A Additional Implementation Details

We use the Adam optimizer Kingma & Ba (2014) with learning rate $\lambda_G = 1e10^{-4}$ and $\lambda_D = 5e10^{-4}$ for the generator and the discriminator, respectively. The discriminator is updated twice for each generator update.

We use orthogonal initialization for all the weights in our model and use spectral normalization both in the generator and the discriminator. We only use the first singular value to normalize the weights. Different from DVD-GAN, we do not use weight moving averages nor orthogonal penalities.

Conditional batch normalization layers use the input noise as the condition, concatenated with the class label when applicable. Features are normalized with a per-frame mean and standard deviation.

To unroll a generator beyond its training temporal horizon, we apply it convolutionally over longer input sequences. We perform 200 "dummy" forward passes to recompute the per-timestep batch normalization statistics at test time.

All convolutions in our models use 3x3 or 3x3x3 filters with padding=1 and stride=1, for 2D or 3D convolutions respectively. All models were implemented in PyTorch.

B Model Architecture Details

For the rest of the section, we use B to denote the batch size, T for the number of frames or timesteps, C for the number of channels, H for the height of the frame and W is the width of the frame.

B.1 Stage 1 architecture

Our first stage model is based on a re-implementation of DVD-GAN Clark et al. (2019). In all our experiments, the first stage produces 32x32/25 outputs.

Generator The generator is composed by a stack of units where each unit is comprised of a ConvGRU layer and two 2D-ResNet upsampling blocks. We follow the nomenclature of Brock et al. (2018); Clark et al. (2019) and describe our network using a base number of channels ch and the channel multipliers associated with each unit. Our stage-1 generators is formed by 4 units with channel multipliers $[8, 8, 4, 2]$. The base number of channel is 128.

The first input of this network is of size BxTx(8xch)x4x4. This input is obtained by first embedding the class label onto a 128 dimensional space, then concatenating the embedding to a 128 dimensional noise vector. This concatenation is mapped to a Bx(8xch)x4x4 tensor with a linear layer and a reshape, and then the final tensor is obtained by replicating the output of the linear layer T times.

The ConvGRU layer Ballas et al. (2015) follows the ConvGRU implementation of Clark et al. (2019) and uses a ReLU non-linearity to compute the ConvGRU update.

The 2D ResNet blocks are of the norm-act-conv-norm-act-conv style. We use conditional batch normalization layers, ReLU activations and standard 2D convolutions. Before the first convolution operation and after the first normalization and activation, there is an optional upsampling operation when increasing the resolution of the tensor. We use standard nearest neighbor upsampling. Except for the last unit, all units perform this upsampling operation. The conditional batch normalization layers receive the embedded class label (if applicable) and the input noise as a condition and map it to the corresponding gain and bias term of the normalization layer using a learned linear transformation. The 2D ResNet blocks process all frames independently by reshaping their input to be (B*T)xCxHxW.

The output of the last stack goes through a final norm-relu-conv-tanh block that maps the output tensor to RGB space with values in the [-1, 1] range.

Discriminator There are two discriminators, a 2D discriminator and a 3D discriminator. The 2D discriminator is composed of 2D ResNet blocks. Each ResNet block is formed by a sequence of relu-conv-relu-conv layers. There are no normalization layers in the discriminator. After the last conv in each block there is an optional downsampling operation, which is implemented with average pooling layers. The 2D discriminator receives as input 8 randomly sampled frames from real or generated samples.

The 3D discriminator is equal to the 2D discriminator except that its first two layers are 3D ResNet blocks, implemented by replacing 2D convolutions with regular 3D convolutions. The 3D discriminator receives as input a spatially downsampled (by a factor of two) real or generated sample. The 2D blocks process different timesteps independently.

We concatenate the output of both discriminators and use a geometric hinge loss. The loss is averaged over samples and outputs.

We use 128 as base number of channel for both discriminators, with the following channel multipliers for each ResNet block: $[16, 16, 8, 4, 2]$

B.2 Upsampling stage architecture

The upsampling stage models follow the same architecture as the first stage with the following modifications.

Generator The generator units replace the ConvGRU layers with a Separable 3D convolution. We first convolve over the temporal dimension with a 1D temporal kernel of size 3 and then convolve over the spatial dimension with 2D 3x3 kernel. We empirically compare generators with ConvGRU and separable convolutions in section C.

We add residual connections at the end of each 3D and 2D ResNet block to an appropriately resized version of $\mathbf{x}^l_{w_i}$. We use nearest neighbor spatial downsampling for this operation, and we use nearest neighbor temporal interpolation to increase the number of frames of $\mathbf{x}^l_{w_i}$. We then map the residual to the appropriate number of channels using a linear 1x1 convolution. We do not add $\mathbf{x}^l_{w_i}$ residual connections to feature maps with spatial resolutions (HxW) greater than the resolution of $\mathbf{x}^l_{w_i}$.

Discriminator We reuse the same 2D and 3D discriminators as for the first stage. Additionally, we add a matching discriminator that discriminates $(\mathbf{x}^l_{w_i}, \mathbf{x}_{w_i})$ pairs. The matching discriminator utilizes the same architecture as the 3D discriminator. It receives as input a concatenation of $\mathbf{x}^l_{w_i}$ and a downsampled version of $\mathbf{x}_{w_i}$ to match the resolution of $\mathbf{x}^l_{w_i}$. We concatenate the outputs of all three networks and use a geometric hinge loss as for the first stage discriminator. The overall loss is averaged over samples and output locations.

For 128x128 generations on Kinetics, the generator uses 128 as base number of filters with the following channel multipliers $[8, 8, 4, 2, 1]$. All discriminators have 96 base channels and the following channel multipliers $[1, 2, 4, 8, 16, 16]$. All our models at 128x128 are two-stage models. We train models to upsample $\mathbf{x}^l_{w_i}$ of sizes 32x32/3 or 32x32/6 to 128x128/6 or 128x128/12, respectively. Since we train our first stage for 32x32/25 outputs, our two-stage models can generate 128x128/50 outputs when unrolled.

For 128x128 generations on BDD100K, the generator uses 96 as the base number of channels with channel multipliers $[8, 4, 4, 2, 2]$. All discriminators have 64 base channels and channel multipliers $[1, 2, 4, 8, 8, 16]$. Our 128x128 models upsample 64x64/6 $\mathbf{x}^l_{w_i}$ inputs to 128x128/12, and can generate outputs of up to 128x128/100.

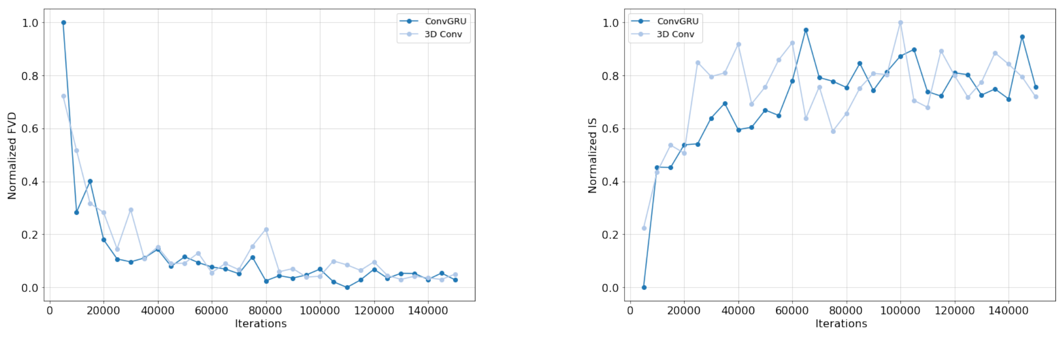

Figure 6: **Comparison of recurrent layers** We compare two variants of the same generator, one with a single ConvGRU layer per generator block and one with a separable 3D convolution per generator block. On the left we show the evolution of the FVD score during training, and on the right we show the Inception Score. Both scores are normalized to the [0, 1] range where 1 is the highest score obtained by these models and 0 the lowest. Both models have similar behaviour and computational costs, but the 3D convolution processes inputs in parallel.

C Comparison of Recurrent Layers

In this section we justify the change of the ConvGRU for Separable 3D convolutions in upsampling stages. In Figure 6 we compare the evolution of two metrics (IS and FVD) during training for two variants of the same two-stage model, one using ConvGRUs and one using separable 3D convolutions. Both models show similar behavior during training and achieve similar final metrics. However, ConvGRUs perform sequential operations over time whereas 3D convolutions can be parallelized.

D Additional Samples

Additional samples can be found in .mp4 format along with this appendix in the supplementary materials file. These videos show multiple samples from our two-stage model for Kinetics and our three-stage model for BDD. For each sample, we show the output of each stage in a row.

We have included 3 videos. One video shows samples from our 128x128/12 model unrolled to generate 128x128/100 samples on BDD. Another video shows samples from our 128x128/12 model unrolled to generate 128x128/50 samples on Kinetics. Finally, we include a video with samples from the same model but without the matching discriminator (baseline no matching disc suffix filename).

We include some additional samples for our Kinetics 128x128/12 model and BDD 128x128/12 model below.

For all evaluations, we sample from an isotropic Gaussian with unit variance for ease of comparison and reproducibility. Samples for figures and the provided video files were produced by sampling with standard deviation $\sigma = 0.5$. We observed that noise samples with reduced variance produce higher quality samples but are slightly less diverse.

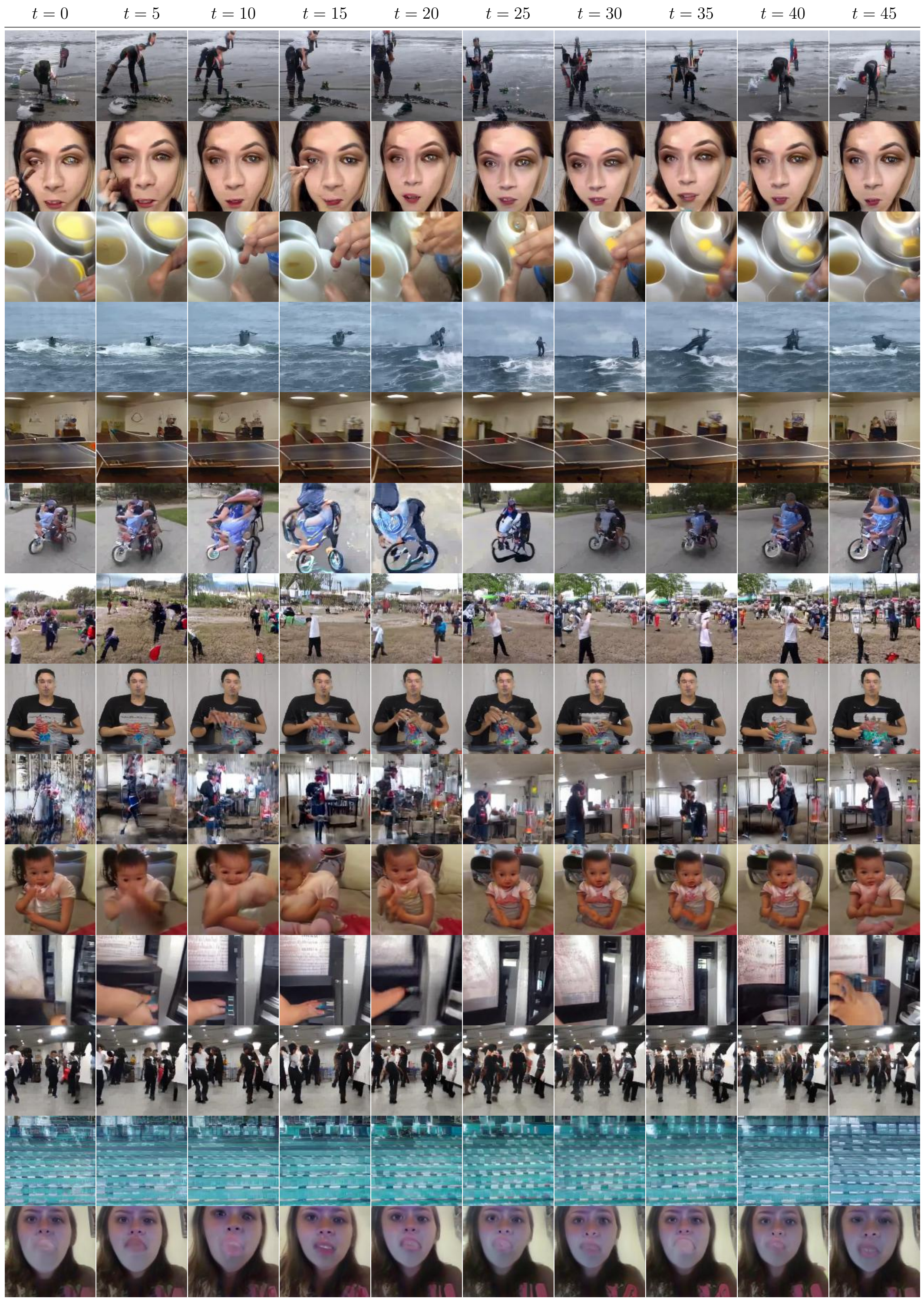

Figure 7: **Additional samples for Kinetics 128x128/12** We show additional samples from our two-stage Kinetics 128x128/12 model unrolled to generate 128x128/50 videos. More samples can be found in the supplementary videos.

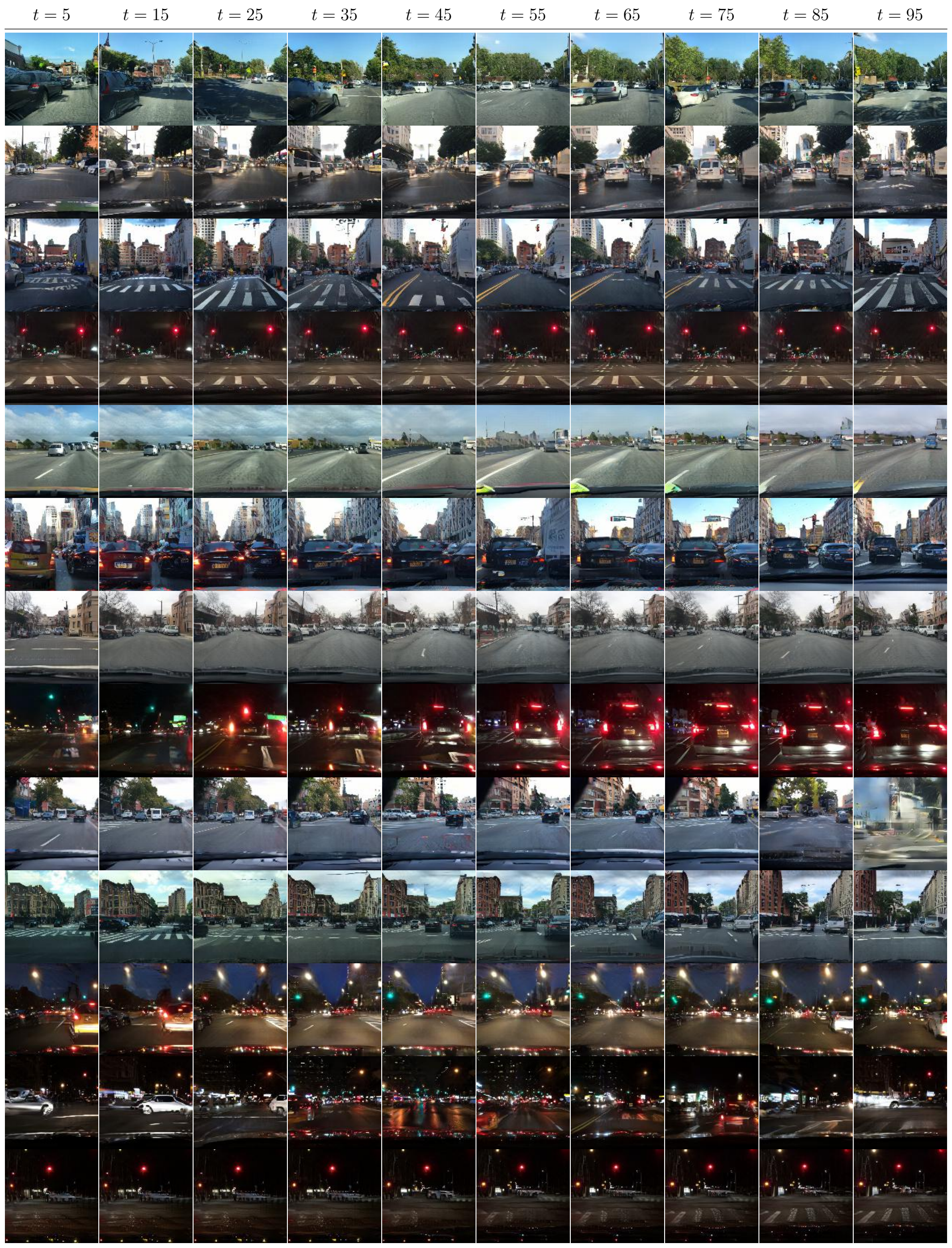

Figure 8: **Additional samples for BDD 128x128/100** We show additional samples from our three-stage BDD 128x128/12 model unrolled to generate 128x128/100 videos.

E Broader Impact

Here we discuss the broader ethical impacts of this work. SSW-GAN is a generative model for video. As with other generative models, there is a chance that similar methods as the one we propose might be used to create "deepfakes". Note however that it would require extensive follow-up work and that it would not be a direct application of our methods. Furthermore, it is not possible to know in advance what a generation is going to look like, and therefore our model could produce results that could be nonsensical or in bad taste.

At the same time, generative models for video have multiple positive applications. First, their generations can be used to train better classifiers. They can also be used to facilitate content creation for visual artists. Furthermore, variants of our model could be used to better compress videos, which has advantages such as reduced bandwidth requirements to transmit video data.

Overall our model would require follow-up work to enable some of the positive and negative applications described, as in its current form it is a class-conditional generative model with its main ability being that of generating samples similar to those used for its training.