CS 229 Project Report: Generating Video from Images

Project Category: Computer Vision

Geoff Penington (geoffp), Mae Hwee Teo (maehwee) and Chao Wang (cwang15)

June 9th, 2019

1 Introduction

In this project we used deep learning techniques to generate moving videos from still images. Specifically, we considered the task of taking two images, the initial image and the final image, and generating a moving video that realistically interpolated from the initial image to the final image. In previous studies, this task was known as video completion [1].

Generative video creation remains a major challenge, even for modern deep-learning techniques. In large part, this is because the space of possible output videos is exceptionally large, even when the videos are short and low resolution. For example, we attempted to generate 30 frame RGB color videos, where each frame consisted of 64×64 pixels. The space of such videos has dimension $30 \times 64 \times 64 \times 3 = 368640$.

To deal with this high-dimensionality, while keeping the number of trainable parameters manageable, we used three-dimensional convolutional neural networks. To generate the videos, we used the two dominant state-of-the-art techniques for conditional deep generative learning. These are conditional variational autoencoders (CVAEs) and conditional generative adversarial networks (CGANs).

2 Related Works

The task of generating a video from a single initial frame was previously attempted in [2] and [3]. [2] trained a generative adversarial network (GAN) on two million unlabelled videos in order to generate short videos of up to a second at full frame rate based on a static image. A two-stream architecture was used where the foreground and background were generated separately. [3] trained a conditional variational autoencoder (CVAE) on thousands of videos. Both achieved some success, although the generated videos were still very unrealistic compared to images generated by modern deep learning techniques.

The task of video completion, as done in this project, was previously addressed in [1] and [4]. Both papers used an architecture where the initial and final images were encoded in a latent space (that, for example, described the state of a figure in the image), a sequence of actions were generated in the latent space and then, finally, the latent sequence was translated into an output video. Both works also used GANs as part of their architectures. However, [1] used a recurrent neural network (RNN), whereas [4] adopted a fully convolutional approach, as done in our project.

In contrast to previous work, we always worked directly at the level of pixels, without any handcrafted features, such as a separation of foreground and background features or a handcrafted latent space designed to capture the relevant degrees of freedom in the image. This makes the task of video completion more challenging in many ways. However, it has often turned out in the past that, as computational power has increased, techniques that relied heavily on handcrafted features have ended up being outperformed by "simpler" end-to-end deep learning methods, see, for example [5].

3 Datasets and Features

We used the Moments in Time Dataset [6] for both training and testing. This dataset consists of 1,000,000 videos, divided into 339 categories.

In order to make video completion more achievable, we trained our neural networks to generate videos from a single category at a time. Specifically, we trained and tested (separate) networks for the “skiing” and “erupting” categories.

For the “skiing ” category, we divided the dataset into 1800 training videos and 81 test videos; for the “erupting” category, we divided the dataset into 2000 training videos and 100 test videos.

To reduce the requirement for computational time and resources, we first downsized all videos, which originally consisted of 90 frames with 256×256 pixels in each frame, to 30 frames with 64×64 pixels in each frame. The pixel values were rescaled to lie between 0 and 1. All videos were in RGB format.

4 Baseline Model

Our baseline model was a simple linear interpolation between the initial image and the final image. We also attempted to use off-the-shelf video interpolation functions (ffmpeg) as an additional baseline model, but these functions generally failed completely when only given two frames of a video.

5 Main Method I: Conditional Variational Autoencoder (CVAE)

Our first approach used a conditional variational autoencoder (CVAE). A variational autoencoder (VAE) encodes some data (in this case a video) as a Gaussian distribution, i.e. a vector of means and standard deviations. A decoder then tries to recover the original data from a random sample from this distribution. An additional KL divergence term is included in the loss function to encourage the encoded Gaussian distribution to be as close as possible to the unit Gaussian. To generate data, one simple inputs random noise, sampled from a unit Gaussian, into the decoder.

In a CVAE, the encoder and decoder are conditioned on some shared additional data, in this case the initial and final images. Again, to generate videos from an initial and final image, one simply inputs the images, together with a random noise vector, into the decoder.

In our case, the video encoder was a three-dimensional convolutional neural network followed two fully-connected layers. The video decoder/generator took as inputs the random noise, together with the initial and final frame, encoded using a convolutional neural network. These inputs were concatenated and a fully-connected layer, followed by a series of three-dimensional deconvolutional layers were applied, in order to produce an output video. The details for the basic CVAE architecture are presented in Table 1.

The training objective function was given by

$$\min \mathbb{E}_{X \sim p(X)} \{ D_{KL}[\mathcal{N}(\mu(X), \Sigma(X) || \mathcal{N}(0, I)] + \lambda_1 ||X - G||_2^2 + \lambda_2 (||x_i - G_i||_2^2 + ||x_f - G_f||_2^2) \}, \qquad (1)$$

where X is the training video (with initial and final frames x_i and x_f), μ and Σ are the mean and covariance matrix for the latent space distribution conditioned on input video, G is the generated video (with initial and final frames G_i and G_f), and λ_1 and λ_2 are hyperparameters.

For the “skiing” videos, we used a slightly more advanced architecture, where the full, unencoded initial and final images were added to the decoder network as additional channels at an intermediate deconvolutional layer. At the same layer, we also added an additional channel containing the (rescaled) frame number.

6 Main Method II: Generative Adversarial Network (GAN)

A generative adversarial network (GAN) can be formulated as a minimax adversarial game between a generator (G) and a discriminator (D), where D’s goal is to tell apart real videos from generated ones and G’s goal is to fool D. This game has a unique Nash equilbrium, where the distribution of videos generated by G is the same as the training distribution.

Just like for the CVAE, we made the generator G be conditioned on the initial and final image, which were pre-processed by an image encoder and then used to generate an output video. In this sense, our network formed a conditional GAN (CGAN).

However, unlike in a traditional CGAN, the discriminator D was not given the input images. The discriminator D simply tries to distinguish between real and generated videos without any knowledge about what the initial and final frames should be. Instead, as in [2], we simply include terms in the loss function

Image Encoder

Input:	Image x, $H_0 \times W_0 \times \{1, 3\}$
L1:	Conv2D(x, C=16, K=4, S=2)
L2:	Conv2D(L1, C=32, K=3, S=1)
L3:	Conv2D(L2, C=64, K=4, S=2)
L4:	Conv2D(L3, C=128, K=3, S=1)
L5:	Linear(Flatten(L4), C = 256)
L6:	Linear(L5, C = image encoding dimension = 1000)
Output:	Feature map $E_I(x)$=L6

Video Encoder

Input:	Video X, $30 \times H_0 \times W_0 \times \{1, 3\}$
L1:	Conv3D(X, C=64, K=(3,4,4), S=(1,2,2))
L2:	Conv3D(L1, C=128, K=(3,4,4), S=(1,2,2))
L3:	Conv3D(L2, C=256, K=(3,4,4), S=(1,2,2))
L4:	Conv3D(L3, C=512, K=(3,4,4), S=(1,2,2))
L5:	Concat([Flatten(L4), $E_I(x_i)$, $E_I(x_f)$])
L6:	Linear(L5, C=1024)
L7:	Linear(L6, 2× encoding dimension = 2×600)
Output:	Vectors $\mu(X), \sigma(X)$ = L7, softplus applied for $\sigma(X)$

Video Generator

Input:	Video latent representation z, encoded image vectors $E_I(x_i)$, $E_I(x_f)$
L1:	Concat([z, $E_I(x_i)$, $E_I(x_f)$])
L2:	Flatten(Linear(L1, C = $256 \times 5 \times H_0/8 \times W_0/8$))
L3:	TransposedConv3D(L2, C=128, K=(1,2,2), S=(1,1,1))
L4:	TransposedConv3D(L3, C=64, K=(3,2,2), S=(3,2,2))
L5:	TransposedConv3D(L4, C=32, K=(3,2,2), S=(1,2,2))
L6:	TransposedConv3D(L5, C=16, K=(3,4,4), S=(1,2,2))
L7:	Sigmoid(TransposedConv3D(L6, C=3, K=(2,4,4), S=(2,1,1)))
Output:	Video $G(z)$=L7, $30 \times H_0 \times W_0 \times \{1, 3\}$

Table 1: The details of the CVAE architecture. We have used "same" padding for all convolutional and transposed convolutional layers. This is used for the training in the "erupting" category. For the "skiing" videos, parts of this architecture are altered (see main text). The CGAN architecture is also largely similar to this (also see main text).

for the generator G that are proportional to the L_2 squared distance between the original and generated initial/final frames in order to make these frames be correct in the generated videos.

The objective function is therefore given by

$$\begin{aligned}\min_{w_G} \max_{w_D} & \mathbb{E}_{X\sim p_X(X)}[\log D(X; w_D)] + \\ & \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z; w_G); w_D))] + \mathbb{E}_{X\sim p_X(X)}[\lambda(||x_i - G_i(z; w_G)||_2^2 + ||x_f - G_f(z; w_G)||_2^2)], \quad (2)\end{aligned}$$

where w_G, w_D are weights for G and D, X is the training video (with initial and final frames x_i and x_f), z is a latent "code" sampled from a Gaussian distribution with zero mean and unit variance, G is the generated video (with initial and final frames G_i and G_f), and λ is a hyperparameter.

The architecture that we used for the CGAN was very similar to the architecture used for the CVAE. The discriminator D used the same architecture as the CVAE video encoder, except with the encoded images $E_I(x_i), E_I(x_f)$ removed from the fully connected layer, and the output Gaussian distribution replaced by a single output logit. The generator G used the same image encoder and video generator as in CVAE, except

L_2 distance	Interpolation	CVAE	CGAN
Erupting	62 $\pm$58	180 $\pm$110	223 $\pm$96
Skiing	153 $\pm$89	197 $\pm$80	N/A

Table 2: As a simple and somewhat naive performance evaluation metric, we calculated the L_2 distance between the generated and real videos, averaged over 100 test videos for “erupting”, 81 test videos for “skiing”. The L_2 distances were calculated with pixel values rescaled to lie between 0 and 1. The numbers in this table have been divided by 100.

that we removed the last 2D convolution layer, and also for simplicity we reduced the number of channels in some of the layers. For the convolutional layers in the generator, we used leaky ReLUs rather than ReLUs, as in [4].

GANs are notoriously difficult to train. One of the main problems we ran into was an instability where during training the discriminator's performance became so good that the generator G never managed to improve significantly. This was true even though we used a non-saturating loss function for G, as was proposed in [7] and has since become common practice. To address this issue, we adopted the “one-sided label smoothing” method as in [8], where the labels for the real videos were changed from 1 to a number slightly less than 1, such as 0.9. This enabled us to train our CGAN on a small set of “erupting” training videos; we report our results in the next section.

7 Experiments and Results

All of our code (which is available at https://github.com/geoff-penington/gifify/) is based on and adapted from a github repository of tensorflow generative models [9]. The training and testing of the models were performed on Google Cloud Platform.

We trained our CVAE model on 2000 “erupting” videos, using the architecture in Table 1. For the training objective, we chose hyperparameters $\lambda_1 = \lambda_2 = 2.0$ (see Eqn. 1). We performed mini-batch gradient descent with 20 videos per batch, using Adam optimizer with $\beta_1 = 0.8$ and learning rate 0.0001. The total loss function converged reasonably well in 100 epochs.

We also trained the more advanced CVAE model, discussed in Section 5, on 1800 “skiing” videos. The hyperparameters were otherwise unchanged. Compared to the baseline model, as measured by L_2 distance, the performance of this model was considerably better than the original CVAE. However, we did not have time to train the more advanced model on the “erupting” videos.

We also trained our CGAN model on 120 “erupting” videos. Since CGAN training takes thousands of epochs for the generator to make meaningful improvement, and we were unable to train on larger datasets due to computational constraints. We used the “one-sided label smoothing” technique [8] to prevent the discriminator from getting too good which would lead to generator failing to make any significant improvements. For the objective function, we chose hyperparameters $\lambda = 0.02$. We updated the discriminator D 4 times for every update on generator G, using the same learning rate 0.0002 for both D and G. Due to time constraint, we stopped training at 1000 epochs, even though the GAN was still far from Nash equilibrium.

The performance of the models was first examined by looking at the generated videos on both the training and the testing datasets. The CVAEs trained on “erupting” and “skiing” achieved some plausible success. The CGAN trained on “erupting” videos managed to make some significant progress on generating videos that are in the training set, however, the testing set performance was very poor. This is likely because, due to time and GPU resource limits, and the large number of epochs for GAN to train well, we were only able to train on 120 videos for 1000 epochs.

One simple quantitative metric for performance is the L_2 distance between generated and real videos. The mean and standard deviations of the L_2 distance measured on a small test set, for the interpolation method, CVAE and CGAN, are presented in Table 2. In this metric, interpolation method appears to give best performance. However, a qualitative examination of the generated videos makes it clear that the interpolation method completely fails to capture any interesting dynamics and instead outperforms our main methods, by this simple metric, simply by perfectly reproducing the initial and final frame images. We therefore do not think that the L_2 distance metric should be treated as definitive. However, the results using this metric are consistent with our qualitative and subjective evaluations that the CVAE appears to significantly outperform the CGAN.

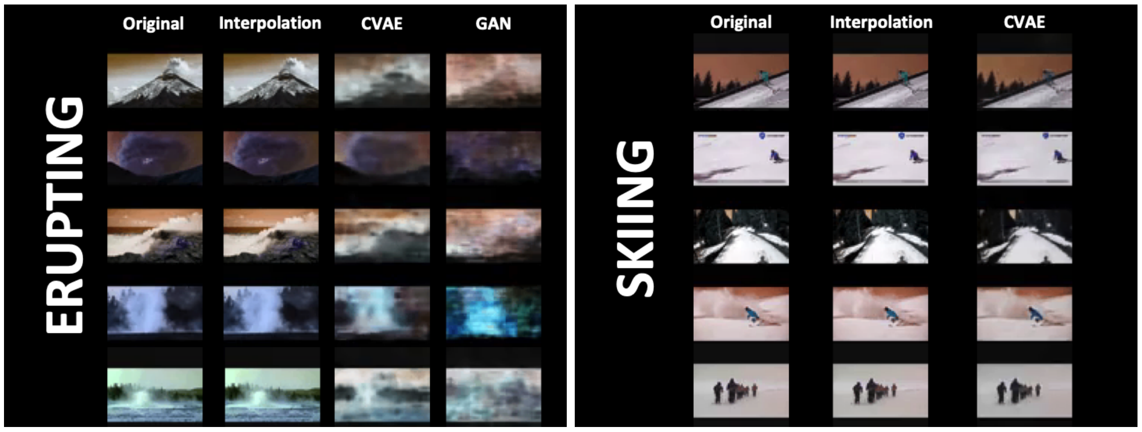

Figure 1: Examples of videos generated given an initial frame and a final frame. Generated videos from interpolation (baseline), a trained CVAE, and a trained GAN are presented. Videos can be viewed at `http://stanford.edu/~maehwee/documents/poster_demo.pptx` or `poster_demo.pptx` in our github repository at `https://github.com/geoff-penington/gifify`.

8 Conclusions and Future Work

Using CVAEs, we were able to produce output videos were approximately based on the initial and final images and which, in certain cases, appeared to have some realistic dynamics. However, at least by the crude metric of L_2 distance, our techniques were outperformed by a simple baseline of a linear interpolation between the initial and final frame.

It seems hopeful that performance could be considerably improved by further tuning of hyperparameters. In particular, the CVAE version used on the skiing videos was only run on one set of hyperparameters for the "skiing" videos and was not tested at all on the "erupting" videos. For CGANs, with more computational power one would be able to train the model on a much larger training dataset and for many more epochs.

Finally, we used only convolutional neural networks to generate videos, which meant that the video length was always fixed. It may be interesting to explore incorporating a recurrent neural network (RNN) to be more flexible about video length and be better at unveiling the temporal correlations. Given the recent rapid development in this field, there are a lot more remaining to be explored.

9 Contributions

All members of the group made significant contributions to the brainstorming, code development, training and testing, analysis and visualization tasks for this project. All of our code ((which is available at https://github.com/geoff-penington/gifify/) is based on and adapted from a github repository of tensorflow generative models [9]. We acknowledge generous support from the Google Cloud Platform.

References

[1] Haoye Cai, Chunyan Bai, Yu-Wing Tai, and Chi-Keung Tang. Deep video generation, prediction and completion of human action sequences. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 366–382, 2018.

[2] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pages 613–621, 2016.

[3] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision*, pages 835–851. Springer, 2016.

[4] Yunpeng Li, Dominik Roblek, and Marco Tagliasacchi. From here to there: Video inbetweening using direct 3d convolutions. *arXiv preprint arXiv:1905.10240*, 2019.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[6] Mathew Monfort, Alex Andonian, Bolei Zhou, Kandan Ramakrishnan, Sarah Adel Bargal, Yan Yan, Lisa Brown, Quanfu Fan, Dan Gutfreund, Carl Vondrick, et al. Moments in time dataset: one million videos for event understanding. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[8] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[9] Tensorflow generative model collections. `https://github.com/hwalsuklee/tensorflow-generative-model-collections/`.