TGANv2: Efficient Training of Large Models for Video Generation with Multiple Subsampling Layers

Masaki Saito Shunta Saito
Preferred Networks, Inc.
{msaito, shunta}@preferred.jp

Abstract

In this paper, we propose a novel method to efficiently train a Generative Adversarial Network (GAN) on high dimensional samples. The key idea is to introduce a differentiable subsampling layer which appropriately reduces the dimensionality of intermediate feature maps in the generator during training. In general, generators require large memory and computational costs in the latter stages of the network as the feature maps become larger, though the latter stages have relatively fewer parameters than the earlier stages. It makes training large models for video generation difficult due to the limited computational resource. We solve this problem by introducing a method that gradually reduces the dimensionality of feature maps in the generator with multiple subsampling layers. We also propose a network (Temporal GAN v2) with such layers and perform video generation experiments. As a consequence, our model trained on the UCF101 dataset at 192×192 *pixels achieves an Inception Score (IS) of 24.34, which shows a significant improvement over the previous state-of-the-art score of 14.56.*

1. Introduction

This study focuses on unsupervised learning for videos with Generative Adversarial Network (GAN) [13, 12]. Unlike the field of image generation where many studies have succeeded in generating high-resolution and high-fidelity images [33, 32, 23, 5], video generation with unconditional GANs is still a challenging problem. One of the main reasons for the difficulty is the fact that videos require larger memory and computational costs than static images. It brings various limitations to the design choice of the network architecture and the training scheme of the models. Therefore, many existing studies on video generation [21, 48, 40, 47] only deal with the small resolution (*e.g.*, 64×64 pixels).

It also brings a limitation on the maximum batch size. Conditional GANs which exploit strong prior such as object contours and semantic labels can be trained with a small number of batch size [20, 49, 50]. However, contrary to those methods, GANs without such strong prior generally require large batch size to train models properly [54, 5]. This situation makes the training of unconditional GANs for video generation more challenging. Therefore, establishing a novel method to efficiently train large models with large batch size will widen the potential of unconditional GANs for videos. For example, it can be useful for high resolution video generation, and it may also be effective for other problems dealing with "large samples" which have high dimensionality, *e.g.*, long-range video prediction [11, 31], large image generation [23, 5], and 3D model generation [52].

Based on this background, in this study we propose a method to efficiently train a large network dealing with large samples. The key idea is to introduce a *subsampling layer*, which stochastically subsamples feature maps along specific dimensions to reduce the dimensionality of intermediate feature maps of the generator during training. We call those target feature maps of subsampling "*abstract maps*". A subsampling layer in the generator reduces the data size of final generated samples so that the input for the discriminator also becomes much lighter. Therefore, it reduces the computational cost to compute the gradient of the model, resulting in shorter training time.

This concept can be extended further with multiple subsampling layers. Contrary to Convolutional Neural Networks (CNNs) for image classification, a generator used in GAN generally has a property that the computational cost and required memory increase as going to the latter blocks close to the output layer, because the spatial sizes of intermediate feature maps are larger at the latter blocks. Here, we assume that the earlier blocks need to learn complicated functions with strong non-linearity, whereas the latter blocks only learn relatively primitive features. Based on this assumption, we introduce multiple subsampling layers to feature maps at different levels in the generator, and subsample feature maps more aggressively in the latter blocks than the earlier blocks. In this study, we represent each subsampling layer as a function that reduces both the batch size and frame rate. Especially, we consider that reducing the batch size multiple times at the different levels in the generator (*adaptive batch reduction*) is applicable to other problems using GANs.

We also propose a new model for large video generation (*Temporal GAN v2*) that employs techniques as mentioned above. Using this model, we succeeded in efficiently training a large network which had originally required substantial computational costs. Specifically, our model can train a generator that outputs a video having 16 frames and 192×192 pixels with four GPUs, and achieved the state-of-the-art inception score (22.70) on UCF101 dataset [43]. It is approximately 83% higher than a previous score published on a conference [47] and 56% higher than a score published on arXiv [1]. By increasing the batch size four times, our model finally achieved the inception score of 24.34.

Our contributions are summarized as follows: (i) the subsampling layer to accelerate the training of large models, (ii) the multi-scale model combining multiple subsampling layers, (iii) demonstrating that subsampling of frames and adaptive batch reduction are useful for video generation problem, and (iv) achieving the state-of-the-art score that significantly outperforms the other existing methods.

2. Related work

Image generation Many applications of the Generative Adversarial Network [13] mainly focus on image generation problem; this is primarily because that DCGAN [37] demonstrated that the GAN is quite effective in image generation. After that, the DCGAN was extended to a network called SA-GAN [54] that includes spectral normalization [33] and self-attention blocks [51]. BigGAN [5] succeeded in generating high fidelity images by introducing several tricks such as orthogonal regularization that stabilizes the training with large batch size. In image-to-image translation problem that transfers an input image from another domain, there exists many studies for transforming high resolution images [20, 50, 58, 28, 19]. These studies and findings are applied to a video-to-video translation problem to be described later.

Multi-scale GANs Our approach is related to methods in which a generator produces multi-scale images. LAPGAN [8] first generates a coarse image and updates it by using a difference of an initial image. StackGAN [56, 55] directly generates multi-scale images, and the corresponding discriminator returns a score from each image. ProgressiveGAN [23] generates high-resolution images by growing both the generator and the discriminator. We will describe the difference between our method and existing ones in Section 4.4.

Video prediction Video prediction, which estimates subsequent images from a given few frames, is one of the major problems in computer vision. Although there is no unified view on how to model the domain of video, many studies dealing with video prediction problem directly predict the next frame with recurrent networks such as LSTM [38, 34, 44, 21, 11, 30, 10, 3, 6, 9, 25]. Another well-known approach is to predict intermediate features of videos such as optical flow [27, 29, 15, 26]. Some studies introduce adversarial training to avoid generating a blurred image caused by the image reconstruction loss [31, 27, 25].

Video-to-video translation Recently, several studies have succeeded in converting high-resolution videos into other ones in a different domain. RecycleGAN [4] extends a concept of CycleGAN [58], and solves a video retargeting problem with two video datasets for which correspondences are not given. Vid2Vid [49] learns a model that maps a source domain into an output one from a pair of videos and generates high resolution videos. Contrary to these models that can be trained well with a small batch size, the generative model for videos generally requires a large batch size.

Video generation There are two major approaches in the field of video generation using GANs. One is a study for generating plausible videos by limiting a video dataset to a specific area such as human pose and face [7, 57, 53]. Another one is a study that can handle any datasets without such restriction [48, 40, 35, 47, 1]. Our study is related to the latter. VGAN [48] exploits 3D convolutional networks to generate videos. TGAN [40] first generates a set of latent vectors corresponding to each frame and then transforms them into actual images. MoCoGAN [47] produces videos more efficiently by decomposing the latent space into the motion and the content subspaces. Although these models illustrate that models using GAN are also effective in video generation, they have a problem of requiring a relatively large computational cost. Acharya *et al*. [1] proposed a method of generating high-resolution videos with ProgressiveGAN [23]. We will show several advantages of our method against ProgressiveGAN in Section 4.4.

3. Abstract map and subsampling layer

In this section, we describe the concept of an abstract map and subsampling layer. After that, we show that this concept is also applicable to multi-scale rendering with multiple subsampling layers.

3.1. Single subsampling layer

For simplicity, we first describe the training of GAN with a single subsampling layer. The training of GAN uses two types of networks called generator and discriminator. The generator G takes a noise vector $\mathbf{z}$ randomly drawn from a specified distribution p_z and yields a sample such as a video. The discriminator D takes a sample denoted by $\mathbf{x}$, and identifies whether each sample is a sample from a given dataset or the generator. The training of these networks is performed by alternately maximizing and minimizing the following objective:

$$\mathbb{E}_{\mathbf{x}\sim p_d}[\ln D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\ln(1 - D(G(\mathbf{z})))], \qquad (1)$$

where p_d is the data distribution.

In our method, the generator consists of two blocks: *abstract block* g^A and *rendering block* g^R. While the abstract block is a function that computes the latent feature map we call *abstract map*, the rendering block generates a sample from the abstract map. In the following, we assume that the computational speed of the abstract block is relatively fast, whereas that of the rendering block is much slower than the abstract block. In the inference, the generation process of samples by the generator is equivalent to the generator in the conventional GAN; that is, $G(\mathbf{z})$ can be represented by

$$\mathbf{x} = G(\mathbf{z}) = \left(g^R \circ g^A\right)(\mathbf{z}). \tag{2}$$

The conventional discriminator directly computes a score from its detailed sample, however, it causes a significant decrease in the computational speed and an increase in the amount of memory consumption in the case where the dimension of the target domain is quite large. To cope with this problem, in the training stage, we introduce a *subsampling layer* for the generator $\mathcal{S}^G$ that randomly subsamples the abstract map with a given function (*e.g.*, crop function where rectangle size is fixed but the position is randomly determined, or reduction of the frame rate) and creates a smaller one. From this smaller abstract map, the rendering block efficiently generates a smaller sample. Namely, in our method, the subsampled data $\mathbf{x}'$ generated by the generator G' can be formulated as

$$\mathbf{x}' = G'(\mathbf{z}) = \left(g^R \circ \mathcal{S}^G \circ g^A\right)(\mathbf{z}). \tag{3}$$

As the number of dimensions of $\mathbf{x}$ is reduced by $\mathcal{S}^G$, we have to modify the discriminator in Equation (1). Unlike the original discriminator that takes a detailed sample as an argument, our discriminator D' efficiently computes the score from a subsampled data. Regarding the sample in the dataset, we introduce another subsampling layer $\mathcal{S}^D$ that corresponds to $\mathcal{S}^G$ and transforms it into the subsampled data. In other words, in the training stage of our method, the objective of Equation (1) can be rewritten by

$$\mathbb{E}_{\mathbf{x}\sim p_d}[\ln D'(\mathcal{S}^D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_z}[\ln(1 - D'(G'(\mathbf{z})))]. \tag{4}$$

To generate consistent samples in both inference and training, g^R, $\mathcal{S}^G$, and $\mathcal{S}^D$ need to satisfy the following three conditions: (i) g^R can accept both the original abstract map and the reduced one, (ii) $\mathcal{S}^G$ and $\mathcal{S}^D$ are differentiable with respect to the input, and (iii) the subsampled regions of $\mathcal{S}^G$ and $\mathcal{S}^D$ are stochastically determined. For example, in the case where $\mathcal{S}^G$ and $\mathcal{S}^D$ are crop functions with a fixed rectangle, its crop position is stochastically determined. g^R needs to be a block consisting of fully-convolutional layers.

The practical advantage of introducing a subsampling layer is that it enables to train networks which generate large samples that have too high dimensionality to fit the GPU memory. Another advantage is the efficient training of a large network; choosing appropriate $\mathcal{S}^G$ and g^R, we can update the parameters much quicker than the conventional training procedures. When using a single subsampling layer, the disadvantage is that after the same number of training iterations, the performance of our method using stochastic subsampling tends to be lower than the conventional one. However, it can be alleviated by introducing multiple levels of abstract maps described in the following.

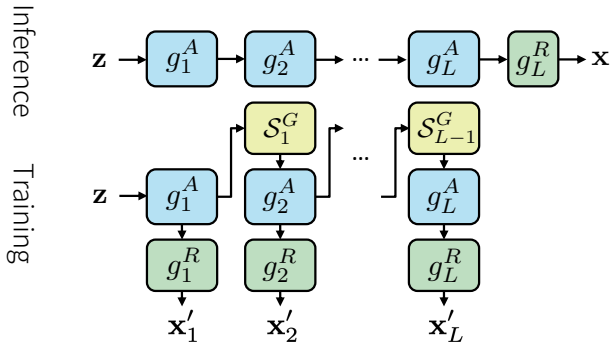


Figure 1. The generator using multiple subsampling layers.

3.2. Training with multiple subsampling layers

There exists a trade-off between the efficient computation with a subsampling layer and the quality of a generated sample. When the dimensionality of a sample comes from $\mathcal{S}$ is small, we can efficiently train the network, but the quality tends to decrease. Thus, it requires careful hyper-parameter tuning to balance the trade-off when using a single subsampling layer. However, by introducing a generator that yields several samples rendered at multiple levels (*e.g.*, multiple resolutions or multiple frame rates), we can efficiently train the network without significantly sacrificing the quality.

Here we show its basic concept in this section. Almost all generators including DCGAN [37] and SA-GAN [54] have a network structure that first yields a spatially small feature map with many channels and gradually transforms it into a larger one with fewer channels. In this case, the initial block in the generator has a large number of parameters, but its computational cost is relatively low. On the other hand, although the number of parameters in the latter block is quite smaller, it takes a lot of time for the computation. Therefore, when using the subsampling layer, reducing the abstract map in the initial block is not advantageous from the viewpoint of computational cost, while it is desirable to reduce the abstract maps in the latter blocks aggressively.

We solve this problem with multiple abstract maps and corresponding rendering blocks (Figure 1). Specifically, in our method, we introduce the generator consisting of L abstract blocks, L rendering blocks, and $L-1$ subsampling layers. We respectively denote them as g_l^A, g_l^R, and $\mathcal{S}_l^G$. In the inference, $\mathbf{x}$ can be evaluated simply by sequentially

applying abstract blocks and rendering with a single g_L^R, *i.e.*,

$$\mathbf{x} = \left(g_L^R \circ g_L^A \circ g_{L-1}^A \circ \cdots \circ g_1^A\right)(\mathbf{z}). \quad (5)$$

In the training, we exploit L generators denoted as $G'_l(\mathbf{z})$. G'_1 is a function for generating coarse samples; it makes an initial abstract map with g_1^A, and then transforms it into a coarse sample defined by $\mathbf{x}'_1$. In contrast, G'_2 first randomly subsamples the above abstract map with S_1, and creates a smaller map. After that, it is converted to another abstract map by g_2^A, and then transformed to a more detailed sample $\mathbf{x}'_2$ by g_2^R. Repeating this procedure iteratively, we finally have a set of samples at multiple levels. These procedures are also represented as follows:

$$\begin{aligned} G'_1 &= g_1^R \circ g_1^A \\ G'_2 &= g_2^R \circ g_2^A \circ \left(\mathcal{S}_1^G \circ g_1^A\right) \\ &\vdots \\ G'_L &= g_L^R \circ g_L^A \circ \left(\mathcal{S}_{L-1}^G \circ g_{L-1}^A\right) \circ \cdots \circ \left(\mathcal{S}_1^G \circ g_1^A\right). \end{aligned} \quad (6)$$

In this method, while any subsampling layers are not used to generate an initial sample, an abstract map made from a subsequent block is aggressively reduced by multiple subsampling layers. Reducing abstract maps that require a lot of computation while preserving layers that do not require much calculation, we can train the network more efficiently while keeping the computational cost low.

Multiple discriminators As the generator generates multiple samples, the discriminator also needs to take multiple samples but output a single score representing whether these samples are generated from the generator or not. We model a discriminator with multiple sub-discriminators. Specifically, let D'_l be the l-th sub-discriminator that takes a l-th sample and returns a scalar value. With this notation, the discriminator D' is represented by

$$D'(\mathbf{x}'_1, \ldots, \mathbf{x}'_L) = \sigma\left(\sum_{l=1}^{L} D'_l(\mathbf{x}'_l)\right), \quad (7)$$

where $\sigma(\cdot)$ denotes a sigmoid function.

To create $\mathbf{x}'_l$ from a sample in the dataset, we introduce other L subsampling layers described as $\mathcal{S}_l^D$. The specific algorithm of $\mathcal{S}_l^D$ depends on the contents of g_l^A and $\mathcal{S}_l^G$; for example, if all $\mathcal{S}_l^G$'s are crop functions and g_l^A contains some upscaling function, $\mathcal{S}_l^D$ consists of a stochastic crop function and a resize function.

3.3. Adaptive batch reduction

In the above discussion, it is implicitly assumed that subsampling layers are functions for reducing an abstract map per sample. However, in the actual training of neural networks, the network practically received input as a mini-batch in which a plurality of samples are concatenated. Therefore, by extending the subsampling layers in batch-axis (*i.e.*, reducing the batch size in addition to the abstract map when applying subsampling layers), we can reduce the computational cost of the latter block while increasing the batch size in the initial block. We call this method "*adaptive batch reduction*".

The method of reducing the size of mini-batch only in the latter blocks implicitly assumes that it does not need to excessively take *diversity* caused by a large mini-batch into account. Although it is difficult to prove this validity theoretically, it can be partially confirmed from the findings of existing methods so far.

As we described in Section 2, while it has been shown that the large batch size is significant for GANs [5] to improve the quality of generated images, the size of mini-batch required by conditional GANs such as Pix2Pix [20] and Vid2Vid [49] is extremely small (*e.g.*, 1 to 4). Also, the initial block in the generator has to output various feature maps from the input (this is the reason why the number of parameters of the initial block in many generators is quite large), whereas the latter blocks do not need to generate such various maps. From these viewpoints, we consider that setting the batch sizes for the initial block to be large and setting it for the latter blocks to be small have some validity as well as computational efficiency.

4. Temporal GAN v2

Based on the above discussion, we introduce the model for video generation called *Temporal GAN v2*.

4.1. Generator

Suppose a generator yielding a video with T frames and $W \times H$ pixels. Unlike other conventional models that generate videos with 3D convolutional networks [48, 35, 1], as with the TGAN [40], our generator first yields T latent feature maps from a noise vector and then transforms each map into a corresponding frame. The specific network structure is shown in Figure 2.

Here we describe the flow of the inference. Given d-dimensional noise vector $\mathbf{z}$ randomly drawn from the uniform distribution within a range of $[-1, 1]$, the generator converts it into a feature map of $(W/64) \times (H/64)$ pixels with a fully-connected layer. A recurrent block consisting of a Convolutional LSTM [42] receives this feature map as an input, and then returns another feature map with the same shape. Note that at $t = 0$ the CLSTM receives the map from $\mathbf{z}$ as input, but at $t \geq 1$ it always gets a map from a zero vector (*i.e.*, $\mathbf{z}$ is used to initialize the state of the CLSTM). After that, each feature map is transformed into another feature map with $W \times H$ pixels by six upsampling blocks, and a rendering block renders it into the frame. That is, the upsampling block is a function that outputs double the resolution

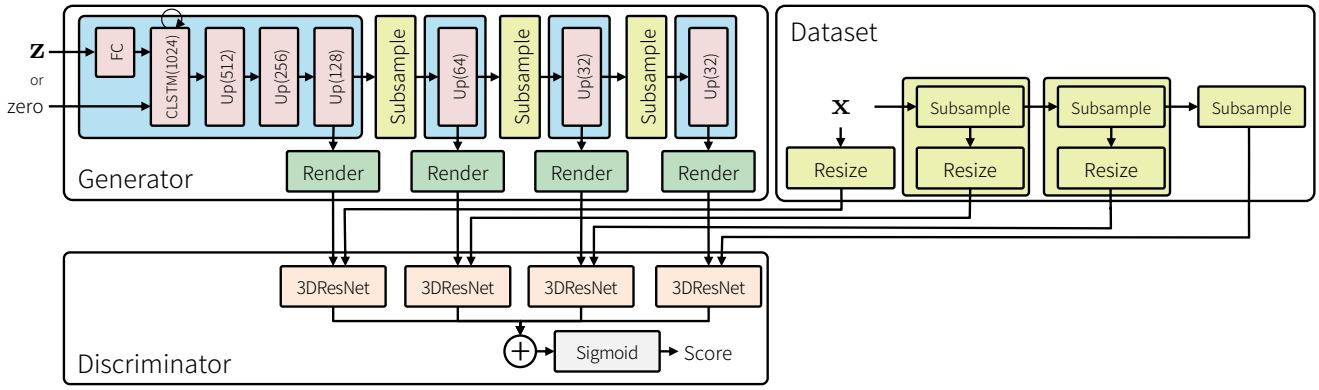


Figure 2. Network configuration of our model. "CLSTM(C)" represents the convolutional LSTM with C channels and 3×3 kernel. "Up(C)" means the upsampling block that returns a feature map with C channels and twice the resolution of the input.

of the input feature map, and the rendering block converts it to the image while maintaining the resolution. We show the network configurations of the upsampling and the rendering blocks in the supplementary material.

In the training stage, the generator progressively reduces the size of abstract maps with three subsampling layers. Each layer has a role of simultaneously reducing batch size and the number of frames. Suppose abstract map $\mathbf{h}$ with a shape of $(N_h \times C_h \times T_h \times H_h \times W_h)$, where each variable denotes batch size, channels, number of frames, height, and width. We represent each element of this tensor by $h_{n,c,t,h,w}$. The subsampling layer reduces the shape of $\mathbf{h}$ to $(\lceil N_h/s_n \rceil \times C_h \times \lceil T_h/s_t \rceil \times H_h \times W_h)$; specifically, we define output $\mathbf{h}'$ reduced by the subsampling layer as

$$h'_{m,c,\tau,h,w} = \sum_{n,t} B(m,\tau,n,t) h_{n,c,t,h,w}, \tag{8}$$

where B is a Boolean function denoted by

$$B(m,\tau,n,t) = \begin{cases} 1 & \text{if } \ m = ns_n \text{ and } \tau = ts_t + b_t \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

$b_t \sim \mathcal{U}\{0, s_t - 1\}$ is an offset value and randomly drawn from a discrete uniform distribution. Note that using the slicing notation of NumPy [36], the above equation is simply represented by `hd = h[::sn, :, bt::st]`.

We used $s_n = 2$ and $s_t = 2$ in all the experiments; that is, the dimensionality of the four rendered videos are almost equivalent since the upsampling block doubles width and height while the number of frames and the batch size are halved. This strategy to make the size of rendered tensor constant also has the advantage that the total amount of consumed memory increases only linearly, even if the resolution of the generated video is doubled.

4.2. Discriminator

As we described in Section 3.2, our discriminator consists of several sub-discriminators. In the experiments, we use four 3D ResNet models, where each model has several spatiotemporal three-dimensional residual blocks [16, 17] and one fully-connected layer. The network configuration of 3D ResNet itself is almost the same as the discriminator used in Miyato *et al*. [33] except the kernel of all convolutional layers is replaced with (3×3) to $(3 \times 3 \times 3)$. The detail of the network configuration of the 3D ResNet is given in our supplementary material.

Although all sub-discriminators have the same network architecture, it is considered that those roles are different. The initial sub-discriminator refers to the overall action from all the frames in a given video, whereas the last one focuses on improving the quality by looking at only a few frames stochastically extracted from the video.

4.3. Dataset

To make four subsampled videos from an example in the dataset, we introduce three subsampling functions. For the sample used in the initial sub-discriminator, we only apply a resize function to lower the resolution by one eighth. On the other hand, we apply three subsampling layers defined in Equation (8) to the input example to create an input for the last sub-discriminator. It reduces the number of frames to one eighth while maintaining the resolution.

4.4. Difference from other existing methods

StackGAN++ Although our approach and its network structure is similar to StackGAN++ [55], the most significant difference lies in the existence of sampling layers; it is useful for problems where the number of dimensions in a sample is quite large. In particular, we consider that the adaptive batch reduction is also applicable to other generation problems with StackGAN++-like models. Another difference is that our method does not require problem-specific heuristics such as a color-consistency regularization to stabilize the training.

ProgressiveGAN Our method has two advantages compared to ProgressiveGAN [23, 1]. The first advantage is that our method does not require hyperparameters to grow the

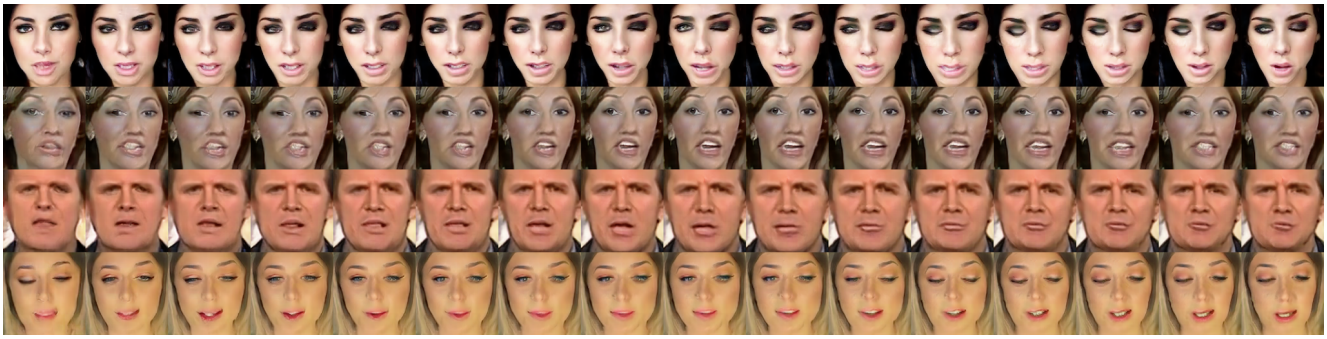


Figure 3. Example of videos generated by TGANv2. Due to the size of the paper, each frame is resized to 128×128 pixels.

network according to the number of iterations dynamically. It also means that it does not need to dynamically change the batch size according to the number of iterations. The second is that our method can generate a large sample from the beginning regardless of the number of iterations. It is useful for evaluating inception score [41] using a pre-trained model accepting only a sample with a fixed shape.

4.5. Regularizer

To improve the performance of the generator, we add a zero-centered gradient penalty with regard to the data distribution [32] to the loss of the discriminator. This regularization term R_1 can be formulated as follows:

$$R_1 = \lambda \sum_{l=1}^{L} \sum_{i=1}^{n_l} \|\nabla D'_l(\mathbf{x}'_l)\|^2, \tag{10}$$

where λ is a weight and n_l represents the size of mini-batch at level l.

Unlike the original definition of [32], Equation (10) uses summation over batch size instead of average ($\mathbb{E}_{\mathbf{x} \sim p_d}[\cdot]$). This is because all the mini-batches now have the same number of elements, so we expect that each gradient will be propagated equally by the summation operator.

5. Experiments

5.1. Datasets

UCF101 UCF101 is a common video dataset that contains 13,320 videos with 320×240 pixels and 101 different sport categories such as *Baseball Pitch* [43]. In the experiments, we randomly extracted 16 frames from the training dataset, cropped a rectangle with 240×240 pixels from the center, resized it to 192×192 pixels, and used for training. The values of all the samples are normalized to $[-1, 1]$. To amplify the samples we randomly flipped video during the training.

FaceForensics Following to Wang *et al*. [49], we created the facial videos from FaceForensics [39] containing 854 news videos with different reporters. Specifically, we first identified the position of the face with a mask video in the dataset, cropped only the area of the face, and resized it to 256×256 pixels. In training, we randomly extracted 16 frames from them and sent it to the discriminator. As with UCF101 dataset, all the values are normalized to $[-1, 1]$.

5.2. Hyperparameters and training statistics

We used the Adam [24] optimizer with the learning rate of 1.0×10^{-4}, decayed linearly to 0 over 100K iterations. We also employed $\beta_1 = 0.0$ and $\beta_2 = 0.9$ in the Adam optimizer. The batch size was selected so it fills the GPU memory of NVIDIA Tesla P100 (12Gb). Since the actual batch size depends on the experiments, we will describe each value in the following. The number of updates of the discriminator for each iteration was set to one. The number of dimensions of $\mathbf{z}$ was set to 256, and λ in Equation (10) was 0.5. We implemented all models using Chainer [45] and ChainerMN [2], and run almost all the experiments with data parallelism across four GPUs.

The total number of parameters in our model is about 2.0×10^8, which is larger than the number of parameters used in BigGAN (1.6×10^8) [5]. The total training time consumed in quantitative experiments with the UCF101 dataset was about three days including the time for taking snapshots and calculating inception scores.

5.3. Qualitative experiments

Using the FaceForensics [39], we confirmed our model successfully generated videos with high resolution (256×256 pixels). We used eight GPUs in this experiment and set the batch size in the initial abstract map per GPU to 8, which means that the total batch size is 64, and it gradually decreases to 32, 16, and 8 as the level goes deeper.

The generated videos are shown in Figure 3. It shows that our model successfully generated facial videos with high-resolution without causing a large mode collapse. One of the main reasons why it could produce such various videos even if the dataset has high resolution videos is that our method allows large batch size in the beginning part of the generator. For example, even if the resolution was only 64×64 pixels, the batch sizes of videos used in the MoCoGAN [47] and the TGAN [40] were 3 and 8, respectively. Although we used eight GPUs to set the batch size to 64, we also confirmed

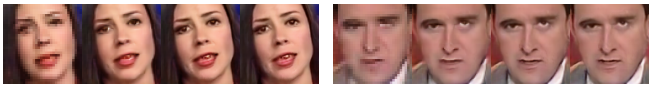


Figure 4. Example of videos generated at different levels. The leftmost image represents the frame of the video generated by the initial rendering block, and the rightmost one denotes the final generated image.

Method	N_1	N_2	N_3	N_4
TGANv2(bs=64)	64	32	16	8
- frame subsampling only	8	8	8	8
- adaptive batch reduction only	8	4	4	4
- without subsampling layers	4	4	4	4
TGANv2(bs=256)	256	128	64	32

Table 1. A list of batch sizes used in each block. N_l is the batch size at level l. "bs" is a batch size used in the initial abstract map.

that our method was able to train models with similar quality even when only four GPUs were used.

To check whether each rendering block generates the same content under the same $\mathbf{z}$, we visualized the result of each rendering block in Figure 4. Interestingly, every rendering block in our method correctly generated the same content even though it does not perform a color-consistency regularization introduced in StackGAN++ [55].

5.4. Quantitative experiments

We also confirmed that the quality of samples generated by the proposed model is superior to other existing models with Inception Score (IS) [41] and Fréchet Inception Distance (FID) [18], which measure the quality of generated samples. We describe the details of the experimental environment used for measuring IS and FID in the following.

As with other comparable methods [40, 47, 1], we employed the UCF101 dataset [43] for the quantitative experiments. Regarding the classifier used for computing IS and FID, we used a pre-trained model of Convolutional 3D (C3D) network [46], which was trained with Sports-1M dataset [22] and then fine-tuned on the UCF101 dataset[1]. We used this pre-trained model as is, and did not update any parameters in the classifier. As the resolution and the number of frames in a video used in the classifier are 128×128 pixels and 16 frames, we resized the generated video with 192×192 pixels to 128×128 pixels and regarded it as the input of the classifier. 2,048 samples were used for computing IS and FID. The standard deviation was calculated by performing the same procedure 10 times.

In comparative methods, we employed the following existing methods: VGAN [48], TGAN [40], MoCoGAN [47], ProgressiveVGAN [1], and ProgressiveVGAN with a Sliced Wasserstein Loss (SWL). The inception scores for the VGAN and the TGAN were obtained from the values

[1] The pre-trained model itself can be downloaded from http://vlg.cs.dartmouth.edu/c3d/

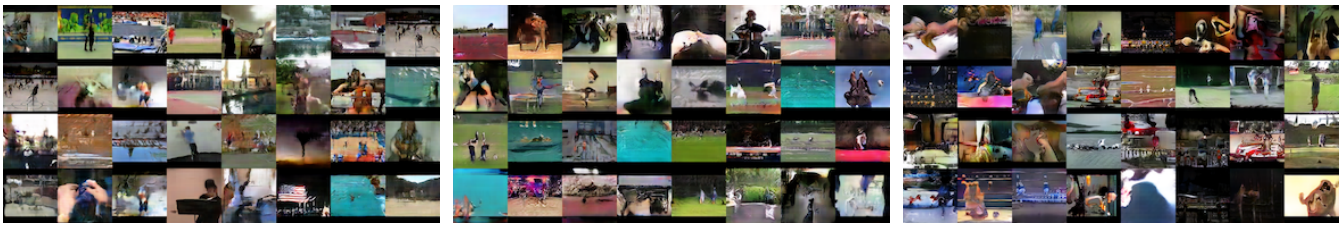
Method	IS	FID
VGAN [48]	$8.31 \pm .09$	
TGAN [40]	$11.85 \pm .07$	
MoCoGAN [47]	$12.42 \pm .03$	
ProgressiveVGAN [1]	$13.59 \pm .07$	
ProgressiveVGAN w/ SWL [1]	$14.56 \pm .05$	
TGANv2(bs = 64)	$\mathbf{22.70 \pm .19}$	3591 ± 24
- frame subsampling only	$21.01 \pm .30$	3723 ± 20
- adaptive batch reduction only	$14.30 \pm .22$	4525 ± 26
- without subsampling layers	$12.10 \pm .10$	5222 ± 20
TGANv2(bs = 256)	$\mathbf{24.34 \pm .35}$	3620 ± 26

Table 2. Inception Score and Fréchet Inception Distance on UCF101 dataset.

reported in [40]. We also used the inception scores for the MoCoGAN and PVGAN reported from the original paper. The VGAN, the TGAN, and the MoCoGAN generate videos with 64×64 pixels, whereas the resolution of videos generated by ProgressiveVGAN is 128×128 pixels. It should be noted that since all the inception scores shown in their papers were computed in the same procedure as those used in this quantitative experiments (*i.e.*, all the generated videos are resized to 128×128 pixels and then sent to the pre-trained classifier), we can compare with these inception scores regardless of resolution. As described above, the batch size in single GPU was selected to fit each GPU memory. In the experiments with UCF101 dataset having 192×192 pixels, we set the batch size in the initial abstract map per GPU to 16. It is equivalent to setting the batch size to 64 in total.

To confirm the effect of the sampling layers, we also conducted similar experiments under the following three conditions. One is a naive model that consists of four abstract blocks and four rendering blocks but does not introduce any sampling layers (in this sense, the approach of this model is somewhat similar to the StackGAN++ [55]). The second one is a model with three subsampling layers that apply the adaptive batch reduction but do not perform subsampling of the frame. The last one is a model in which each subsampling layer only reduces the number of frames. We respectively set the batch size of the model without subsampling layers, the model with the adaptive batch reduction only, and the model with frame subsampling only to 1, 2, 4. These numbers are significantly smaller than batch size 16 used in the proposed model. Note that when the batch size in each GPU is only one, the subsampling layer does not perform the batch reduction. We summarize the batch size used in each abstract block of each model in Table 1. The authors of BigGAN [5] reported that increasing the batch size significantly increased the inception score. To measure this effect, we also ran additional experiments regarding our proposed model with 16 GPUs (*i.e.*, we set the total batch size to 256).

The quantitative results are shown in Table 2. It indicates that the IS and the FID computed by our model with four GPUs improved over any other existing methods. In par-



(a) Number of samples = 101 (b) Number of samples = 505 (c) Full

Figure 5. Changes of generated samples when changing the number of samples used for training.

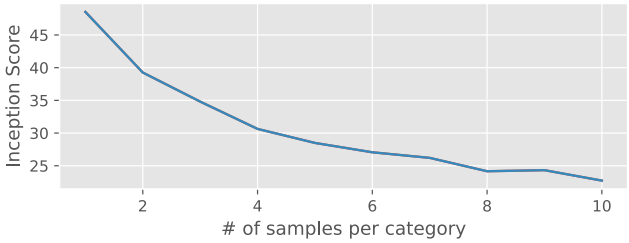


Figure 6. Transition of inception score when the number of samples per category is limited. As the number of categories in the dataset is 101, the total number of samples used for training is $101 \times N$.

ticular, the IS of our model is about 83% higher than the MoCoGAN [47], which was published in 2018 and reported the 5% higher score than the TGAN in 2017 [40]. It shows that our proposed method succeeded in generating higher quality videos by efficiently training the large model with the large batch size. It should be noted that this score cannot be achieved by simply making the output of the generator high resolution. For example, the resolution of the ProgressiveVGAN is 128×128 pixels, which matches the resolution of the pre-trained classifier, but its reported score is 36% lower than the proposed method. It implies that our method using subsampling layers is more effective for training the large GAN dealing with videos. We also observed that the IS of our model could be improved by increasing the total batch size from 64 to 256. Although there was no significant improvement in FID (it implies that increasing the batch size improved the image quality itself, but did not improve diversity), this finding is consistent with that of the BigGAN to some extent.

The effectiveness of subsampling layers can also be confirmed from the model where subsampling layers were not used. Especially, the inception score of the model without subsampling layers was almost the same as that of MoCoGAN. This suggests the difficulty of training in high-resolution samples. It is particularly interesting to see that while the model only performing the batch reduction is the same as the naive model except the batch size in the initial block (*c.f*. Table 1), its FID and IS were significantly improved. It implies that the batch size in the initial block is vital in training. Table 2 also indicates that frame subsampling contributed most to the improvement of the IS and the FID. We inferred that this is because our method succeeded in making each discriminator have a different role by introducing the frame subsampling (*c.f*. Section 4.2).

5.5. Experiments of memory capacity

Although our model succeeded in generating more detailed shapes such as person's shadow than those in TGAN and Progressive VGAN, the generated video itself still has room for improvement. Based on the background where this problem was not clearly explained in the existing studies, in this section we discuss the hypothesis that the main reason is due to the *memory capacity*. Unlike the FaceForensics [39] containing many similar parts, the UCF101 dataset has many diverse videos that do not include overlapping clips. Thus, measuring the inception score when reducing the number of videos per category, we can roughly estimate the memory capacity in the model.

The results are shown in Figures 5 and 6. When the number of samples per category is small, the quality of the generated video is clearly higher than that trained with the original dataset. It means that the model has potential power of expression to generate natural videos. The score gradually decreased as the number of samples gradually increased, and it almost converged to a constant when the number of samples exceeded 1,000. This implies that the problem of the current model is primarily due to memory capacity.

The generator for videos clearly needs more memory capacity than that for still images. When linearly moving $\mathbf{z}$ of the GAN for still images, a created image is one like a morphing connecting two images smoothly. We consider that our model needs more memory capacity in order to support many essential transformations that cannot be created by such morphing. Considering that the current parameter size in the generator is only 312MB while the uncompressed UCF101 dataset used for training is about 200GB, we predict it needs to drastically increase the number of parameters to generate high fidelity natural videos.

6. Summary

We proposed the method for efficiently training the model that generates large samples by introducing subsampling layers. We also developed the extended method with multiple subsampling layers. Through experiments of our model (TGANv2), we confirmed its effectiveness.

Acknowledgements First, we would like to acknowledge Takeru Miyato, Sosuke Kobayashi, and Shoichiro Yamaguchi for helpful discussions. We also would like to thank the developers of Chainer [45, 2].

References

[1] D. Acharya, Z. Huang, D. P. Paudel, and L. V. Gool. Towards High Resolution Video Generation with Progressive Growing of Sliced Wasserstein GANs. In *Arxiv preprint arXiv:1810.02419*, 2018. 2, 4, 5, 7

[2] T. Akiba, K. Fukuda, and S. Suzuki. ChainerMN: Scalable Distributed Deep Learning Framework. In *Proceedings of Workshop on ML Systems in NIPS*, 2017. 6, 9

[3] M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell, and S. Levine. Stochastic Variational Video Prediction. In *ICLR*, 2018. 2

[4] A. Bansal, S. Ma, D. Ramanan, and Y. Sheikh. Recycle-GAN: Unsupervised Video Retargeting. In *ECCV*, 2018. 2

[5] A. Brock, J. Donahue, and K. Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *Arxiv preprint arXiv:1809.11096*, 2018. 1, 2, 4, 6, 7, 11

[6] W. Byeon, Q. Wang, R. K. Srivastava, and P. Koumoutsakos. ContextVP: Fully Context-Aware Video Prediction. In *ECCV*, 2018. 2

[7] H. Cai, C. Bai, Y.-W. Tai, and C.-K. Tang. Deep Video Generation, Prediction and Completion of Human Action Sequences. In *ECCV*, 2018. 2

[8] E. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep Generative Image Models Using a Laplacian Pyramid of Adversarial Networks. In *NIPS*, 2015. 2

[9] E. Denton and R. Fergus. Stochastic Video Generation with a Learned Prior. In *Arxiv preprint arXiv:1802.07687*, 2018. 2

[10] F. Ebert, C. Finn, A. X. Lee, and S. Levine. Self-Supervised Visual Planning with Temporal Skip Connections. In *Conference on Robot Learning (CoRL)*, 2017. 2

[11] C. Finn, I. Goodfellow, and S. Levine. Unsupervised Learning for Physical Interaction through Video Prediction. In *NIPS*, 2016. 1, 2

[12] I. Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. In *Arxiv preprint arXiv:1701.00160*, 2016. 1

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In *NIPS*, 2014. 1, 2

[14] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved Training of Wasserstein GANs. In *NIPS*, 2017. 11

[15] Z. Hao, X. Huang, and S. Belongie. Controllable Video Generation with Sparse Trajectories. In *CVPR*, 2018. 2

[16] K. Hara, H. Kataoka, and Y. Satoh. Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet? In *CVPR*, 2018. 5

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 5

[18] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *NIPS*, 2017. 7

[19] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz. Multimodal Unsupervised Image-to-Image Translation. In *ECCV*, 2018. 2

[20] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *CVPR*, 2017. 1, 2, 4

[21] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. Video Pixel Networks. In *arxiv preprint arXiv:1610.00527*, 2016. 1, 2

[22] A. Karpathy, S. Shetty, G. Toderici, R. Sukthankar, T. Leung, and Li Fei-Fei. Large-scale Video Classification with Convolutional Neural Networks. In *CVPR*, 2014. 7

[23] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *ICLR*, 2018. 1, 2, 5

[24] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 6

[25] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine. Stochastic Adversarial Video Prediction. In *Arxiv preprint arXiv:1804.01523*, 2018. 2

[26] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Flow-Grounded Spatial-Temporal Video Prediction from Still Images. In *ECCV*, 2018. 2

[27] X. Liang, L. Lee, W. Dai, and E. P. Xing. Dual Motion GAN for Future-Flow Embedded Video Prediction. In *ICCV*, 2017. 2

[28] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised Image-to-Image Translation Networks. In *NIPS*, 2017. 2

[29] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video Frame Synthesis using Deep Voxel Flow. In *ICCV*, 2017. 2

[30] W. Lotter, G. Kreiman, and D. Cox. Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning. In *ICLR*, 2017. 2

[31] M. Mathieu, C. Couprie, and Y. LeCun. Deep Multi-Scale Video Prediction beyond Mean Square Error. In *ICLR*, 2016. 1, 2

[32] L. Mescheder, S. Nowozin, and A. Geiger. Which Training Methods for GANs do actually Converge? In *ICML*, 2018. 1, 6, 11

[33] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral Normalization for Generative Adversarial Networks. In *ICLR*, 2018. 1, 2, 5, 11

[34] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh. Action-Conditional Video Prediction using Deep Networks in Atari Games. In *NIPS*, 2015. 2

[35] K. Ohnishi, S. Yamamoto, Y. Ushiku, and T. Harada. Hierarchical Video Generation from Orthogonal Information: Optical Flow and Texture. In *AAAI*, 2018. 2, 4

[36] T. E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition, 2015. 5

[37] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *ICLR*, 2016. 2, 3

[38] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (Language) Modeling: A Baseline for Generative Models of Natural Videos. *arXiv preprint arXiv:1412.6604*, 2014. 2

[39] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner. FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces. In *Arxiv preprint arXiv:1803.09179*, 2018. 6, 8

[40] M. Saito, E. Matsumoto, and S. Saito. Temporal Generative Adversarial Nets with Singular Value Clipping. In *ICCV*, 2017. 1, 2, 4, 6, 7, 8, 11

[41] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved Techniques for Training GANs. In *NIPS*, 2016. 6, 7

[42] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In *NIPS*, 2015. 4

[43] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. *arXiv preprint arXiv:1212.0402*, 2012. 2, 6, 7

[44] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised Learning of Video Representations using LSTMs. In *ICML*, 2015. 2

[45] S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a Next-Generation Open Source Framework for Deep Learning. In *Proceedings of Workshop on Machine Learning Systems in NIPS*, 2015. 6, 9

[46] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning Spatiotemporal Features with 3D Convolutional Networks. In *ICCV*, 2015. 7

[47] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. MoCoGAN: Decomposing Motion and Content for Video Generation. In *CVPR*, 2018. 1, 2, 6, 7, 8

[48] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating Videos with Scene Dynamics. In *NIPS*, 2016. 1, 2, 4, 7

[49] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz, and B. Catanzaro. Video-to-Video Synthesis. In *Arxiv preprint arXiv:1808.06601*, 2018. 1, 2, 4, 6

[50] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In *CVPR*, 2018. 1, 2

[51] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local Neural Networks. In *CVPR*, 2018. 2

[52] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *NIPS*, 2016. 1

[53] C. Yang, Z. Wang, X. Zhu, C. Huang, J. Shi, and D. Lin. Pose Guided Human Video Generation. In *ECCV*, 2018. 2

[54] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-Attention Generative Adversarial Networks. In *Arxiv preprint arXiv:1805.08318*, 2018. 1, 2, 3

[55] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas. StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. In *arXiv preprint arXiv:1710.10916*, 2017. 2, 5, 7

[56] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. In *ICCV*, 2017. 2

[57] L. Zhao, X. Peng, Y. Tian, M. Kapadia, and D. Metaxas. Learning to Forecast and Refine Residual Motion for Image-to-Video Generation. In *ECCV*, 2018. 2

[58] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *ICCV*, 2017. 2

Supplementary Note

A. Network configuration

We show the detail of the network components in Figure 7. As we described in our main paper, the network configuration of 3D ResNet is almost the same as the discriminator used in Miyato *et al*. [33] except the kernel of all convolutional layers (3×3) are replaced with $(3\times3\times3)$. In the initializers, we use the GlorotUniform initializer with a scale $\sqrt{2}$ in the residual paths of both the "Up(C)" and the "Down(C)", and the normal GlorotUniform initializer in the shortcut paths.

B. Additional results

As with the FaceForensics dataset in our main paper, we confirmed that our model successfully generated the videos with high resolution in the UCF101 dataset. The results are shown in Figure 8. It shows that the qualitative quality is clearly better than that of the videos shown in the project page of TGAN [40]. Regarding quantitative results, we have already shown in the main paper.

C. Trials and their results

Layers and parameters The behaviour when increasing the number of layers of the generator and the discriminator in our model was consistent with the findings of BigGAN [5]. That is, although we increased the number of layers of both the generator and the discriminator, we observed that the inception score of generate samples was lower than that of our main paper. We also observed that simply increasing the number of channels in each layer contributed the most improvement in the inception score.

CLSTM layers We tried stacked Convolutional LSTM layers but observed that a single LSTM with many channels worked better. We also tried a dilated CLSTM layer but the plain CLSTM was better. As with the above, we observed that simply increasing the number of channels in the single CLSTM contributed the most improvement in the score.

Generation of longer videos We observed that our model may be able to stably generate longer videos than the number of frames used for training (*i.e*., 16 frames), while some other methods may not. Although this behavior depends on the hyperparameters and the dataset, it is unclear under what conditions our model can generate stably[2].

Spectral normalization We tried to insert the Spectral Normalization [33] into the 3D convolutional layers in the discriminator, but its performance decreased significantly.

[2]Note that we mention videos after 16 frames. Up to 16 frames, our model has succeeded in generating stable videos regardless of the dataset.

Gradient penalty Although we initially used a gradient penalty based on WGAN-GP [14] as a regularizer, but later adopted a simpler zero-centered gradient penalty. Through the grid search we finally confirmed that $\lambda = 0.5$ is the best, but also observed that the training itself can be performed normally without the gradient penalty. It may indicate that the method using multiple discriminators contributed the stabilization of training.

Batch normalization Mescheder *et al*. [32], the authors of the zero-centered gradient penalty, reported that they succeeded in generating high fidelity images without Batch Normalization (BN) layers in the generator, but in our experiments without BN the performance significantly decreased.

3D discriminator The inception score when using a simple discriminator consisting of several 3D convolutional layers without any residual blocks is lower than the 3D ResNet discriminator, but the computational speed is faster. We used this simple discriminator temporarily for making the trial-and-error loops for searching a good model faster, and adopted the 3D ResNet for the final evaluation.

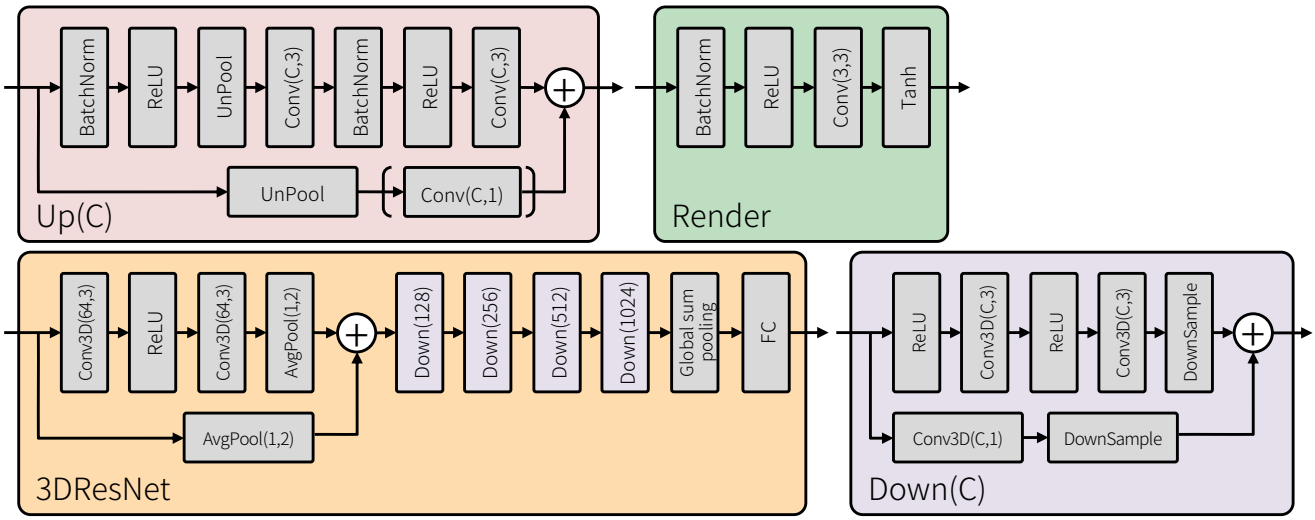


Figure 7. Details of the blocks used in the main paper. "Conv(C, k)" denotes a 2D convolutional layer with C channels and $(k \times k)$ kernel. "Conv3D(C, k)" denotes a 3D convolutional layer with C channels and $(k \times k \times k)$ kernel. "UnPool" denotes a 2D unpooling layer with (2×2) kernel and stride 2. "AvgPool2D" denotes an average pooling layer along the spatial dimensions with (2×2) kernel and stride 2. Note that it does not perform pooling operation along the temporal dimension. "DownSample" means the downsampling operator. If the size of each dimension of the input 3D feature maps is larger than one, this operator performs the average pooling along its axis (if the size is odd when performing the average pooling, the padding of the target axis is set to one). Otherwise, average pooling is not performed for that axis. $(\cdot)$ in "Up(C)" means that the blocks in the bracket are not inserted if the number of input channels is equivalent to that of output channels.

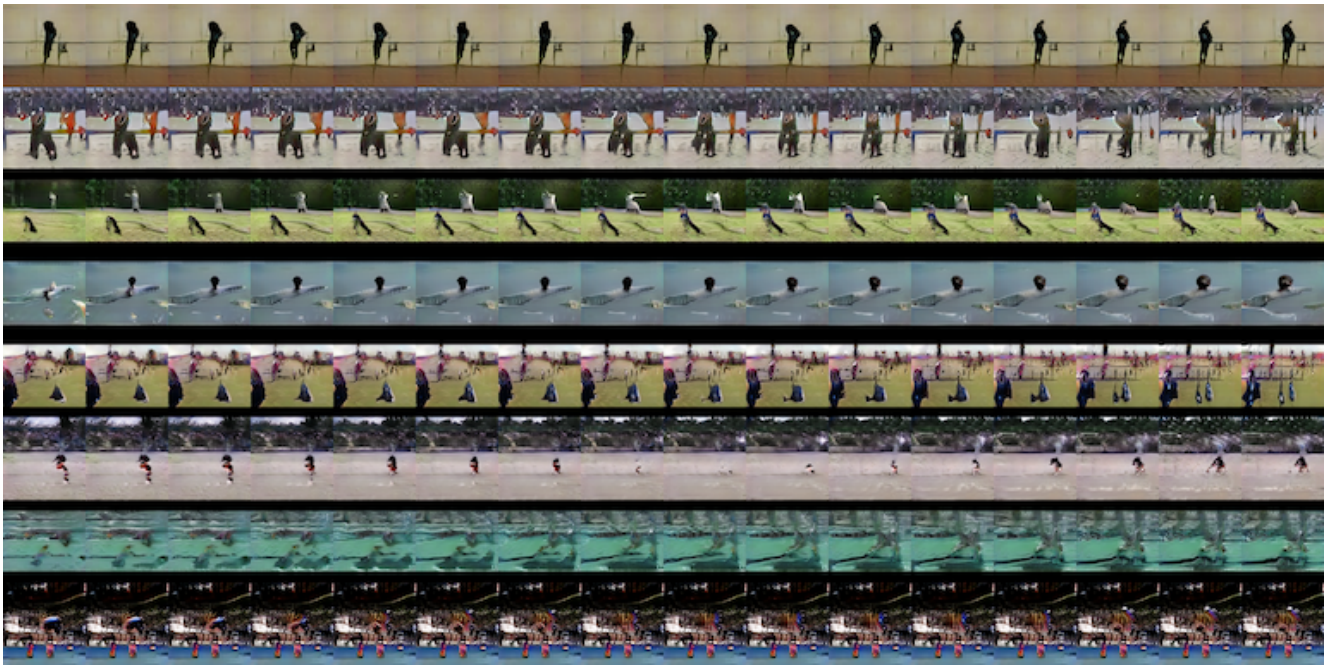


Figure 8. Examples of generated videos with UCF101. Due to the size of the paper, each frame is resized to 48×48 pixels (originally 192×192).