Temporally and Spatially Novel Video Frame Synthesis using 4D Video Autoencoder

Bidipta Sarkar
Stanford University
bidiptas@stanford.edu

Xinyi Wang
Stanford University
xinyiwang1@stanford.edu

Feiyang (Kathy) Yu
Stanford University
fyu9@cs.stanford.edu

Abstract

We propose a model that captures a 4D understanding of RGB video by decomposing a scene into a base 3D voxel representation, a camera trajectory, and a voxel flow. We merge the fundamental principles of the Video Autoencoder and Deep Voxel Flow methods by creating a flow network that captures the flow of the Video Autoencoder's voxel representations. To train our model, we try a variety of losses to capture the quality of voxel reconstruction given just a flow. We also analyze the importance of a given voxel to the final scene reconstruction, and use this to ensure that the flow captures the true dynamics of a scene. We train on the HMDB-51 dataset using videos of humans doing cartwheels, running, and walking, and we test our models using videos of humans throwing objects. We find that our flow autoencoder is able to capture the general dynamics of a scene to generate interpolating frames and perform novel view synthesis. Our code and models are available at https://github.com/KathyFeiyang/cs231n-project.

1. Introduction

The human understanding of the visual world is inherently 4-dimensional, with three spatial dimensions and one time dimension. When we see a video of a camera moving towards a ball, we understand that the ball is not growing in size but it is instead physically moving closer to the camera. The visual world is also always in motion. Some motion is very subtle and local, like leaves waving in the wind, while other motions can result in large changes in the scene, like people walking. In general, we can interpret a lot of the motion of a 3D scene as a flow.

Most video files only represent 3-dimensional discrete information, with two "compressed" spatial dimensions as the frame and one time dimension. However, each video is only a projection of a more complex 4-dimensional scene. Recovering the 4D scene from a video would allow us to see new views in space and time that were not directly captured in one video. A tool for recovering 4D scenes would be incredibly useful for artists who want to animate 3-d characters from a simple 2-d video, and it would be useful for video ML tasks which could use an understanding of scene flow to parse out the dynamics of actors in a scene.

We build our algorithm on top of the pretrained models for the 3D Video Autoencoder [1] by first pretending that the scene is static for short intervals of time. We add our own dynamics model by creating a new autoencoder that can represent the change in two scenes as a flow. We train and evaluate the model on splits of HMDB51, which allow us to get quantitative and qualitative results on its potential as an autoencoder and as a form of novel view synthesis.

Real-world videos contain rich dynamics, including moving cameras and objects that can change over time. These dynamics can be broken down to the (1) 3D structure of the scene, (2) trajectory of the camera, and (3) 3D flow of dynamic objects. An understanding of these components will enable us to generate new video frames that are novel spatially and temporally. In this work, we build a video autoencoder that captures the 3 components in a feedforward manner, and allows novel video frame synthesis at arbitrary views and time points. The contributions we aim to make are:

- Improve upon NeRF [2] [3] [4] [5] [6] [7] based models by allowing feedforward video synthesis, without needing to train on a series of input video frames.
- Improve upon the Video Autoencoder [1] by capturing dynamic objects and allowing interpolation of new video frames.
- Improve on 2D frame interpolation [8] by creating a 3D structure of the video, thereby allowing novel-view synthesis across space.

The inputs for the full video autoencoder are videos of shape $\mathbb{R}^{t\times c\times h\times w}$, where t represents the number of frames. The first output of the base video encoder is a deep voxel representation of the first frame of shape $C \times H \times W \times D$, where C is the deep voxel channel dimension. The second output of the base video encoder is a trajectory prediction of

shape $t \times 6$ that represents the rotation and translation of the camera at every timestep. We add an additional flow output at each timestep of shape $H \times W \times D \times 3$. To decode, we use all of the encoder outputs and try to reconstruct the original video to the best of our abilities.

Our best performing model yielded an 5.0% improvement in LPIPS, 3.6% improvement in PSNR and 6.1% improvement in SSIM relative to the baseline 3D Video Autoencoder. Additionally, it performed visibly better at capturing scene dynamics on HMDB51 videos, producing a flow of motion, while maintaining good performance at preserving the static scene and adjusting for camera pose trajectory.

2. Related Work

2.1. NeRF and NeRF-Styled Models

Neural Radiance Fields [9] have been able to create photorealistic novel view reconstructions by encoding a 3D scene as a deep neural network that maps 5D input coordinates (spatial location and viewing direction) to a volume density and view-dependent radiance. Among the papers inspired by NeRF, STaR [5] is able to reconstruct a dynamic scene with a single rigid object in motion with self-supervised learning. NR-NeRF [2] proposes a reconstruction and novel view synthesis approach by disentangling the dynamic scene into a canonical volume and its deformation. Dynamic-NeRF [4] generates novel views given a mononular view of a dynamic scene by jointly training a time-invariant static NeRF and a time-varying dynamic NeRF. iNeRF [6] performs pose estimation on objects or scenes represented as trained NeRF. NeRFies [7] makes use of an additional continuous volumetric deformation field to reconstruct non-rigidly deforming scenes. Finally, D-NeRF [3] extends the idea of NeRF to support scenes that evolve over time by adding an additional input dimension for time and creating a deformation network that encodes how a point in space moves over time.

The technique from D-NeRF can be used to solve the problem that we have, but it has some notable drawbacks. Most importantly, NeRF-based models require training a new neural network for every single scene. Because of this, trying to do novel view reconstruction based on a video is not as simple as a sequence of feed-forward networks.

2.2. Video Autoencoder for 3D structure

One of the primary inspirations for this project is the work by Lai et al. on Video Autoencoders [1]. Their work encodes a video as a deep voxel representation of the 3D scene along with a 3D trajectory of the camera motion. The deep voxel representation is generated by applying two 3D deconvolutions to the ResNet-50 [10] feature representation of the first input frame. The trajectory encoder concatenates the first input frame with the input frame at a different time step to predict a 6 DOF transformation of rotation and translation. The decoder takes the transformation and the deep voxels for the first frame and reconstructs the image for that corresponding frame.

This model allows for novel view reconstruction since the user can pick an arbitrary pose and evaluate a given image from there. It can also generate an accurate prediction of the trajectory without ever needing ground-truth labels for training. The biggest limitation of this work is that it cannot handle non-static scenes due to its assumption that the voxels from the first frame capture most relevant information of future frames.

2.3. Deep Voxel Flow and Spatial Transformation

Deep voxel flow by Liu et al. [8] is a technique to generate new frames by learning a 3D Voxel Flow across the two spatial dimensions and the time dimension. The voxel flow can be used to generate a realistic interpolation of existing frames, and it can even extrapolate the flow beyond existing frames to predict future frames.

The biggest drawback to this technique is its lack of explicit understanding of the 3D structure of the world. Although their frame interpolation techniques can be considered a form of novel view reconstruction across time, it cannot be used for novel view reconstruction across space.

Also modeling frame-to-frame transformations, Spatial Transformer Networks by Jaderberg et al. [11] are neural networks that spatially transform feature maps and provide spatial invariance to generic transformations. Their core mechanism is a sampling grid that specifies a set of input points to be sampled to generate transformed output points. Our work takes inspiration from this and uses grid sampling to transform video frames according to camera poses and dynamics.

3. Methods

Our work mostly builds on top of the Video Autoencoder [1] repository to enable view reconstruction across time and space. Our main additions include the ability to incorporate dynamics as a deep scene flow representation and interpolating scenes over time. Figure 4 in the appendix shows a graphical representation of the encoder and decoder.

3.1. Task Definition

Using the static Video Autoencoder, we can calculate the 3D voxel representation of any given frame and predict the camera pose transformation between any two frames. The base autoencoder can use the voxel representation of the first frame and the camera transformation to make a prediction of the other frame with the assumption that the world is static. We want to create a flow encoder and decoder module that can correct the difference between the true voxel

representation and the generated voxels using a prediction of the scene flow.

Symbolically, let $\mathcal{F}_{\text{3D}}$ be the static 3D voxel generator that takes an image as input, and let $\mathcal{H}$ be the camera pose estimator that takes in the two images as input. On the decoding side, we let $\mathcal{R}$ represent the function to rotate voxels according to the camera pose transformation, and we let $\mathcal{G}$ be the decoder that takes in voxels and outputs a reconstructed image. In this work, we treat the functions specified above as given since we build on top of pretrained models from the static Video Autoencoder. Suppose we are trying to calculate the voxel flow representation from frame 1 to frame i, which are represented as images I_1 to I_i. Like in the static case, we calculate $z = \mathcal{F}_{\text{3D}}(I_1)$ as the base deep voxel representation of the first frame, and $e_i = \mathcal{H}(I_1, I_i)$ as the predicted pose of the camera given the two input frames. The goal is to recover the "true" voxel representation at frame i, which is $z_i = \mathcal{F}_{\text{3D}}(I_i)$. Using just the information from the static autoencoder, we can make an initial estimate for the voxels as $z'_i = \mathcal{R}(z, e_i)$.

In order to improve the initial estimate z'_i to get a better final understanding of scene dynamics, we create a flow encoder-decoder pair, V_{enc} and V_{dec}. The flow encoder calculates $f = V_{\text{enc}}(z'_i, z_i)$, which is the representation of the flow of voxels from the initial prediction to the true voxels. We can also consider this flow as a representation of the change in scene between timestep 1 and timestep i. The flow decoder calculates $z^*_i = V_{\text{dec}}(z'_i, f)$, which applies the flow to the original voxels to reconstruct new voxels. This encoder-decoder pair is successful if $z^*_i \approx z_i$ since this implies that the flow encodes all the information necessary to construct the new voxels.

For the rest of this paper, we will consistently use z_i to refer to the voxel representation of image I_i, which we can consider the "ground truth" voxels. We will use $z'_{i,j}$ to refer to the static Video Autoencoder's prediction of the voxels at image I_i, which is a rigid transformation of the voxel representation of image I_j. Finally, we will use $z^*_{i,j}$ to refer to the output of the flow autoencoder, which tries to correct $z'_{i,j}$ by applying a dynamic flow.

3.2. Flow Encoder Architecture

The flow encoder, V_{enc}, calculates $f = V_{\text{enc}}(z'_i, z_i)$. Both z_i and z'_i are voxels of shape $C \times H \times W \times D$, and the flow is a tensor of shape $H \times W \times D \times 3$. The last dimension of the flow is a $(\Delta x, \Delta y, \Delta z)$ triplet between -1 and 1 representing the displacement of the given voxel in the final representation, where -1 and 1 represent the edges of the voxel grid.

The flow encoder first concatenates the two voxel representations along the first dimension (the channels), because the voxels are already aligned by the rotation transformation, $\mathcal{R}$. We then perform two strided 3D convolutions with a kernel size of 3, a standard 3D convolution of size 3, and then two transposed convolutions to restore the shape of the flow. The 3D convolutions are each followed by a ReLU nonlinearity, but the transposed convolutions use a tanh nonlinearity since we want the output to be between -1 and 1. In order for the flow to be realistically smooth, we remove very large changes in displacements between voxels that are close together. We remove high-frequency details by adding a simple 3D box filter of size 3, which is equivalent to a 3D convolution with all weights being 1 and a kernel size of 3.

3.3. Flow Decoder Architecture

The flow decoder treats the output of the flow encoder as a true "flow" and use the PyTorch grid_sample [11] [12] function to apply this flow to the voxels. Note that this function is used in Deep Voxel Flow [8] to model the flow of changes in the scenes in pixel-space. However, we use the 5D version of this function whereas Deep Voxel Flow uses the 4D version since we add an extra spatial dimension. This function is also used in the original static Video Autoencoder for the $\mathcal{R}$ function to model affine transformations. This part of the function has no learnable parameters, but it allows the encoder's output to be interpretable as a flow. This also means that we can do different operations on the output of the encoder to manipulate the flow in a geometrically meaningful way, such as halving the values of the flow to predict an intermediate flow between the start and end voxel representations.

Although this flow function gives a valid voxel representation, the grid_sample function may not give a perfect output. Specifically, it performs a trilinear interpolation of the voxel features when the input and output voxels do not perfectly align. This type of interpolation is typically acceptable in the image domain since it just results in a natural "blend" of RGB values, but this is not as good in the deep voxel domain since the features might have important properties that are destroyed after performing averaging with other feature vectors. To adjust for this, we add two 3D convolution layers and add a residual connection from the output of the grid_sample function to the final output. Intuitively, these new convolution layers act as a flow correction term for the final voxel representation.

When training, we use both the final output (flow-corrected output) and the intermediate output (from just after the grid_sample function) to calculate losses. By using the intermediate representation, we reinforce the idea that the output of the encoder represents the flow. We also saw that training became more stable once we included both representations in the loss.

3.4. Ideal Losses

For this section, assume that the voxel representations z_i and z'_i are perfect. In other words, z_i perfectly captures all features of the 3D scene at timestep i and z'_i perfectly captures all features of the 3D scene at timestep 1 but oriented to be in the same coordinate system as z_i based on the camera pose trajectory. This means that in a static scene, every feature in every voxel is identical, so $z_i = z'_i$. We can create a loss to capture the quality of the voxel flow between timesteps j and i as

$$\mathcal{L}_{flow} = ||V_{\text{dec}}(z'_{i,j}, V_{\text{enc}}(z'_{i,j}, z_i)) - z_i||_1$$

where $z'_{i,j}$ assumes that j is the starting timestep of the sequence. The original static Video Autoencoder has a loss function that tries to minimize $||z'_{i,j} - z_i||_1$ as a form of consistency loss in the static case, so we also use the L1 loss to emulate this consistency loss in the dynamic case. Intuitively, we use the L1 loss because individual outliers in our final voxel representation are not as harmful to the final image as small differences in features across many voxels. When training, we use consecutive video frames of length t, which we set to 4 to fit memory constraints unless otherwise specified. Since the flow should be reasonable between any pair of frames (including a frame with itself), we can consider every pair of frames within a sequence and minimize the sum of losses over all pairs. This way, we extract t^2 samples from a single video sequence of length t and efficiently make use of the training data.

Another loss we consider is the quality of interpolation. Assume that frames smoothly change over time. If we want to reconstruct the voxel representation of frame i given frames $i-1$ and $i+1$, we can define $z'_i = \mathcal{R}(z_{i-1}, \frac{1}{2}\mathcal{H}(I_{i-1}, I_{i+1}))$. In other words, the static components should be able to interpolate the middle frame by applying $\frac{1}{2}$ of the camera pose transformation of the surrounding frames. Beyond accounting for the camera pose trajectory, we also want the flow from frame $i-1$ to i to be half of the flow from $i-1$ to $i+1$, resulting in the interpolation loss

$$\mathcal{L}_{interp} = ||V_{\text{dec}}(z'_i, \frac{1}{2}V_{\text{enc}}(z'_{i+1,i-1}, z_{i+1})) - z_i||_1$$

which is based on the assumption that smooth flow is locally linear. When training with this loss, we can consider every set of consecutive triplets within a video sequence and also evaluate the loss on the training clip playing backwards. Therefore, if we have clips of length t, we can construct $2(t-2)$ interpolation losses.

3.5. Imperfect Voxel Representations

In the previous section, we assumed that all voxel representations were perfect. However, the real outputs of voxel construction have many flaws. For starters, we have to predict the values of voxels that are occluded by foreground objects, which can be impossible without extra information. Furthermore, the feature vectors associated with deep voxels do not have clean interpretations, and some information may not be as crucial to preserve.

When experimenting the qualities of these deep voxels, we noticed a few patterns emerge. First, we observed that voxels with very small values as absolute maximum components (or small l^∞ norm) could be set to 0 without much noticable changes to the final output image. Next, we observed that among voxels that are not clipped to zero, the absolute scale is not important to the final reconstruction, because we can scale each feature of all voxels by a constant and obtain a similar output. Figure 8 in the appendix shows how clipping voxels affects reconstruction quality. Figure 9 in the appendix shows how scaling voxels does not impact the hues of the output image. These results indicate that the original L1 losses are good for ensuring the scales are correct, but preserving cosine similarity is also important since this indicates that voxels have similar features. However, we note that clipping the voxels with small l^∞ norms to 0 is necessary since otherwise the cosine similarity values can be distorted by voxels with little importance to the output image. After applying these changes, we can be more confident that the losses ensure that voxels that represent the same underlying information are not penalized as harshly.

Next, we still need to deal with occlusions and determining which voxels to preserve. One way to verify if a voxel is consequential is to perturb it slightly and examine how much the output image changes in response to the perturbation. This is equivalent to finding the gradient of each channel of the voxel with respect to the image. If we let $\hat{I}_i = G(z_i)$ be the reconstructed (or interpolated) pixel image from a voxel representation and we let $\hat{s}_i$ be the sum of all pixels across all channels for $\hat{I}_i$, then $M = abs(\frac{\partial \hat{s}_i}{\partial z_i})$ can be interpreted as an "importance mask" over all channels for each voxel we are trying to reconstruct, where abs is the element-wise absolute value function. Voxel channels with a low importance mask should not make a big impact on the reconstruction and interpolation losses, since this means that they are either occluded or unimportant to the output image for some reason. This technique is very similar to saliency via backpropagation in the image classification domain [13], but here we consider how individual voxel channels relate to the final image instead of how individual pixels relate to a class score.

Unfortunately, we cannot directly use this mask to weight the reconstructed (or interpolated) voxel representations because this still allows for large changes in the image that would be rendered by the voxel representation. For example, if we only care about the voxels and channels that are important to the ground-truth representation

z_i, it is possible for the predicted z_i^* to have an occluding wall in front of those important voxels, resulting in a fundamentally bad reconstruction. To fix this, we can also set up an importance mask for the reconstructed voxel representation $M^* = abs(\frac{\partial s_i^*}{\partial z_i^*})$ and define a new mask as $M_{\text{use}} = max(M, M^*)$, where max is the element-wise max function. This way, the importance mask gives weight to voxels that impact either of the renderings.

To use the masks, we perform loss computations using $z_i \odot M_{\text{use}}$ and $z_i^* \odot M_{\text{use}}$ (where $\odot$ is the element-wise multiplication operator) instead of their original values.

3.6. Robust Losses

Using the lessons from the Imperfect Voxel Representations section, we can now make changes to the ideal losses to make them more robust. For the reconstruction loss, we define $z_i^* = V_{\text{dec}}(z'_{i,j}, V_{\text{enc}}(z'_{i,j}, z_i))$, and for the interpolation loss, we define $z_i^* = V_{\text{dec}}(z'_i, \frac{1}{2}V_{\text{enc}}(z'_{i+1,i-1}, z_{i+1}))$. First, we clip the features with an l^∞ norm less than a threshold to 0 in z_i^* and z_i to get $z^*_{i,\text{clip}}$ and $z_{i,\text{clip}}$. After analyzing many voxel reconstructions, we decided that a value of 0.005 would be a good cutoff point for voxels that do not impact reconstruction (or interpolation) quality. Next, we calculate the mask M_{use} using z_i^* and z_i. Finally, we apply the mask to $z^*_{i,\text{clip}}$ and $z_{i,\text{clip}}$ using element-wise multiplication to get $z^*_{i,\text{m}}$ and $z_{i,\text{m}}$. For both the flow reconstruction and interpolation losses, we can rewrite the loss as follows:

$$\mathcal{L}_{\text{vox}} = \text{cos\_sim}(z^*_{i,\text{m}}, z_{i,\text{m}}) + \lambda_1 ||z^*_{i,\text{m}} - z_{i,\text{m}}||_1$$

where cos_sim calculates the sum of cosine similarities between two voxels along the channel dimension, which is $\frac{x_1 \cdot x_2}{\max(||x_1||_2 \cdot ||x_2||_2, 10^{-6})}$ for each voxel's channels. Note that we have a small epsilon term of 10^{-6} to avoid dividing by zero. For the λ_1 hyperparameter, we chose $\lambda_1 = 1000$ to ensure that both parts of the loss have similar magnitudes and therefore similar weighting.

3.7. Perceptual Loss

To ensure that the voxels give a reasonable result for the downstream task of frame reconstruction, we have an additional perceptual loss term [14] that computes the similarity between the decoded voxels and the true frame. When starting this project, we believed that this loss alone would solve the task, but we saw that the flow would consistently converge to 0. With just the perceptual loss, the flow encoder would learn that the static video autoencoder's result was already very similar to the ground truth frame, and adding some flow would just disturb this local minima.

To integrate this loss with our voxel loss, we just define the total loss as

$$\mathcal{L} = \mathcal{L}_{\text{vox}} + \lambda_{\text{perc}} \mathcal{L}_{\text{perc}}$$

where $\mathcal{L}_{\text{perc}}$ is the perceptual loss, and $\lambda_{\text{perc}} = 0.1$ is the weight of the perceptual loss on the final loss. Note that this loss formulation applies to both the interpolation and flow losses.

4. Datasets

To train and evaluate our model, we need datasets that contain dynamic scenes. The static portions of the pretrained Video Autoencoder are trained on RealEstate10k, and our voxel flow components are trained on HMDB51.

4.1. RealEstate10k

The RealEstate10k dataset [15] contains 10 million frames derived from 80,000 YouTube real estate videos with dynamic camera trajectories and static scenes. The original static Video Autoencoder was trained on this dataset and we utilized its pretrained weights.

4.2. HMDB51

In this work, we use the HMDB51 dataset [16] as the main dataset, which consists of 6,849 videos from 51 action categories. To investigate whether the model captures dynamic objects, we focused on action categories with prominent motions, as opposed to categories with subtle motions such as facial motions. Specifically, we used the "run" (232 raw videos), "stand" (154 raw videos), and "cartwheel" (107 raw videos) action categories for training. [1] We used the "throw" action category as an external test dataset. We converted the videos in ".avi" format into a sequence of individual video frames in ".png" format. Figure 5 shows some example frames from the dataset.

4.3. Data Preprocessing

The data preprocessing pipeline resizes each frame to 256×256 spatial dimension with 3 color channels (RGB). We perform a sanity check on each video to keep for use the ones that do not contain significant noise after reconstructed with the original Video Autoencoder pipeline. During training, we split videos into clips each consisting of 4 frames at an interval of 1.[2]

We further cleaned the dataset by removing videos for which reconstructed videos of the original RealEstate10k pretrained Video Autoencoder differ drastically from the real videos in terms of the Frobenius norm. These video often exhibit distinct artifacts during reconstruction, such as the noise block at the top left corner of the video shown

[1] We used the "run" and "stand" action categories for most experiments. We used the "cartwheel" dataset for two experiments.

[2] We set the clip length to 6 in a few initial experiments we ran when training the model. Unless otherwise specified, the clip length is set to 4.

in Figure 7. These artifacts prevent the formulation of a meaningful scene flow and would likely degrade training. Therefore, videos containing these artifacts were removed from the training set.

5. Experiments/Results/Discussion

5.1. Model Design and Loss Functions

As a baseline, we adapted the original static Video Autoencoder to interpolate the middle frame without capturing voxel flow. We used the RealEstate10K-pretrained Video Autoencoder to generate 3D representations and camera pose transformation for a start frame and an end frame. We applied $\frac{1}{2}$ of the transformation to the start frame to generate the middle frame.

For our design with voxel flow, we built variations of the model exploring different architectures and loss functions. Specifically, we experimented with models without and with the Flow Correction component and different combination schemes of the loss functions, including (1) robust reconstruction loss; (2) robust reconstruction and interpolation losses; (3) robust reconstruction and interpolation losses, and perceptual loss. Note that the reconstruction and interpolation losses entail four components: the L1 distance loss and cosine similarity loss of predicted voxel representation with respect to the ground-truth representation before and after applying flow correction. The different model variations are shown in Table 1.

Method	Method Description
M0	Original static Video Autoencoder
M1	(Ideal L1 + robust cos_sim.) recon. loss
M2	(Ideal L1 + robust cos_sim.) recon. loss
M3	Robust recon. loss, with flow correct.
M4	Robust recon. + inter. loss, with flow correct.
M5	M4, with perceptual loss

Table 1. Variations of model architecture and combinations of loss functions that were experimented. "Cos. sim." stands for "cosine similarity"; "recon." stands for "reconstruction"; "inter."" stands for "interpolation"; "flow correct."" stands for "flow correction."

5.2. Implementation Details

We implemented variations of model using the PyTorch deep learning framework. We initialized the weights of the 3D Encoder, Trajectory Encoder, Decoder and Rotate components using the RealEstate10K [15] pretrained checkpoint created by [1] and froze them. We trained the weights for the Flow Encoder, Flow and Flow Correction components, which capture and apply scene dynamics, from scratch.

We trained the models using the Adam optimizer with learning rates of $1e-4$ and $2e-4$ depending on the model variation and batch size, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-8$, and 0 weight decay for 100 epochs. Within one epoch, we trained on 100 video frame sequences that were randomly sampled from the training set. We used a video frame sequence length of 3 or 4 and a batch size of 1 or 2 depending on the model variation. The choice of the small batch size was motivated by the memory intensive computation of reconstruction loss over every pair of video frames within a sequence, which amounts to $O(BL^2)$ computations where B is the batch size and L is the sequence length. We believed it to be more important to preserve a moderate sequence length to help the model learn long-distance transformations at the cost of batch size given hardware constraints. For more memory intensive experiments (M5) where the video frame sequence length had to be set to 3, we set a video frame interval of 2, which means that we sampled every other frame, to facilitate long-distance learning. Training was executed on 1 NVIDIA T4 GPU. Additional details can be found in the appendix in Table 3.

For combining losses, we set a weight of 1000 for the L1 distance loss, a weight of 1 for the cosine similarity and a weight of 0.1 for the perceptual loss if it is included. By doing so, we ensured that each loss contributes nontrivially to the overall loss despite their raw values being on different orders of magnitude.

To make training more efficient, we reused the voxel masks that were computed to ensure loss robustness and trained for multiple inner steps using the same voxel masks. This procedure effectively increases training speed because the voxel mask computation requires an expensive forward and backward computation through the Rotate and Decoder components. This procedure also has minimal impact on the quality of each training step, because given the small video frame sequence length, the relative importance of different voxels should remain similar across video frames. For training with only reconstruction loss, we trained for 10 inner steps. For training with both reconstruction and interpolation losses, we alternated between training for 5 inner steps on the reconstruction loss and training for 3 inner steps on the interpolation loss.

During testing, we tested on 3 different datasets from HMDB51: the "run" and "stand" (384 videos), "cartwheel" (98 videos) and "throw" (100 videos) action categories. A frame limit of 30 was applied on each video, and a frame interval of 1 was used. Videos that produced invalid evaluation outputs were excluded.

5.3. Middle Frame Synthesis Task

The goal of this work is to synthesize temporally and spatially novel frames. We define this goal as middle frame interpolation: The model takes in a start frame and an end

frame as input, and generates the middle frame between the start and end frames as output. Note that the model offers the flexibility of interpolating arbitrary frames. We evaluated the model by comparing the generated middle frame against the ground-truth middle frame from the original video. We quantified this comparison using three commonly used metrics in video synthesis: LPIPS (Perceptual Similarity) [14], PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity) [17].

5.4. Model Performance

We evaluated the models' ability to interpolate temporally and spatially novel middle frames. Given a video frame sequence, we extracted triples of consecutive video frames to form start, middle and end frames, had the model interpolate the middle frame, and quantitatively evaluated the interpolated middle frame against the real middle frame. With a sequence of length T, we interpolated the middle $T-2$ frames and kept the original first and last frames. To ensure fair comparisons between models, we evaluated the models on (1) the "run" and "stand" action categories, which most models except the baseline (M0), M1 and M2 were trained on; (2) the "cartwheel" action category, which M1 and M2 were trained; and (3) the "throw" action category, which is an unseen dataset for all models. Here, we discuss the results on the external "throw" dataset, as shown in Table 2. The other two sets of results are shown in Table 4 and 5 in the appendix.

As shown in Table 2, we observe that most variations of the 4D Video Autoencoder improve upon the baseline, suggesting the effectiveness of capturing scene dynamics. In particular, M5 performed the best across all metrics with a LPIPS of 1.503, PSNR of 24.010 and SSIM of 0.682. This yields an 5.0% improvement in LPIPS, 3.6% improvement in PSNR and 6.1% improvement in SSIM relative to the baseline. Additionally, M5's improved performance upon earlier model variations suggests that the combination of the robust version of reconstruction and interpolation losses, perceptual loss and flow correction works well in helping the model learn scene dynamics and representing this dynamics in a way that is meaningful with scalar multiplication. However, we also note that through the model progression from M1 to M4, where loss functions become more sophisticated, the change in performance was not pronounced. We observe a slight performance decrease in terms of LPIPS, a slight performance increase in terms of PSNR and no obvious trend with SSIM. Therefore, it is difficult to ascertain which single strategy is the key to improving voxel flow learning. Our experiments show that the various strategies and training hyperparameters used in M5 works well in conjunction. In addition, M4's suboptimal performance suggests that incorporating interpolation loss may not necessarily improve model learning. On

Method	LPIPS (↓)	PSNR (↑)	SSIM (↑)	Ext. Test
M0	1.582	23.167	0.643	T
M1	1.556	23.392	0.652	T
M2	1.560	23.416	0.652	T
M3	1.560	23.455	0.653	T
M4	1.889	21.858	0.586	T
M5	**1.503**	**24.010**	**0.682**	T

Table 2. Performance of model variations on middle frame synthesis for the HMDB51 "throw" action category (100 videos). A lower LPIPS score is better; higher PSNR and SSIM scores are better. The "external test" column indicates whether the model was not trained on this dataset.

the "run" and "stand" action categories (Table 4), which are representative of common human movements, and the "cartwheel" action category (Table 5), which contains large-scale motion, we observe consistent patterns as we did for "throw." Consistent performance improvements across three test datasets indicate that the 4D Video Autoencoder indeed allows for better representation of scene dynamics, with generalization to other unseen dynamics. The consistent relative performances of different models also suggest that M5 is a robustly more optimal design.

Beyond the three metrics, we visually examined the ability of different models to capture scene dynamics in middle frame interpolation: as in training, here we used every pair of ground-truth t and $t+2$ frames to interpolate the middle frame. Example interpolation results are shown in Figure 13 in the appendix. We observed that M5 was the only model that visibly described scene dynamics as indicated by the slight blur of motion, which in the example was the person moving backwards towards the camera; other models failed to capture this dynamics and simply described the person as being static. In addition, we observed that M5 produced less distortion to the static scene background compared with the baseline. This is likely the result of the 4D Video Autoencoder separating the static from the dynamic scene and therefore allowing the camera pose transformation to be less affected by the moving object. Meanwhile, we note that M5's representation of scene dynamics is still far from perfect: Because perceptual loss depends on image similarity, it tends to encourage blurring of image features as a conservative way to decrease image pixel distance. Therefore, although the flow-enabled model is able to represent dynamics, its representation is coarse.

5.5. Qualitative Reconstruction Results

To determine whether the models learn a flow that improves upon the static baseline, we have a testing task to reconstruct a frame given an initial image representation from at most 4 frames ago along with the model's predicted

Figure 1. **Frame reconstruction results (Jump)** From left to right: initial frame, baseline (M0) output, M5 output, and ground truth. We try to reconstruct frame 17 given frame 13 of HMDB video New_polish_best_goalkeeper_-_FC_Barcelona_jump_f_cm_np1_le_bad_0.avi.

Figure 2. **Frame reconstruction results (Cartwheel)** From left to right: initial frame, baseline (M0) output, M5 output, and ground truth. We try to reconstruct frame 13 given frame 9 of HMDB video Acrobacias_de_un_fenomeno_cartwheel_f_cm_np1_ba_bad_8.avi.

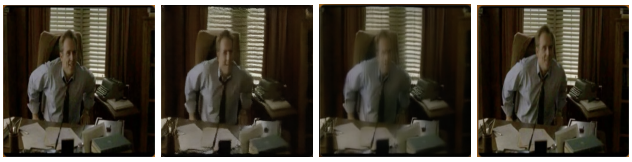

Figure 3. **Frame reconstruction results (Stand)** From left to right: initial frame, baseline (M0) output, M5 output, and ground truth. We try to reconstruct frame 29 given frame 25 of HMDB video A_Beautiful_Mind_6_stand_u_cm_np1_fr_med_3.avi.

flow to the current frame. This task is quite tricky since many aspects of the scene can change within 4 frames, so this is more of a stress test compared to the standard frame interpolation task. In all three examples above, the static frame autoencoder barely changes the pose of the human from the initializing frame, but it might change the camera pose slightly. The baseline images are very crisp, but they are completely incorrect given the ground truth we are trying to reconstruct. Our M5 is able to construct a flow that makes the reconstruction more similar to the ground truth.

From the Jump task 1, we see that the legs have actually moved towards the proper position, indicating that the model is able to capture the flow at this level. However, it is still unable to capture the change in hand movement, because that part of the flow is high frequency with respect to the scale of the voxels.

From the Cartwheel task 2, we see a very large deformation in the human model, which is somewhat captured as a blurry motion in the M5 output. Note how most of the blurring is around the human model instead of the entire scene, indicating that the flow is localized to the areas of motion. This reconstruction is quite poor since it essentially erases the subject from the picture, but it is showing signs of a proper flow encoding.

From the Stand task 3, we see a human figure standing up from a chair. Again, our M5 is able to capture some of the deformation as blurring, but it is unable to fully reconstruct the person's motion. Note that the static parts of the scene are still crisp in the M5 reconstruction, indicating that the flow is localized to the moving subject.

In general, we can see that the model is learning a flow representation that brings the reconstruction closer to the ground truth. The model tends to leave blurring and ghosting artifacts in dynamic regions of a scene, so this flow representation is able to correctly identify the moving voxels of the scene. The reconstructed videos are publicly available at the following Google Drive link: https://drive.google.com/drive/u/0/folders/13vXKMavrRmxpKWDcMiP8hAy_uOOQTCZE.

6. Conclusion/Future Work

We present 4D Video Autoencoder that generates a 4D understanding of the world from a video of a dynamic scene by disentangling the scene into a base scene geometry, the camera motion, and the scene flow. Extending prior work on 3D Autoencoder with flow encoder and decoder modules, this model is trained to capture the changes in the scene as voxel flow without using explicit flow supervision. Our model demonstrates superior ability of generating coherent videos with interpolated middle frames and smooth scene flow. Our model also achieves sufficiently better performance than the baseline on all evaluation metrics.

Future work could look into how the deep voxel flow captured by our model could be used to inform video classification into different action categories. The extracted scene flow could also enable better compression and decompression of videos with dynamic scenes. We would also like to see if a different 3D representation, like interpretable voxels or point clouds, could result in a better flow that does not suffer from the blurring artifacts of the current deep voxel representation.

Contributions & Acknowledgements

Our work builds on the Video Autoencoder repository on GitHub: https://github.com/zlai0/VideoAutoencoder/. We use their pre-trained models and add on to their general training scripts to incorporate our new flow modules.

References

[1] Z. Lai, S. Liu, A. A. Efros, and X. Wang, "Video autoencoder: self-supervised disentanglement of 3d structure and

motion," in *ICCV*, 2021. 1, 2, 6

[2] E. Tretschk, A. Tewari, V. Golyanik, M. Zollhöfer, C. Lassner, and C. Theobalt, "Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video," 2020. 1, 2

[3] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-nerf: Neural radiance fields for dynamic scenes," *CoRR*, vol. abs/2011.13961, 2020. 1, 2

[4] C. Gao, A. Saraf, J. Kopf, and J. Huang, "Dynamic view synthesis from dynamic monocular video," *CoRR*, vol. abs/2105.06468, 2021. 1, 2

[5] W. Yuan, Z. Lv, T. Schmidt, and S. Lovegrove, "Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering," *CoRR*, vol. abs/2101.01602, 2021. 1, 2

[6] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin, "iNeRF: Inverting neural radiance fields for pose estimation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021. 1, 2

[7] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla, "Nerfies: Deformable neural radiance fields," *ICCV*, 2021. 1, 2

[8] Z. Liu, R. Yeh, X. Tang, Y. Liu, and A. Agarwala, "Video frame synthesis using deep voxel flow," in *ICCV*, pp. 4473–4481, 10 2017. 1, 2, 3

[9] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020. 2

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. 2

[11] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *CoRR*, vol. abs/1506.02025, 2015. 2, 3

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019. 3

[13] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014. 4

[14] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018. 5, 7

[15] R. Tucker and N. Snavely, "RealEstate10K: A large dataset of camera trajectories." 5, 6

[16] H. Kuhne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb: A large video database for human motion recognition," in *IEEE International Conference on Computer Vision (ICCV)*, 2011. 5

[17] A. Horé and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th International Conference on Pattern Recognition*, pp. 2366–2369, 2010. 7

Appendix

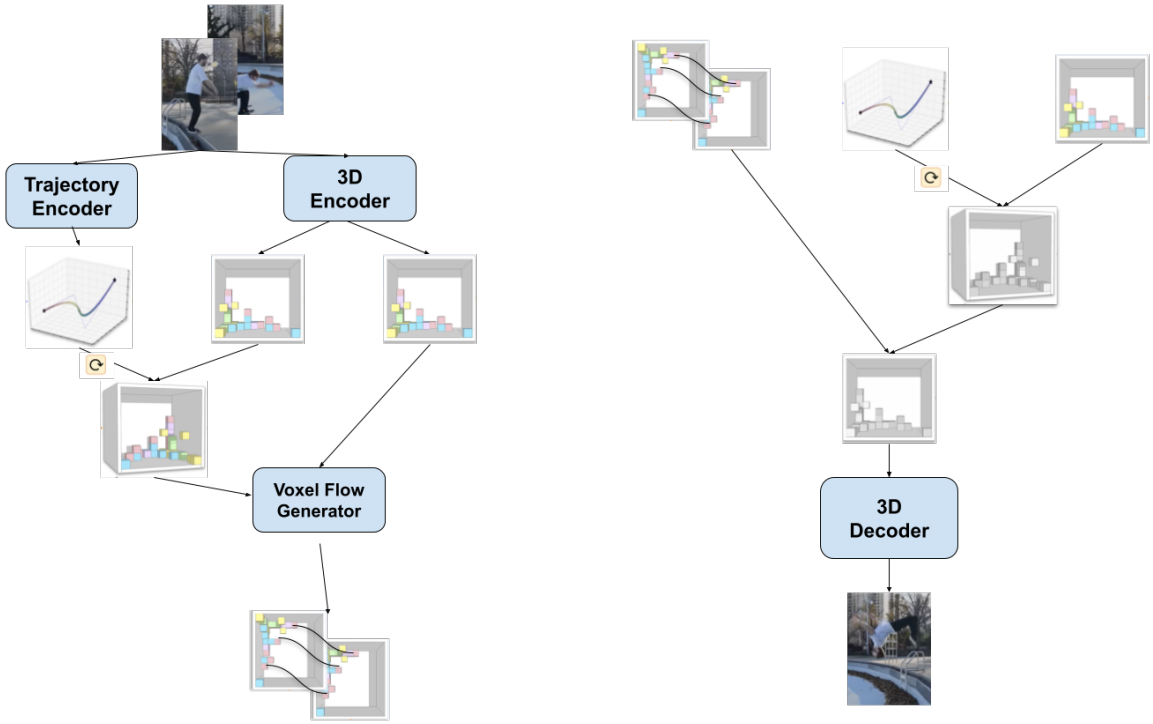

Figure 4. **Structure of the Encoder:** With the first input frame and the end frame stacking together as the input to the encoder, we generate the 3D deep voxel representations of both frames using the 3D encoder and the camera trajectory from the first frame to the end frame using the Trajectory encoder. We then re-entangle the 3D structure of the first frame and the camera trajectory to render an end frame that disregards the motion of the object(s) in the scene. Our voxel flow generator then takes in the voxel representations of the transformed frame and the end frame to generate the voxel flow of the moving object(s) in the scene.
Structure of the Decoder: The decoder first takes in the voxel representation of the initial frame and the camera trajectory (either taken from the encoder or a novel one) to generate the voxel representation of the middle frame between the first and the end frames, disregarding the change of the moving object(s). Then we apply the voxel flow (either taken from the encoder or a novel one) to the 3D structure of the middle frame to get the middle frame with the dynamic object(s) transformed, which is inputted into a 3D decoder to reconstruct the pixel representation of the interpolated middle frame.

Figure 5. **Sample frames:** The first two frames are taken from "stand" category and the last two frames are taken from "run" category in the HMDB dataset.

Figure 6. **Successful 3D Autoencoder:** From left to right: reconstructed first frame, ground truth first frame, reconstructed final frame, ground truth final frame. As we can see, there is no clear noise and the final reconstruction is reasonable using the base 3D Autoencoder that has no understanding of dynamic scenes. Notice that the reconstructed final frame is able to recover the camera alignment even though the goalie has moved.

Figure 7. **Unsuccessful 3D Autoencoder:** From left to right: reconstructed first frame, ground truth first frame, reconstructed final frame, ground truth final frame. There is clear noise at the top left corner of the reconstructed images, implying that the video autoencoder trained on RealEstate10K is sometimes not robust to the HMDB data.

Method	Training Details
M0	N/A
M1	Created clips of length 6 from training videos and took the first 4 frames to form training sequences. Used a learning rate of $1e-4$ and batch size of 2.
M2	Created clips of length 6 from training videos and took the first 4 frames to form training sequences. Used a learning rate of $2e-4$ and batch size of 2.
M3	Created clips of length 6 from training videos and took the first 4 frames to form training sequences. Used a learning rate of $2e-4$ and batch size of 1 due to memory constraints.
M4	Used clip length and frame sequence length of 4. Used a learning rate of $2e-4$ and batch size of 1.
M5	Used clip length and frame sequence length of 3 due to memory constraints. Used a frame interval of 2 to counteract the decreased sequence length. Used a learning rate of $2e-4$ and batch size of 1.

Table 3. Training details for variations of the models.

Figure 8. **Voxel Clip Rates:** The left set of images represents the reconstructions from voxels with no clipping (0% of voxels removed). The second set of images has all voxels with an absolute maximum value of channels below 0.0025 clipped to 0 ($\sim$10% of voxels removed). The third set of images sets the clipping value to 0.005 ($\sim$50% of voxels removed). The final set of images sets the clipping value to 0.01 ($\sim$75% of voxels removed).

Figure 9. **Voxel Scale Rates:** The left set of images represents the reconstructions from voxels with no scaling (scaling of 1). The second set of images has all channels of all voxels scaled by 0.5. The third set of images scales all voxels by 1.5. The final set of images scales all voxels by 2. Note how hues are mostly consistent across all images, but the scaling affects the contrast and saturation.

Figure 10. **Frame reconstruction results (Jump)** From left to right: baseline (M0) output, M1, M2, M3, M4, M5 output, and ground truth. We try to reconstruct frame 17 given frame 13 of HMDB video New_polish_best_goalkeeper_-_FC_Barcelona_jump_f_cm_np1_le_bad_0.avi.

Figure 11. **Frame reconstruction results (Cartwheel)** From left to right: baseline (M0) output, M1, M2, M3, M4, M5 output, and ground truth. We try to reconstruct frame 13 given frame 9 of HMDB video Acrobacias_de_un_fenomeno_cartwheel_f_cm_np1_ba_bad_8.avi.

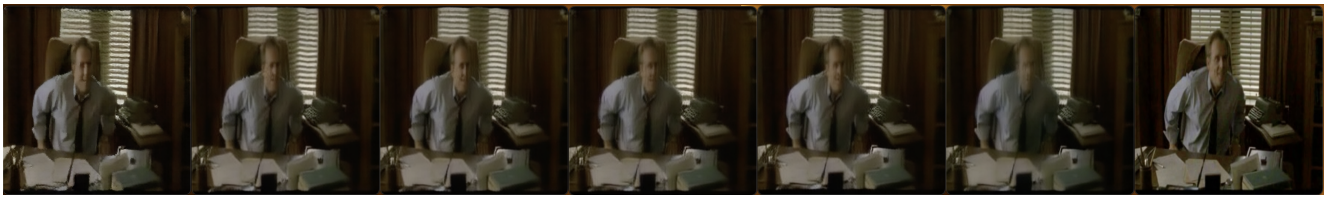

Figure 12. **Frame reconstruction results (Stand)** From left to right: baseline (M0) output, M1, M2, M3, M4, M5 output, and ground truth. We try to reconstruct frame 29 given frame 25 of HMDB video A_Beautiful_Mind_6_stand_u_cm_np1_fr_med_3.avi.

Figure 13. **Middle frame synthesis results (Throw)** From left to right: ground truth, baseline (M0) output, and M5 output. The results were generated for a video frame sequence with interval of 3. This is to make the discrepancies between models more pronounced and because video frames in the dataset often had identical consecutive frames.

Method	LPIPS (↓)	PSNR (↑)	SSIM (↑)	Ext. Test
M0	1.642	24.508	0.619	T
M1	1.620	24.800	0.629	T
M2	1.624	24.827	0.630	T
M3	1.628	24.831	0.629	F
M4	1.925	23.308	0.567	F
M5	**1.590**	**25.308**	**0.658**	F

Table 4. Performance of model variations on middle frame synthesis for the HMDB51 "run" and "stand" action categories (384 videos). A lower LPIPS score is better; higher PSNR and SSIM scores are better. The "external test" column indicates whether the model was not trained on this dataset.

Method	LPIPS (↓)	PSNR (↑)	SSIM (↑)	Ext. Test
M0	1.624	23.654	0.690	T
M1	1.596	24.061	0.705	F
M2	1.604	24.068	0.706	F
M3	1.605	24.074	0.704	T
M4	1.876	22.635	0.649	T
M5	**1.590**	**24.477**	**0.719**	T

Table 5. Performance of model variations on middle frame synthesis for the HMDB51 "cartwheel" action category (98 videos). A lower LPIPS score is better; higher PSNR and SSIM scores are better. The "external test" column indicates whether the model was not trained on this dataset.