Natural Video Synthesis with Generative Adversarial Networks

by

Nicholas R. Egan

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author .....
Department of Electrical Engineering and Computer Science
May 24, 2019

Certified by .....
Antonio Torralba
Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Natural Video Synthesis with Generative Adversarial Networks

by

Nicholas R. Egan

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Generative Adversarial Networks (GANs) are the state of the art neural network models for image generation, but the use of GANs for video generation is still largely unexplored. This thesis introduces new GAN based video generation methods by proposing the technique of model inflation and the segmentation-to-video task. The model inflation technique converts image generative models into video generative models, and experiments show that model inflation improves training speed, training stability, and output video quality. The segmentation-to-video task is that of turning an input image segmentation mask into an output video matching that segmentation. A GAN model was created to perform this task, and its usefulness as a creative tool was demonstrated.

Thesis Supervisor: Antonio Torralba
Title: Professor

Acknowledgments

I would like to thank Hang Zhao and Jun-Yan Zhu for their mentorship and guidance with this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Generative Adversarial Networks (GANs) [12] are a powerful class of generative neural network models, and recent literature [2, 31] has shown that they can be used to generate images of unprecedented quality. Applying GANs to the task of video generation has achieved decent results [5, 23, 27, 19, 33], but still lacks far behind the state of the art in image generation. For my Masters of Engineering thesis research, I worked on pushing forward the use of GANs for video generation. My two contributions are improving video GAN training through model inflation, and applying GANs to the task of generating videos from image segmentation masks.

We define model inflation for GANs as pretraining an image GAN on frames of a video dataset, turning that image GAN into a video GAN, and then finetuning the video GAN on videos from the same dataset. Training a video GAN from scratch is difficult [5], so by breaking up the training procedure in this way we hope to make training faster and more stable. And since the GAN first learns to create realistic frames and then learns to make them move, we expect the videos generated at the end of this procedure to be of higher quality.

The segmentation-to-video task is that of making a model that can map between an input image segmentation mask and an output video, such that the frames of the output video match the input segmentation. A model that can perform this task well is a useful creative tool, as it would allow anybody to turn a hand-drawn sketch into a realistic looking video. We used a GAN to solve this task, which, to our knowledge,

is the first example of an image-to-video GAN in the research literature.

In Chapter 2, an overview of the advancements of GAN research is provided. We start with an overview of the GAN framework purely from the standpoint of a generative model, and then discuss how GANs have been applied to generative tasks for images and videos in conditional and unconditional settings. In Chapter 3, I describe in detail how model inflation and segmentation-to-video work. I also discuss the specifics of the video GAN architectures tested in the experiments. In Chapter 4, the dataset preparation and experimental procedures used to test our models on inflation and segmentation-to-video are laid out. Results and analysis from these experiments are then provided. In Chapter 5, I discuss the impact of this research and provide a roadmap for future work.

Chapter 2

Background

2.1 Generative Adversarial Networks

Generative Adversarial Networks [12] are a class of generative neural network models consisting of a generator and a discriminator. Given a data distribution p_r (the "real" data), the generator G is tasked with generating "fake" data resembling the real data by mapping input noise $z \sim p_z$ through the network G, where p_z is usually a unit Gaussian. Notice that G implicitly defines a model distribution p_g. The discriminator network D is tasked with the classification problem of determining if a given data sample is real or fake. In the original GAN paper, the optimization problem is framed as a minimax game with the following logistic value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_r} \log D(x) + \mathbb{E}_{z \sim p_z} \log(1 - D(G(z)))$$

The training procedure for this system consists of alternating stochastic gradient descent updates to the discriminator and generator. The discriminator's update is calculated from the gradient of the empirical objective $\hat{J}$ with respect to the model's parameters, where $\hat{J}$ is defined as:

$$\hat{J}(D) = \frac{1}{n} \sum_{i=1}^{n} \log D(x_i) + \frac{1}{n} \sum_{i=1}^{n} \log(1 - D(G(z_i)))$$

Each x_i is a sample from p_r and each z_i is a sample from p_z. The generator's update is calculated from the negative gradient of the empirical loss $\hat{L}$ with respect to the generator network's parameters, where $\hat{L}$ is defined as:

$$\hat{L}(G) = \frac{1}{n}\sum_{i=1}^{n}\log(1 - D(G(z)))$$

Notice that at each gradient update for the generator, it uses "feedback" from the discriminator by backpropagating from its weights and vice versa. Also, these gradient descent updates are generally calculated for mini-batches of the dataset as opposed to the whole dataset at once.

2.1.1 Convergence

In a non-parametric setting with infinite capacity and training time, the generator will eventually converge to a global optimum of $p_g = p_r$ if the discriminator is allowed to converge to its optimum at every training step. The original GAN paper proves this by showing that for a fixed generator, the optimal discriminator is:

$$D^*_G(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

which is the target function of our logistic loss. Given that the discriminator is at its optimum, the GAN value function can be shown to be:

$$V(D^*, G) = -\log(4) + 2JSD(p_r, p_g)$$

Where JSD denotes the Jensen-Shannon divergence. Since the Jensen-Shannon divergence has a unique minimum at $p_g = p_r$, the optimal GAN is one that perfectly replicates the underlying real data distribution. And because our value function is convex over the parameters of G given a discriminator fixed at D^*, we are guaranteed to reach the optimal discriminator with sufficiently small gradient descent updates in this setting. In practice however, GANs as neural network models are notoriously dif-

ficult to train due to instability between the alternating updates to the discriminator and generator, which is the focus of many recent developments in GAN research.

2.1.2 Alternative Loss Functions

Square Loss

Since the seminal GAN paper published in 2014, several variations on this GAN objective have been proposed. The Least-Squares GAN (LSGAN) [35] proposes a squared loss based value function, defining the discriminator loss as:

$$L_{LS}(D) = \frac{1}{2}\mathbb{E}_{x\sim p_r}(D(x) - b)^2 + \frac{1}{2}\mathbb{E}_{z\sim p_z}(D(G(z)) - a)^2$$

where a and b are the labels denoting fake and real data. The generator loss is defined as:

$$L_{LS}(G) = \frac{1}{2}\mathbb{E}_{z\sim p_z}(D(G(z)) - c)^2$$

where c is a label for fake data that doesn't necessarily equal a. It can be shown that when $(a, b, c) = (-1, 1, 0)$, our value function equals $\frac{1}{2}\chi^2(p_r + p_g\big\|2p_g)$ where χ^2 denotes the Pearson Chi-Squared divergence.

Hinge Loss

The Geometric GAN (GeoGAN) [14] proposes a hinge loss based value function. where the discriminator loss is defined as:

$$L_{hinge}(D) = \mathbb{E}_{x\sim p_r}|1 - D(x)|_+ + \mathbb{E}_{z\sim p_z}|1 + D(G(z))|_+$$

with $|x|_+ = max(0, x)$. The generator loss is defined as:

$$L_{hinge}(G) = \mathbb{E}_{z\sim p_z}D(G(z))$$

Wasserstein Loss

The Wasserstein GAN (WGAN) [21] defines a deceptively simple looking loss function of

$$L_W(D, G) = \mathbb{E}_{x \sim p_r} D(x) - \mathbb{E}_{x \sim p_g} D(x)$$

which can be shown through Kantorovich-Rubinstein duality to be equivalent to the Earth-Mover or Wasserstein-1 distance

$$W(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y) \sim \gamma} \big|\big| x - y \big|\big|$$

provided our discriminator is 1-Lipschitz. In the above equation, $\Pi(p_r, p_g)$ refers to the set of all joint distributions with marginals p_r and p_g. In order to enforce this Lipschitz constraint, the original WGAN paper clips the weights of the discriminator.

The WGAN Gradient Penalty (WGAN-GP) [13] takes the WGAN formulation and replaces weight clipping with a regularization term on the gradient of the discriminator, resulting in a loss function of:

$$L_{W-GP}(D, G) = \mathbb{E}_{x \sim p_g} D(x) - \mathbb{E}_{x \sim p_r} D(x) + \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}} \left(\big|\big| \nabla_{\hat{x}} D(\hat{x}) \big|\big|_2 - 1 \right)^2$$

where $p_{\hat{x}}$ is an implicitly defined distribution formed by sampling uniformly across lines connecting points sampled from p_r and p_g. The WGAN-GP loss formulation helps significantly to stabilize GAN training in practice.

Stability Loss

The theoretical analysis of GAN loss functions has usually included the assumption that the data and generator distributions are absolutely continuous. It was recently shown [20] that this rather unrealistic assumption is actually necessary for proving that GAN training can converge for the original GAN loss formulation and others. The authors suggest a regularization term on the original GAN logistic discriminator loss:

$$R(D) = \frac{\gamma}{2}\mathbb{E}_{x\sim p_r}\left|\left|\nabla_x D(x)\right|\right|^2$$

and show that this regularization term allows for convergence. Additionally, they performed extensive experiments to demonstrate that this regularized loss is helpful in stabilizing training. We refer to this regularized logistic loss as "stability loss."

2.1.3 Conditional GANs

So far our discussion of GANs has been in an unconditional setting, where our generator model samples from p_g based on noise. Some of the most exciting applications of GANs however have been in conditional GANs, where the generator instead samples from the conditional distribution $p_{g|y}$ by taking in both a noise vector z and some other input y. We can think of x and y as being paired elements of two sets, where the generator is tasked with learning the mapping from one to the other. The task of the discriminator is thus to determine if an input x is real or fake when paired with y, estimating $p_{r|y}(x|y)$. The objective of a conditional GAN with the original logistic loss is:

$$\min_G \max_D V_c(D,G) = \mathbb{E}_{(x,y)\sim p_{r,y}} \log D(x,y) + \mathbb{E}_{z\sim p_z, y\sim p_y} \log(1 - D(G(y,z),y))$$

2.2 GANs for Image Generation

While the GAN framework can, in theory, be used to model any continuous data distribution, they have been primarily applied to the task of image generation. Unconditional image GANs are tasked with taking input noise and generating images matching those found in a given dataset. Given a large dataset of, say cat photos, an unconditional GAN can be trained to implicitly learn what cats look like and be able to provide a method for sampling cat photos from its learned cat distribution. Conditional image GANs, on the other hand, generate images given both input noise

and some other input. The most common form of a conditional image GAN is an image-to-image GAN, which learns to map one kind of image to another. For example, it could learn to map black and white images to color images, day photos to night photos, or semantic segmentation masks to images.

2.2.1 Unconditional Image GANs

Direct Convolutional GAN

The first popularized and effective GAN architecture for image generation is the Deep Convolutional GAN (DCGAN) [1], which solely uses strided convolutions for the discriminator to downsample an image into single unit output representing a classification prediction, and fractionally strided convolutions for the generator to upsample the input latent vector into an output image. The architectures of the DCGAN Generator and Discriminator are mirrors of each other, with the same layer sizes going between input and image.

Progressive Growing GAN

While the DCGAN can effectively generate images up to size 64×64, training becomes highly unstable for images of higher resolutions. To get around this, the Progressive Growing GAN (ProGan, PGAN, or PGGAN) [31] starts by generating images of size 4×4, and progressively adds new layers once it learns to generate images in its current resolution. When new layers are added, they are faded in smoothly by adding the output of the new convolution (for the discriminator) or transposed convolution (for the generator) to a downsampled or upsampled version of the previous layer's output, where each of these outputs are weighed in such a way that gradually moves towards solely using the new layer. This method allows highly photorealistic images to be generated of sizes up to 1024×1024.

Self-Attention GAN

A recently developed alternative to the DCGAN or DCGAN based architectures like the PGGAN is the Self-Attention GAN (SAGAN) [10]. The SAGAN is composed of two components: residual blocks and self-attention blocks. The residual block is inspired by Resnet [18], a popular CNN architecture originally developed for computer vision tasks like image classification and object detection. The key idea behind Resnet is that very deep networks are harder to train because of vanishing/exploding gradients and degradation, so it may be better to think of each component of a network as performing simple modifications that are added to the input via "skip-connection." If the input to a residual block is x, then the output is $f(x) + x$, where f is a combination of operations like convolution and non-linearities. In the case of SAGAN, the skip connection also needs to include an upsampling or downsampling operation.

The self-attention blocks apply the idea of non-local operations to GANs. Non-local operations [34] allow the output at a given unit to be the weighted sum of an operation applied to every unit in the previous layer, where in this case the weighting between unit i in the previous layer and unit j in the next layer is computed as

$$\frac{\exp(x_i^T W_f^T W_g x_j)}{\sum_{i=1}^{N} \exp(x_i^T W_f^T W_g x_j)}$$

where x is the output of the previous layer and W_f, W_G are learned parameters. The idea of non-local operations comes from the popular idea of self-attention [3], where models learn which parts of the input are important to "attend" to.

The SAGAN architecture uses several residual blocks with one self-attention block in the middle. More recently, the BigGAN paper [2] showed that they were able to scale up the performance of this architecture by using very large amounts of compute power to achieve state of the art image generation results. And while it doesn't quite match the performance of BigGAN, the Stability GAN [20] achieves high quality results by using residual blocks without self-attention. In later sections, we will refer to the architecture of the Stability GAN as "Resnet."

2.2.2 Image-to-Image Models

pix2pix

The first major image to image GAN model is pix2pix [26], which uses a dataset of paired images, such as the same scene at night and during the day, or a picture of a handbag and a picture of its outline. The pix2pix model uses the conditional GAN logistic loss with an added L_1 distance term for the generator loss:

$$L_{L1}(G) = \mathbb{E}_{(x,y)\sim p_{r,y},z\sim p_z}||x - G(y,z)||_1$$

The L_1 loss incentivizes the generator to generate output images that have a similar layout to the input images, capturing low frequency correctness. The discriminator therefore only needs to focus on ensuring the output images have the correct texture, capturing high frequency correctness. The authors of pix2pix thus use a discriminator they call PatchGAN with a DCGAN-like architecture to classify small square patches of the output image as real or fake.

The generator architecture, called U-net [24], is structured like an autoencoder: it downsamples the input image then upsamples it to create the output image. In order to better incentivize the generator to use low level features from the input image, the U-net also includes Resnet-like skip connections between the layers across the network of equal size. While an image to image generator would ideally take in both an input image and a noise vector and produce different output images when different noise vectors are used, the authors of pix2pix noticed that the generator learned to simply ignore the noise vector, so they removed the noise vector from the generator model.

SPADE

SPADE [29] represents the current state of the art in image to image GAN models. The core idea behind SPADE is that by using autoencoder models like that of pix2pix, we lose semantic information as the input image is processed by normalization layers like InstanceNorm. For example, if we are trying to turn semantic segmentation masks into images and we input a segmentation mask consisting of a single label, then

applying InstanceNorm would result in the output being completely zero. To fix this, they replace the normalization layers in the generator with what they call Spatially-Adaptive (DE)normalization (SPADE) layers, that normalize the input channel-wise and applies a pixel-wise affine transformation to the output. This affine transformation, of multiplying by $\gamma^i_{c,y,x}$ and adding $\beta^i_{c,y,x}$ to pixel (c, y, x) on layer i, is learned from the input image through two convolutional layers.

The generator uses a Resnet-like architecture similar to the ones used by SA-GAN and Stability GAN, but with self-attention blocks removed and InstanceNorm or BatchNorm replaced by SPADE. It takes in a noise vector as input and modulates it with SPADE to produce the output image. The discriminator uses a PatchGAN architecture that uses both the input mask y and the output image x to determine if x is real or fake. In addition to the generator and discriminator, SPADE introduces an optional encoder network that captures the style of an image and outputs a noise vector that can be fed into the generator.

CycleGAN

The CycleGAN [16] model contrasts with pix2pix and SPADE in that it uses a dataset of unpaired images, such as pictures of horses and pictures of zebras that are not in the same pose, orientation, or background. We use two generators, one that turns zebras into horses and one that turns horses into zebras, and a discriminator specifically for horses and zebras. In addition to the original GAN generator loss that measures the ability of the generator to generate images of its corresponding class, the CycleGAN uses the cycle consistency loss, which measures the ability of the horse to zebra generator to take a horse photo and generate a zebra photo that can be taken as input by the zebra to horse generator to generate the original horse photo. While there isn't a clear theoretical backing for why this approach would work, it has been shown to be a very powerful technique in practice.

2.2.3 Evaluation Metrics

It is hard to quantify how good a GAN is at generating images in the conditional or unconditional setting. Qualitative visual inspection of generated images is often the main criteria used when comparing different GAN models. Other times people would perform a Turing test with Amazon Turk workers to decide if a real or fake image looks more realistic.

Unconditional GAN Metrics

In the unconditional setting, two automated quantitative metrics have been widely used: the Inception Score (IS) [32] and the Frechet Inception Distance (FID) [22]. They both are based on the idea that a classification model run on generated images should report similar dataset statistics as the classification model run on the training dataset.

To calculate the Inception Score, we train both a GAN and the Inception v3 network [7] on the ImageNet dataset [25]. The Inception Score for our generator is:

$$IS(G) = \exp(\mathbb{E}_{x \sim p_g} KL(p(y|x) \big|\big| p(y)))$$

where $p(y|x)$ is the output of the Inception network as a conditional class distribution, $p(y) = \int_x p(y|x) p_g(x)$ is the marginal class distribution, and KL denotes Kullback-Leibler divergence. A high Inception Score implies that the network is confident that the image represents a single object class and that there is a high diversity of object classes represented in the generator distribution.

To calculate the Frechet Inception Distance, we extract features from the Inception v3 network trained on the ImageNet dataset, and calculate the mean μ and covariance Σ of these features for the real and fake data. Our FID is:

$$d^2((\mu_r, \Sigma_r), (\mu_g, \Sigma_g)) = ||\mu_r - \mu_g||_2^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_f)^{1/2})$$

This formulation comes from the Frechet or Wasserstein-2 distance between two Gaus-

sians.

While these scores have been shown to correlate with human judgment, they can be exploited [28] by constructing GANs that generate adversarial examples for the Inception network, and thus should not be taken as a source of truth. These metrics, in their original form, also require the GAN to be trained on ImageNet. We can compute similar scores using a pretrained classification network on a different dataset, but this requires the dataset to be labeled. Due to these challenges, we have decided to not use these metrics in our later analysis.

Image-to-Image GAN Metrics

As an image-to-image model is tasked with mapping from one image class to another, one way to measure the performance of the mapping is to see how well the reverse mapping can get us back to the original input image. For instance, if our GAN is tasked with turning semantic segmentation masks into images, we can run a semantic segmentation algorithm on the generated images and measure how similar they are to the input segmentation masks. This agreement can be measured by pixel-wise accuracy or intersection over union (IoU).

2.2.4 Latent Vector Interpretation

One of the interesting uses of GANs is for learning a feature representation as the latent space of the generator. It has been shown that the latent vectors can potentially be used in a similar way to word embeddings, where they follow simple rules of arithmetic. For example, the DCGAN paper [1] showed that if you take the latent vector for a man with glasses and subtract a man without glasses then add a woman without glasses, you may get a woman with glasses. Another interesting use of GANs is for interpolation between images: by interpolating between two latent vectors, the images generated along the way from those latent vectors tend to smoothly transition between the two original images for those latent vectors.

2.3 GANs for Video Generation

A natural extension of applying GANs to the task of generating images is the application of GANs to the task of generating videos. Generating a video is much more challenging than generating images: in addition to creating a realistic looking picture with spatial consistency, you also need to convey realistic dynamics with temporal consistency.

2.3.1 Unconditional Video Generation

VGAN

The first approach to extending GANs to video in the unconditional setting was done here in Professor Torralba's lab, in "Generating Video with Scene Dynamics" [5]. For this project, two datasets were constructed of 32 frame long videos of 64×64 resolution at 25 FPS downloaded from Flickr. The first dataset contains unfiltered, unlabeled videos, and the second dataset uses a model pretrained on the Places2 dataset [4] to create a labeled dataset with four categories: golf course, hospital rooms/babies, beaches, and train stations. These videos were then stabilized by extracting SIFT keypoints [8] and estimating homography with RANSAC, discarding videos when the homography had too large of a re-projection error.

Two video GAN (VGAN) architectures were tested on this dataset. The idea of the "one-stream" architecture is to simply turn the DCGAN's 2D "spatial convolutions" into 3D "spatio-temporal convolutions" in both the discriminator and generator. The "two-stream" architecture takes a more structured approach of enforcing a static background with a moving foreground. The generator consists of learning a 3D DCGAN for the foreground $f(z)$, a 3D DCGAN for the mask $m(z)$, and a 2D DCGAN for the background $b(z)$ where the output of the DCGAN is used in every frame of the video. The final GAN output is then

$$G(z) = m(z) \odot f(z) + (1 - m(z)) \odot b(z)$$

where $\odot$ denotes element-wise multiplication.

The general impression one gets from viewing these generated videos is that they often look somewhat realistic, but have unrealistic appearances or dynamics that give them away as fake. To evaluate the performance of the VGAN, an Amazon Mechanical Turk test was conducted asking which video looked the most realistic between videos from the dataset, the one-stream architecture, the two-stream architecture, and a Variational-Autoencoder used as a baseline. Overall, the VGAN two-stream was preferred over real 18% of the time, VGAN one-stream was preferred over real 16% of the time, and the Autoencoder was preferred over real 3% of the time. Interestingly, the two-stream architecture was preferred over the one-stream architecture only 53% of the time, suggesting that the added complexity of separating the foreground from the background leads to only modest improvement.

An additional task tackled by this project was the task of generating future frames of video from a static image, which is quite hard since the GAN would be attempting to predict the future. In order to do this, they condition their VGAN models on the first frame. The resultant frames are not usually correct but seemed plausible.

TGAN

The Temporal GAN (TGAN) [23] is a different approach to generating videos, consisting of a temporal generator and an image generator. The temporal generator takes a single latent vector as input and outputs a sequence of latent vectors, each of which is concatenated to the original latent vector and fed into the image generator in order to generate the frames of a video. The temporal generator architecture is similar to that of a DCGAN except with 1D deconvolutions denoting convolution in the temporal direction. The image generator uses the standard 2D DCGAN architecture, and the discriminator uses the 3D DCGAN used in VGAN.

One novel application that the TGAN is particularly well suited to is that of video frame interpolation. By creating frame latent vectors by way of linear interpolation in the latent space, frames can be generated that semantically move from one frame to another.

MoCoGAN

With the Motion and Content Decomposed GAN (MoCoGAN) [27], we model videos as consisting of a sample from an underlying "content space" and a series of samples from an underlying "motion space." In this way, we are explicitly disentangling the latent vector into two parts, where one represents what the video is about, and the other represents the video dynamics. The architecture of MoCoGAN is similar to that of TGAN except for the use of a single layer GRU [17] as a recurrent neural network model for the temporal generator. Note that by using a recurrent model as opposed to a convolutional model for generating the frame latent vectors, the generator can theoretically generate videos of any duration, and during training they pick the video duration randomly. The MoCoGAN uses both an image discriminator and a video discriminator in order to ensure that each frame looks realistic in addition to the generated video itself.

FTGAN

The Flow and Texture GAN (FTGAN) [19] has a similar philosophy as MoCoGAN in that the generative process for videos should model dynamics and content separately. The FTGAN takes a different approach to this problem however by using two GANs: the FlowGAN and the TextureGAN, where the FlowGAN creates optical flows and the TextureGAN converts the optical flow into RGB video. An optical flow is represented in this case by the edges of moving objects, so the TextureGAN can be interpreted as a pix2pix model that maps optical flows to video. In this way the FTGAN decomposes the problem of unconditional generation into an unconditional generation of an intermediate representation followed by a conditional generation of the final video. The FlowGAN architecture is similar to the two-stream architecture used in VGAN except for the absence of a background generator, since optical flows have a background of 0 denoting no movement. The TextureGAN also resembles the conditional two-stream architecture of VGAN where the optical flow is taken as input, but with a foreground generator that resembles U-net [24], with downsampling fol-

lowed by upsampling and skip connections between layers of equal size. The authors trained the FlowGAN and TextureGAN separately before joining them together to fine-tune end to end.

2.3.2 Video-to-Video Models

The video-to-video problem is a natural extension of the image-to-image problem to video: we have paired video data, such as video segmentation masks and videos they correspond to, and wish to map from one to the other.

vid2vid

Video-to-Video Synthesis (vid2vid) [33] is the first GAN model to tackle this problem successfully. The vid2vid model makes the Markovian assumption that the current frame we wish to generate depends on the current source frame, the previous L source frames, and the previous L generated frames. The authors use the observation that videos consist largely of reused pixels across time, and thus generate the new frame x_t as

$$x_t = (1 - m_t) \odot w_{t-1}(x_{t-1}) + m_t \odot h_t$$

In this equation, $\odot$ represents element-wise multiplication, w_{t-1} is an estimated optical flow from x_{t-1} to x_t as predicted by an optical flow network, h_t is an image consisting of new pixels, and m_t is an occlusion mask as predicted by a mask network. Pixels where m_t is higher are predicted to be new, indicating that something has emerged in the video that was not visible in the previous frame. Our new pixels h_t are generated by a similar two-stream architecture as VGAN, in that we separately generate a foreground and a background.

Chapter 3

Methods

The goal of this research is to extend and improve upon prior work of applying GANs to video. The first contribution is the idea of model inflation, and the second is the image segmentation to video task.

3.1 Model Inflation

3.1.1 Motivation

GANs have shown incredible success in the task of image generation. On the unconditional side, models like Progressive GAN and Style GAN have made news headlines due to their uncanny ability to create fake people [15]. And on the conditional side, pix2pix and CycleGAN have been shown to be incredibly powerful in applications ranging from super-resolution [6] to artistic tools [11]. It would thus be quite helpful if we could take these image GANs and somehow convert them into video GANs.

At the architecture level, this seems fairly straightforward: we can simply take all the layers in the network and extend them into the time dimension. The convolutional layers, which are the main building blocks of GANs, would use 3D kernels as opposed to 2D kernels, and the other layers can be sized up accordingly. This idea is essentially that of VGAN's 1 stream architecture [5], which is simply DCGAN but with 3D layers. The results of this paper show however that the increased complexity of training our

GAN to generate video results in a degraded frame quality compared to using the DCGAN for image generation. Training this network from scratch may be too much to ask.

Instead, we can view this task as a transfer learning problem. We could pretrain a 2D GAN model to generate images, turn that 2D model into a 3D model, and finetune that model on videos. For our dataset, we use videos to train our 3D model, and we use frames of those videos to train our 2D model. If everything is working properly, the newly formed video generator will start by generating videos consisting of a still image, and the newly formed video discriminator will start by classifying these still videos in the same way it classified the still image. Over time, the discriminator will learn that movement is a factor separating real videos from fake ones, and eventually teach the generator to create realistic looking movements. Overall, we would expect this training procedure to be faster and more stable than training a video GAN from scratch.

3.1.2 Inflation Procedure

The procedure of turning a 2D model into a 3D model involves copying model weights in such a way that the output of layer ℓ in the 3D model is identical to the output of layer ℓ in the 2D model except duplicated in the time dimension, when given an input that is identical except duplicated in the time dimension. In the following sections I describe the two particular layer classes that are used the most in our GANs: convolutional layers and spectrally normalized convolutional layers.

Inflating convolutional Layers

To inflate a convolutional layer, note that the output y of a 2D convolution for an input x with weights w and biases b can be computed as follows: we line up the weight matrix such that it is centered at pixel i, j of x, and then we simply multiply each w value with its corresponding x value. We sum these values up, and add b to get the output y at i, j. A 3D convolution is the same, except our weight matrix is

3D, and this sum is across 3 dimensions. Thus if our weight kernel is of dimensions $H \times W \times T$, we can transfer the 2D weights to the 3D weights with the following formula:

$$w^{3D}_{i,j,t} = \frac{1}{T} w^{2D}_{i,j}$$

The reason we need to divide by T is because the output at every pixel is T times larger due to this summing across a 3rd dimension: the inputs x are the same across this dimension, and there are T duplicates.

If the temporal width T is greater than 1, we cannot use zero padding, because this would result in this would result in the pixels on the starting and ending frames of the video being dimmer since we're adding zero instead of a replication of the input frame. In this case we either need to use replication padding, reflection padding, or not pad at all.

Inflating Spectrally Normalized Layers

Spectral Normalization [30] is a stability technique used in GAN training in which the weights of layer W are regularized to move towards $W_{SN} = W/\sigma$ where σ denotes the largest singular value of W. Instead of directly computing the SVD of W, the spectral normalization algorithm runs a single step of power iteration every gradient update by keeping a running u and v that approximate the first left and right singular vectors respectively. Thus $\sigma \approx u^T W v$. When spectral normalization is used on convolutional layers, the weight matrix W is reshaped to be 2D, with the first dimension having a size equal to the number of input channels.

When we inflate a 2D spectrally normalized convolutional layer W_2 to become a 3D spectrally normalized convolutional layer W_3, we would like to calculate a new u_3 and v_3 such that $\sigma_3 \approx u_3^T W_3 v_3$, where σ_3 is the first singular value of W_3. This would allow for the power iteration to continue where it left off, and avoid a sudden destabilization in the training after inflation. Below is a proof that $u_3 = u_2$ and that v_3 is the result of stacking v_2 vertically T times and dividing by $\sqrt{T}$:

Proof. As explained in the previous subsection, W_3 consists of T copies of W_2, each

divided by T, where T denotes the expansion in the time dimension. In the context of spectral normalization where each matrix is reshaped to be 2 dimensional, we get a W_3 of the form:

$$W_3 = \begin{bmatrix} \frac{1}{T}W & \frac{1}{T}W & \dots \end{bmatrix}$$

To determine the proper value of σ_3, observe that

$$W_3 W_3^T = \sum_{i=1}^{T} \frac{1}{T^2} W_2 W_2^T = \frac{1}{T} W_2 W_2^T$$

For every eigenvalue λ_2 of $W_2 W_2^T$ with eigenvector x_2, it thus must be the case that $W_3 W_3^T$ has a corresponding eigenvalue $\lambda_3 = \frac{1}{T}\lambda_2$ with eigenvector $x_3 = x_2$. The singular values of matrix W are simply the square roots of the eigenvalues of WW^T, which means that $\sigma_3 = \frac{1}{\sqrt{T}}\sigma_2$. The left singular vectors of matrix W are the eigenvectors of WW^T, which means that $u_2 = u_3$.

Now consider the matrix $W_3^T W_3$: we can write it as a $T \times T$ block matrix, where each block is the matrix $\frac{1}{T^2}W_2$. Let x_2 be an eigenvector of $W_2^T W_2$ with eigenvalue λ_2. Let x_3 be defined as the vector formed by stacking T copies of x_2 and dividing by $\frac{1}{\sqrt{T}}$:

$$x_3 = \begin{bmatrix} \frac{1}{\sqrt{T}}x_2 \\ \vdots \end{bmatrix}$$

Observe that:

$$W_3^T W_3 x_3 = \begin{bmatrix} \sum_{i=1}^{T} \frac{1}{T^2} W_2^T W_2 \frac{1}{\sqrt{T}} x_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \frac{1}{T\sqrt{T}} x_2 \\ \vdots \end{bmatrix} = \frac{1}{T} x_3$$

Thus $\lambda_3 = \frac{1}{T}$ is an eigenvalue of $W_3^T W_3$ with eigenvector x_3. Since the right singular vectors of matrix W are the eigenvectors of $W^T W$, we find that $v_3 = \frac{1}{\sqrt{T}} v_2$. $\square$

3.2 Segmentation-to-Video

A semantic segmentation map for a given image contains a class label for every pixel in the image (see figure 3-1 for an example). The task of producing a segmentation for an image is an old one that been attempted with various computer vision techniques over the years, with deep learning based methods representing the state of the art. More recently, the reverse task of producing an image for a given image segmentation has been done by image-to-image GANs like pix2pix and SPADE, and the task of producing a video for a given video segmentation has been done by vid2vid.

Figure 3-1: *Top row:* 4 sample images from the Nature Video dataset. *Bottom row:* Semantic segmentation maps of the sample images, with different colors representing different object classes. Note that this segmentation was performed by an algorithm as opposed to a human, and thus may contain errors.

In this thesis, I am exploring a new problem: given a image segmentation mask, can we design a model to produce a video for it, where each frame of the video has a semantic segmentation given by the input? The input for this model would be an image and the output would be a video, putting this model in the "image-to-video" category. In terms of model complexity, we would expect it to be more complex than image-to-image, but simpler than video-to-video.

The only image-to-video models that I am aware of are those of next frame prediction. The next frame prediction problem asks a model to predict the next frame or frames of a video given an input frame or frames. Generally, these models require multiple frames as input however in order to properly predict motion trajectories. Furthermore, this task of segmentation to video is fundamentally different in that each output frame has a structure matching that of the input, as opposed to having

to worry about how the input structure will warp.

One of the biggest motivations for this model is that it can turned into a useful creative tool. A user could draw a segmentation mask, run it through the generator, and produce an output video. Drawing a single segmentation mask is much easier and more intuitive than having to draw many segmentation masks for different frames. Once this is working, there are further extensions of this project proposed that could also generate sound to accompany this generated video.

In terms of constructing a dataset to train this model on, we need to find videos that have very little changes in the frame segmentation mask. This implies that the videos satisfy the following properties:

1. There is no camera movement over the course of the video
2. The video does not zoom in or out
3. No objects change in how they overlap with each other

While these requirements may seem fairly stringent, there are many video classes that satisfy them all, such as typical videos of beaches, forests, waterfalls, fields, and mountains. All of these videos have the property that the video consists of natural objects like oceans, skies, or grass that change in appearance within themselves without actually changing their placement within the frame: oceans create waves, skies move clouds, and grass rustles in the wind, but none of these grow significantly larger or smaller in the process. This works because our segmentation algorithm groups together waves into an "ocean" category as opposed to labelling individual waves, groups clouds into the "sky" category, and considers each blade of grass a single object of class "grass."

3.3 Architectures

In order to demonstrate the usefulness of our methods, we need GAN architectures for generating video. We chose to use 3D versions of DCGAN, Resnet, and SPADE, with the details described in the following subsections.

3.3.1 DCGAN

As our first model we use the DCGAN architecture [1], in both its original 2D form as well as in its 3D form of the 1-stream VGAN [5]. To improve model performance, we added spectral normalization [30] to the convolutional layers. The 2D and 3D generators are shown in figure 3-2, and the 2D and 3D discriminators are shown in figure 3-3. In these diagrams, convolutional and transposed convolutional layers are denoted as K-Conv$X$D-F and K-Deconv$X$D-F respectively, where K is the kernel dimensions, X is 2 or 3, and F is the number of output filters. "BN" denotes batch normalization.

These models shown here are for images of dimensions 128×128, and videos of dimensions $32 \times 128 \times 128$ (32 frames, each of dimensions 128×128). Upsampling and downsampling of the image comes from fractionally strided convolutions or transposed convolutions. Turning the 2D model into the 3D model is fairly straightforward, as all of the layers are convolutional.

3.3.2 Resnet

To test the idea of weight inflation on an unconditional model, we use the Resnet GAN architecture of SA-GAN [10] and GAN Stability [20]. The 2D and 3D generators are shown in figure 3-4, the 2D and 3D discriminators are shown in figure 3-5, and the 2D and 3D Resblocks used by the generators and discriminators are shown in figure 3-6. K-AvgPool-$\downarrow S$ denotes average pooling with kernel size K and stride S, and Block$X$D-F denotes a 2D or 3D Resblock with F output filters.

From the outside, the Resnet architecture looks fairly similar to that of DCGAN: the generator and discriminator are mirror images of each other in that they use layers of roughly the same size to sample an image up or down. The power of this architecture however comes from the Resblocks being used instead of mere convolutional layers, where each Resblock models a small change being added to the input image. For Resblocks where the number of input and output filters stays the same, the convolutional layer on the skip connection shown in the diagram is removed.

The models shown here are for images of dimensions 128×128, and videos of dimensions $8 \times 128 \times 128$. While it would be ideal to generate longer videos, GPU memory constraints make longer videos prohibitively expensive. To generate longer or shorter videos, one would simply change more Upsample$-1 \times 2 \times 2$ layers into Upsample$-2 \times 2 \times 2$ layers in the generator and more $1 \times 3 \times 3-$AvgPool$- \downarrow 1 \times 2 \times 2$ layers into $2 \times 3 \times 3-$AvgPool$- \downarrow 2 \times 2 \times 2$ layers in the Discriminator.

3.3.3 SPADE

To test weight inflation and develop an architecture for the segmentation to video task, we adapted the SPADE architecture [29] to generate video output for image input. The 2D and 3D generators are shown in figure 3-7, the 2D and 3D discriminators are shown in figure 3-8, the 2D and 3D encoders are shown in figure 3-9, the 2D and 3D generator blocks are shown in figure 3-10, and the 2D and 3D SPADE normalization blocks are shown in figure 3-11. "IN" denotes instance normalization, and "Sync BN" denotes synchronized batch normalization. Although it isn't specifically noted in the diagrams, SPADE uses spectral normalization for convolutional layers.

The SPADE generator is fairly similar in design to that of Resnet, but the SPADE discriminator instead takes its inspiration from the PatchGAN of pix2pix [26]. The idea is that since SPADE uses an L1 loss term to incentivize the overall structure of the input segmentation mask to match the overall structure of the output image, the discriminator simply needs to focus on making sure the finer textures of the output image look realistic. The SPADE model uses two discriminators that operate at different scales of the input. The 2D SPADE model has one discriminator that takes an input image and segmentation of dimensions 128×128 and outputs a tensor of decisions of dimensions 16×16, and a second discriminator with 64×64 input and 8×8 output. The 3D SPADE model has one discriminator with $8 \times 128 \times 128$ input and $8 \times 16 \times 16$ output, and a second discriminator with $4 \times 64 \times 64$ input and $4 \times 8 \times 8$ output.

Turning the image-to-image SPADE model into a image-to-video SPADE model required us to change the layers producing the generator output or processing the

discriminator or encoder input from 2D to 3D. The layers processing the input segmentation mask however remained 2D, since the mask is still an image. In our model, the SPADE normalization layers learn a single pixel-wise affine transformation that is applied to every frame of the video because the semantics of the segmentation stay the same for the full video duration.

The 2D SPADE discriminator expects the channel-wise concatenation of an image and a segmentation. Adapting this model to our 3D case unfortunately required us to make a duplicate of the segmentation mask for each frame so that they can be concatenated.

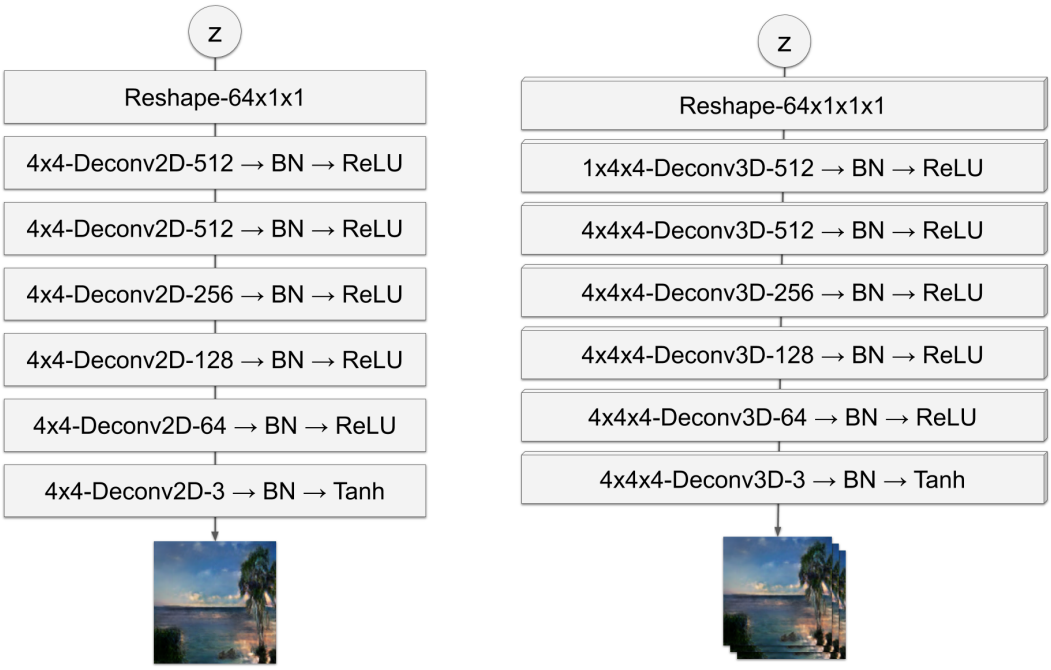

Figure 3-2: *Left:* Architecture of 2D DCGAN Generator. *Right:* Architecture of 3D DCGAN Generator.

Figure 3-3: *Left:* Architecture of 2D DCGAN Discriminator. *Right:* Architecture of 3D DCGAN Discriminator.

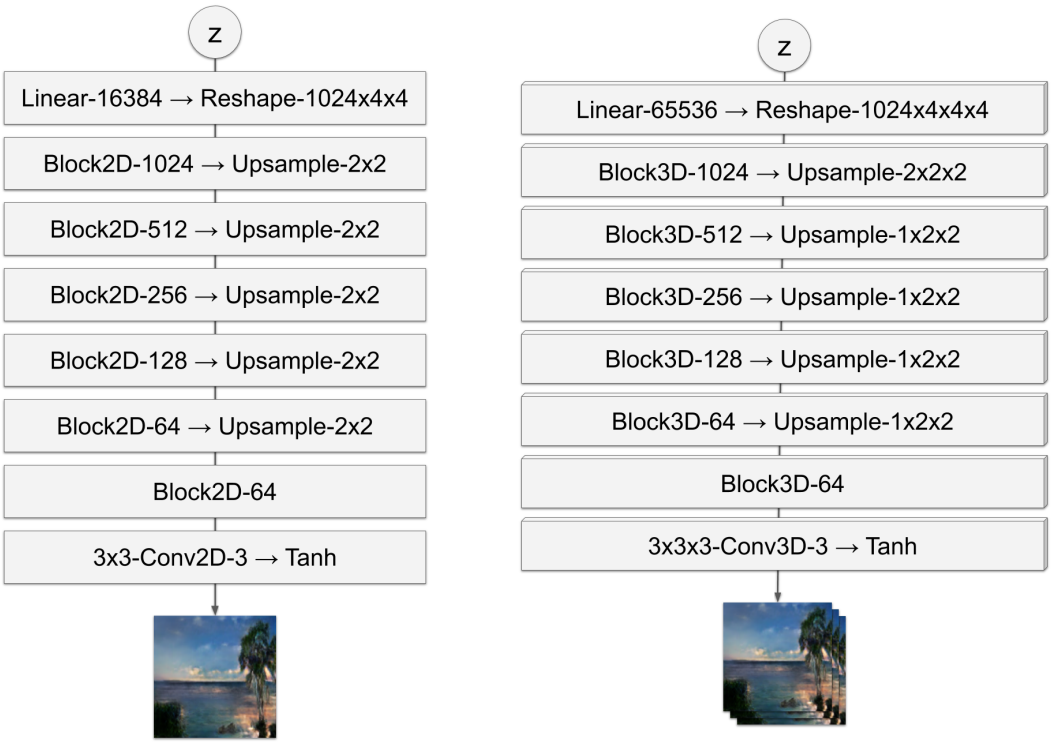

Figure 3-4: *Left:* Architecture of 2D Resnet Generator. *Right:* Architecture of 3D Resnet Generator.

Figure 3-5: *Left:* Architecture of 2D Resnet Discriminator. *Right:* Architecture of 3D Resnet Discriminator.

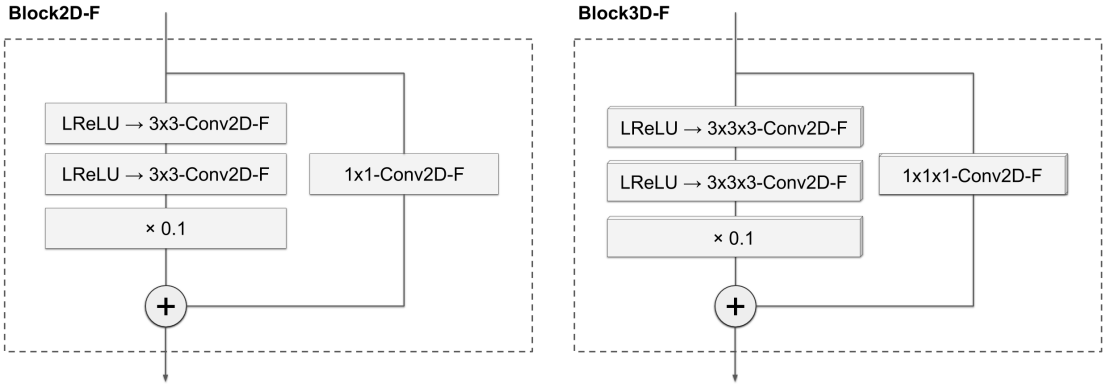

Figure 3-6: *Left:* Architecture of 2D Resnet Block. *Right:* Architecture of 3D Resnet Block.

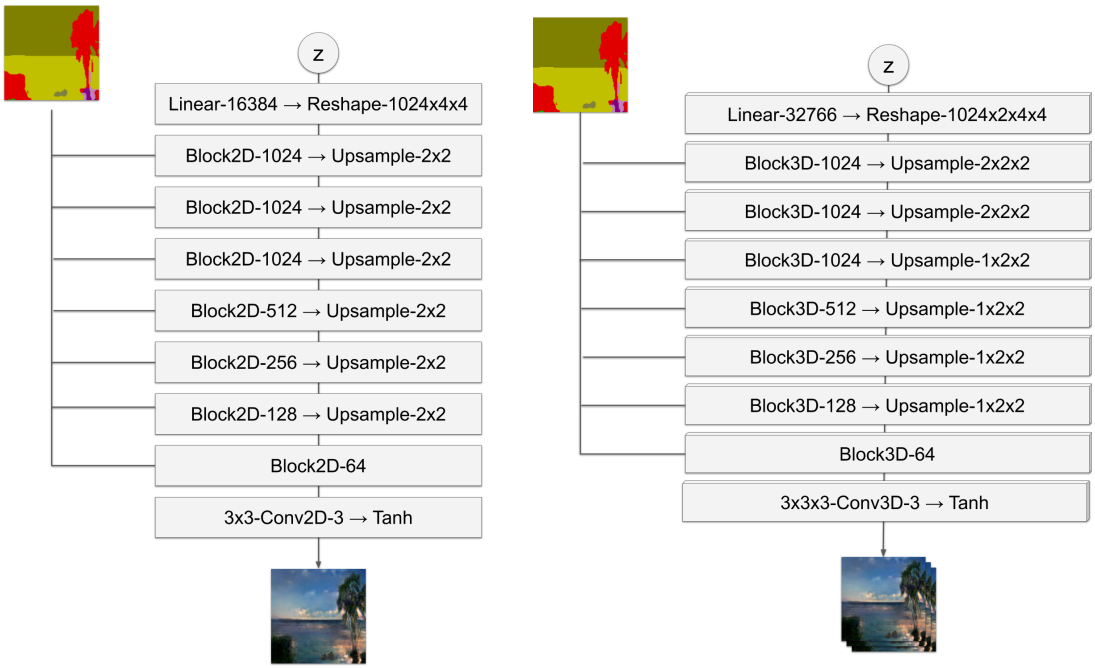

Figure 3-7: *Left:* Architecture of 2D SPADE Generator. *Right:* Architecture of 3D SPADE Generator.

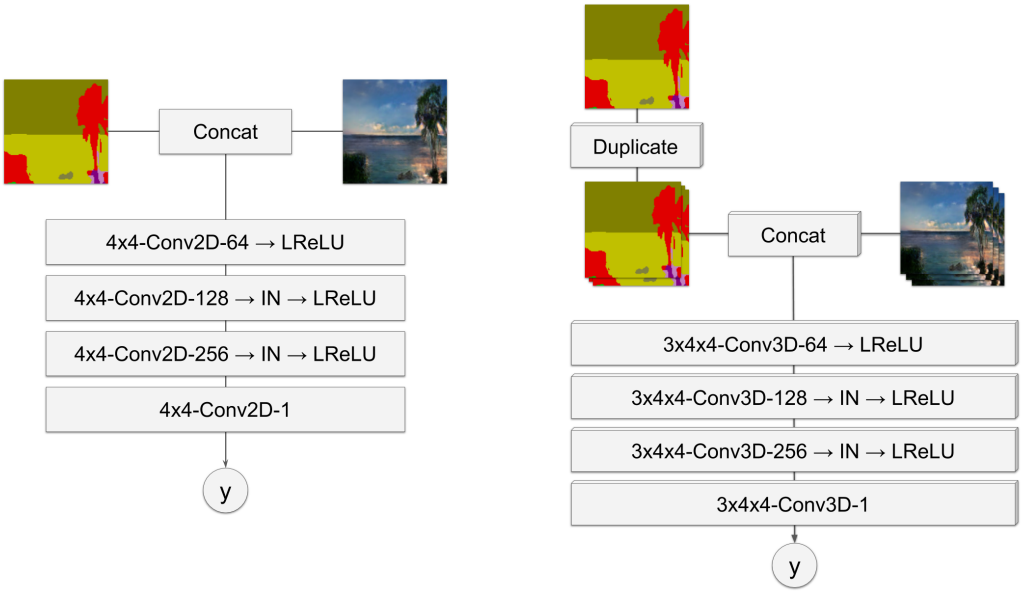

Figure 3-8: *Left:* Architecture of 2D SPADE Discriminator. *Right:* Architecture of 3D SPADE Discriminator.

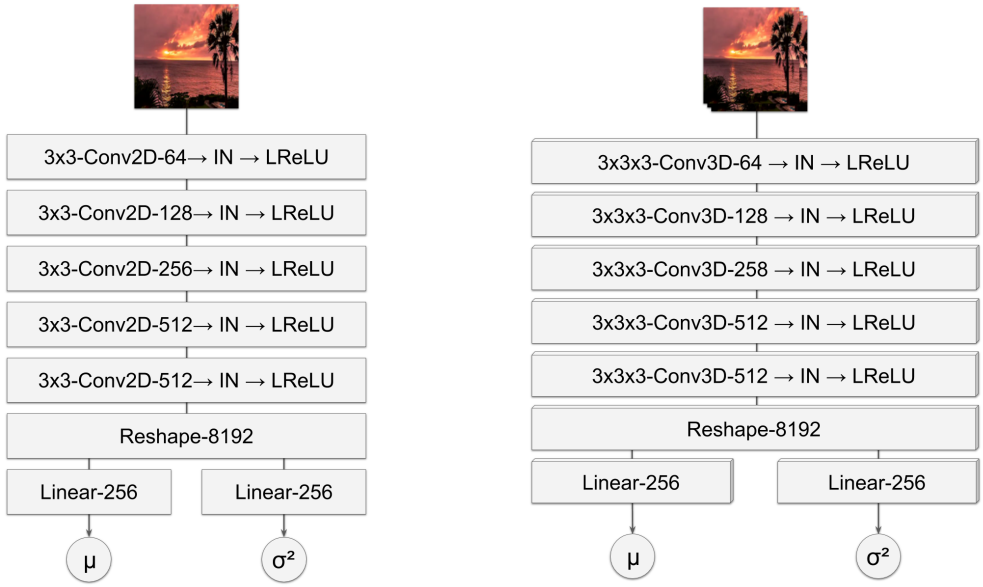

Figure 3-9: *Left:* Architecture of 2D SPADE Encoder. *Right:* Architecture of 3D SPADE Encoder.

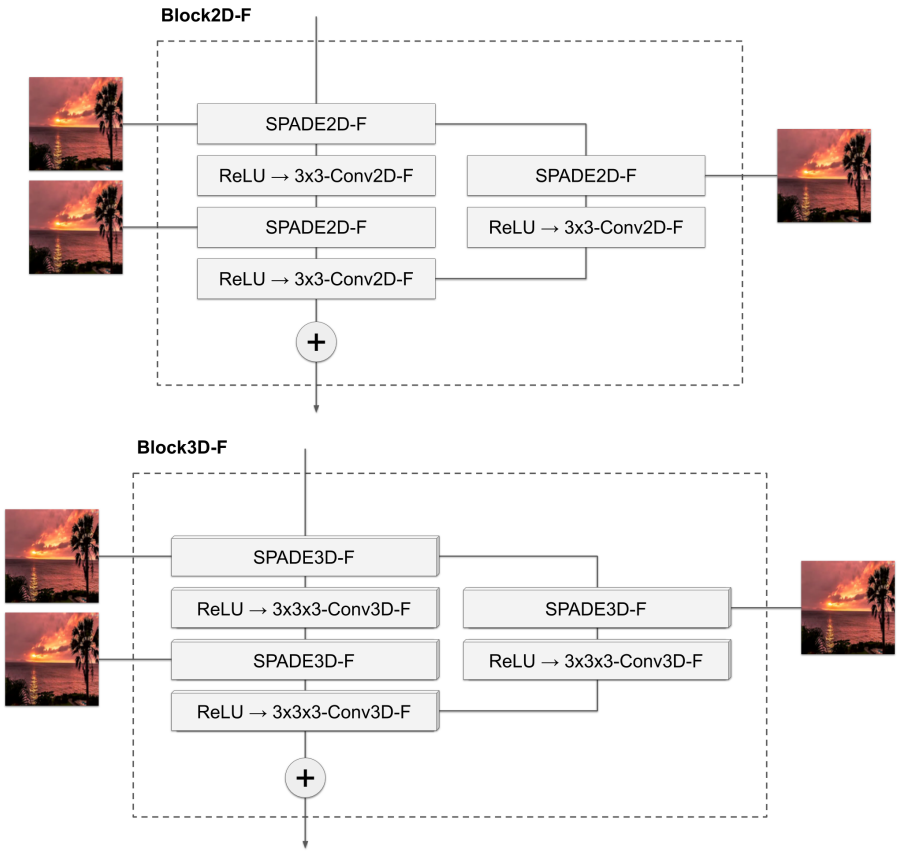

Figure 3-10: *Top:* Architecture of 2D SPADE Generator Block. *Bottom:* Architecture of 3D SPADE Generator Block.

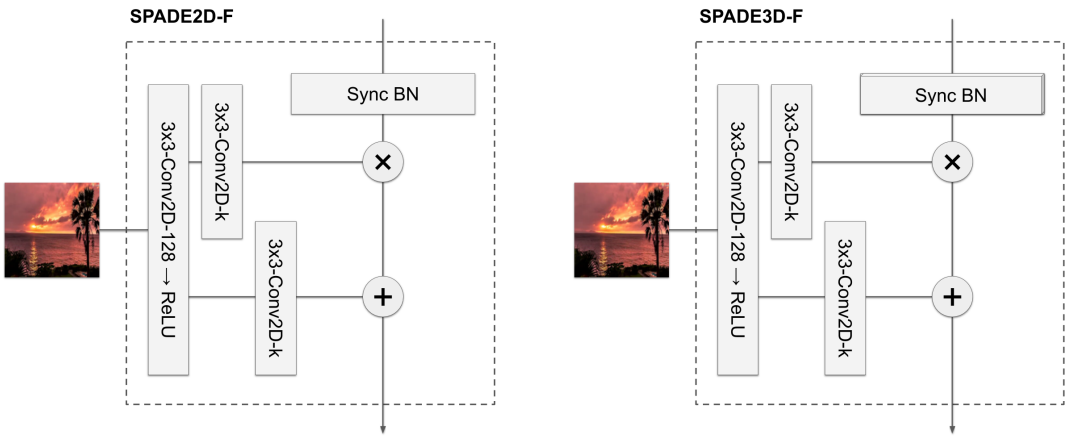

Figure 3-11: *Left:* Architecture of 2D SPADE normalization block. *Right:* Architecture of 3D SPADE normalization block.

Chapter 4

Experiments

In order to test the technique of model inflation and the segmentation image to video task, I sought to answer the following questions:

1. How effective is model inflation in helping video quality, training speed, and training stability?

2. Can we successfully train a model to turn segmentation images into videos with high frame quality and realistic dynamics?

To answer these questions, I designed experiments in which I gathered video datasets, implemented GAN models, trained the models on the datasets, and generated output videos.

4.1 Dataset Preparation

4.1.1 Nature Video

One of the contributions of this work is producing a high quality dataset of nature videos, with sample clips shown in figure 4-2 and sample frames shown in 4-1. Each of these videos depicts a different natural scene, such as a beach, a waterfall, a forest, a field, or a river.

Data Collection

In order to make this dataset useful for the segmentation image to video task, these videos were selected according to the criteria outlined in section 3.2: the camera stays fixed, and movement only occurs within semantic sections of the video. In addition, I made sure to not include any timelapse videos, as they would make the dataset videos inconsistent in their time scale. These criteria filtered out videos with people in them, because people tend to move around, as well as amateur videography, as these videos tend to be shaky. To find suitable videos I searched YouTube for queries like "natural scenes," and found many hour long relaxation videos with titles like "Nature Sounds Beach Relaxation Mindfulness."

In total, 213 high quality videos were acquired. Most of these videos were long and consisted of multiple shots, where each shot was a different scene. We ran a shot detection algorithm on these videos to split them into their individual shots, since different shots would have different segmentation masks, finding 34,413 total shots across our videos. We then sampled a 3 second clip at an FPS of 24 from each of these shots. The dataset was split into 184 training videos consisting of 28,369 training clips, and 29 validation videos consisting of 6044 validation clips. Figure 4-1 shows a large sample of frames from this dataset, and figure 4-2 shows sample video clips.

Segmentation

In order to produce segmentation masks for each clip, we used the Semantic Segmentation Pytorch repository with the Resnet 50 Dilated encoder and the PPM Deepsub decoder, pretrained on the MIT ADE20K dataset [37] [36]. We ensured that the resultant segmentation mask remained roughly the same during the whole duration of a clip, and saved the segmentation of the middle frame as the clip's ground truth label.

The MIT ADE20K dataset uses 150 different semantic classes, and of these only 14 seemed relevant to our dataset. However, 61 classes existed in our dataset, which

seems to be the result of the dataset containing some classes with very low frequency (like the fire and chair labels), as well as the segmentation algorithm making mistakes (like the curtain and rug labels). The top 40 most frequent segmentation class in the dataset are shown in table 4.1.

4.1.2 Synthetic Trains

The usefulness of the Nature Videos dataset comes from the fact that each clip is paired with a segmentation mask image. In the unconditional video generation setting, we decided it would be useful to start with a much simpler video dataset that we could use to debug our networks.

The Synthetic Trains was constructed by taking images from the "train" object category of the LSUN dataset [9], zooming into them, and panning the viewing window horizontally across the image. In total, 100,005 synthetic videos were collected, each consisting of 32 frames of dimensions 128×128. Figure 4-3 shows a large sample of frames from this dataset (which are simply cropped samples from the LSUN trains dataset), and figure 4-4 shows sample video clips.

Table 4.1: Segmentation labels in the Nature Videos dataset

Name	Proportion	Label
sea	36.44%	26
sky	26.13%	2
sand	11.22%	46
rock	7.41%	34
tree	3.92%	4
mountain	3.92%	16
water	3.08%	21
plant	1.59%	17
grass	1.42%	9
river	1.06%	60
tree	0.78%	72
earth	0.58%	13
waterfall	0.57%	113
sofa	0.15%	23
fence	0.14%	32
door	0.12%	14
field	0.12%	29
rug	0.09%	28
chair	0.08%	19
desk	0.07%	33
floor	0.06%	3
lamp	0.06%	36
house	0.06%	25
mirror	0.06%	27
pedestal	0.05%	40
person	0.05%	12
armchair	0.05%	30
chest	0.05%	44
ceiling	0.05%	5
road	0.04%	6
seat	0.04%	31
table	0.04%	15
painting	0.03%	22
column	0.03%	42
curtain	0.03%	18
shelf	0.03%	24
car	0.03%	20
sidewalk	0.03%	11
signboard	0.03%	43
cabinet	0.03%	10

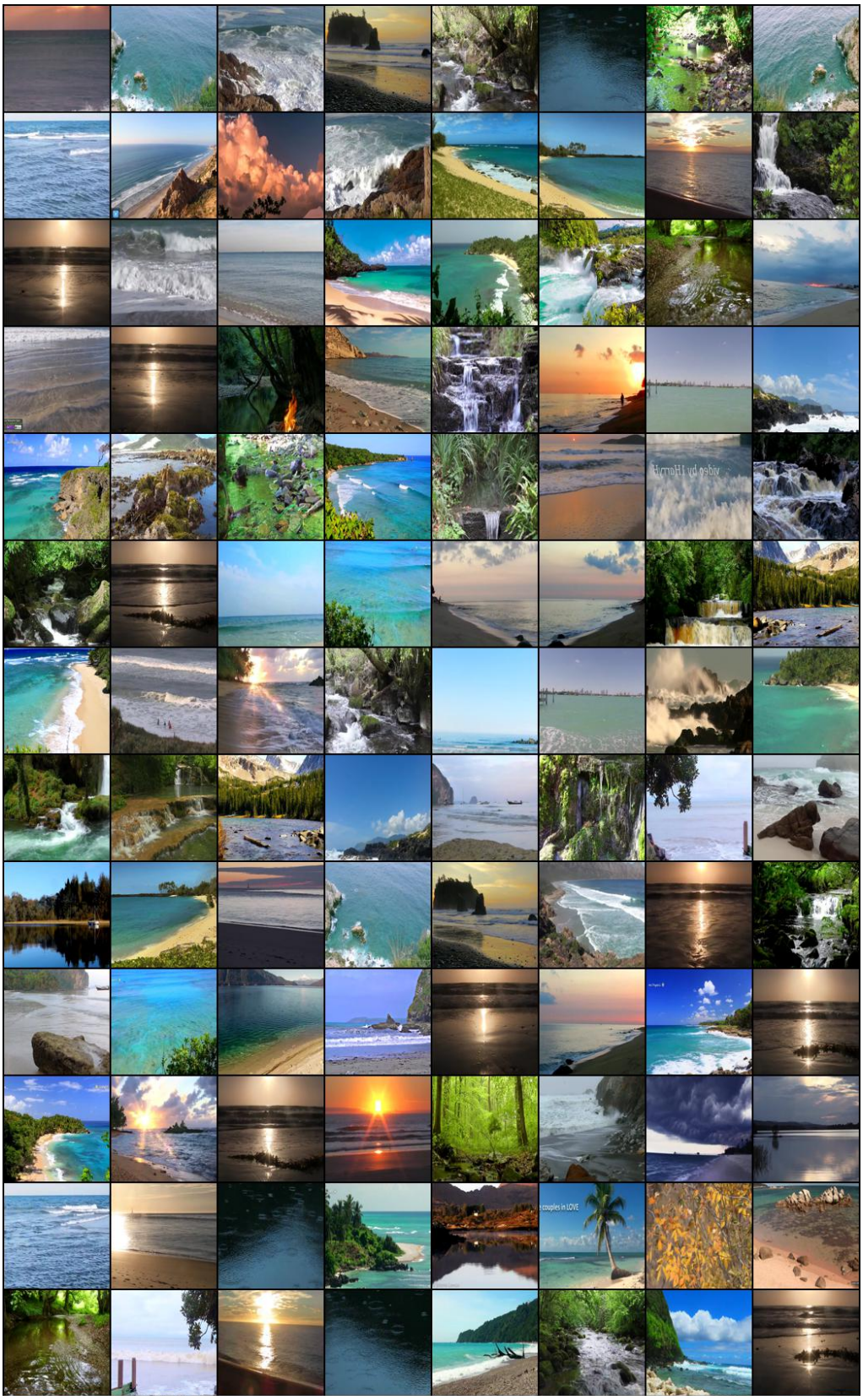

Figure 4-1: Sample frames from the nature videos dataset.

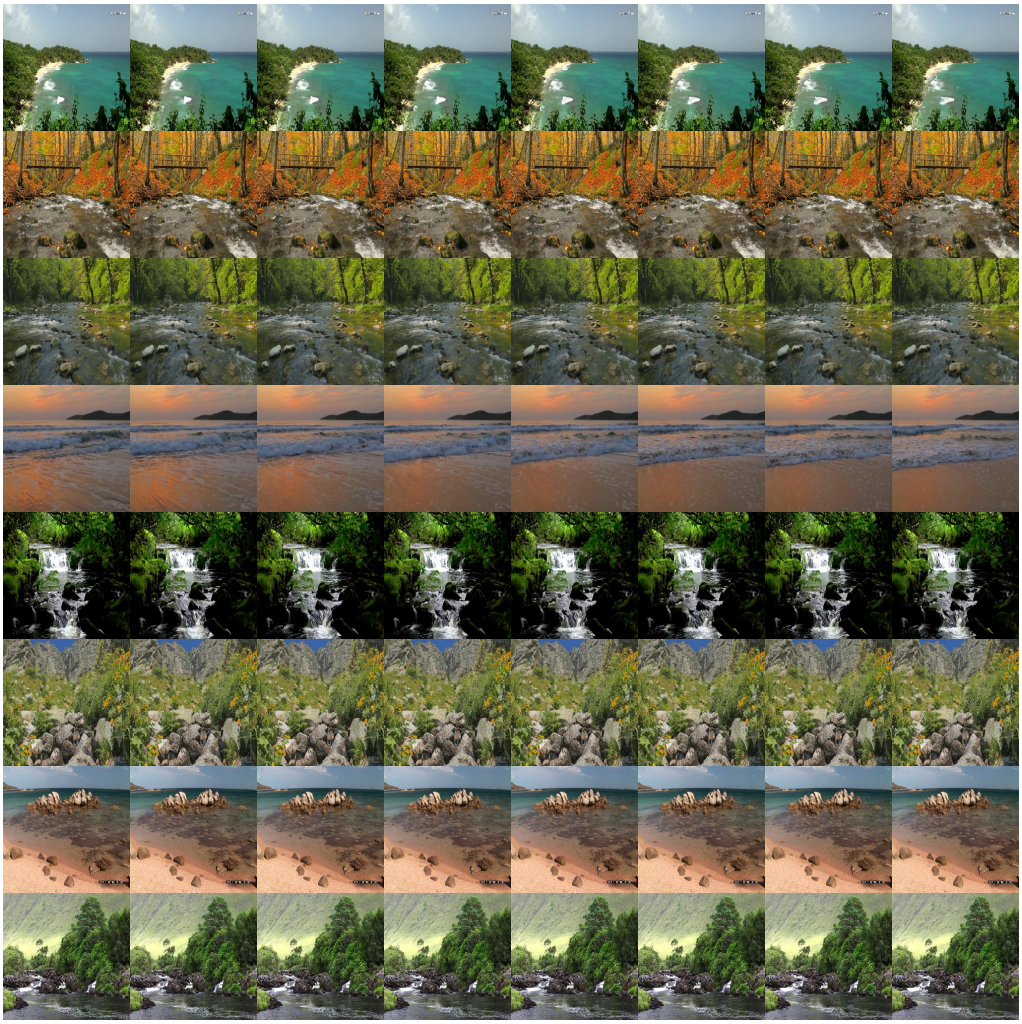

Figure 4-2: Sample clips from the nature videos dataset. Each row shows a different clip, with the columns showing different frames of these videos. The original clips are 72 frames long, but every 9 frames are shown here. This figure can be watched as a video at `http://people.csail.mit.edu/negan/thesis/figures.html`.

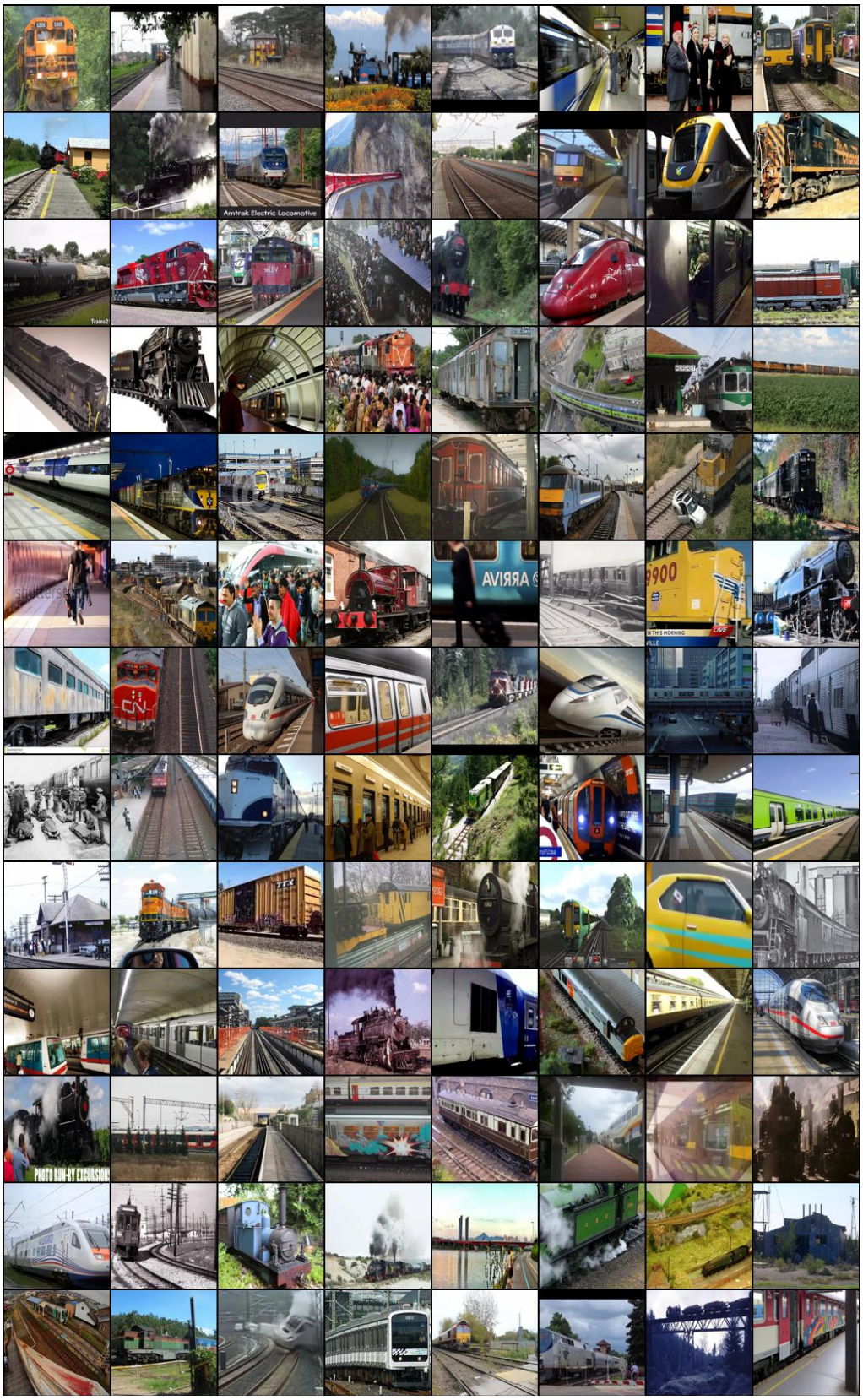

Figure 4-3: Sample frames from the Synthetic Trains dataset.

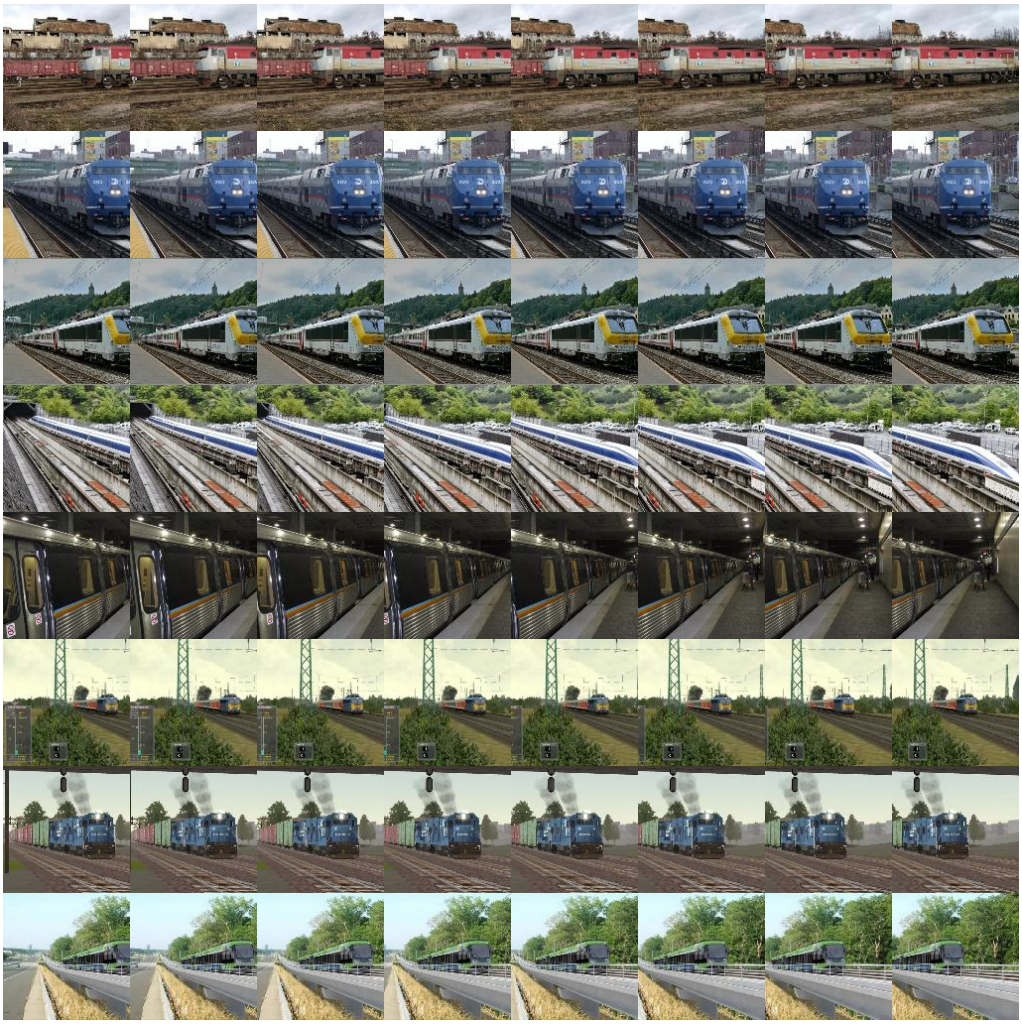

Figure 4-4: Sample clips from the Synthetic Trains dataset. Each row shows a different video, with the columns showing different frames of these videos. The original clips are 32 frames long, but every 4 frames are shown here. This figure can be watched as a video at `http://people.csail.mit.edu/negan/thesis/figures.html`.

4.2 Experimental Setup

4.2.1 Model Implementation and Training

Each of the models outlined in section 3 were implemented in PyTorch. The overall training framework was heavily influenced by the structure of the CycleGAN [16] and Pix2Pix [26] PyTorch repository, with the 2D Resnet architecture lifted from the GAN Stability [20] repository, and the SPADE architecture lifted from the SPADE [29] repository.

Each model was trained using 2 to 4 Nvidia GPUs depending on the size of the model, with GPUs including the GTX Titan, the GeForce GTX 1080, the Tesla K40c, the GeForce GTX Titan X, the Titan X Pascal, and the Titan Xp. The batch size was generally chosen to maximize VRAM usage.

During training, images or videos were resized to be of dimensions 160×160, and then randomly cropped to be of dimensions 128×128. Random horizontal flipping was also performed. When training image models, the dataloader sampled training images by first uniformly sampling a video, then uniformly sampling a clip from that video, and finally sampling a frame from that clip. Training the DCGAN and video model followed a similar procedure, except the whole clip was used as a training video. The SPADE and Resnet video models however generate videos 8 frames in duration as opposed to 32 due to memory constraints, so to train these model we uniformly sampled a starting frame from the first 8 frames of a clip, and then used every 3 frames in the video beyond that. By doing so, we effectively lowered the framerate of the video to allow for the video to contain significant movement.

The DCGAN models were trained with hinge loss [14]. The Resnet models were trained with the original GAN [12] logistic loss, using the GAN Stability [20] regularization term with a weight of 10. The SPADE models were trained with hinge loss, VGG loss with a weight of 10, and L1 loss with a weight of 10. The VAE SPADE models also included a KL Divergence loss term with a weight of .05.

The DCGAN models were trained using the Adam optimizer with $\beta = (0, 0.999)$ and a learning rate of $2 \cdot 10^{-4}$. The Resnet models were trained with the RMSProp

optimizer, with the image model using a learning rate of $1 \cdot 10^{-4}$ and the video model using a generator learning rate of $1 \cdot 10^{-5}$ and discriminator learning rate of $3 \cdot 10^{-5}$. The SPADE models were trained using the Adam optimizer with $\beta = (0, 0.9)$ and a learning rate of $2 \cdot 10^{-4}$.

4.2.2 Experimental Procedures

The first step to answering the questions outlined at the beginning of this chapter was to train several image GAN models that could later be inflated. The 2D DCGAN and Resnet models as described in section 3.3 were trained on the Synthetic Trains and Nature Video datasets by randomly sampling a frame within the video clips. Since these are unconditional models, they ignored the image segmentation masks of the Nature Video dataset. The 2D SPADE model was trained on the Nature Video dataset only. Since the Synthetic Trains dataset is only intended to be used for debugging, the results for it are not included in this paper.

When training the SPADE model, we found that the architecture described in the paper and section 3.3 was too big for our relatively small dataset, leading to overfitting in which the model memorized training examples and thus performed poorly at test time. As a result, we trained two smaller sizes of the SPADE architecture: one with the number of Generator convolutional filters cut in half (the "mid" model), and one with the Generator convolutional filters cut by a factor of 4 and the second block removed (the "small" model). We also wanted to see if including the encoder network and training SPADE as an autoencoder would help SPADE produce better results, so in total we had 4 SPADE models: small, small VAE, mid, and mid VAE. The relative sizes of all the models we trained is shown in table 4.2.

After training our models, we trained each video model both through inflating the corresponding image model as well as by training from scratch. We produced test videos by sampling from the unconditional video models and using validation segmentation masks for SPADE, and analyzed these videos to test our hypotheses.

Table 4.2: Model sizes, measured in millions of parameters

Model	Generator	Discriminator	Encoder	Total
DCGAN Image	7.5	7.0		14.5
Resnet Image	33.3	29.1		62.4
SPADE Image Small	12.5	1.7		14.2
SPADE Image Small VAE	12.5	1.7	1.3	15.5
SPADE Image Mid	35.2	1.7		36.9
SPADE Image Mid VAE	35.2	1.7	3.1	40.0
DCGAN Video	28.3	27.8		56.1
Resnet Video	102.7	85.8		188.0
SPADE Video Small	19.5	5.1		24.6
SPADE Video Small VAE	19.5	5.1	1.8	26.4
SPADE Video Mid	70.3	5.1		75.4
SPADE Video Mid VAE	70.3	5.1	5.0	80.4

4.3 Results

4.3.1 Unconditional Image Generation

The DCGAN and Resnet image models were trained on the Nature Videos training frames. The DCGAN was trained for 100 epochs, and the Resnet model was trained for 400 epochs, with a decaying learning rate for the last 200 epochs. Figures 4-5 and 4-6 show sample images from the DCGAN and Resnet models after training respectively.

4.3.2 Unconditional Video Generation

A 3D DCGAN was trained from scratch for 80 epochs, and another 3D DCGAN was trained by inflating the 2D DCGAN and finetuning on the Nature Videos training clips for 80 epochs. Figure 4-7 shows test videos sampled from the inflated model, and figure 4-8 shows test videos sampled from the model trained from scratch.

The inflated 3D Resnet model was trained from inflating the 2D Resnet model after it completed training, and finetuning for 20 epochs. We attempted to also train the 3D Resnet model from scratch, but training collapsed during the first epoch. Different learning rate settings were attempted to try to stabilize the training, but

nothing worked. Figure 4-9 shows generated videos from the two models. The first row shows what every video generated by the model trained from scratch looks like, and the following rows show videos generated by the inflated model.

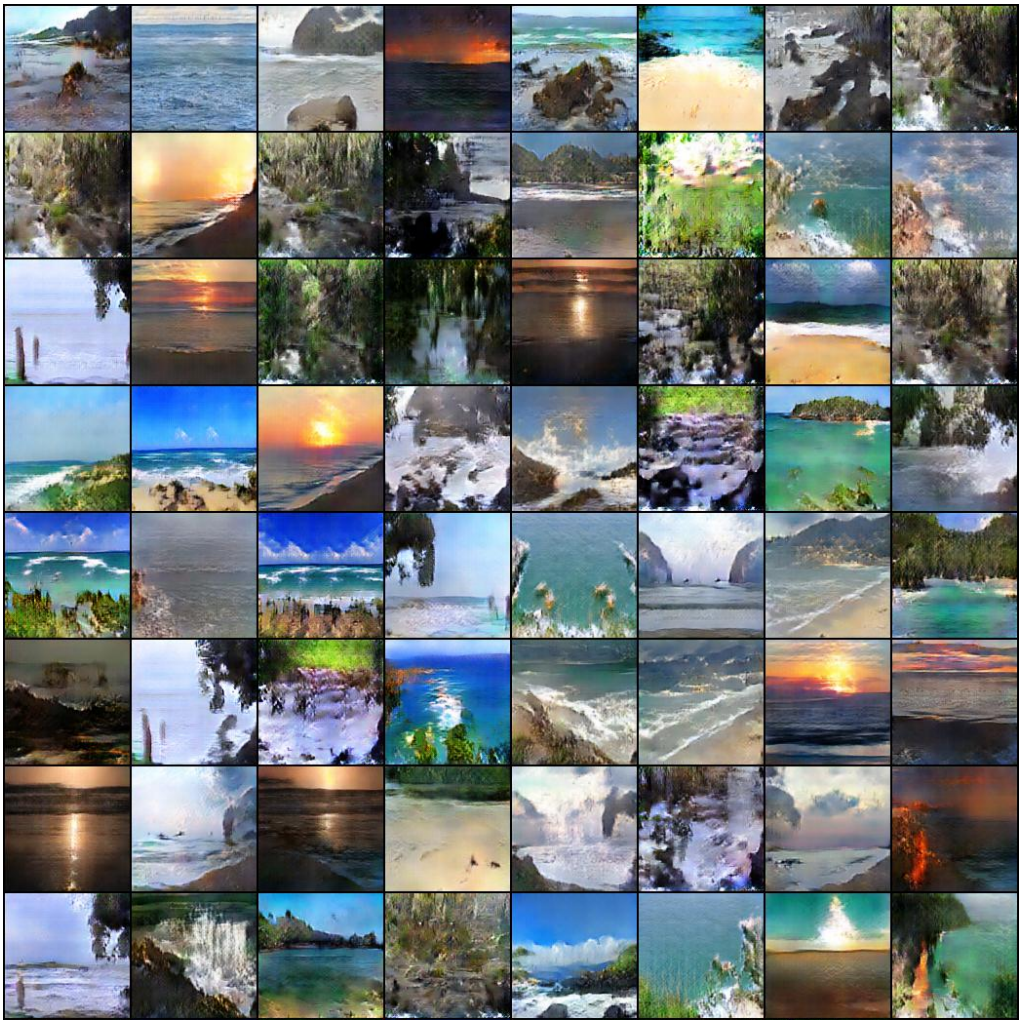

Figure 4-5: Test images generated by 2D DCGAN.

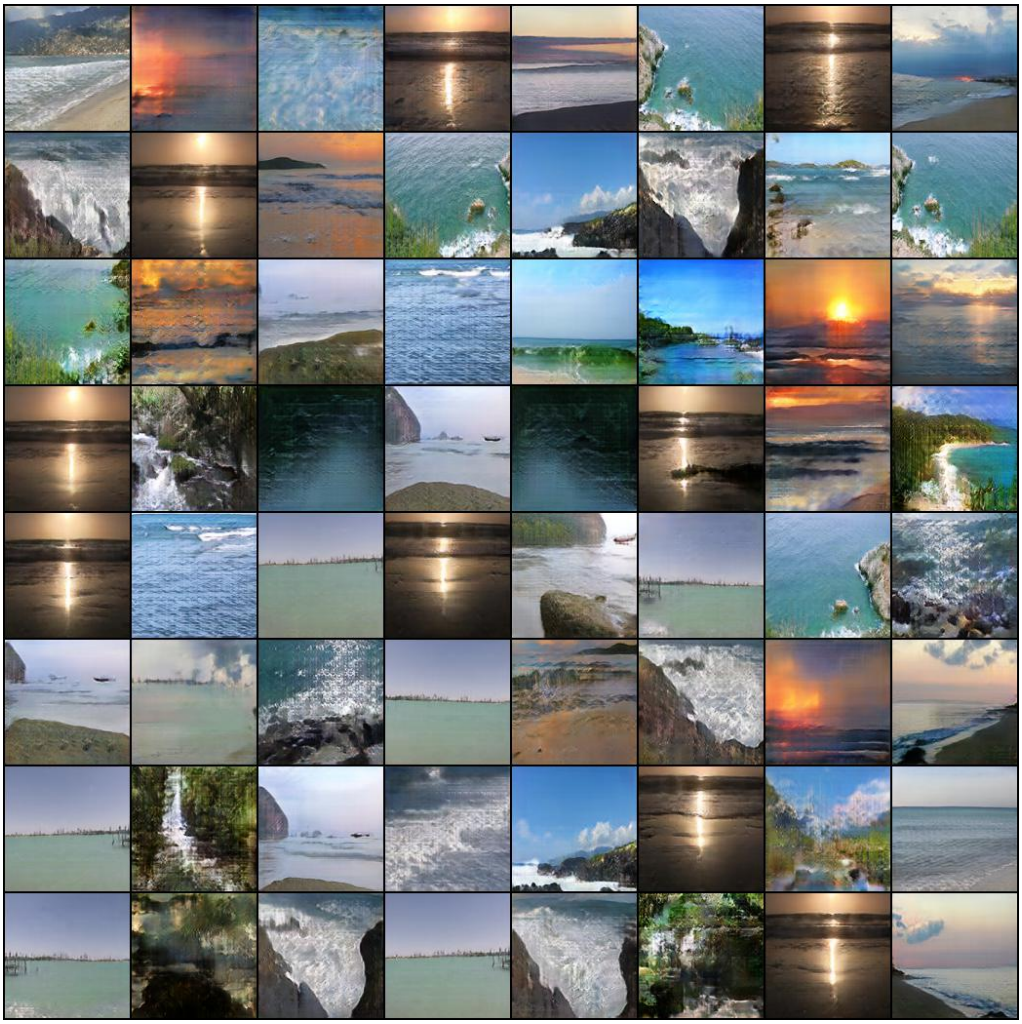

Figure 4-6: Test images generated by 2d Resnet.

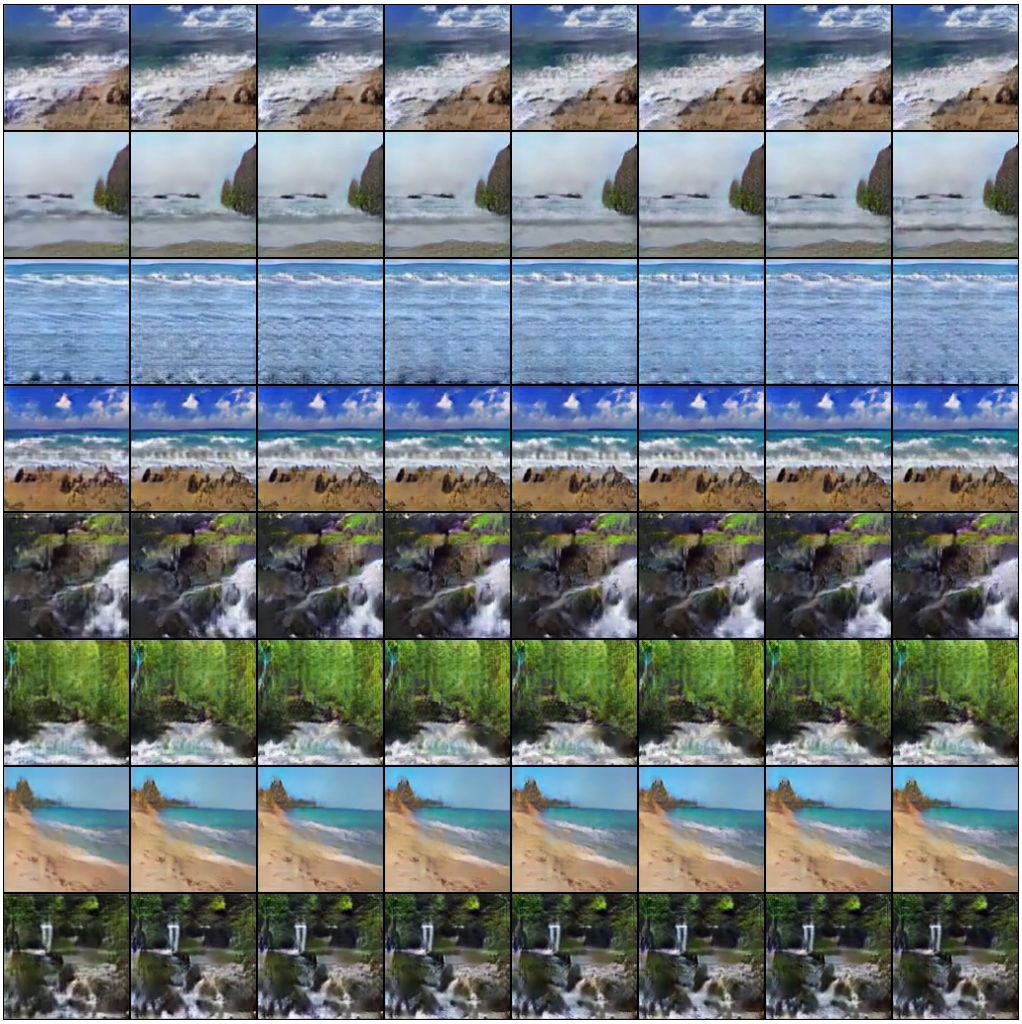

Figure 4-7: Test videos generated by the inflated 3D DCGAN. Each row shows frames of a different video. The original video is 32 frames long, so every 4 frames are shown here. This figure is best watched as a video at `http://people.csail.mit.edu/negan/thesis/figures.html`.

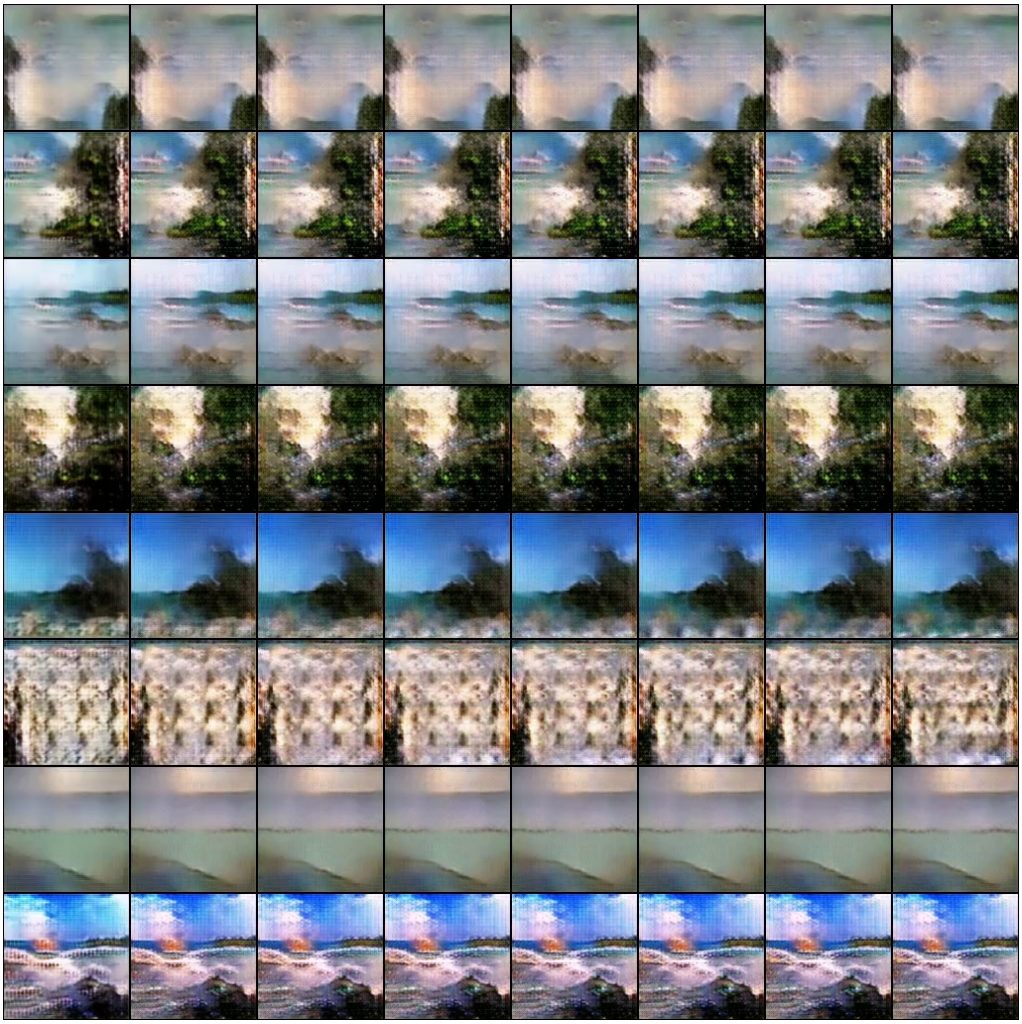

Figure 4-8: Test videos generated by the 3D DCGAN trained from scratch. Each row shows frames of a different video. The original video is 32 frames long, so every 4 frames are shown here. This figure is best watched as a video at `http://people.csail.mit.edu/negan/thesis/figures.html`.

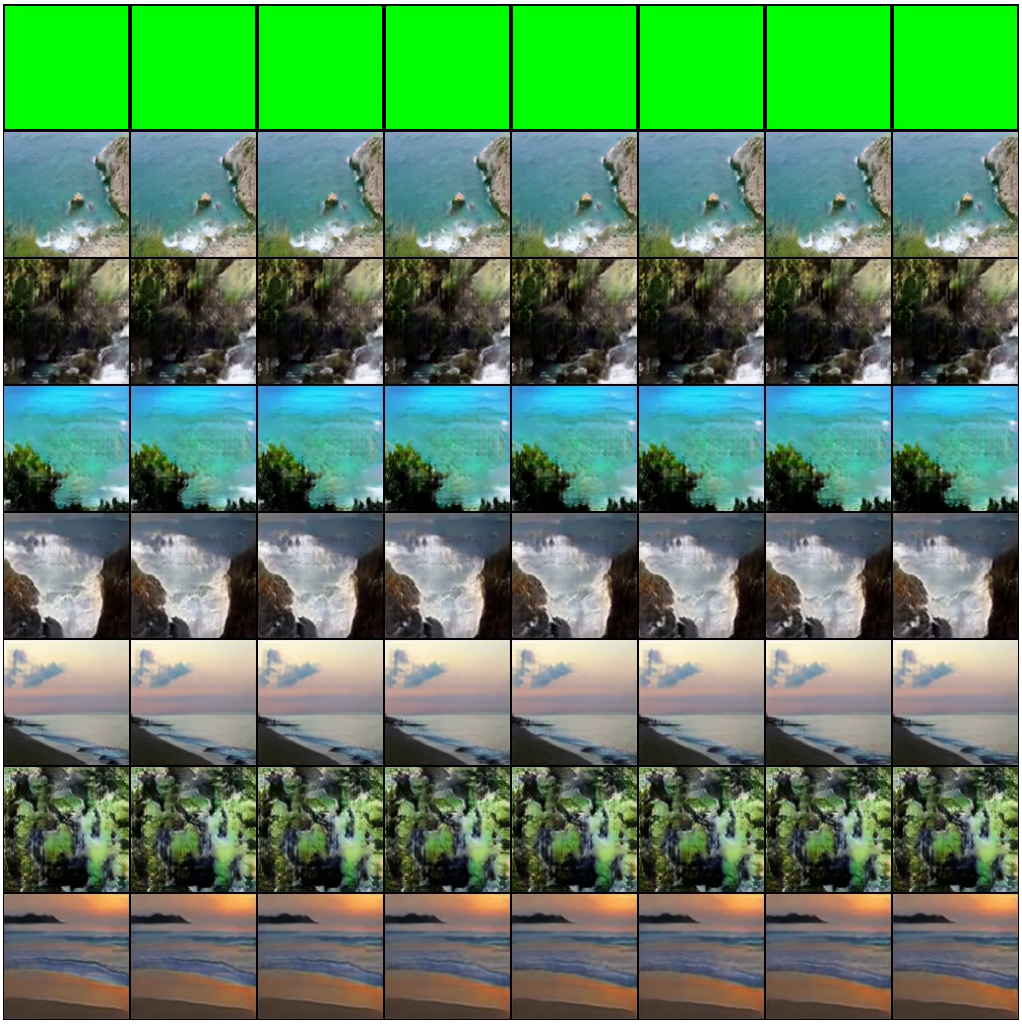

Figure 4-9: *Top Row:* A test video generated by 3D Resnet trained from scratch. *Other Rows:* test videos generated by the inflated 3D Resnet model. Each row shows frames of a different video. This figure is best watched as a video at `http://people.csail.mit.edu/negan/thesis/figures.html`.

4.3.3 Segmentation-to-Image

Each variant of the SPADE image model was trained for 40 epochs on the Nature Videos training dataset. For every 5 epochs of the model, I sampled 353 video frames from the validation set and ran them through the trained SPADE generator to produce test images. I then computed the mean IoU and pixelwise accuracy for each of these images by computing a segmentation mask for each image and comparing it to the segmentation mask from the validation dataset. Since the mean IoU averages the IoU across every semantic class, but not every semantic class is represented in the dataset, I only considered classes that occur with a frequency of at least 1% in the dataset, which narrowed it down to the top 10 classes. These results are shown in table 4.3, with the highest scores for each variant bolded. Figure 4-10 shows example images from each 2D SPADE variant, using the epoch with the highest mean IoU score.

When analyzing these mean IoU and pixelwise accuracy scores, it's important to note that they are noisy metrics because we are using an imperfect segmentation model to calculate them. While they can be used to see overall trends in the data, small differences in scores are most likely not meaningful. Visually inspecting the images produced by the different variants seems to suggest that they all perform similarly well, and that the differences in metrics are largely noise.

The VAE SPADE model as described in the SPADE paper [29] is supposed to allow for different images to be generated from the same input label by using the encoder model to generate an input latent vector. However, we found that our VAE models produced the same image at test time regardless of the input noise vector.

4.3.4 Segmentation-to-Video

After training the 2D SPADE models, these models were inflated using the weights of the epoch that maximized mean IoU, and then finetuned on the Nature Videos clips for 5 epochs. We also trained the 3D SPADE models from scratch for 5 epochs. Since using the encoder network didn't help our metrics when training images and also didn't allow us to produce different outputs, we didn't use VAE video models.

Table 4.3: 2D SPADE Performance, measured by mean IoU and pixelwise accuracy

Variant	Epoch	mIoU	Accuracy
small	5	39.07	75.52
small	10	39.73	75.45
small	15	43.16	77.14
small	20	42.70	77.30
small	25	44.23	77.40
small	30	**45.68**	**78.05**
small	35	45.07	77.84
small	40	44.30	77.36
small	45	43.67	77.07
mid	5	41.08	75.96
mid	10	**44.89**	**77.74**
mid	15	42.21	76.70
mid	20	42.63	76.54
mid	25	42.83	77.23
mid	30	43.09	77.09
mid	35	43.81	77.26
mid	40	43.68	76.27
small vae	5	38.95	75.60
small vae	10	39.39	75.33
small vae	15	43.38	76.48
small vae	20	44.00	77.23
small vae	25	41.68	76.23
small vae	30	41.82	76.66
small vae	35	**44.12**	**77.35**
small vae	40	42.32	76.35
mid vae	5	40.45	75.68
mid vae	10	42.78	76.81
mid vae	15	42.67	76.83
mid vae	20	42.60	77.44
mid vae	25	43.98	77.17
mid vae	30	**45.00**	**78.42**
mid vae	35	42.25	76.33
mid vae	40	41.73	76.26

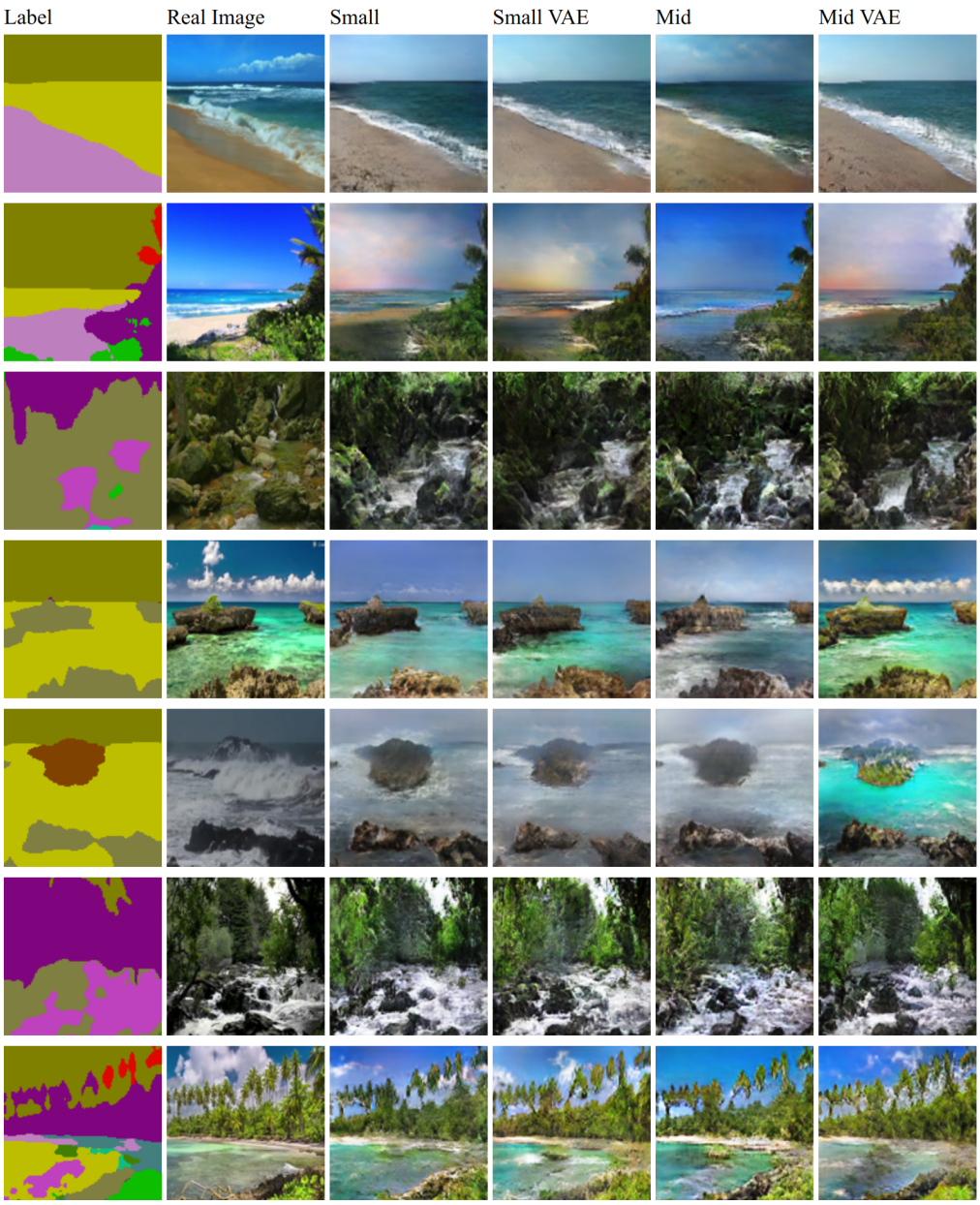

Figure 4-10: Test images generated by 2D SPADE. Each row comes from a different input segmentation mask.

Figures 4-11 and 4-12 show a comparison of generated test videos between the different models at epochs 1 and 5 respectively.

One way to quantitatively measure the quality of these generated videos is through the mean IoU and pixelwise accuracy of the video frames. These metrics were calculated for each of the 5 epochs of the models, and are shown in table 4.4. The warning given in the previous section about reading too much into these metrics holds here as well. It is also important to note that these metrics simply measure the frame quality of the generated videos, and don't pay attention to the dynamics or temporal consistency.

Table 4.4: 3D SPADE framewise performance

Variant	**Epoch**	**mIoU**	**Accuracy**
small inflate	1	17.18	33.11
small inflate	2	21.09	45.73
small inflate	3	19.56	37.69
small inflate	4	21.64	47.13
small inflate	5	19.77	44.56
small scratch	1	16.49	41.84
small scratch	2	17.12	42.86
small scratch	3	18.57	46.25
small scratch	4	20.32	43.84
small scratch	5	18.64	41.41
mid inflate	1	23.67	48.68
mid inflate	2	21.10	49.37
mid inflate	3	17.15	37.56
mid inflate	4	17.79	40.19
mid inflate	5	20.02	44.54
mid scratch	1	19.02	43.61
mid scratch	2	19.34	45.71
mid scratch	3	19.67	43.87
mid scratch	4	17.99	35.04
mid scratch	5	20.29	40.41

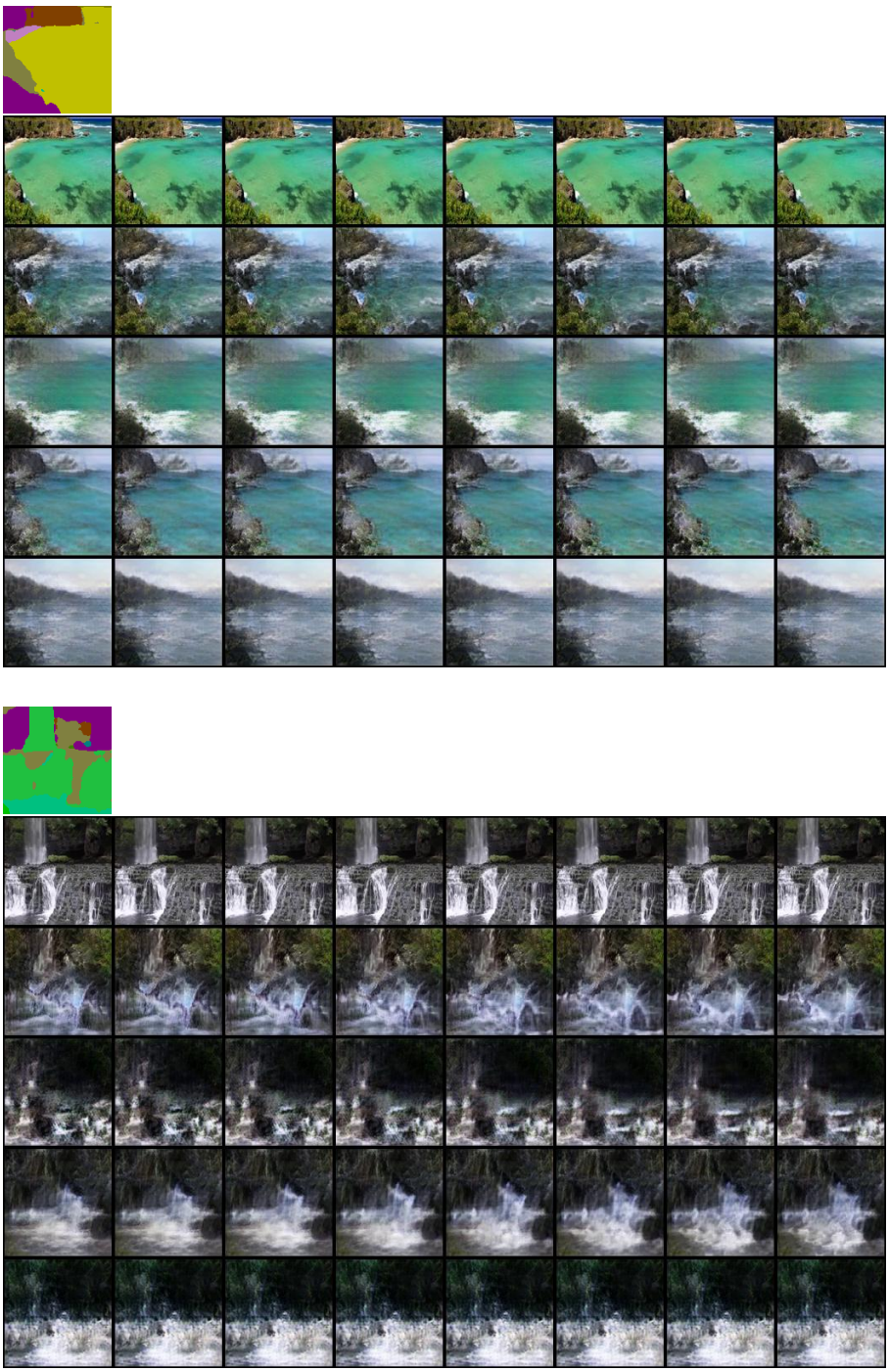

Figure 4-11: 2 examples of test videos generated by 3D SPADE at 1 epoch. On top of both of these examples is the input label from the test dataset. Each row represents the frames of a video, which from top to bottom are real, small inflated fake, small scratch fake, mid inflated fake, mid scratch fake. This figure is best watched as a video at `http://people.csail.mit.edu/negan/thesis/figures.html`.

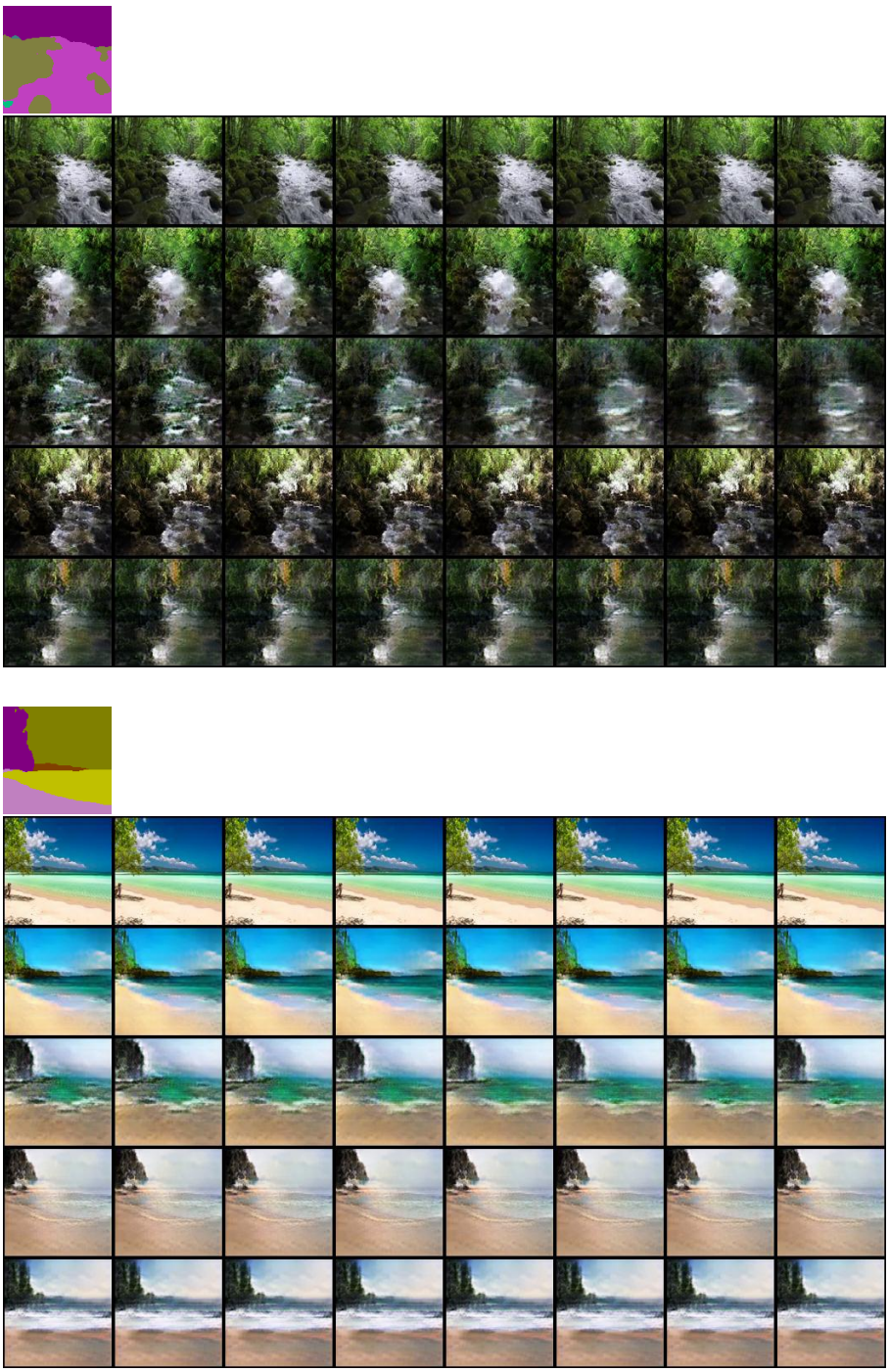

Figure 4-12: 2 examples of test videos generated by 3D SPADE at 5 epochs. On top of both of these examples is the input label from the test dataset. Each row represents the frames of a video, which from top to bottom are real, small inflated fake, small scratch fake, mid inflated fake, mid scratch fake. This figure is best watched as a video at `http://people.csail.mit.edu/negan/thesis/figures.html`.

4.4 Analysis

4.4.1 Model Inflation

For unconditional video generation, model inflation leads to dramatically better results. This was particularly true for the 3D Resnet model, which completely collapsed when trained from scratch. To investigate the cause of this training collapse, we can observe the training loss graphs in figure 4-13. The top graph, showing training with inflation, is an example of GAN training gone well: after a brief unstable period at the beginning, the generator loss, discriminator real loss, and discriminator fake loss eventually center themselves at around $-\log(\frac{1}{2}) = 0.693$. Since we're using logistic loss, this corresponds to a discriminator that can only make correct predictions around 50% of the time. The oscillation around this $-\log(\frac{1}{2})$ value shows that neither the generator nor the discriminator has the upper hand, and that they continue to learn from one another.

The lower graph, showing training from scratch, paints a rather different picture. The generator loss tends to stay at values around 10, which corresponds to the generator only being able to fool the discriminator around $\exp(-10) = 0.0045\%$ of the time. The spikes in the generator loss graph show a sudden increase then decrease in generator loss, which could suggest that the discriminator overfits itself to the videos that the generator is currently generating. The generator quickly learns to generate something different (like generating green squares instead of blue squares), which temporarily confuses the discriminator. This dynamic is unproductive in reaching the eventual goal of generating videos matching the dataset, and could suggest that the 3D Resnet model is too complex to train from scratch.

The 3D DCGAN also benefited from inflation, but to a lesser extent. The inflated model was able to hit the ground running, and in relatively few epochs started generating videos with realistic dynamics while retaining most of the original frame quality. The model trained from scratch however was very slow to train. In the same number of epochs the inflated model took to finetune, the model trained from scratch could barely generate frames resembling the dataset.

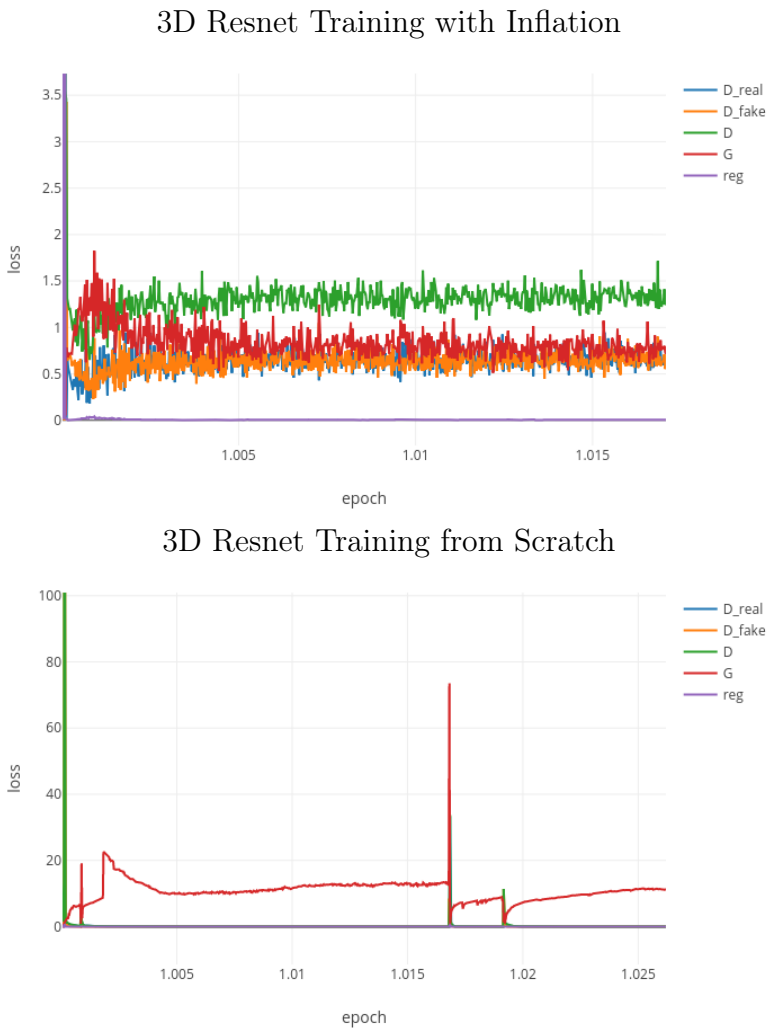

Figure 4-13: Graphs displaying the various losses from training 3D Resnet from an inflated 2D Resnet (top) and from scratch (bottom). *D_real* refers to discriminator loss on real videos, *D_fake* refers to discriminator loss on generated videos, *D* refers to the total discriminator loss, *G* refers to the generator loss, and *reg* refers to the GAN Stability regularization term.

In the segmentation-to-video setting, the differences between the inflated models and the models trained from scratch are more subtle. They perform similarly in the mean IoU and pixelwise accuracy metrics as shown in 4.4, but to my eye the inflated model produces more realistic videos, especially when comparing the two mid sized models. So while inflation is not necessary for 3D SPADE training speed or the prevention of collapse, I would argue it helps video quality.

Overall, the effectiveness of model inflation seems to be tied to the complexity of

the learning problem. Since unconditional GANs are harder to train than conditional GANs, they benefit more from inflation. Additionally, models with more parameters seem to struggle more when trained from scratch, as shown in table 4.2.

4.4.2 Segmentation-to-Video

Since one of the goals of this project is to create an artistic tool that can allow users to hand draw a segmentation mask and turn it into a video, I created some segmentation masks and ran them through the small inflated model. The results are shown in figure 4-14.

What is interesting to see in these results is that the model doesn't strictly obey segmentation boundaries. In the first video, the tree seems to have been extended rightwards; in the fourth video, rocks have been covered by water in some places and rocks have been put in water in others; and in the 5th video, the grass drawn at the bottom has been turned into a reflection of grass in the river. This is likely a result of the training segmentation masks being noisy: oftentimes the segmentation algorithm we used would fail to label smaller details in images like the fine lines between rocks and water, or make bigger mistakes simply due to the difficulty of the segmentation problem. Our GANs have learned to recognize that the input segmentations are not always completely correct, and has figured out how to work around that.

The last two videos were created to test how our model would deal with unrealistic inputs. The second from the bottom segmentation has sky below sand, and shows that the model is unable to output something appropriate, which is a result of no video in the training dataset containing sky in the bottom of the image or sand in the top of the image. The bottom segmentation has vertical bars of sea, sky, sand, rock, plant, and mountain, and the model appears to have ignored some of these labels for the reasons described in the previous paragraph.

This mismatch between the segmentation mask and the generated video is in many ways a good thing: drawing fine borders between semantic labels is hard for people to do. By correcting the borders to be more realistic, the model is helping the user to create a better output video. The example test videos as well as the example custom

videos show, in my opinion, that we indeed have succeeded in creating models that can map between image segmentation masks and videos matching that mask.

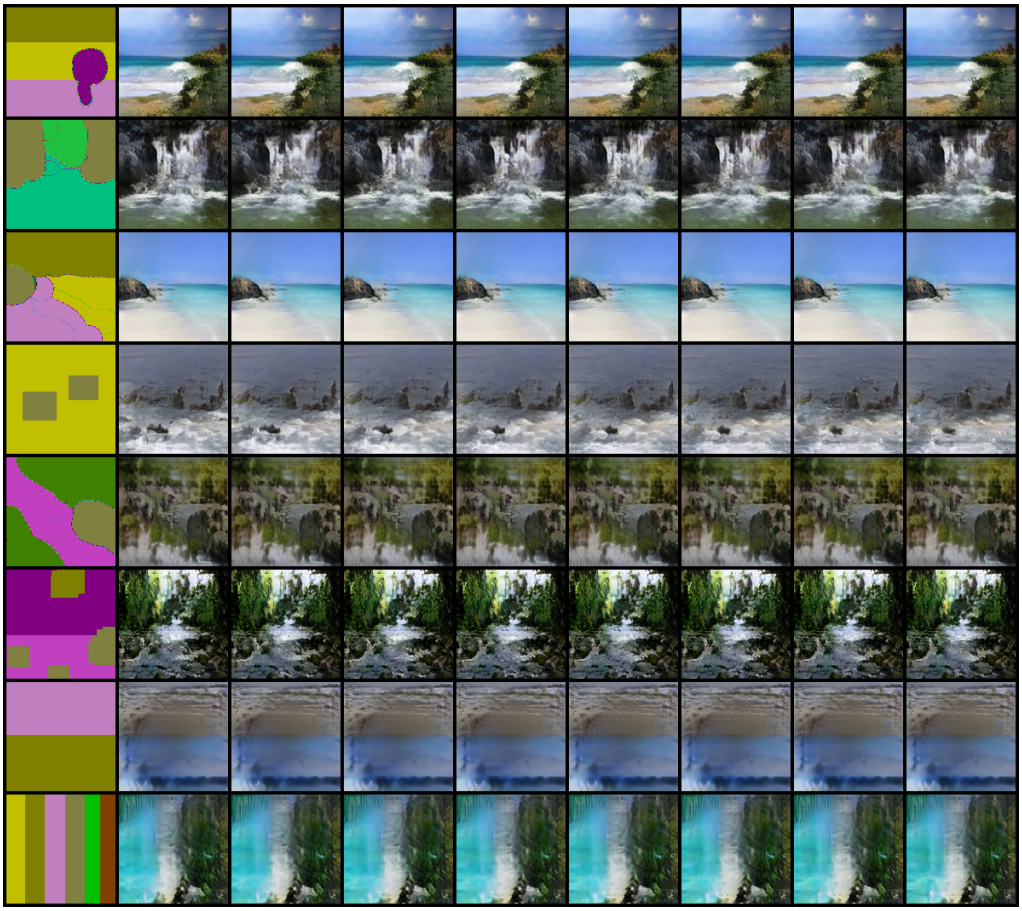

Figure 4-14: Videos created by custom segmentation masks that I hand drew. Each row represents a different video, with the leftmost image showing the input segmentation. This figure is best watched as a video at `http://people.csail.mit.edu/negan/thesis/figures.html`.

Chapter 5

Conclusion

This thesis has proved the effectiveness of model inflation as a technique to train 3D GAN models for video generation by constructing 3D GAN architectures, deriving the model inflation procedure, and evaluating model inflation through several experiments. This thesis has also created a GAN that can effectively perform the segmentation-to-video task through the process of assembling a suitable dataset, implementing an image-to-video GAN model, and testing the trained model on real and handmade inputs.

This working algorithm for model inflation is significant because it could potentially allow for parts of the huge body of image GAN research to be ported over to video GANs. Stability problems in 3D GAN training, especially with larger models accomplishing more demanding tasks, may be able to be fixed by first training the GANs in 2D form. Testing model inflation on different GANs, or even on neural network models other than GANs, is a future research problem to be explored. It would also be useful to study in greater detail the training dynamics of model inflation, and to see if image quality can be better transferred to video quality.

The segmentation-to-video models trained in this thesis are a useful artistic tool that can bring to life anybody's idea for a natural scene. No special skills are required to create, through hypothetical extensions of this model, a hit movie featuring CGI effects. One of the greatest strengths of this model as a creative tool comes from the fact that it maps between images, which are easy for humans to draw, and videos,

which are pleasing for humans to watch. Future research can explore related image-to-video models that can create videos out of inputs like outlines or cartoons.

Bibliography

[1] Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*, 2015.

[2] Andrew Brock, Jeff Donahue, Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *International Conference on Learning Representations (ICLR)*, 2018.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[4] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[5] Carl Vondrick, Hamed Pirsiavash, Antonio Torralba. Generating Videos with Scene Dynamics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[6] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[8] David G Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)*, 1999.

[9] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, Jianxiong Xiao. LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop. In *arXiv:1506.03365*, 2015.

[10] Han Zhang, Ian Goodfellow, Dimitris Metaxas, Augustus Odena. Self-Attention Generative Adversarial Networks. In *arXiv preprint arXiv:1805.08318*, 2018.

[11] Holly Grimm. Training on Art Composition Attributes to Influence CycleGAN Art Generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[13] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[14] Jae Hyun Lim, Jong Chul Ye. Geometric GAN. In *preprint arXiv:1705.02894*, 2017.

[15] James Vincent. ThisPersonDoesNotExist.com uses AI to generate endless fake faces. *The Verge*, 2019.

[16] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[17] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In *Advances in Neural Information Processing Systems (NeurIPS) Workshop*, 2014.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[19] Katsunori Ohnishi, Shohei Yamamoto, Yoshitaka Ushiku, Tatsuya Harada. Hierarchical Video Generation from Orthogonal Information: Optical Flow and Texture. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2017.

[20] Lars Mescheder, Sebastian Nowozin, Andreas Geiger. Which Training Methods for GANs do actually Converge? In *International Conference on Machine Learning (ICML)*, 2018.

[21] Martin Arjovsky, Soumith Chintala, LÃľon Bottou. Wasserstein GAN. In *International Conference on Machine Learning (ICML)*, 2017.

[22] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[23] Masaki Saito, Eiichi Matsumoto, Shunta Saito. Temporal Generative Adversarial Nets with Singular Value Clipping. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[24] Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. In *arXiv:1409.0575*, 2014.

[26] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[27] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, Jan Kautz. MoCoGAN: Decomposing Motion and Content for Video Generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[28] Shane Barratt, Rishi Sharma. A Note on the Inception Score. In *arXiv:1801.01973*, 2018.

[29] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, Jun-Yan Zhu. Semantic Image Synthesis with Spatially-Adaptive Normalization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[30] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[31] Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *International Conference on Learning Representations (ICLR)*, 2017.

[32] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[33] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, Bryan Catanzaro. Video-to-Video Synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[34] Xiaolong Wang, Ross Girshick, Abhinav Gupta, Kaiming He. Non-local Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[35] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, Stephen Paul Smolley. Least Squares Generative Adversarial Networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2016.

[36] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[37] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal on Computer Vision (IJCV)*, 2018.