspondences established in the adjacent frames of a video using pixel-level codes. For the sake of visualization, we select 256 random regions on the moving objects in a scene.



first frame (labeled) middle frame last frame

Figure 24. **Instance Mask Propagation:** Given the first labeled frame, we propagate the instance labels using the pixel codes. We show here the labels in the first frame on the left. We show the propagated labels from the first frame to the middle and last frame of the video.



first frame (labeled) middle frame last frame

Figure 25. **Instance Mask Propagation:** Given the first labeled frame, we propagate the instance labels using the pixel codes. We show here the labels in the first frame on the left. We show the propagated labels from the first frame to the middle and last frame of the video.

(a) input **(b)** output

Figure 26. **10X Spatial Super Resolution:** We train video-specific autoencoders using original videos. The reprojection property allows us to get hi-res outputs even when inputting a low-res sample at test time.

[29] Sean J. Liu, Maneesh Agrawala, Stephen DiVerdi, and Aaron Hertzmann. View-dependent video textures for 360° video. In *ACM UIST*, 2019. 1

[30] Yu-Lun Liu, Wei-Sheng Lai, Ming-Hsuan Yang, Yung-Yu Chuang, and Jia-Bin Huang. Learning to see through obstructions with layered decomposition. In *CVPR*, 2020. 1

[31] Ziwei Liu, Raymond A Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *ICCV*, 2017. 7

[32] Gucan Long, Laurent Kneip, Jose M Alvarez, Hongdong Li, Xiaohu Zhang, and Qifeng Yu. Learning image matching by simply watching video. In *ECCV*. Springer, 2016. 7

[33] Erika Lu, Forrester Cole, Tali Dekel, Weidi Xie, Andrew Zisserman, David Salesin, William T. Freeman, and Michael Rubinstein. Layered neural rendering for retiming people in video. *ACM Trans. Graph.*, 2020. 1

[34] Dhruv Mahajan, Fu-Chung Huang, Wojciech Matusik, Ravi Ramamoorthi, and Peter Belhumeur. Moving gradients: a path-based method for plausible image interpolation. *ACM Trans. Graph.*, 2009. 1

[35] Tomer Michaeli and Michal Irani. Blind deblurring using internal patch recurrence. In *ECCV*. Springer, 2014. 1

[36] Alasdair Newson, Andrés Almansa, Matthieu Fradet, Yann Gousseau, and Patrick Pérez. Towards fast, generic video inpainting. In *European Conference on Visual Media Production*, 2013. 1

[37] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *CVPR*, 2017. 7

[38] Junheum Park, Keunsoo Ko, Chul Lee, and Chang-Su Kim. Bmbc: Bilateral motion estimation with bilateral cost volume for video interpolation. In *ECCV*. Springer, 2020. 7

[39] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv:1704.00675*, 2017. 4, 5, 7, 8, 9, 10, 15

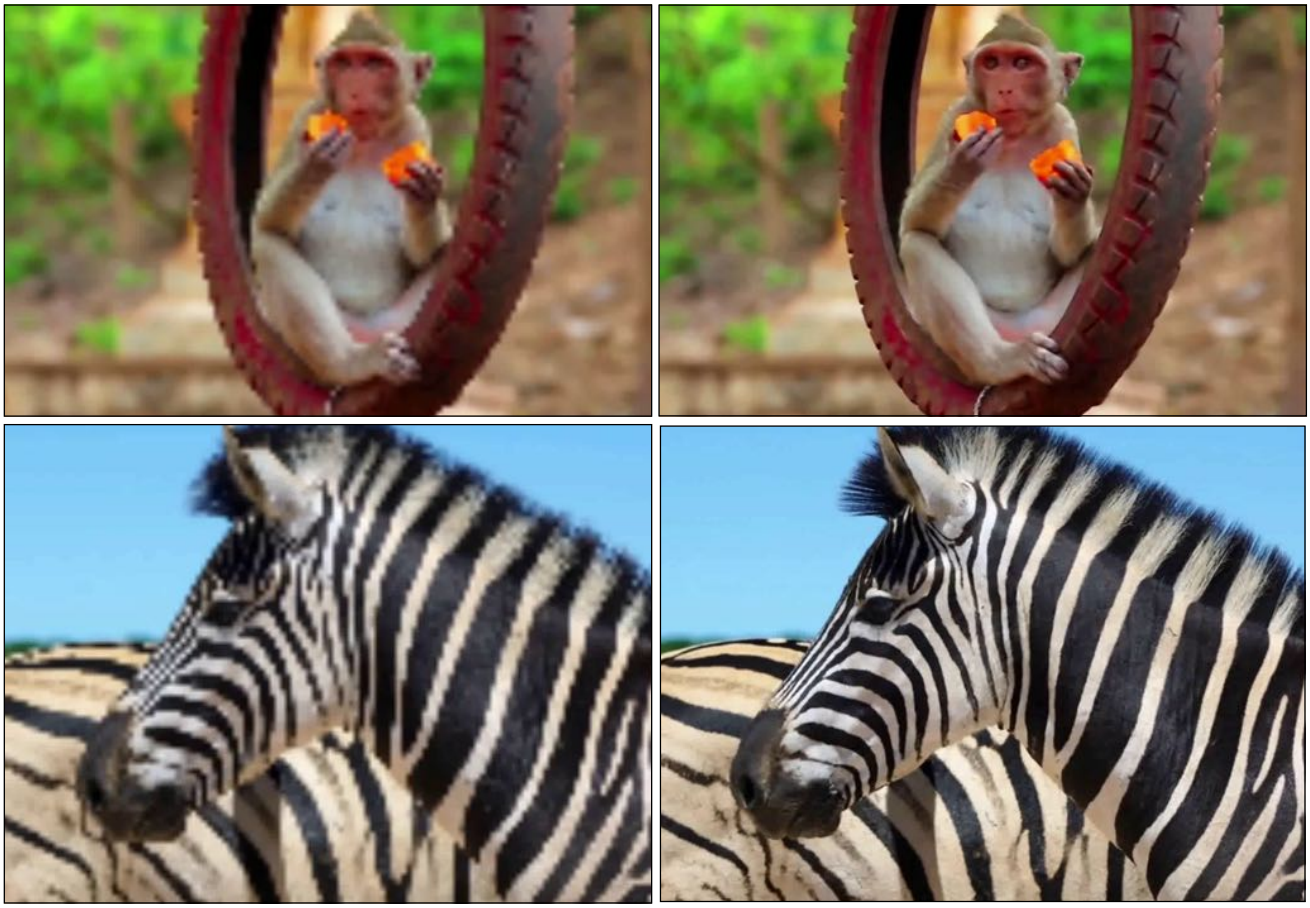
[40] Senthil Purushwalkam, Tian Ye, Saurabh Gupta, and Abhinav Gupta. Aligning videos in space and time. In *ECCV*, 2020. 1

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 2015. 9, 10

[42] Arno Schödl, Richard Szeliski, David H Salesin, and Irfan Essa. Video textures. *ACM Trans. Graph.*, 2000. 1, 5, 6

[43] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *ICCV*, 2019. 1

[44] Eli Shechtman, Yaron Caspi, and Michal Irani. Increasing space-time resolution in video. In *ECCV*. Springer, 2002. 1



(a) input **(b)** output

Figure 27. **10X Spatial Super Resolution:** We show more examples of 10X super-resolution with various animal videos. Once trained, the reprojection property of the autoencoder allows us to input low-res samples and get hi-res details in the outputs.

[45] Eli Shechtman, Yaron Caspi, and Michal Irani. Space-time super-resolution. *TPAMI*, 2005. 1

[46] Assaf Shocher, Shai Bagon, Phillip Isola, and Michal Irani. Ingan: Capturing and remapping the" dna" of a natural image. In *ICCV*, 2019. 1

[47] Assaf Shocher, Nadav Cohen, and Michal Irani. "zero-shot" super-resolution using deep internal learning. In *CVPR*, 2018. 1

[48] D Ulyanov, A Vedaldi, and V Lempitsky. Deep image prior. *IJCV*, 2020. 1

[49] Minh Vo, Srinivasa G Narasimhan, and Yaser Sheikh. Spatiotemporal bundle adjustment for dynamic 3d reconstruction. In *CVPR*, 2016. 1

[50] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *ECCV*. Springer, 2016. 1

[51] Jacob Walker, Kenneth Marino, Abhinav Gupta, and Martial Hebert. The pose knows: Video forecasting by generating pose futures. In *ICCV*, 2017. 1

[52] Xiaolong Wang, Allan Jabri, and Alexei A Efros. Learning correspondence from the cycle-consistency of time. In *CVPR*, 2019. 1

[53] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *ECCVW*, 2018. 8

[54] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *TPAMI*, 2007. 1

[55] Xiaoyu Xiang, Yapeng Tian, Yulun Zhang, Yun Fu, Jan P Allebach, and Chenliang Xu. Zooming slow-mo: Fast and accurate one-stage space-time video super-resolution. In *CVPR*, 2020. 1

[56] Rui Xu, Xiaoxiao Li, Bolei Zhou, and Chen Change Loy. Deep flow-guided video inpainting. In *CVPR*, 2019. 1

[57] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *IJCV*, 2019. 1

[58] Gengshan Yang and Deva Ramanan. Volumetric correspondence networks for optical flow. In *NeurIPS*, 2019. 1

[59] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6

input: 32x32 output: 1024x1024

Figure 28. **32X Spatial Super Resolution without Iterative Reprorjection:** We demonstrate that one can also train an autoencoder with noisy or low-res input samples during training. This enables us to input very low resolution inputs at test time, and get a hi-res output in a single iteration. We train an autoencoder using the low-res 32×32 input and a hi-res 1024×1024 output. This enables the model to be robust to low-res samples. We show the result of our approach on an unseen 32×32 input that yields a 1024×1024 output.

[60] Weiyu Zhang, Menglong Zhu, and Konstantinos G Derpanis. From actemes to action: A strongly-supervised representation for detailed action understanding. In *ICCV*, 2013. 6, 7, 9, 10, 14

[61] Jun-Yan Zhu, Yong Jae Lee, and Alexei A Efros. Averageexplorer: Interactive exploration and alignment of visual data collections. *ACM Trans. Graph.*, 2014. 1, 4

[62] Xiaoou Tang Yiming Liu Ziwei Liu, Raymond Yeh and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *ICCV*, 2017. 1

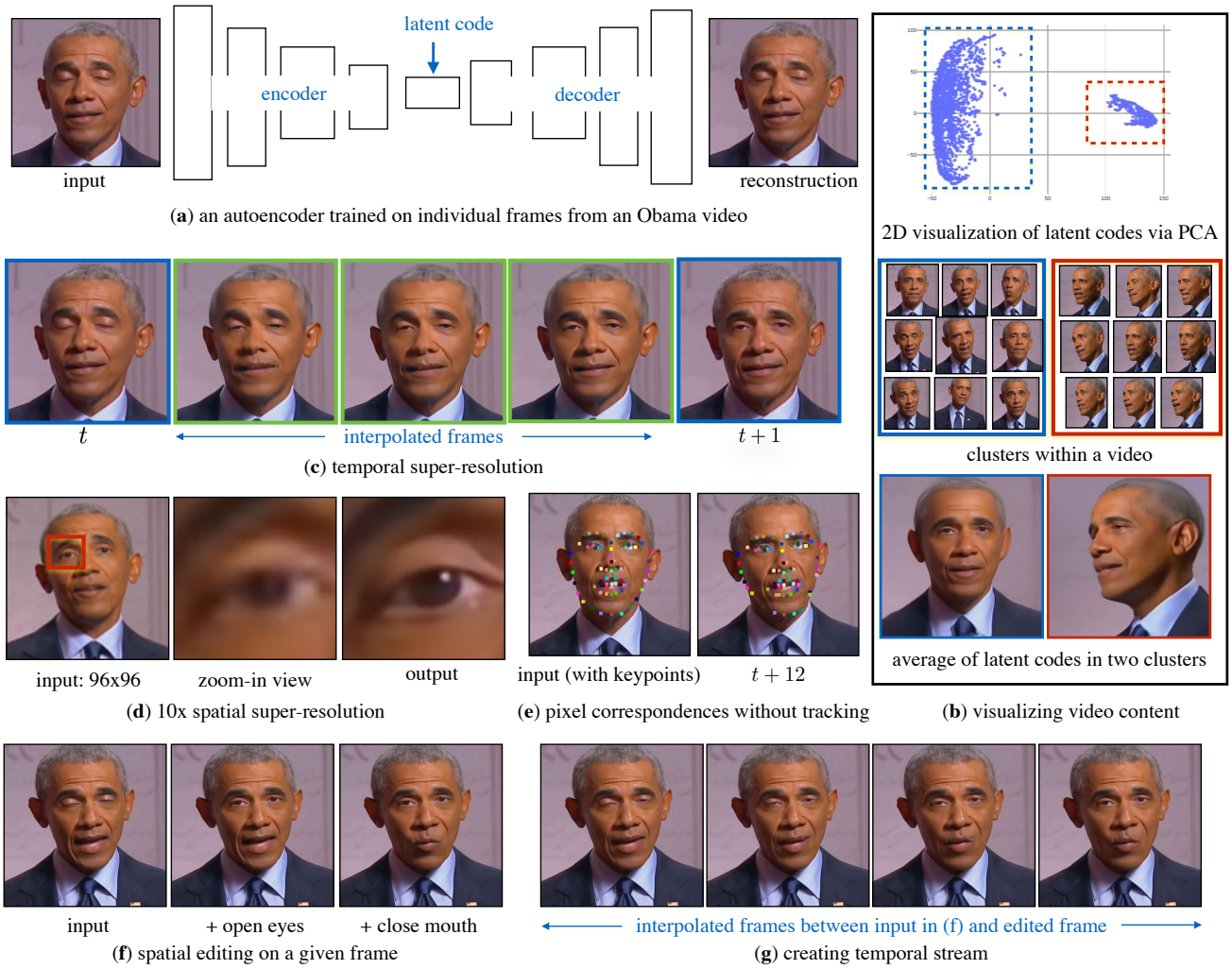
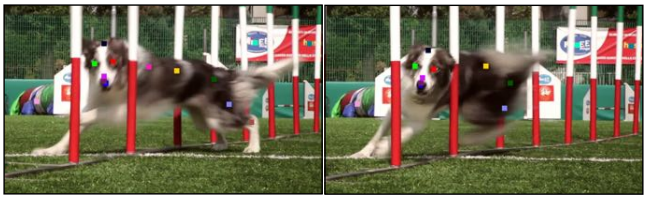


Figure 29. **What can we do with a video given a single video representation? (a)** We train a video-specific autoencoder on individual frames from a target video (here, 300 unordered 1024×1024 frames). **(b)** We use Principal Component Analysis (PCA) to **visualize latent codes** of the frames of this video in a 2D space. This 2D visualization shows two different data distributions within the video. We visualize the **clusters within the video** and **average of latent codes** in these two clusters. **(c)** We interpolate the latent codes of adjacent frames (and decode them) for **temporal super-resolution**. **(d)** By linearly upsampling low-res 96×96 image frames to 1024×1024 blurry inputs and passing them through the autoencoder, we can "project" such noisy inputs into the high-res-video-specific manifold, resulting in high quality 10X **super-resolution**, even on subsequent video frames not used for training. **(e)** We use hypercolumn features from the encoder and can do **pixel-level correspondences** between two frames in a video. **(f)** We can also do **spatial editing** on a given frame of video. Shown here is an input frame where eyes are closed. We copy open eyes from another frame and close mouth from a yet another frame, and pass it through the autoencoder to get a consistent output. **(g)** We can further **create temporal stream** between the original frame and edited frame by interpolating the latent code.

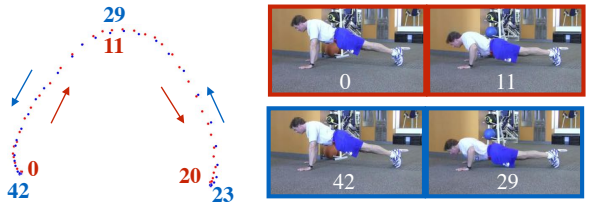
(**a**) temporal super-resolution

(**b**) 10x spatial super-resolution

(**c**) object removal



(**d**) pixel correspondences



(**e**) 2D visualization of latent codes

(**f**) average image of a video

(**g**) aligning two videos

Figure 30. We demonstrate a remarkable number of video processing tasks enabled by a simple video-specific representation: an *image* autoencoder trained on frames from a target video. **(a)** By interpolating latent codes of adjacent frames (and decoding them), one can perform **temporal super-resolution**. **(b)** By linearly upsampling low-res 96×96 image frames to 1024×1024 blurry inputs and passing them through the autoencoder, we can *project* such blurry inputs into the high-res-video-specific manifold, resulting in high quality **10X super-resolution**, even on subsequent video frames not used for training. **(c)** Manifold projection can also be used to **remove objects** marked with a bounding box. **(d)** Hypercolumn features of the autoencoder can be used to establish **per-pixel correspondences** across frames of a video via feature matching. **(e)** Multidimensional scaling of latent codes (via 2D Principal Component Analysis) allows for "at-a-glance" interactive video exploration; one can summarize visual modes and discover repeated frames that look similar but are temporally distant. **(f)** We can **summarize** videos by decoding the average latent codes (of all the frames in a video), which compares favorably to a naive per-pixel average image. **(g)** Manifold projection allow us to **align** two semantically similar videos and **retarget** from one to another (here, two baseball games).

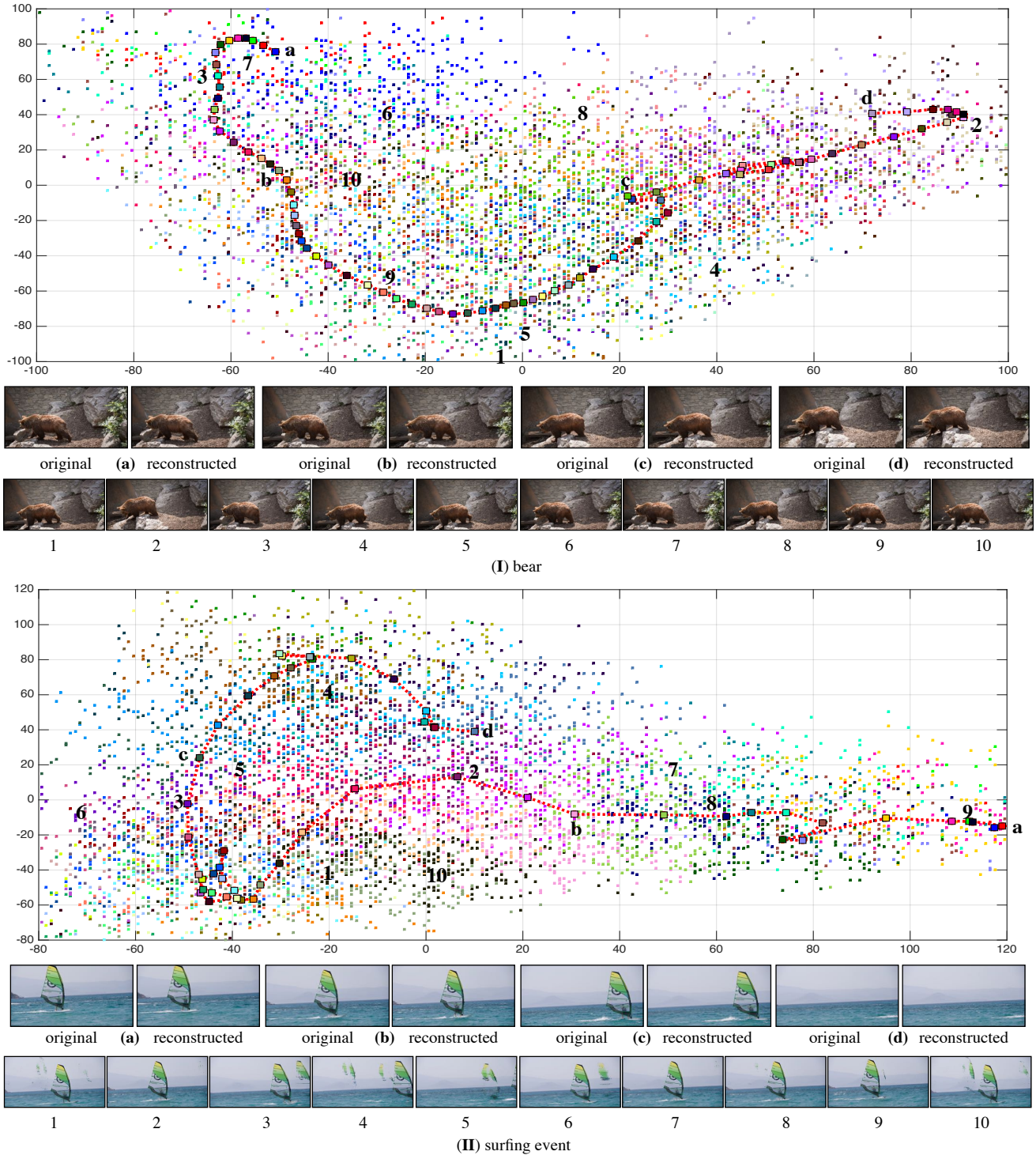


Figure 31. **Exploring the Manifold**: We train two video instance specific autoencoders on: **(I)** 82 individual frames from a bear sequence; and **(II)** 55 individual frames from a surfing event. We visualize the latent codes of original points on a 2D plot for two sequences using the bold squares. Each square represents an original frame in the video and is shown using a different color. The red line connecting the squares show the temporal sequence. We show **original** images and **reconstructed** images for four points: **(a)**, **(b)**, **(c)**, and **(d)**. We show random points on manifold M colored by the closest original frame. We visualize image reconstruction of a random subset of 10 in the bottom row. Note that latent coordinates even far away from the original frames tend to produce high-quality image reconstructions.

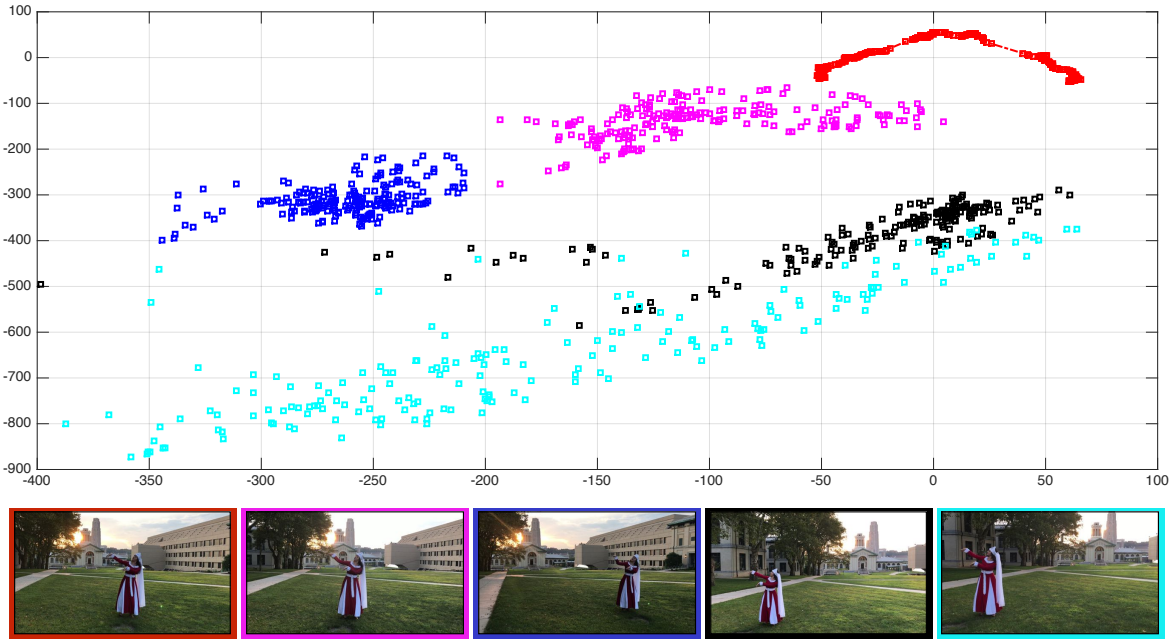


Figure 32. **Perturbation via Multi-Views:** We train a video-specific autoencoder with a sequence captured via a stationary camera (a frame of the sequence as shown in the red box). The 2D visualization of this sequence is shown with the red points in the visualization. Once trained, we take inputs from multi-views (shown by views from other colors). We observe that the points move farther away from the red points as we move away from the original camera/sequence that was used for training the video-specific autoencoder. For e.g., magenta points are close to the red points because the frames from two sequences look more similar than the frames representing cyan points.

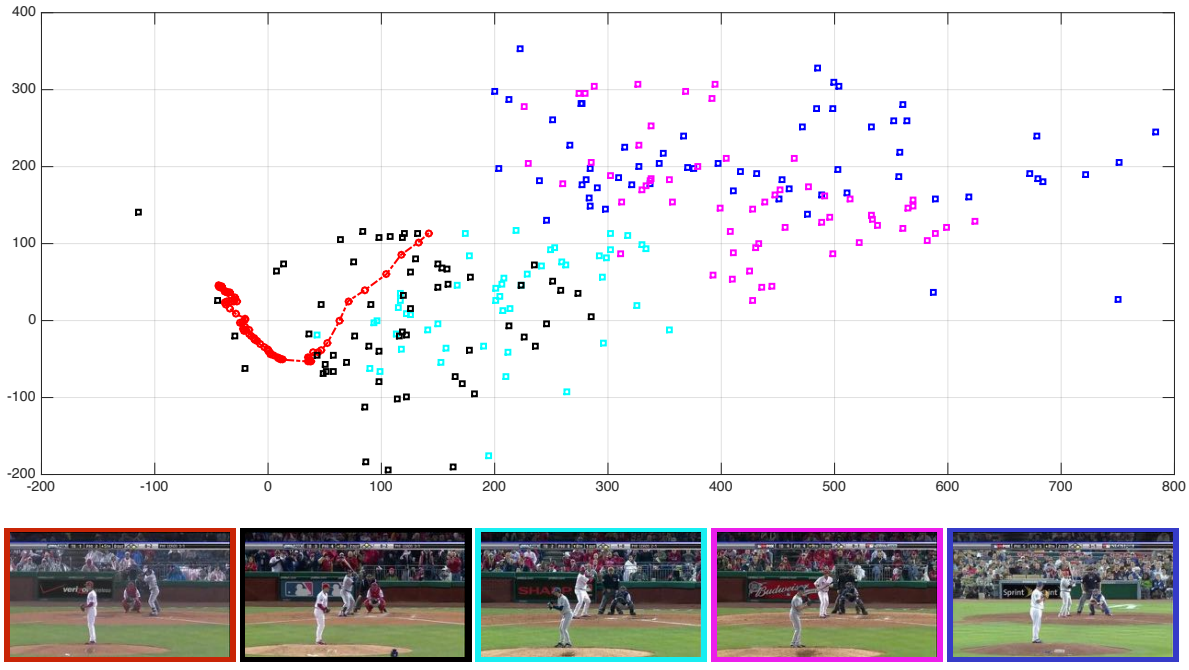


Figure 33. **Perturbation via Semantically Similar Events:** We train a video-specific autoencoder with one baseball game video (a frame of the sequence is shown in the red box). The 2D visualization of this sequence is shown with the red points in the 2D visualization of latent codes. Once trained, we take inputs from other baseball games (shown by views marked with other colors). We observe that the points move farther from the red points as the inputs become less similar to the original sequence.

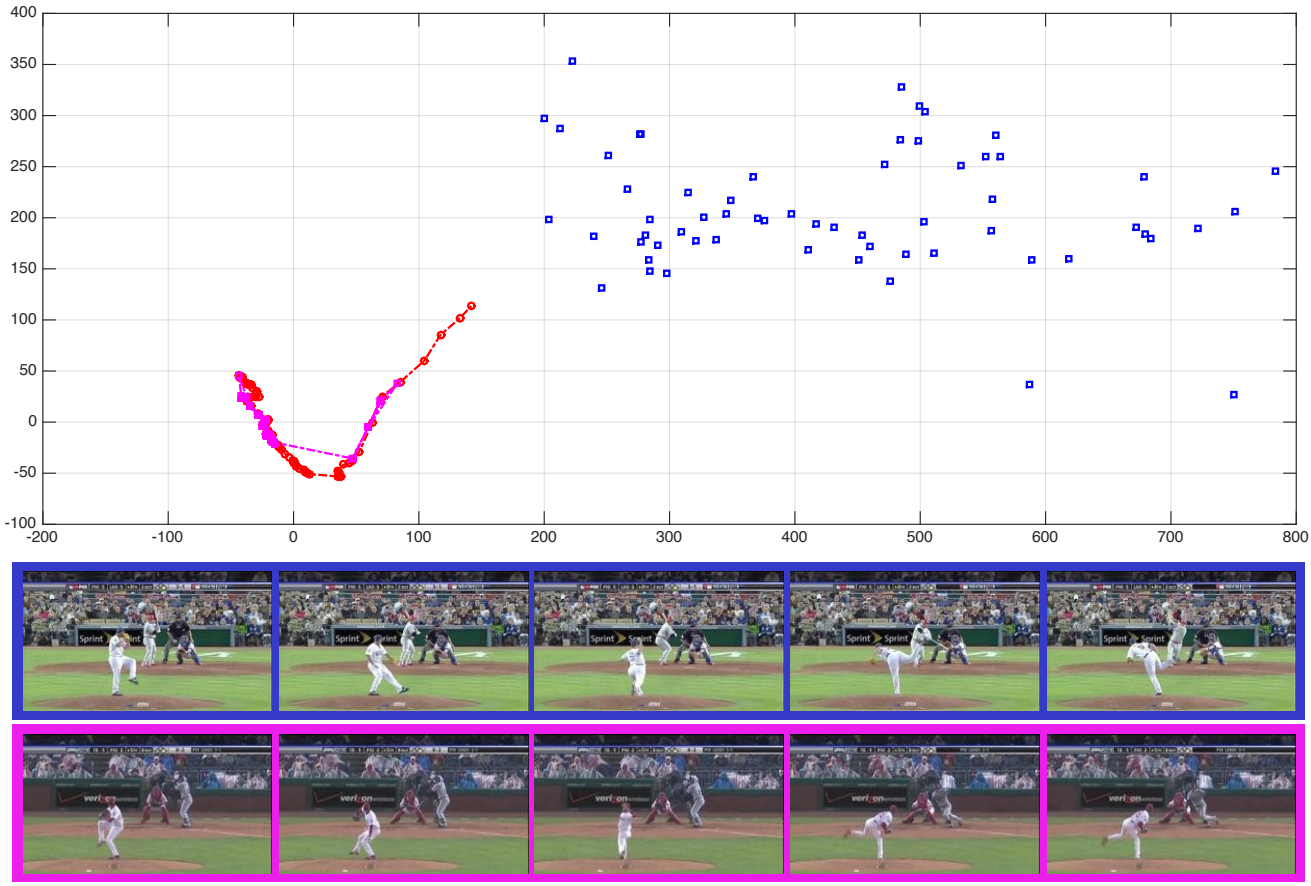


Figure 34. **Iterative Reprojection of Semantically Similar Events:** We train a video-specific autoencoder with one baseball game video. The 2D visualization of this sequence is shown with the red points. Once trained, we take inputs from other another baseball game (shown by blue points). We iteratively reproject the input and observe that one can align two semantically similar videos (shown by magenta points). We also show a few example inputs from the "unknown" sequence and the reconstructed images (after 25^{th} iteration) from the autoencoder. We observe temporally coherent outputs.

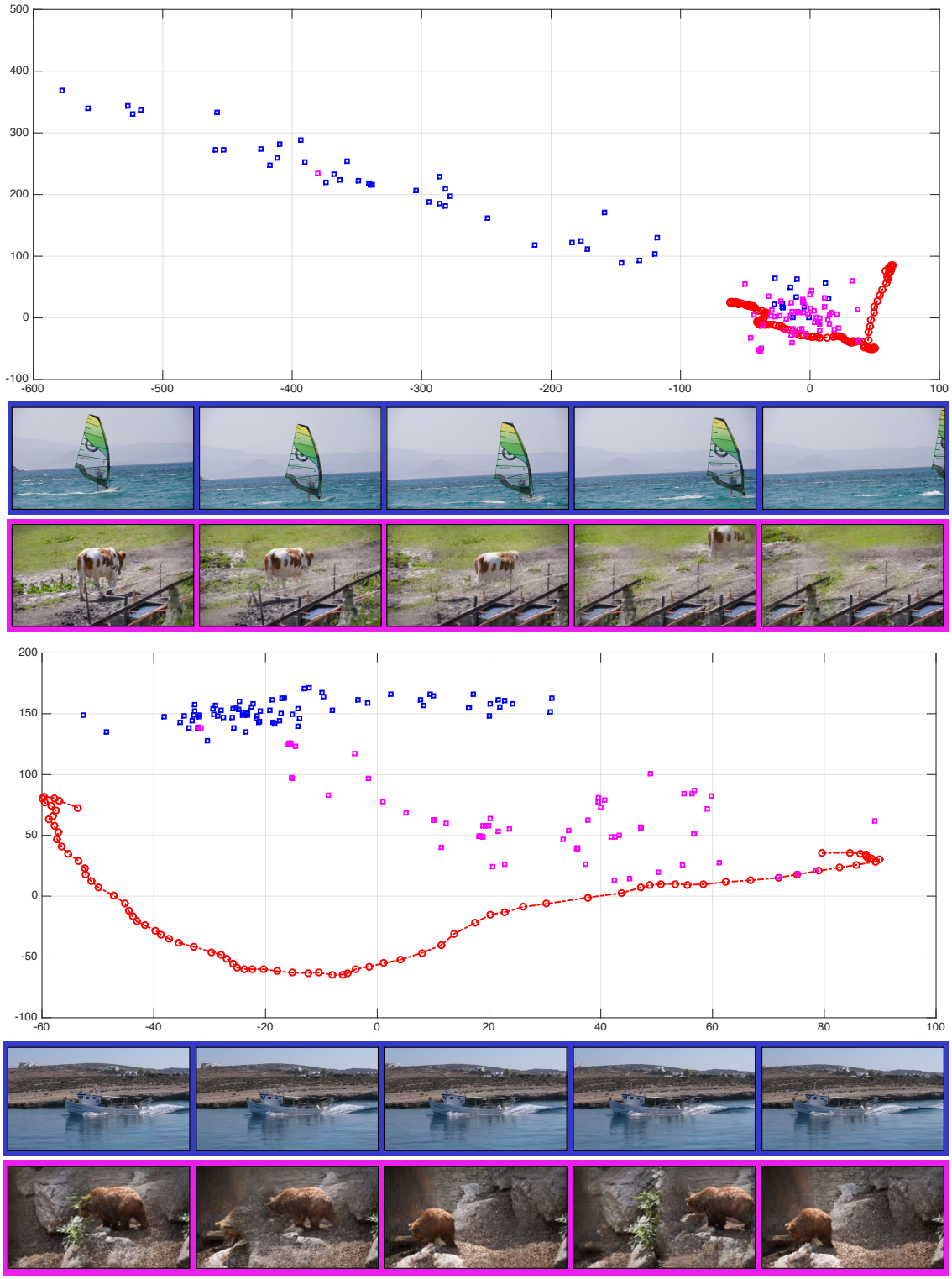


Figure 35. **Extreme Perturbation via Random Videos:** We train a video-specific autoencoder using a cow sequence. The 2D visualization of this sequence is shown with the red points. Once trained, we take inputs from a completely different video. For e.g., a surfing event in top example. The blue points show the projection of this video in the first iteration, and yields noisy output. Iteratively projecting these inputs bring them close to the original points. Here we show the results of 51^{st} iteration using magenta points. We also show a few example inputs from the sequence and the reconstructed images (after 51^{st} iteration). The outputs are noisy and does not have a temporal coherence.

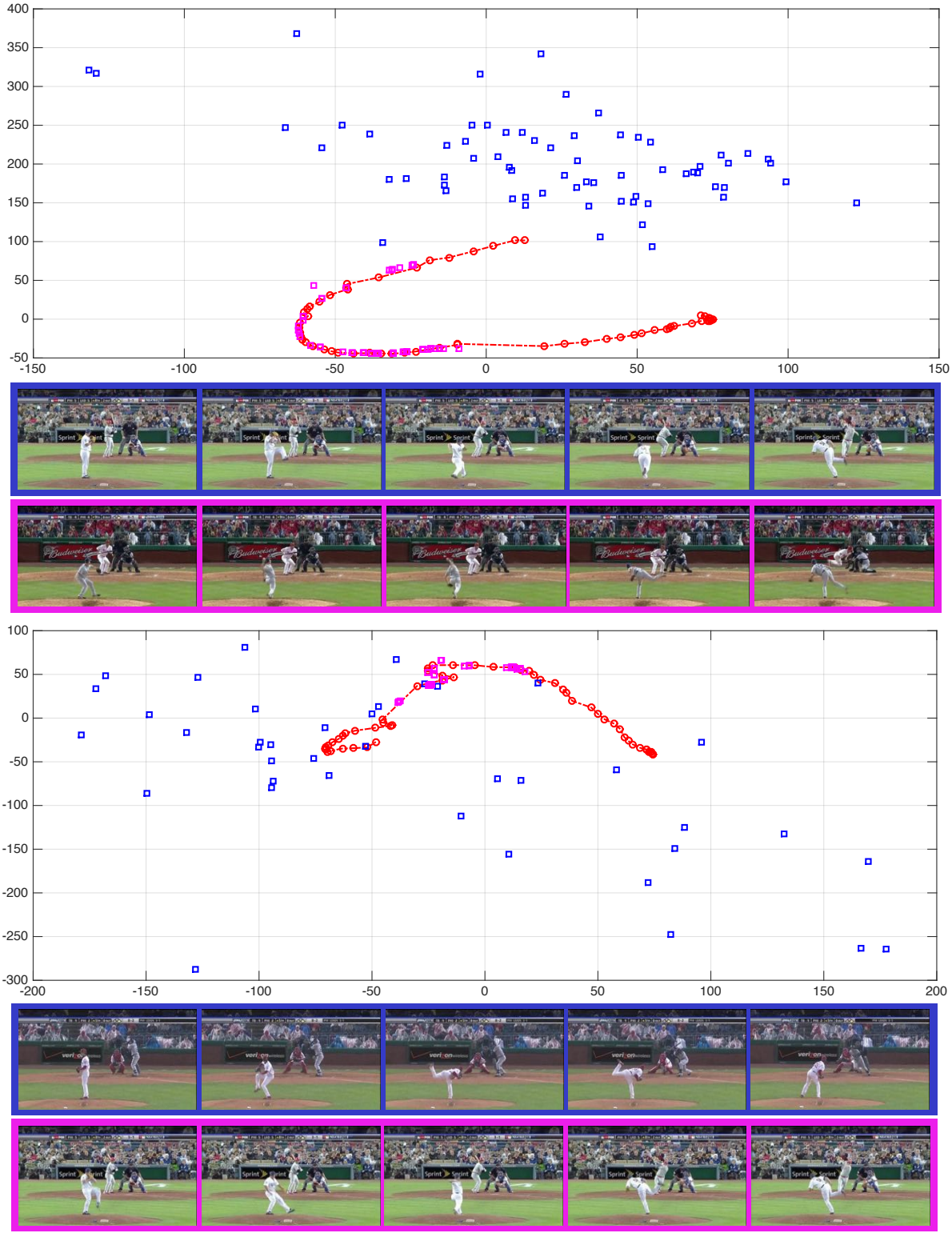


Figure 36. **Frame Correspondence across Videos:** Given a video-specific manifold for a baseball video (red points), we iteratively reproject the unknown video (blue points) to the manifold (shown using magenta points). This allow us to get corresponding frames in two videos via cosine similarity. We also show reconstructed images from the frames of unknown video using the video-specific autoencoder.

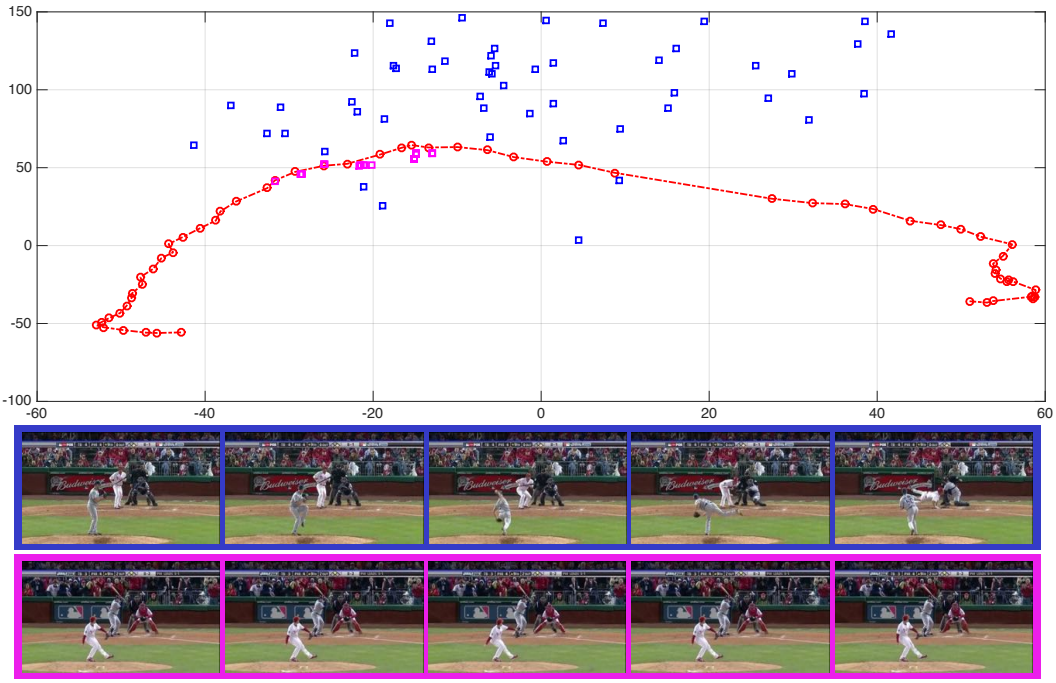


Figure 37. **Failure Case of Frame Correspondence:** We observe mapping (blue points to magenta points) collapse to a few similar points. This means there are no guarantees that we can align two videos using iterative reprojection property. However, one can see the success or failure of frame correspondences via 2D visualization of the latent codes.

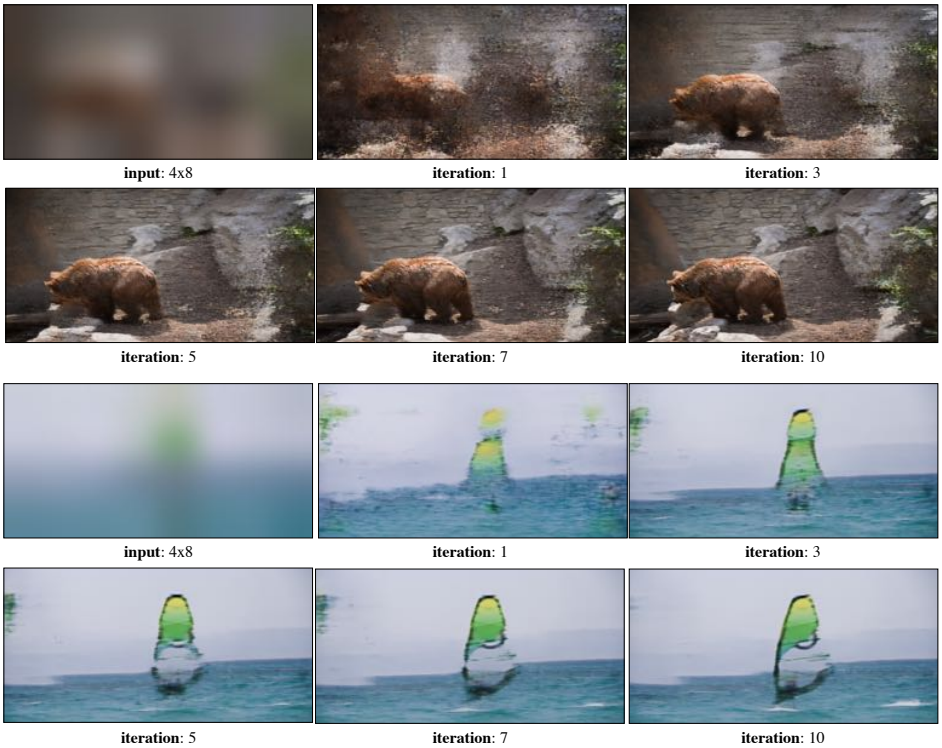


Figure 38. **Iterative Improvement via Reprojection Property**: We input a low-res 4×8 image and iteratively improve the quality of outputs. The reprojection property allows us to move towards a good solution with every iteration. At the end of the tenth iteration, we observe a sharp but *plausible* hi-res (256×512) output. However, it may not be an actual solution.

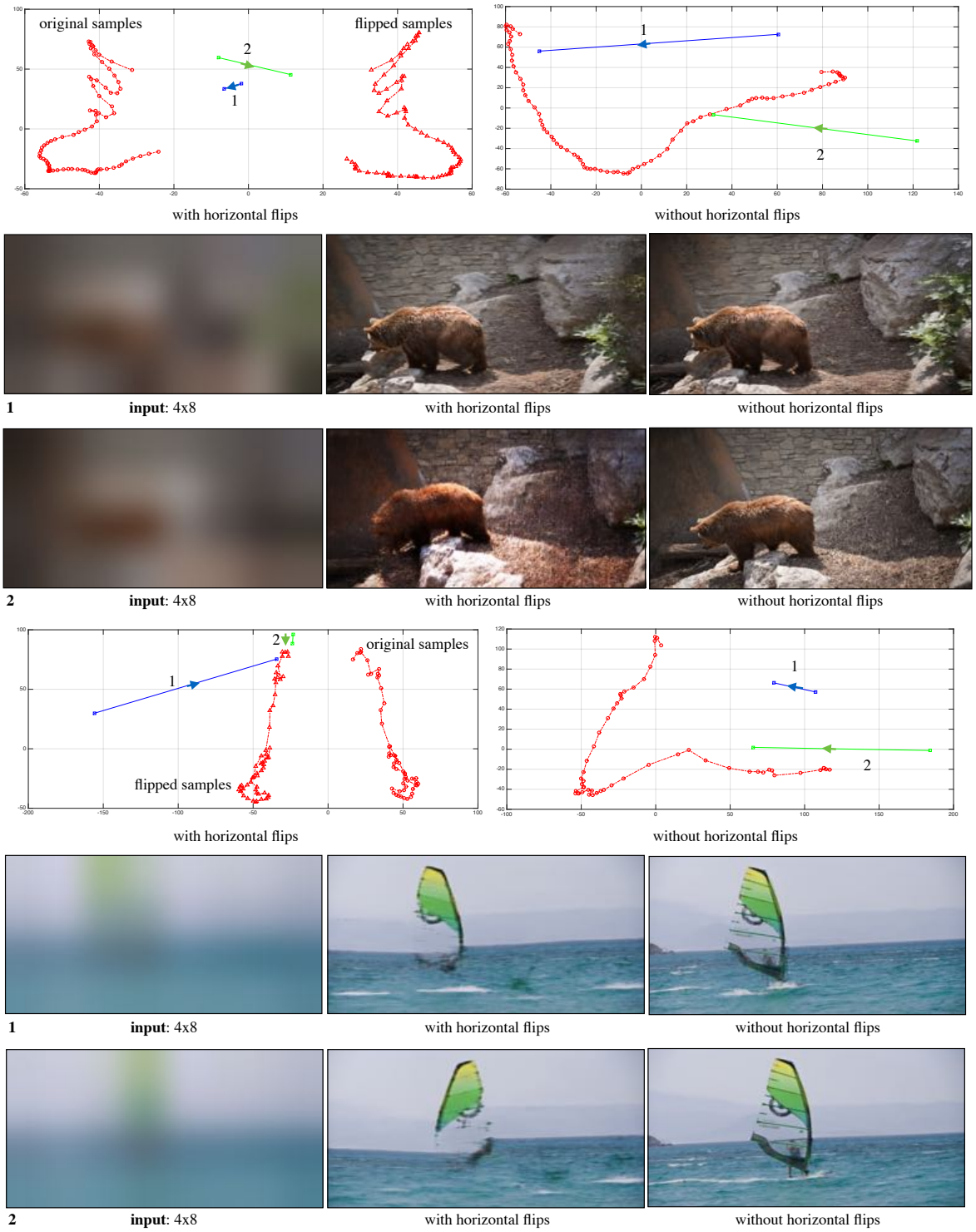


Figure 39. **Influence of random horizontal flips during training on video manifold**: We study the influence of random horizontal flips when training a video-specific autoencoder. We train two autoencoder, one with random horizontal flips and other without horizontal flips. The video manifold of an autoencoder with horizontal flips learns separate spaces for original and flipped samples. This is, however, not true for the model trained without random horizontal flips. We input a low-res 4×8 image and iteratively improve the quality of outputs. Due to two separate spaces in the first model, the autoencoder is confused in which direction to move the noisy input sample and thereby leads to slow movement. The other autoencoder is, however, able to move quickly towards a good solution. We observe sharp 256×512 output at the end of tenth iteration.