Conditional Generative Adversarial Nets

Mehdi Mirza
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7
mirzamom@iro.umontreal.ca

Simon Osindero
Flickr / Yahoo Inc.
San Francisco, CA 94103
osindero@yahoo-inc.com

Abstract

Generative Adversarial Nets [8] were recently introduced as a novel way to train generative models. In this work we introduce the conditional version of generative adversarial nets, which can be constructed by simply feeding the data, y, we wish to condition on to both the generator and discriminator. We show that this model can generate MNIST digits conditioned on class labels. We also illustrate how this model could be used to learn a multi-modal model, and provide preliminary examples of an application to image tagging in which we demonstrate how this approach can generate descriptive tags which are not part of training labels.

1 Introduction

Generative adversarial nets were recently introduced as an alternative framework for training generative models in order to sidestep the difficulty of approximating many intractable probabilistic computations.

Adversarial nets have the advantages that Markov chains are never needed, only backpropagation is used to obtain gradients, no inference is required during learning, and a wide variety of factors and interactions can easily be incorporated into the model.

Furthermore, as demonstrated in [8], it can produce state of the art log-likelihood estimates and realistic samples.

In an unconditioned generative model, there is no control on modes of the data being generated. However, by conditioning the model on additional information it is possible to direct the data generation process. Such conditioning could be based on class labels, on some part of data for inpainting like [5], or even on data from different modality.

In this work we show how can we construct the conditional adversarial net. And for empirical results we demonstrate two set of experiment. One on MNIST digit data set conditioned on class labels and one on MIR Flickr 25,000 dataset [10] for multi-modal learning.

2 Related Work

2.1 Multi-modal Learning For Image Labelling

Despite the many recent successes of supervised neural networks (and convolutional networks in particular) [13, 17], it remains challenging to scale such models to accommodate an extremely large number of predicted output categories. A second issue is that much of the work to date has focused on learning one-to-one mappings from input to output. However, many interesting problems are more naturally thought of as a probabilistic one-to-many mapping. For instance in the case of image labeling there may be many different tags that could appropriately applied to a given image, and different (human) annotators may use different (but typically synonymous or related) terms to describe the same image.

One way to help address the first issue is to leverage additional information from other modalities: for instance, by using natural language corpora to learn a vector representation for labels in which geometric relations are semantically meaningful. When making predictions in such spaces, we benefit from the fact that when prediction errors we are still often 'close' to the truth (e.g. predicting 'table' instead of 'chair'), and also from the fact that we can naturally make predictive generalizations to labels that were not seen during training time. Works such as [3] have shown that even a simple linear mapping from image feature-space to word-representation-space can yield improved classification performance.

One way to address the second problem is to use a conditional probabilistic generative model, the input is taken to be the conditioning variable and the one-to-many mapping is instantiated as a conditional predictive distribution.

[16] take a similar approach to this problem, and train a multi-modal Deep Boltzmann Machine on the MIR Flickr 25,000 dataset as we do in this work.

Additionally, in [12] the authors show how to train a supervised multi-modal neural language model, and they are able to generate descriptive sentence for images.

3 Conditional Adversarial Nets

3.1 Generative Adversarial Nets

Generative adversarial nets were recently introduced as a novel way to train a generative model. They consists of two 'adversarial' models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. Both G and D could be a non-linear mapping function, such as a multi-layer perceptron.

To learn a generator distribution p_g over data data $\boldsymbol{x}$, the generator builds a mapping function from a prior noise distribution $p_z(z)$ to data space as $G(z;\theta_g)$. And the discriminator, $D(x;\theta_d)$, outputs a single scalar representing the probability that $\boldsymbol{x}$ came form training data rather than p_g.

G and D are both trained simultaneously: we adjust parameters for G to minimize $log(1 - D(G(\boldsymbol{z}))$ and adjust parameters for D to minimize $logD(X)$, as if they are following the two-player min-max game with value function $V(G, D)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \tag{1}$$

3.2 Conditional Adversarial Nets

Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information $\boldsymbol{y}$. $\boldsymbol{y}$ could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding $\boldsymbol{y}$ into the both the discriminator and generator as additional input layer.

In the generator the prior input noise $p_{\boldsymbol{z}}(\boldsymbol{z})$, and $\boldsymbol{y}$ are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. [1]

In the discriminator $\boldsymbol{x}$ and $\boldsymbol{y}$ are presented as inputs and to a discriminative function (embodied again by a MLP in this case).

The objective function of a two-player minimax game would be as Eq 2

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))]. \tag{2}$$

Fig 1 illustrates the structure of a simple conditional adversarial net.

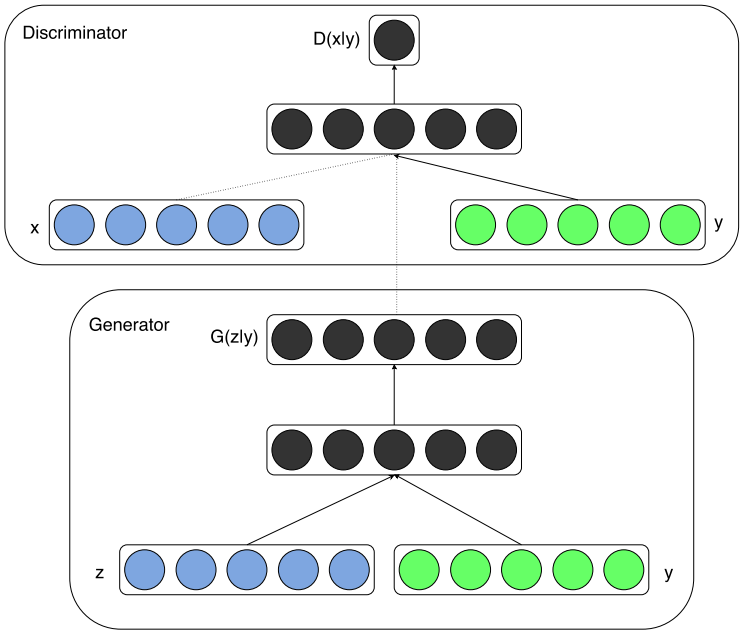


Figure 1: Conditional adversarial net

4 Experimental Results

4.1 Unimodal

We trained a conditional adversarial net on MNIST images conditioned on their class labels, encoded as one-hot vectors.

In the generator net, a noise prior $\boldsymbol{z}$ with dimensionality 100 was drawn from a uniform distribution within the unit hypercube. Both $\boldsymbol{z}$ and $\boldsymbol{y}$ are mapped to hidden layers with Rectified Linear Unit (ReLu) activation [4, 11], with layer sizes 200 and 1000 respectively, before both being mapped to second, combined hidden ReLu layer of dimensionality 1200. We then have a final sigmoid unit layer as our output for generating the 784-dimensional MNIST samples.

[1] For now we simply have the conditioning input and prior noise as inputs to a single hidden layer of a MLP, but one could imagine using higher order interactions allowing for complex generation mechanisms that would be extremely difficult to work with in a traditional generative framework.

Model	MNIST
DBN [1]	138 ± 2
Stacked CAE [1]	121 ± 1.6
Deep GSN [2]	214 ± 1.1
Adversarial nets	225 ± 2
Conditional adversarial nets	132 ± 1.8

Table 1: Parzen window-based log-likelihood estimates for MNIST. We followed the same procedure as [8] for computing these values.

The discriminator maps $\boldsymbol{x}$ to a maxout [6] layer with 240 units and 5 pieces, and $\boldsymbol{y}$ to a maxout layer with 50 units and 5 pieces. Both of the hidden layers mapped to a joint maxout layer with 240 units and 4 pieces before being fed to the sigmoid layer. (The precise architecture of the discriminator is not critical as long as it has sufficient power; we have found that maxout units are typically well suited to the task.)

The model was trained using stochastic gradient decent with mini-batches of size 100 and initial learning rate of 0.1 which was exponentially decreased down to $.000001$ with decay factor of 1.00004. Also momentum was used with initial value of $.5$ which was increased up to 0.7. Dropout [9] with probability of 0.5 was applied to both the generator and discriminator. And best estimate of log-likelihood on the validation set was used as stopping point.

Table 1 shows Gaussian Parzen window log-likelihood estimate for the MNIST dataset test data. 1000 samples were drawn from each 10 class and a Gaussian Parzen window was fitted to these samples. We then estimate the log-likelihood of the test set using the Parzen window distribution. (See [8] for more details of how this estimate is constructed.)

The conditional adversarial net results that we present are comparable with some other network based, but are outperformed by several other approaches – including non-conditional adversarial nets. We present these results more as a proof-of-concept than as demonstration of efficacy, and believe that with further exploration of hyper-parameter space and architecture that the conditional model should match or exceed the non-conditional results.

Fig 2 shows some of the generated samples. Each row is conditioned on one label and each column is a different generated sample.

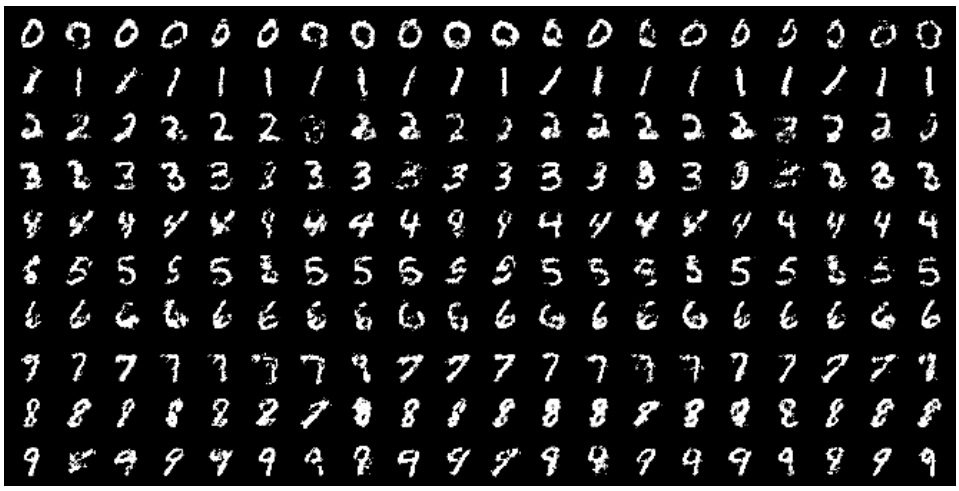


Figure 2: Generated MNIST digits, each row conditioned on one label

4.2 Multimodal

Photo sites such as Flickr are a rich source of labeled data in the form of images and their associated user-generated metadata (UGM) — in particular user-tags.

User-generated metadata differ from more 'canonical' image labelling schems in that they are typically more descriptive, and are semantically much closer to how humans describe images with natural language rather than just identifying the objects present in an image. Another aspect of UGM is that synoymy is prevalent and different users may use different vocabulary to describe the same concepts — consequently, having an efficient way to normalize these labels becomes important. Conceptual word embeddings [14] can be very useful here since related concepts end up being represented by similar vectors.

In this section we demonstrate automated tagging of images, with multi-label predictions, using conditional adversarial nets to generate a (possibly multi-modal) distribution of tag-vectors conditional on image features.

For image features we pre-train a convolutional model similar to the one from [13] on the full ImageNet dataset with 21,000 labels [15]. We use the output of the last fully connected layer with 4096 units as image representations.

For the world representation we first gather a corpus of text from concatenation of user-tags, titles and descriptions from YFCC100M [2] dataset metadata. After pre-processing and cleaning of the text we trained a skip-gram model [14] with word vector size of 200. And we omitted any word appearing less than 200 times from the vocabulary, thereby ending up with a dictionary of size 247465.

We keep the convolutional model and the language model fixed during training of the adversarial net. And leave the experiments when we even backpropagate through these models as future work.

For our experiments we use MIR Flickr 25,000 dataset [10], and extract the image and tags features using the convolutional model and language model we described above. Images without any tag were omitted from our experiments and annotations were treated as extra tags. The first 150,000 examples were used as training set. Images with multiple tags were repeated inside the training set once for each associated tag.

For evaluation, we generate 100 samples for each image and find top 20 closest words using cosine similarity of vector representation of the words in the vocabulary to each sample. Then we select the top 10 most common words among all 100 samples. Table 4.2 shows some samples of the user assigned tags and annotations along with the generated tags.

The best working model's generator receives Gaussian noise of size 100 as noise prior and maps it to 500 dimension ReLu layer. And maps 4096 dimension image feature vector to 2000 dimension ReLu hidden layer. Both of these layers are mapped to a joint representation of 200 dimension linear layer which would output the generated word vectors.

The discriminator is consisted of 500 and 1200 dimension ReLu hidden layers for word vectors and image features respectively and maxout layer with 1000 units and 3 pieces as the join layer which is finally fed to the one single sigmoid unit.

The model was trained using stochastic gradient decent with mini-batches of size 100 and initial learning rate of 0.1 which was exponentially decreased down to $.000001$ with decay factor of 1.00004. Also momentum was used with initial value of $.5$ which was increased up to 0.7. Dropout with probability of 0.5 was applied to both the generator and discriminator.

The hyper-parameters and architectural choices were obtained by cross-validation and a mix of random grid search and manual selection (albeit over a somewhat limited search space.)

5 Future Work

The results shown in this paper are extremely preliminary, but they demonstrate the potential of conditional adversarial nets and show promise for interesting and useful applications.

In future explorations between now and the workshop we expect to present more sophisticated models, as well as a more detailed and thorough analysis of their performance and characteristics.

[2] Yahoo Flickr Creative Common 100M `http://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67`.

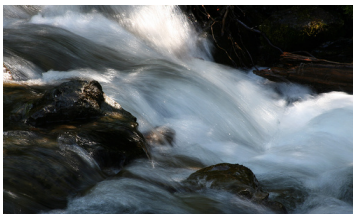
	User tags + annotations	Generated tags
	montanha, trem, inverno, frio, people, male, plant life, tree, structures, transport, car	taxi, passenger, line, transportation, railway station, passengers, railways, signals, rail, rails
	food, raspberry, delicious, homemade	chicken, fattening, cooked, peanut, cream, cookie, house made, bread, biscuit, bakes
	water, river	creek, lake, along, near, river, rocky, treeline, valley, woods, waters
	people, portrait, female, baby, indoor	love, people, posing, girl, young, strangers, pretty, women, happy, life

Table 2: Samples of generated tags

Also, in the current experiments we only use each tag individually. But by using multiple tags at the same time (effectively posing generative problem as one of 'set generation') we hope to achieve better results.

Another obvious direction left for future work is to construct a joint training scheme to learn the language model. Works such as [12] has shown that we can learn a language model for suited for the specific task.

Acknowledgments

This project was developed in Pylearn2 [7] framework, and we would like to thank Pylearn2 developers. We also like to thank Ian Goodfellow for helpful discussion during his affiliation at University of Montreal. The authors gratefully acknowledge the support from the Vision & Machine Learning, and Production Engineering teams at Flickr (in alphabetical order: Andrew Stadlen, Arel Cordero, Clayton Mellina, Cyprien Noel, Frank Liu, Gerry Pesavento, Huy Nguyen, Jack Culpepper, John Ko, Pierre Garrigues, Rob Hess, Stacey Svetlichnaya, Tobi Baumgartner, and Ye Lu).

References

[1] Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013). Better mixing via deep representations. In *ICML'2013*.

[2] Bengio, Y., Thibodeau-Laufer, E., Alain, G., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In *Proceedings of the 30th International Conference on Machine Learning (ICML'14)*.

[3] Frome, A., Corrado, G. S., Shlens, J., Bengio, S., Dean, J., Mikolov, T., *et al.* (2013). Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, pages 2121–2129.

[4] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

[5] Goodfellow, I., Mirza, M., Courville, A., and Bengio, Y. (2013a). Multi-prediction deep boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 548–556.

[6] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013b). Maxout networks. In *ICML'2013*.

[7] Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., and Bengio, Y. (2013c). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*.

[8] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *NIPS'2014*.

[9] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580.

[10] Huiskes, M. J. and Lew, M. S. (2008). The mir flickr retrieval evaluation. In *MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*, New York, NY, USA. ACM.

[11] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *ICCV'09*.

[12] Kiros, R., Zemel, R., and Salakhutdinov, R. (2013). Multimodal neural language models. In *Proc. NIPS Deep Learning Workshop*.

[13] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

[14] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations: Workshops Track*.

[15] Russakovsky, O. and Fei-Fei, L. (2010). Attribute learning in large-scale datasets. In *European Conference of Computer Vision (ECCV), International Workshop on Parts and Attributes*, Crete, Greece.

[16] Srivastava, N. and Salakhutdinov, R. (2012). Multimodal learning with deep boltzmann machines. In *NIPS'2012*.

[17] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*.

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

1 INTRODUCTION

Learning reusable feature representations from large unlabeled datasets has been an area of active research. In the context of computer vision, one can leverage the practically unlimited amount of unlabeled images and videos to learn good intermediate representations, which can then be used on a variety of supervised learning tasks such as image classification. We propose that one way to build good image representations is by training Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), and later reusing parts of the generator and discriminator networks as feature extractors for supervised tasks. GANs provide an attractive alternative to maximum likelihood techniques. One can additionally argue that their learning process and the lack of a heuristic cost function (such as pixel-wise independent mean-square error) are attractive to representation learning. GANs have been known to be unstable to train, often resulting in generators that produce nonsensical outputs. There has been very limited published research in trying to understand and visualize what GANs learn, and the intermediate representations of multi-layer GANs.

In this paper, we make the following contributions

- We propose and evaluate a set of constraints on the architectural topology of Convolutional GANs that make them stable to train in most settings. We name this class of architectures Deep Convolutional GANs (DCGAN)
- We use the trained discriminators for image classification tasks, showing competitive performance with other unsupervised algorithms.
- We visualize the filters learnt by GANs and empirically show that specific filters have learned to draw specific objects.

- We show that the generators have interesting vector arithmetic properties allowing for easy manipulation of many semantic qualities of generated samples.

2 Related Work

2.1 Representation Learning from unlabeled data

Unsupervised representation learning is a fairly well studied problem in general computer vision research, as well as in the context of images. A classic approach to unsupervised representation learning is to do clustering on the data (for example using K-means), and leverage the clusters for improved classification scores. In the context of images, one can do hierarchical clustering of image patches (Coates & Ng, 2012) to learn powerful image representations. Another popular method is to train auto-encoders (convolutionally, stacked (Vincent et al., 2010), separating the what and where components of the code (Zhao et al., 2015), ladder structures (Rasmus et al., 2015)) that encode an image into a compact code, and decode the code to reconstruct the image as accurately as possible. These methods have also been shown to learn good feature representations from image pixels. Deep belief networks (Lee et al., 2009) have also been shown to work well in learning hierarchical representations.

2.2 Generating natural images

Generative image models are well studied and fall into two categories: parametric and non-parametric.

The non-parametric models often do matching from a database of existing images, often matching patches of images, and have been used in texture synthesis (Efros et al., 1999), super-resolution (Freeman et al., 2002) and in-painting (Hays & Efros, 2007).

Parametric models for generating images has been explored extensively (for example on MNIST digits or for texture synthesis (Portilla & Simoncelli, 2000)). However, generating natural images of the real world have had not much success until recently. A variational sampling approach to generating images (Kingma & Welling, 2013) has had some success, but the samples often suffer from being blurry. Another approach generates images using an iterative forward diffusion process (Sohl-Dickstein et al., 2015). Generative Adversarial Networks (Goodfellow et al., 2014) generated images suffering from being noisy and incomprehensible. A laplacian pyramid extension to this approach (Denton et al., 2015) showed higher quality images, but they still suffered from the objects looking wobbly because of noise introduced in chaining multiple models. A recurrent network approach (Gregor et al., 2015) and a deconvolution network approach (Dosovitskiy et al., 2014) have also recently had some success with generating natural images. However, they have not leveraged the generators for supervised tasks.

2.3 Visualizing the internals of CNNs

One constant criticism of using neural networks has been that they are black-box methods, with little understanding of what the networks do in the form of a simple human-consumable algorithm. In the context of CNNs, Zeiler et. al. (Zeiler & Fergus, 2014) showed that by using deconvolutions and filtering the maximal activations, one can find the approximate purpose of each convolution filter in the network. Similarly, using a gradient descent on the inputs lets us inspect the ideal image that activates certain subsets of filters (Mordvintsev et al.).

3 Approach and Model Architecture

Historical attempts to scale up GANs using CNNs to model images have been unsuccessful. This motivated the authors of LAPGAN (Denton et al., 2015) to develop an alternative approach to iteratively upscale low resolution generated images which can be modeled more reliably. We also encountered difficulties attempting to scale GANs using CNN architectures commonly used in the supervised literature. However, after extensive model exploration we identified a family of archi-

tectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models.

Core to our approach is adopting and modifying three recently demonstrated changes to CNN architectures.

The first is the all convolutional net (Springenberg et al., 2014) which replaces deterministic spatial pooling functions (such as maxpooling) with strided convolutions, allowing the network to learn its own spatial downsampling. We use this approach in our generator, allowing it to learn its own spatial upsampling, and discriminator.

Second is the trend towards eliminating fully connected layers on top of convolutional features. The strongest example of this is global average pooling which has been utilized in state of the art image classification models (Mordvintsev et al.). We found global average pooling increased model stability but hurt convergence speed. A middle ground of directly connecting the highest convolutional features to the input and output respectively of the generator and discriminator worked well. The first layer of the GAN, which takes a uniform noise distribution Z as input, could be called fully connected as it is just a matrix multiplication, but the result is reshaped into a 4-dimensional tensor and used as the start of the convolution stack. For the discriminator, the last convolution layer is flattened and then fed into a single sigmoid output. See Fig. 1 for a visualization of an example model architecture.

Third is Batch Normalization (Ioffe & Szegedy, 2015) which stabilizes learning by normalizing the input to each unit to have zero mean and unit variance. This helps deal with training problems that arise due to poor initialization and helps gradient flow in deeper models. This proved critical to get deep generators to begin learning, preventing the generator from collapsing all samples to a single point which is a common failure mode observed in GANs. Directly applying batchnorm to all layers however, resulted in sample oscillation and model instability. This was avoided by not applying batchnorm to the generator output layer and the discriminator input layer.

The ReLU activation (Nair & Hinton, 2010) is used in the generator with the exception of the output layer which uses the Tanh function. We observed that using a bounded activation allowed the model to learn more quickly to saturate and cover the color space of the training distribution. Within the discriminator we found the leaky rectified activation (Maas et al., 2013) (Xu et al., 2015) to work well, especially for higher resolution modeling. This is in contrast to the original GAN paper, which used the maxout activation (Goodfellow et al., 2013).

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

4 DETAILS OF ADVERSARIAL TRAINING

We trained DCGANs on three datasets, Large-scale Scene Understanding (LSUN) (Yu et al., 2015), Imagenet-1k and a newly assembled Faces dataset. Details on the usage of each of these datasets are given below.

No pre-processing was applied to training images besides scaling to the range of the tanh activation function [-1, 1]. All models were trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128. All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02. In the LeakyReLU, the slope of the leak was set to 0.2 in all models. While previous GAN work has used momentum to accelerate training, we used the Adam optimizer (Kingma & Ba, 2014) with tuned hyperparameters. We found the suggested learning rate of 0.001, to be too high, using 0.0002 instead. Additionally, we found leaving the momentum term β_1 at the

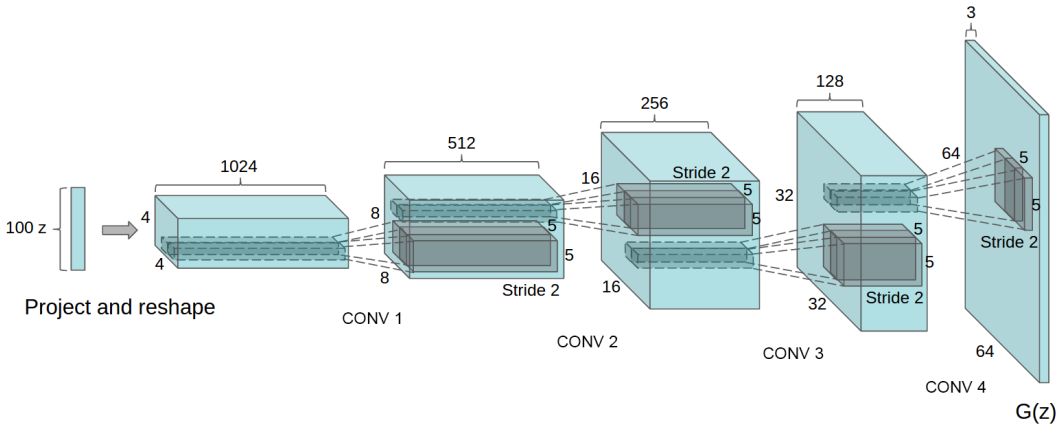


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

suggested value of 0.9 resulted in training oscillation and instability while reducing it to 0.5 helped stabilize training.

4.1 LSUN

As visual quality of samples from generative image models has improved, concerns of over-fitting and memorization of training samples have risen. To demonstrate how our model scales with more data and higher resolution generation, we train a model on the LSUN bedrooms dataset containing a little over 3 million training examples. Recent analysis has shown that there is a direct link between how fast models learn and their generalization performance (Hardt et al., 2015). We show samples from one epoch of training (Fig.2), mimicking online learning, in addition to samples after convergence (Fig.3), as an opportunity to demonstrate that our model is not producing high quality samples via simply overfitting/memorizing training examples. No data augmentation was applied to the images.

4.1.1 Deduplication

To further decrease the likelihood of the generator memorizing input examples (Fig.2) we perform a simple image de-duplication process. We fit a 3072-128-3072 de-noising dropout regularized RELU autoencoder on 32x32 downsampled center-crops of training examples. The resulting code layer activations are then binarized via thresholding the ReLU activation which has been shown to be an effective information preserving technique (Srivastava et al., 2014) and provides a convenient form of semantic-hashing, allowing for linear time de-duplication . Visual inspection of hash collisions showed high precision with an estimated false positive rate of less than 1 in 100. Additionally, the technique detected and removed approximately 275,000 near duplicates, suggesting a high recall.

4.2 Faces

We scraped images containing human faces from random web image queries of peoples names. The people names were acquired from dbpedia, with a criterion that they were born in the modern era. This dataset has 3M images from 10K people. We run an OpenCV face detector on these images, keeping the detections that are sufficiently high resolution, which gives us approximately 350,000 face boxes. We use these face boxes for training. No data augmentation was applied to the images.

Figure 2: Generated bedrooms after one training pass through the dataset. Theoretically, the model could learn to memorize training examples, but this is experimentally unlikely as we train with a small learning rate and minibatch SGD. We are aware of no prior empirical evidence demonstrating memorization with SGD and a small learning rate.

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

4.3 IMAGENET-1K

We use Imagenet-1k (Deng et al., 2009) as a source of natural images for unsupervised training. We train on 32×32 min-resized center crops. No data augmentation was applied to the images.

5 EMPIRICAL VALIDATION OF DCGANS CAPABILITIES

5.1 CLASSIFYING CIFAR-10 USING GANS AS A FEATURE EXTRACTOR

One common technique for evaluating the quality of unsupervised representation learning algorithms is to apply them as a feature extractor on supervised datasets and evaluate the performance of linear models fitted on top of these features.

On the CIFAR-10 dataset, a very strong baseline performance has been demonstrated from a well tuned single layer feature extraction pipeline utilizing K-means as a feature learning algorithm. When using a very large amount of feature maps (4800) this technique achieves 80.6% accuracy. An unsupervised multi-layered extension of the base algorithm reaches 82.0% accuracy (Coates & Ng, 2011). To evaluate the quality of the representations learned by DCGANs for supervised tasks, we train on Imagenet-1k and then use the discriminator's convolutional features from all layers, maxpooling each layers representation to produce a 4×4 spatial grid. These features are then flattened and concatenated to form a 28672 dimensional vector and a regularized linear L2-SVM classifier is trained on top of them. This achieves 82.8% accuracy, out performing all K-means based approaches. Notably, the discriminator has many less feature maps (512 in the highest layer) compared to K-means based techniques, but does result in a larger total feature vector size due to the many layers of 4×4 spatial locations. The performance of DCGANs is still less than that of Exemplar CNNs (Dosovitskiy et al., 2015), a technique which trains normal discriminative CNNs in an unsupervised fashion to differentiate between specifically chosen, aggressively augmented, exemplar samples from the source dataset. Further improvements could be made by finetuning the discriminator's representations, but we leave this for future work. Additionally, since our DCGAN was never trained on CIFAR-10 this experiment also demonstrates the domain robustness of the learned features.

Table 1: CIFAR-10 classification results using our pre-trained model. Our DCGAN is not pre-trained on CIFAR-10, but on Imagenet-1k, and the features are used to classify CIFAR-10 images.

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% ($\pm$0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% ($\pm$0.7%)	3200
View Invariant K-means	81.9%	72.6% ($\pm$0.7%)	6400
Exemplar CNN	84.3%	77.4% ($\pm$0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% ($\pm$0.4%)	512

5.2 CLASSIFYING SVHN DIGITS USING GANS AS A FEATURE EXTRACTOR

On the StreetView House Numbers dataset (SVHN)(Netzer et al., 2011), we use the features of the discriminator of a DCGAN for supervised purposes when labeled data is scarce. Following similar dataset preparation rules as in the CIFAR-10 experiments, we split off a validation set of 10,000 examples from the non-extra set and use it for all hyperparameter and model selection. 1000 uniformly class distributed training examples are randomly selected and used to train a regularized linear L2-SVM classifier on top of the same feature extraction pipeline used for CIFAR-10. This achieves state of the art (for classification using 1000 labels) at 22.48% test error, improving upon another modifcation of CNNs designed to leverage unlabled data (Zhao et al., 2015). Additionally, we validate that the CNN architecture used in DCGAN is not the key contributing factor of the model's performance by training a purely supervised CNN with the same architecture on the same data and optimizing this model via random search over 64 hyperparameter trials (Bergstra & Bengio, 2012). It achieves a signficantly higher 28.87% validation error.

6 INVESTIGATING AND VISUALIZING THE INTERNALS OF THE NETWORKS

We investigate the trained generators and discriminators in a variety of ways. We do not do any kind of nearest neighbor search on the training set. Nearest neighbors in pixel or feature space are

Table 2: SVHN classification with 1000 labels

Model	error rate
KNN	77.93%
TSVM	66.55%
M1+KNN	65.63%
M1+TSVM	54.33%
M1+M2	36.02%
SWWAE without dropout	27.83%
SWWAE with dropout	23.56%
DCGAN (ours) + L2-SVM	22.48%
Supervised CNN with the same architecture	28.87% (validation)

trivially fooled (Theis et al., 2015) by small image transforms. We also do not use log-likelihood metrics to quantitatively assess the model, as it is a poor (Theis et al., 2015) metric.

6.1 Walking in the latent space

The first experiment we did was to understand the landscape of the latent space. Walking on the manifold that is learnt can usually tell us about signs of memorization (if there are sharp transitions) and about the way in which the space is hierarchically collapsed. If walking in this latent space results in semantic changes to the image generations (such as objects being added and removed), we can reason that the model has learned relevant and interesting representations. The results are shown in Fig.4.

6.2 Visualizing the Discriminator Features

Previous work has demonstrated that supervised training of CNNs on large image datasets results in very powerful learned features (Zeiler & Fergus, 2014). Additionally, supervised CNNs trained on scene classification learn object detectors (Oquab et al., 2014). We demonstrate that an unsupervised DCGAN trained on a large image dataset can also learn a hierarchy of features that are interesting. Using guided backpropagation as proposed by (Springenberg et al., 2014), we show in Fig.5 that the features learnt by the discriminator activate on typical parts of a bedroom, like beds and windows. For comparison, in the same figure, we give a baseline for randomly initialized features that are not activated on anything that is semantically relevant or interesting.

6.3 Manipulating the Generator Representation

6.3.1 Forgetting to draw certain objects

In addition to the representations learnt by a discriminator, there is the question of what representations the generator learns. The quality of samples suggest that the generator learns specific object representations for major scene components such as beds, windows, lamps, doors, and miscellaneous furniture. In order to explore the form that these representations take, we conducted an experiment to attempt to remove windows from the generator completely.

On 150 samples, 52 window bounding boxes were drawn manually. On the second highest convolution layer features, logistic regression was fit to predict whether a feature activation was on a window (or not), by using the criterion that activations inside the drawn bounding boxes are positives and random samples from the same images are negatives. Using this simple model, all feature maps with weights greater than zero (200 in total) were dropped from all spatial locations. Then, random new samples were generated with and without the feature map removal.

The generated images with and without the window dropout are shown in Fig.6, and interestingly, the network mostly forgets to draw windows in the bedrooms, replacing them with other objects.

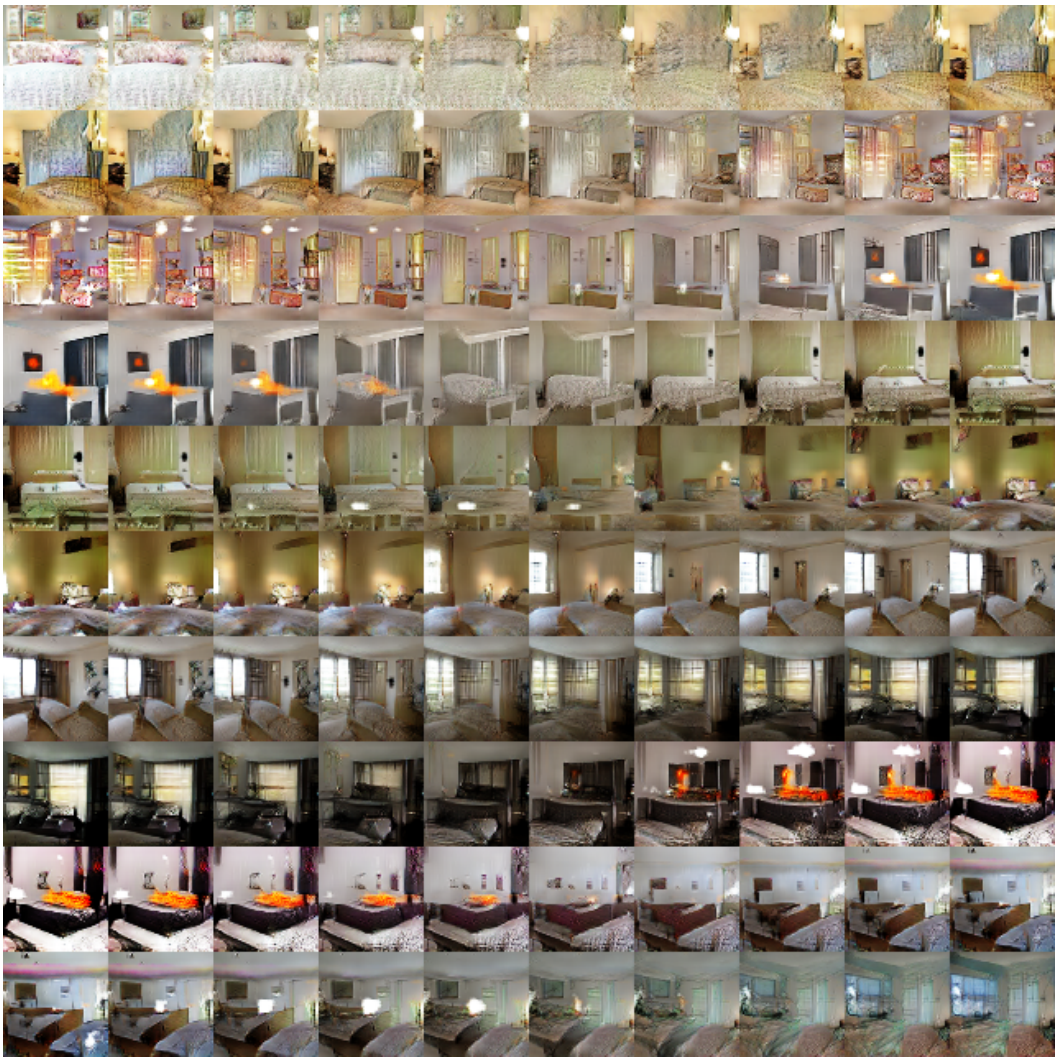


Figure 4: Top rows: Interpolation between a series of 9 random points in Z show that the space learned has smooth transitions, with every image in the space plausibly looking like a bedroom. In the 6th row, you see a room without a window slowly transforming into a room with a giant window. In the 10th row, you see what appears to be a TV slowly being transformed into a window.

6.3.2 VECTOR ARITHMETIC ON FACE SAMPLES

In the context of evaluating learned representations of words (Mikolov et al., 2013) demonstrated that simple arithmetic operations revealed rich linear structure in representation space. One canonical example demonstrated that the vector(”King”) - vector(”Man”) + vector(”Woman”) resulted in a vector whose nearest neighbor was the vector for Queen. We investigated whether similar structure emerges in the Z representation of our generators. We performed similar arithmetic on the Z vectors of sets of exemplar samples for visual concepts. Experiments working on only single samples per concept were unstable, but averaging the Z vector for three examplars showed consistent and stable generations that semantically obeyed the arithmetic. In addition to the object manipulation shown in (Fig. 7), we demonstrate that face pose is also modeled linearly in Z space (Fig. 8).

These demonstrations suggest interesting applications can be developed using Z representations learned by our models. It has been previously demonstrated that conditional generative models can learn to convincingly model object attributes like scale, rotation, and position (Dosovitskiy et al., 2014). This is to our knowledge the first demonstration of this occurring in purely unsupervised

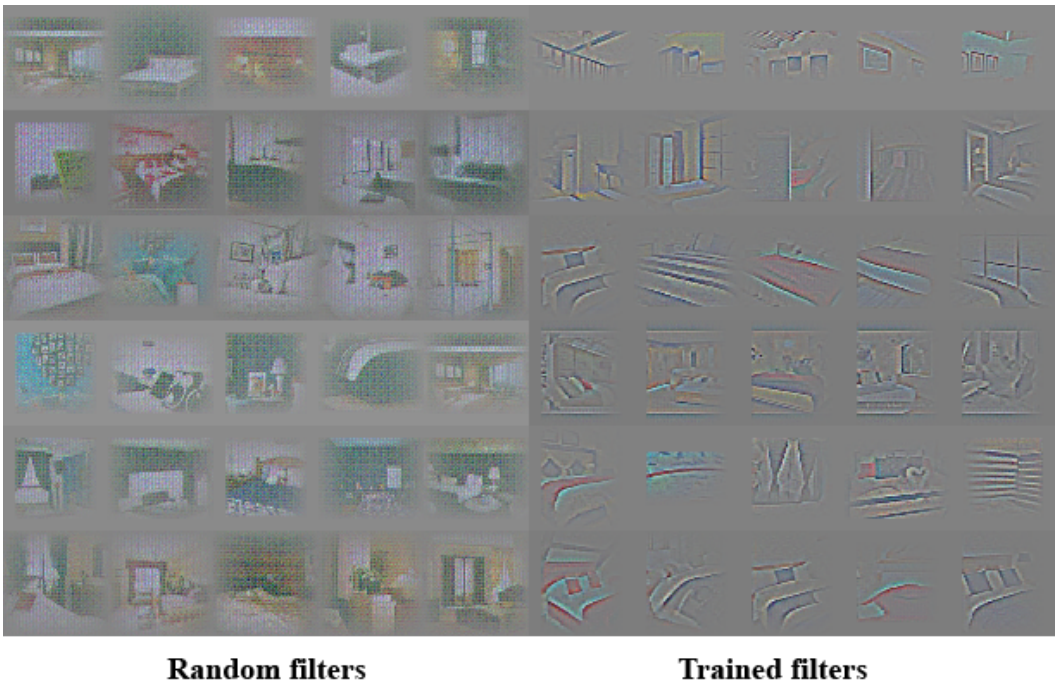


Figure 5: On the right, guided backpropagation visualizations of maximal axis-aligned responses for the first 6 learned convolutional features from the last convolution layer in the discriminator. Notice a significant minority of features respond to beds - the central object in the LSUN bedrooms dataset. On the left is a random filter baseline. Comparing to the previous responses there is little to no discrimination and random structure.

Figure 6: Top row: un-modified samples from model. Bottom row: the same samples generated with dropping out "window" filters. Some windows are removed, others are transformed into objects with similar visual appearance such as doors and mirrors. Although visual quality decreased, overall scene composition stayed similar, suggesting the generator has done a good job disentangling scene representation from object representation. Extended experiments could be done to remove other objects from the image and modify the objects the generator draws.

models. Further exploring and developing the above mentioned vector arithmetic could dramatically reduce the amount of data needed for conditional generative modeling of complex image distributions.

7 Conclusion and Future Work

We propose a more stable set of architectures for training generative adversarial networks and we give evidence that adversarial networks learn good representations of images for supervised learning and generative modeling. There are still some forms of model instability remaining - we noticed as models are trained longer they sometimes collapse a subset of filters to a single oscillating mode.

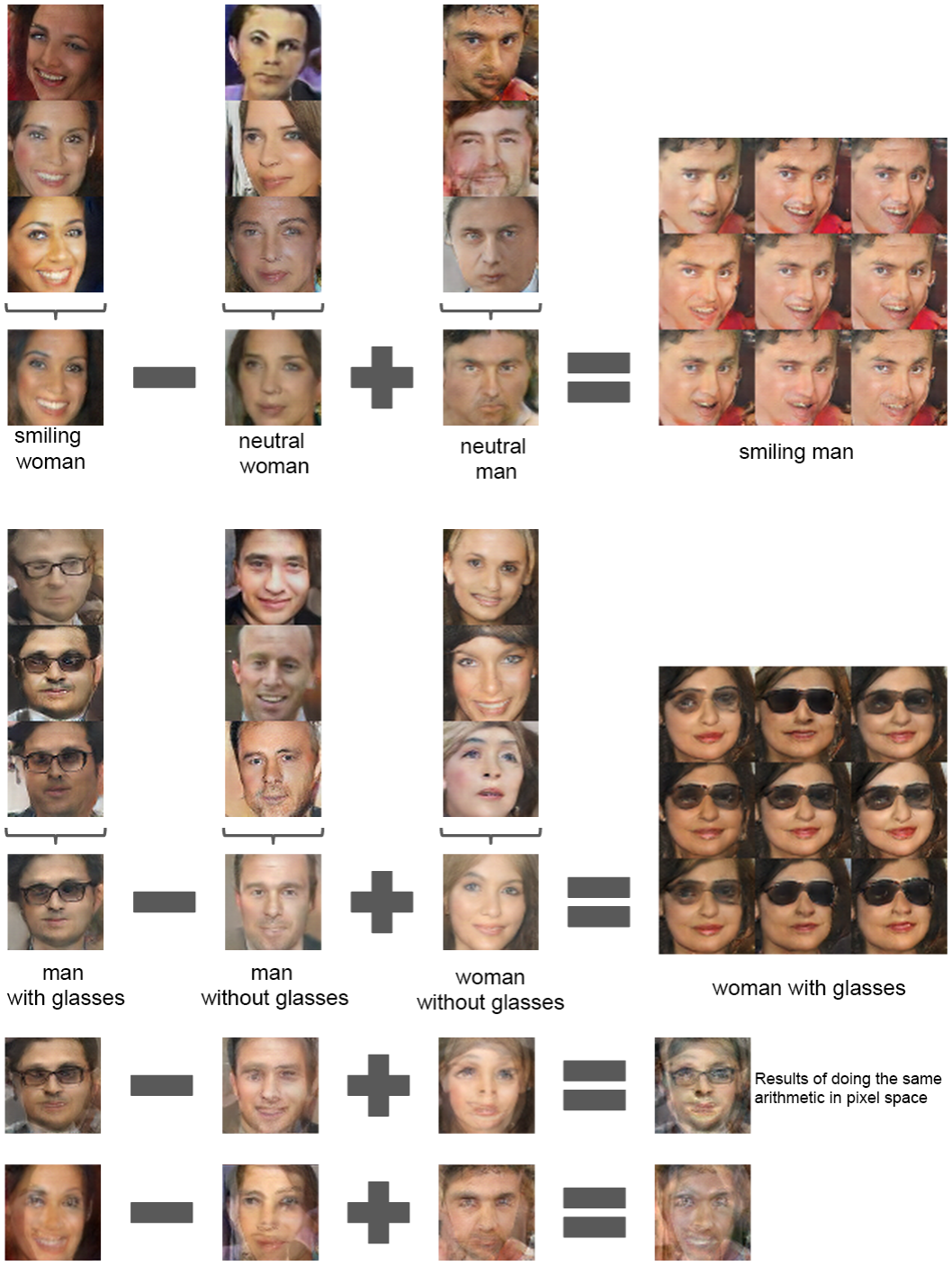


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples) results in noisy overlap due to misalignment.

Further work is needed to tackle this from of instability. We think that extending this framework

Figure 8: A "turn" vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

to other domains such as video (for frame prediction) and audio (pre-trained features for speech synthesis) should be very interesting. Further investigations into the properties of the learnt latent space would be interesting as well.

ACKNOWLEDGMENTS

We are fortunate and thankful for all the advice and guidance we have received during this work, especially that of Ian Goodfellow, Tobias Springenberg, Arthur Szlam and Durk Kingma. Additionally we'd like to thank all of the folks at indico for providing support, resources, and conversations, especially the two other members of the indico research team, Dan Kuster and Nathan Lintz. Finally, we'd like to thank Nvidia for donating a Titan-X GPU used in this work.

REFERENCES

Bergstra, James and Bengio, Yoshua. Random search for hyper-parameter optimization. *JMLR*, 2012.

Coates, Adam and Ng, Andrew. Selecting receptive fields in deep networks. *NIPS*, 2011.

Coates, Adam and Ng, Andrew Y. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, pp. 561–580. Springer, 2012.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Denton, Emily, Chintala, Soumith, Szlam, Arthur, and Fergus, Rob. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*, 2015.

Dosovitskiy, Alexey, Springenberg, Jost Tobias, and Brox, Thomas. Learning to generate chairs with convolutional neural networks. *arXiv preprint arXiv:1411.5928*, 2014.

Dosovitskiy, Alexey, Fischer, Philipp, Springenberg, Jost Tobias, Riedmiller, Martin, and Brox, Thomas. Discriminative unsupervised feature learning with exemplar convolutional neural networks. In *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, volume 99. IEEE, 2015.

Efros, Alexei, Leung, Thomas K, et al. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pp. 1033–1038. IEEE, 1999.

Freeman, William T, Jones, Thouis R, and Pasztor, Egon C. Example-based super-resolution. *Computer Graphics and Applications, IEEE*, 22(2):56–65, 2002.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. Generative adversarial nets. *NIPS*, 2014.

Gregor, Karol, Danihelka, Ivo, Graves, Alex, and Wierstra, Daan. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Hardt, Moritz, Recht, Benjamin, and Singer, Yoram. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240*, 2015.

Hauberg, Sren, Freifeld, Oren, Larsen, Anders Boesen Lindbo, Fisher III, John W., and Hansen, Lars Kair. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. *arXiv preprint arXiv:1510.02795*, 2015.

Hays, James and Efros, Alexei A. Scene completion using millions of photographs. *ACM Transactions on Graphics (TOG)*, 26(3):4, 2007.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Kingma, Diederik P and Ba, Jimmy Lei. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lee, Honglak, Grosse, Roger, Ranganath, Rajesh, and Ng, Andrew Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 609–616. ACM, 2009.

Loosli, Gaëlle, Canu, Stéphane, and Bottou, Léon. Training invariant support vector machines using selective sampling. In Bottou, Léon, Chapelle, Olivier, DeCoste, Dennis, and Weston, Jason (eds.), *Large Scale Kernel Machines*, pp. 301–320. MIT Press, Cambridge, MA., 2007. URL `http://leon.bottou.org/papers/loosli-canu-bottou-2006`.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Mordvintsev, Alexander, Olah, Christopher, and Tyka, Mike. Inceptionism : Going deeper into neural networks. `http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html`. Accessed: 2015-06-17.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, 2010.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5. Granada, Spain, 2011.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, 2014.

Portilla, Javier and Simoncelli, Eero P. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–70, 2000.

Rasmus, Antti, Valpola, Harri, Honkala, Mikko, Berglund, Mathias, and Raiko, Tapani. Semi-supervised learning with ladder network. *arXiv preprint arXiv:1507.02672*, 2015.

Sohl-Dickstein, Jascha, Weiss, Eric A, Maheswaranathan, Niru, and Ganguli, Surya. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Srivastava, Rupesh Kumar, Masci, Jonathan, Gomez, Faustino, and Schmidhuber, Jürgen. Understanding locally competitive networks. *arXiv preprint arXiv:1410.1165*, 2014.

Theis, L., van den Oord, A., and Bethge, M. A note on the evaluation of generative models. arXiv:1511.01844, Nov 2015. URL `http://arxiv.org/abs/1511.01844`.

Vincent, Pascal, Larochelle, Hugo, Lajoie, Isabelle, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

Xu, Bing, Wang, Naiyan, Chen, Tianqi, and Li, Mu. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Yu, Fisher, Zhang, Yinda, Song, Shuran, Seff, Ari, and Xiao, Jianxiong. Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Zeiler, Matthew D and Fergus, Rob. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pp. 818–833. Springer, 2014.

Zhao, Junbo, Mathieu, Michael, Goroshin, Ross, and Lecun, Yann. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015.

8 SUPPLEMENTARY MATERIAL

8.1 EVALUATING DCGANS CAPABILITY TO CAPTURE DATA DISTRIBUTIONS

We propose to apply standard classification metrics to a conditional version of our model, evaluating the conditional distributions learned. We trained a DCGAN on MNIST (splitting off a 10K validation set) as well as a permutation invariant GAN baseline and evaluated the models using a nearest neighbor classifier comparing real data to a set of generated conditional samples. We found that removing the scale and bias parameters from batchnorm produced better results for both models. We speculate that the noise introduced by batchnorm helps the generative models to better explore and generate from the underlying data distribution. The results are shown in Table 3 which compares our models with other techniques. The DCGAN model achieves the same test error as a nearest neighbor classifier fitted on the training dataset - suggesting the DCGAN model has done a superb job at modeling the conditional distributions of this dataset. At one million samples per class, the DCGAN model outperforms InfiMNIST (Loosli et al., 2007), a hand developed data augmentation pipeline which uses translations and elastic deformations of training examples. The DCGAN is competitive with a probabilistic generative data augmentation technique utilizing learned per class transformations (Hauberg et al., 2015) while being more general as it directly models the data instead of transformations of the data.

Table 3: Nearest neighbor classification results.

Model	Test Error @50K samples	Test Error @10M samples
AlignMNIST	-	1.4%
InfiMNIST	-	2.6%
Real Data	3.1%	-
GAN	6.28%	5.65%
DCGAN (ours)	2.98%	1.48%

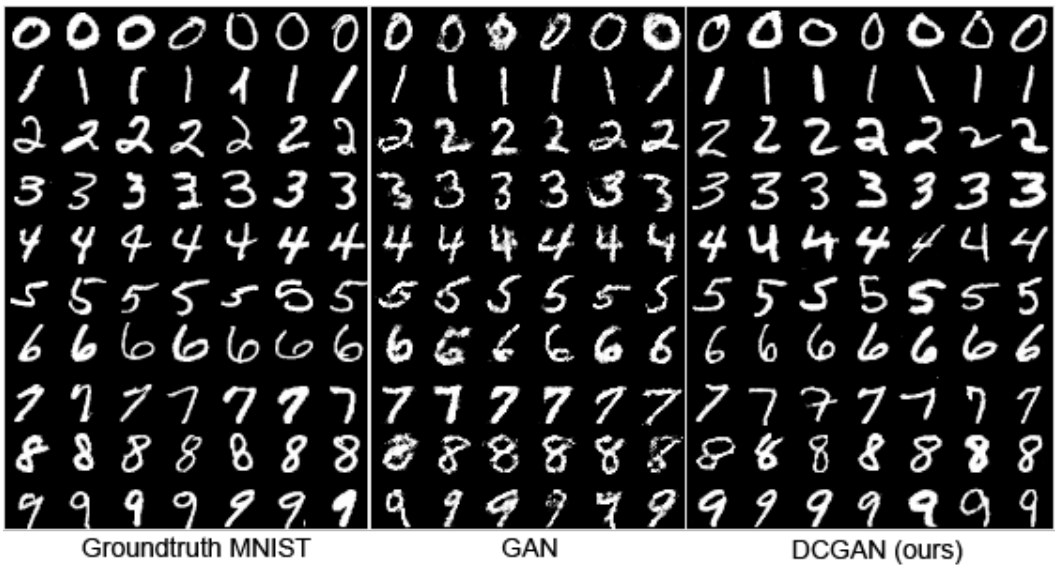


Figure 9: Side-by-side illustration of (from left-to-right) the MNIST dataset, generations from a baseline GAN, and generations from our DCGAN .

Figure 10: More face generations from our Face DCGAN.

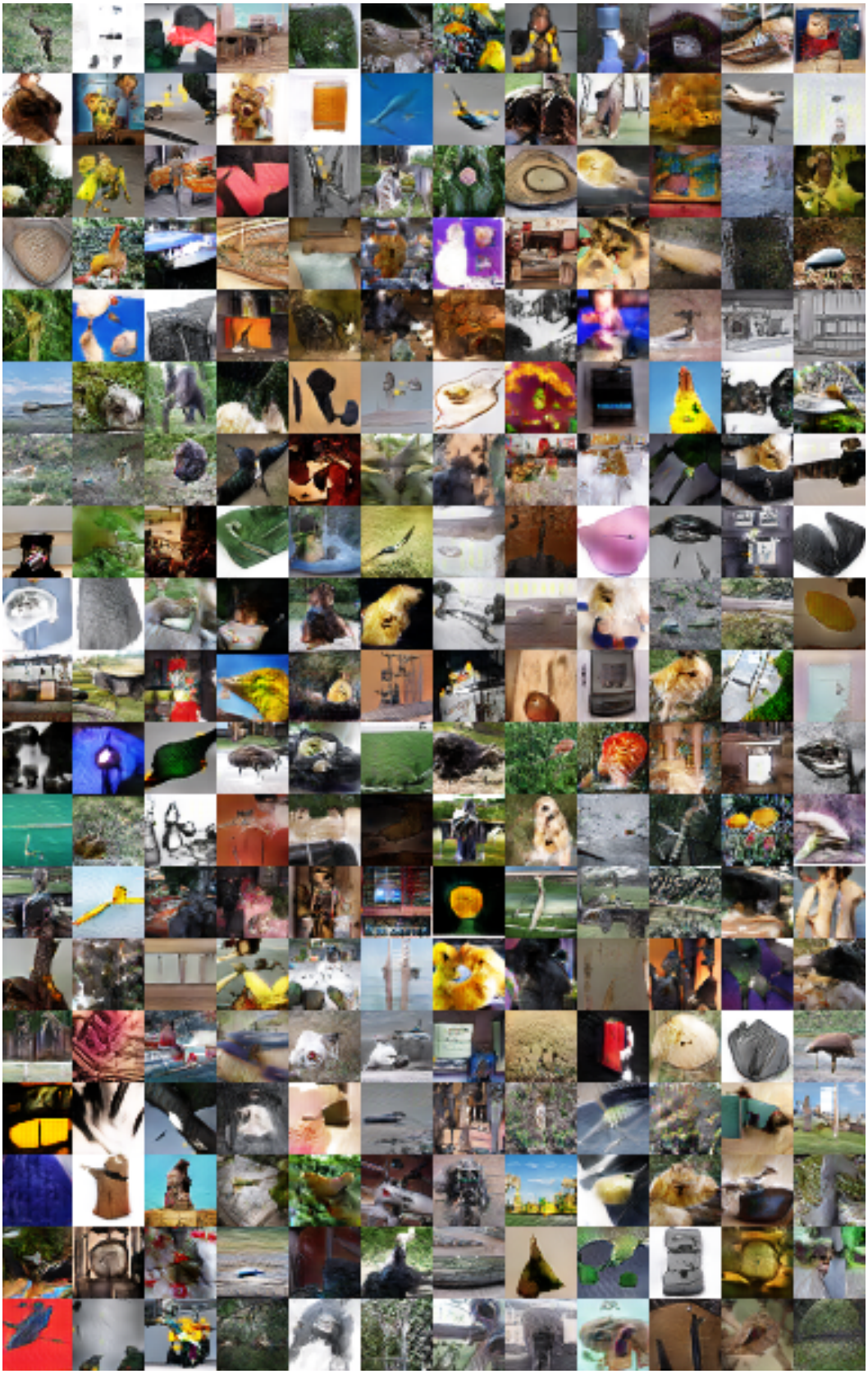


Figure 11: Generations of a DCGAN that was trained on the Imagenet-1k dataset.

Generating Videos with Scene Dynamics

Carl Vondrick
MIT
vondrick@mit.edu

Hamed Pirsiavash
UMBC
hpirsiav@umbc.edu

Antonio Torralba
MIT
torralba@mit.edu

Abstract

We capitalize on large amounts of unlabeled video in order to learn a model of scene dynamics for both video recognition tasks (e.g. action classification) and video generation tasks (e.g. future prediction). We propose a generative adversarial network for video with a spatio-temporal convolutional architecture that untangles the scene's foreground from the background. Experiments suggest this model can generate tiny videos up to a second at full frame rate better than simple baselines, and we show its utility at predicting plausible futures of static images. Moreover, experiments and visualizations show the model internally learns useful features for recognizing actions with minimal supervision, suggesting scene dynamics are a promising signal for representation learning. We believe generative video models can impact many applications in video understanding and simulation.

1 Introduction

Understanding object motions and scene dynamics is a core problem in computer vision. For both video recognition tasks (e.g., action classification) and video generation tasks (e.g., future prediction), a model of how scenes transform is needed. However, creating a model of dynamics is challenging because there is a vast number of ways that objects and scenes can change.

In this work, we are interested in the fundamental problem of learning how scenes transform with time. We believe investigating this question may yield insight into the design of predictive models for computer vision. However, since annotating this knowledge is both expensive and ambiguous, we instead seek to learn it directly from large amounts of in-the-wild, unlabeled video. Unlabeled video has the advantage that it can be economically acquired at massive scales yet contains rich temporal signals "for free" because frames are temporally coherent.

With the goal of capturing some of the temporal knowledge contained in large amounts of unlabeled video, we present an approach that learns to generate tiny videos which have fairly realistic dynamics and motions. To do this, we capitalize on recent advances in generative adversarial networks [9, 31, 4], which we extend to video. We introduce a two-stream generative model that explicitly models the foreground separately from the background, which allows us to enforce that the background is stationary, helping the network to learn which objects move and which do not.

Our experiments suggest that our model has started to learn about dynamics. In our generation experiments, we show that our model can generate scenes with plausible motions.[1] We conducted a psychophysical study where we asked over a hundred people to compare generated videos, and people preferred videos from our full model more often. Furthermore, by making the model conditional on an input image, our model can sometimes predict a plausible (but "incorrect") future. In our recognition experiments, we show how our model has learned, without supervision, useful features for human action classification. Moreover, visualizations of the learned representation suggest future generation may be a promising supervisory signal for learning to recognize objects of motion.

[1] See http://mit.edu/vondrick/tinyvideo for the animated videos.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

The primary contribution of this paper is showing how to leverage large amounts of unlabeled video in order to acquire priors about scene dynamics. The secondary contribution is the development of a generative model for video. The remainder of this paper describes these contributions in detail. In section 2, we describe our generative model for video. In section 3, we present several experiments to analyze the generative model. We believe that generative video models can impact many applications, such as in simulations, forecasting, and representation learning.

1.1 Related Work

This paper builds upon early work in generative video models [29]. However, previous work has focused mostly on small patches, and evaluated it for video clustering. Here, we develop a generative video model for natural scenes using state-of-the-art adversarial learning methods [9, 31]. Conceptually, our work is related to studies into fundamental roles of time in computer vision [30, 12, 2, 7, 24]. However, here we are interested in generating short videos with realistic temporal semantics, rather than detecting or retrieving them.

Our technical approach builds on recent work in generative adversarial networks for image modeling [9, 31, 4, 48, 28], which we extend to video. To our knowledge, there has been relatively little work extensively studying generative adversarial networks for video. Most notably, [22] also uses adversarial networks for video frame prediction. Our framework can generate videos for longer time scales and learn representations of video using unlabeled data. Our work is also related to efforts to predict the future in video [33, 22, 44, 51, 42, 17, 8, 55] as well as concurrent work in future generation [6, 15, 20, 50, 56, 43]. Often these works may be viewed as a generative model conditioned on the past frames. Our work complements these efforts in two ways. Firstly, we explore how to generate videos from scratch (not conditioned on the past). Secondly, while prior work has used generative models in video settings mostly on a single frame, we jointly generate a sequence of frames (32 frames) using spatio-temporal convolutional networks, which may help prevent drifts due to errors accumulating.

We leverage approaches for recognizing actions in video with deep networks, but apply them for video generation instead. We use spatio-temporal 3D convolutions to model videos [40], but we use fractionally strided convolutions [52] instead because we are interested in generation. We also use two-streams to model video [34], but apply them for video generation instead of action recognition. However, our approach does not explicitly use optical flow; instead, we expect the network to learn motion features on its own. Finally, this paper is related to a growing body of work that capitalizes on large amounts of unlabeled video for visual recognition tasks [18, 47, 37, 13, 24, 25, 3, 32, 26, 27, 19, 41, 42, 1]. We instead leverage large amounts of unlabeled video for generation.

2 Generative Models for Video

In this section, we present a generative model for videos. We propose to use generative adversarial networks [9], which have been shown to have good performance on image generation [31, 4].

2.1 Review: Generative Adversarial Networks

The main idea behind generative adversarial networks [9] is to train two networks: a generator network G tries to produce a video, and a discriminator network D tries to distinguish between "real" videos and "fake" generated videos. One can train these networks against each other in a min-max game where the generator seeks to maximally fool the discriminator while simultaneously the discriminator seeks to detect which examples are fake:

$$\min_{w_G} \max_{w_D} \mathbb{E}_{x \sim p_x(x)} \left[\log D(x; w_D)\right] + \mathbb{E}_{z \sim p_z(z)} \left[\log \left(1 - D(G(z; w_G); w_D)\right)\right] \tag{1}$$

where z is a latent "code" that is often sampled from a simple distribution (such as a normal distribution) and $x \sim p_x(x)$ samples from the data distribution. In practice, since we do not know the true distribution of data $p_x(x)$, we can estimate the expectation by drawing from our dataset.

Since we will optimize Equation 1 with gradient based methods (SGD), the two networks G and D can take on any form appropriate for the task as long as they are differentiable with respect to parameters w_G and w_D. We design a G and D for video.

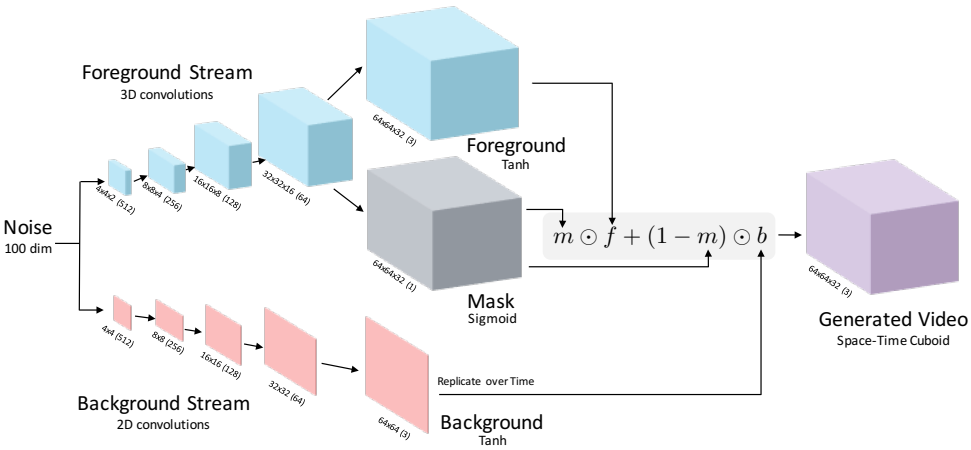


Figure 1: **Video Generator Network:** We illustrate our network architecture for the generator. The input is 100 dimensional (Gaussian noise). There are two independent streams: a moving foreground pathway of fractionally-strided spatio-temporal convolutions, and a static background pathway of fractionally-strided spatial convolutions, both of which up-sample. These two pathways are combined to create the generated video using a mask from the motion pathway. Below each volume is its size and the number of channels in parenthesis.

2.2 Generator Network

The input to the generator network is a low-dimensional latent code $z \in \mathbb{R}^d$. In most cases, this code can be sampled from a distribution (e.g., Gaussian). Given a code z, we wish to produce a video.

We design the architecture of the generator network with a few principles in mind. Firstly, we want the network to be invariant to translations in both space and time. Secondly, we want a low-dimensional z to be able to produce a high-dimensional output (video). Thirdly, we want to assume a stationary camera and take advantage of the the property that usually only objects move. We are interested in modeling object motion, and not the motion of cameras. Moreover, since modeling that the background is stationary is important in video recognition tasks [45], it may be helpful in video generation as well. We explore two different network architectures:

One Stream Architecture: We combine spatio-temporal convolutions [14, 40] with fractionally strided convolutions [52, 31] to generate video. Three dimensional convolutions provide spatial and temporal invariance, while fractionally strided convolutions can upsample efficiently in a deep network, allowing z to be low-dimensional. We use an architecture inspired by [31], except extended in time. We use a five layer network of $4 \times 4 \times 4$ convolutions with a stride of 2, except for the first layer which uses $2 \times 4 \times 4$ convolutions (time $\times$ width $\times$ height). We found that these kernel sizes provided an appropriate balance between training speed and quality of generations.

Two Stream Architecture: The one stream architecture does not model that the world is stationary and usually only objects move. We experimented with making this behavior explicit in the model. We use an architecture that enforces a static background and moving foreground. We use a two-stream architecture where the generator is governed by the combination:

$$G_2(z) = m(z) \odot f(z) + (1 - m(z)) \odot b(z). \tag{2}$$

Our intention is that $0 \geq m(z) \geq 1$ can be viewed as a spatio-temporal mask that selects either the foreground $f(z)$ model or the background model $b(z)$ for each pixel location and timestep. To enforce a background model in the generations, $b(z)$ produces a spatial image that is replicated over time, while $f(z)$ produces a spatio-temporal cuboid masked by $m(z)$. By summing the foreground model with the background model, we can obtain the final generation. Note that $\odot$ is element-wise multiplication, and we replicate singleton dimensions to match its corresponding tensor. During learning, we also add to the objective a small sparsity prior on the mask $\lambda \|m(z)\|_1$ for $\lambda = 0.1$, which we found helps encourage the network to use the background stream.

We use fractionally strided convolutional networks for $m(z)$, $f(z)$, and $b(z)$. For $f(z)$, we use the same network as the one-stream architecture, and for $b(z)$ we use a similar generator architecture to [31]. We only use their architecture; we do not initialize with their learned weights. To create the mask $m(z)$, we use a network that shares weights with $f(z)$ except the last layer, which has only one output channel. We use a sigmoid activation function for the mask. We visualize the two-stream architecture in Figure 1. In our experiments, the generator produces 64×64 videos for 32 frames, which is a little over a second.

2.3 Discriminator Network

The discriminator needs to be able to solve two problems: firstly, it must be able to classify realistic scenes from synthetically generated scenes, and secondly, it must be able to recognize realistic motion between frames. We chose to design the discriminator to be able to solve both of these tasks with the same model. We use a five-layer spatio-temporal convolutional network with kernels $4 \times 4 \times 4$ so that the hidden layers can learn both visual models and motion models. We design the architecture to be reverse of the foreground stream in the generator, replacing fractionally strided convolutions with strided convolutions (to down-sample instead of up-sample), and replacing the last layer to output a binary classification (real or not).

2.4 Learning and Implementation

We train the generator and discriminator with stochastic gradient descent. We alternate between maximizing the loss w.r.t. w_D and minimizing the loss w.r.t. w_G until a fixed number of iterations. All networks are trained from scratch. Our implementation is based off a modified version of [31] in Torch7. We used a more numerically stable implementation of cross entropy loss to prevent overflow. We use the Adam [16] optimizer and a fixed learning rate of 0.0002 and momentum term of 0.5. The latent code has 100 dimensions, which we sample from a normal distribution. We use a batch size of 64. We initialize all weights with zero mean Gaussian noise with standard deviation 0.01. We normalize all videos to be in the range $[-1, 1]$. We use batch normalization [11] followed by the ReLU activation functions after every layer in the generator, except the output layers, which uses tanh. Following [31], we also use batch normalization in the discriminator except for the first layer and we instead use leaky ReLU [49]. Training typically took several days on a GPU.

3 Experiments

We experiment with the generative adversarial network for video (VGAN) on both generation and recognition tasks. We also show several qualitative examples online.

3.1 Unlabeled Video Dataset

We use a large amount of unlabeled video to train our model. We downloaded over two million videos from Flickr [39] by querying for popular Flickr tags as well as querying for common English words. From this pool, we created two datasets:

Unfiltered Unlabeled Videos: We use these videos directly, without any filtering, for representation learning. The dataset is over $5,000$ hours. **Filtered Unlabeled Videos:** To evaluate generations, we use the Places2 pre-trained model [54] to automatically filter the videos by scene category. Since image/video generation is a challenging problem, we assembled this dataset to better diagnose strengths and weaknesses of approaches. We experimented with four scene categories: golf course, hospital rooms (babies), beaches, and train station.

Stabilization: As we are interested in the movement of objects and not camera shake, we stabilize the camera motion for both datasets. We extract SIFT keypoints [21], use RANSAC to estimate a homography (rotation, translation, scale) between adjacent frames, and warp frames to minimize background motion. When the homography moved out of the frame, we fill in the missing values using the previous frames. If the homography has too large of a re-projection error, we ignore that segment of the video for training, which only happened 3% of the time. The only other pre-processing we do is normalizing the videos to be in the range $[-1, 1]$. We extract frames at native frame rate (25 fps). We use 32-frame videos of spatial resolution 64×64.

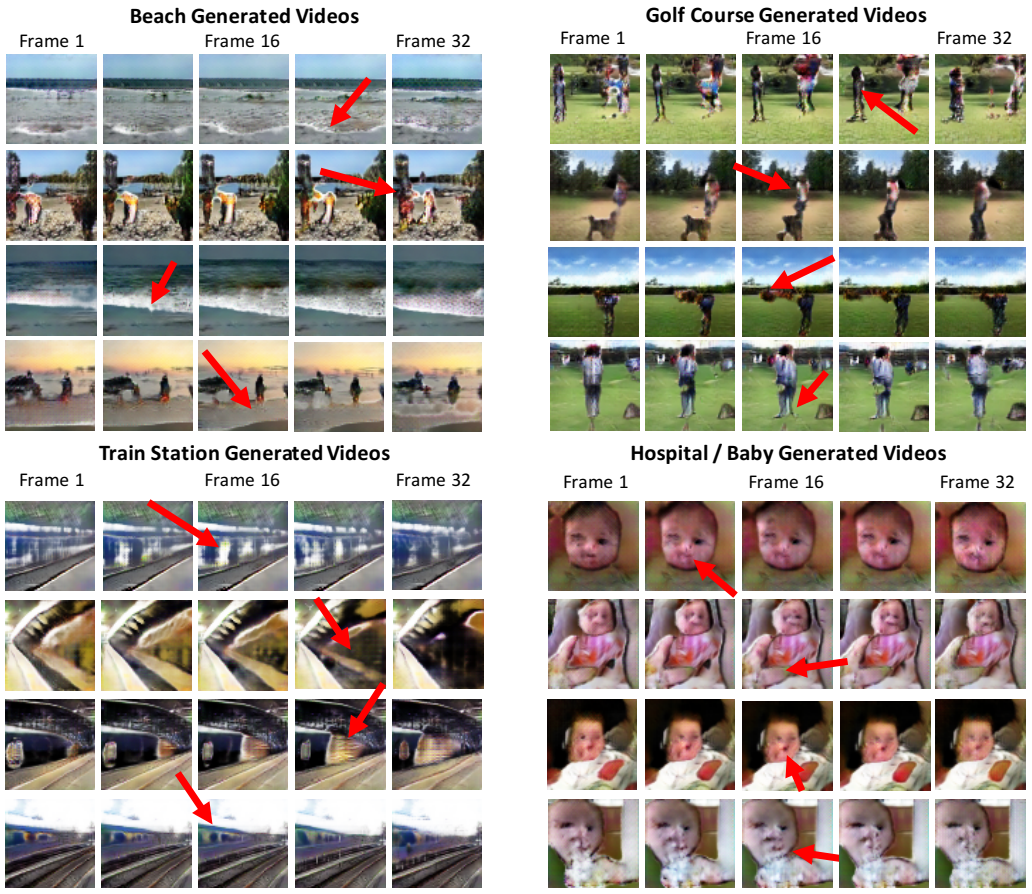


Figure 2: **Video Generations:** We show some generations from the two-stream model. The red arrows highlight motions. Please see `http://mit.edu/vondrick/tinyvideo` for animated movies.

3.2 Video Generation

We evaluate both the one-stream and two-stream generator. We trained a generator for each scene category in our filtered dataset. We perform both a qualitative evaluation as well as a quantitative psychophysical evaluation to measure the perceptual quality of the generated videos.

Qualitative Results: We show several examples of the videos generated from our model in Figure 2. We observe that a) the generated scenes tend to be fairly sharp and that b) the motion patterns are generally correct for their respective scene. For example, the beach model tends to produce beaches with crashing waves, the golf model produces people walking on grass, and the train station generations usually show train tracks and a train with windows rapidly moving along it. While the model usually learns to put motion on the right objects, one common failure mode is that the objects lack resolution. For example, the people in the beaches and golf courses are often blobs. Nevertheless, we believe it is promising that our model can generate short motions. We visualize the behavior of the two-stream architecture in Figure 3.

Baseline: Since to our knowledge there are no existing large-scale generative models of video ([33] requires an input frame), we develop a simple but reasonable baseline for this task. We train an autoencoder over our data. The encoder is similar to the discriminator network (except producing 100 dimensional code), while the decoder follows the two-stream generator network. Hence, the baseline autoencoder network has a similar number of parameters as our full approach. We then feed examples through the encoder and fit a Gaussian Mixture Model (GMM) with 256 components over the 100 dimensional hidden space. To generate a novel video, we sample from this GMM, and feed the sample through the decoder.

Evaluation Metric: We quantitatively evaluate our generation using a psychophysical two-alternative forced choice with workers on Amazon Mechanical Turk. We show a worker two random videos,

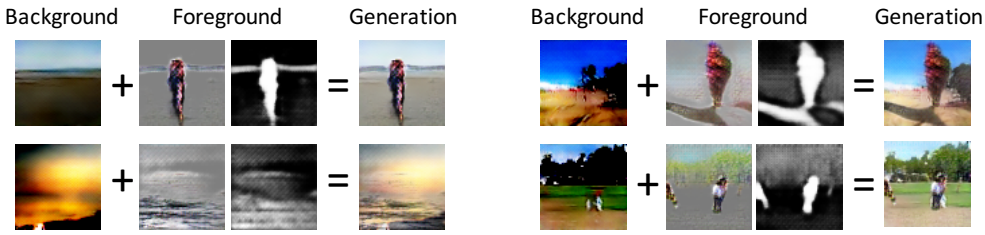


Figure 3: **Streams:** We visualize the background, foreground, and masks for beaches (left) and golf (right). The network generally learns to disentangle the foreground from the background.

	Percentage of Trials				
"Which video is more realistic?"	Golf	Beach	Train	Baby	Mean
Random Preference	50	50	50	50	50
Prefer VGAN Two Stream over Autoencoder	88	83	87	71	82
Prefer VGAN One Stream over Autoencoder	85	88	85	73	82
Prefer VGAN Two Stream over VGAN One Stream	55	58	47	52	53
Prefer VGAN Two Stream over Real	21	23	23	6	18
Prefer VGAN One Stream over Real	17	21	19	8	16
Prefer Autoencoder over Real	4	2	4	2	3

Table 1: **Video Generation Preferences:** We show two videos to workers on Amazon Mechanical Turk, and ask them to choose which video is more realistic. The table shows the percentage of times that workers prefer one generations from one model over another. In all cases, workers tend to prefer video generative adversarial networks over an autoencoder. In most cases, workers show a slight preference for the two-stream model.

and ask them "Which video is more realistic?" We collected over $13,000$ opinions across 150 unique workers. We paid workers one cent per comparison, and required workers to historically have a 95% approval rating on MTurk. We experimented with removing bad workers that frequently said real videos were not realistic, but the relative rankings did not change. We designed this experiment following advice from [38], which advocates evaluating generative models for the task at hand. In our case, we are interested in perceptual quality of motion. We consider a model X better than model Y if workers prefer generations from X more than generations from Y.

Quantitative Results: Table 1 shows the percentage of times that workers preferred generations from one model over another. Workers consistently prefer videos from the generative adversarial network more than an autoencoder. Additionally, workers show a slight preference for the two-stream architecture, especially in scenes where the background is large (e.g., golf course, beach). Although the one-stream architecture is capable of generating stationary backgrounds, it may be difficult to find this solution, motivating a more explicit architecture. The one-stream architecture generally produces high-frequency temporal flickering in the background. To evaluate whether static frames are better than our generations, we also ask workers to choose between our videos and a static frame, and workers only chose the static frame 38% of the time, suggesting our model produces more realistic motion than static frames on average. Finally, while workers generally can distinguish real videos from generated videos, the workers show the most confusion with our two-stream model compared to baselines, suggesting the two-stream generations may be more realistic on average.

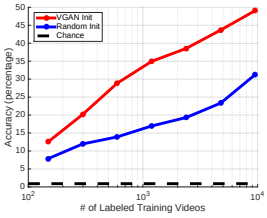
3.3 Video Representation Learning

We also experimented with using our model as a way to learn unsupervised representations for video. We train our two-stream model with over $5,000$ hours of unfiltered, unlabeled videos from Flickr. We then fine-tune the discriminator on the task of interest (e.g., action recognition) using a relatively small set of labeled video. To do this, we replace the last layer (which is a binary classifier) with a K-way softmax classifier. We also add dropout [36] to the penultimate layer to reduce overfitting.

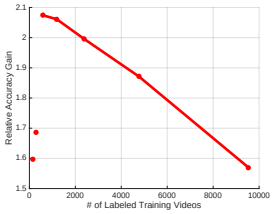
Action Classification: We evaluated performance on classifying actions on UCF101 [35]. We report accuracy in Figure 4a. Initializing the network with the weights learned from the generative adversarial network outperforms a randomly initialized network, suggesting that it has learned an useful internal representation for video. Interestingly, while a randomly initialized network under-performs hand-crafted STIP features [35], the network initialized with our model significantly

Method	Accuracy
Chance	0.9%
STIP Features [35]	43.9%
Temporal Coherence [10]	45.4%
Shuffle and Learn [24]	50.2%
VGAN + Random Init	36.7%
VGAN + Logistic Reg	49.3%
VGAN + Fine Tune	**52.1%**
ImageNet Supervision [46]	91.4%

(a) Accuracy with Unsupervised Methods



(b) Performance vs # Data



(c) Relative Gain vs # Data

Figure 4: **Video Representation Learning:** We evaluate the representation learned by the discriminator for action classification on UCF101 [35]. **(a)** By fine-tuning the discriminator on a relatively small labeled dataset, we can obtain better performance than random initialization, and better than hand-crafted space-time interest point (STIP) features. Moreover, our model slightly outperforms another unsupervised video representation [24] despite using an order of magnitude fewer learned parameters and only 64×64 videos. Note unsupervised video representations are still far from models that leverage external supervision. **(b)** Our unsupervised representation with less labeled data outperforms random initialization with all the labeled data. Our results suggest that, with just 1/8th of the labeled data, we can match performance to a randomly initialized network that used all of the labeled data. **(c)** The fine-tuned model has larger relative gain over random initialization in cases with less labeled data. Note that (a) is over all train/test splits of UCF101, while (b,c) is over the first split in order to make experiments less expensive.

outperforms it. We also experimented with training a logistic regression on only the last layer, which performed worse. Finally, our model slightly outperforms another recent unsupervised video representation learning approach [24]. However, our approach uses an order of magnitude fewer parameters, less layers (5 layers vs 8 layers), and low-resolution video.

Performance vs Data: We also experimented with varying the amount of labeled training data available to our fine-tuned network. Figure 4b reports performance versus the amount of labeled training data available. As expected, performance increases with more labeled data. The fine-tuned model shows an advantage in low data regimes: even with *one eighth* of the labeled data, the fine-tuned model still beats a randomly initialized network. Moreover, Figure 4c plots the relative accuracy gain over the fine-tuned model and the random initialization (fine-tuned performance divided by random initialized performance). This shows that fine-tuning with our model has larger relative gain over random initialization in cases with less labeled data, showing its utility in low-data regimes.

3.4 Future Generation

We investigate whether our approach can be used to generate the future of a static image. Specifically, given a static image x_0, can we extrapolate a video of possible consequent frames?

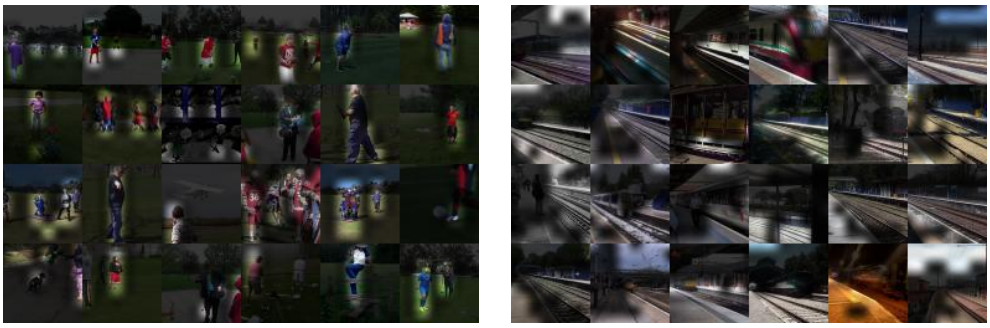
Encoder: We utilize the same model as our two-stream model, however we must make one change in order to input the static image instead of the latent code. We can do this by attaching a five-layer convolutional network to the front of the generator which encodes the image into the latent space, similar to a conditional generative adversarial network [23]. The rest of the generator and discriminator networks remain the same. However, we add an additional loss term that minimizes the L1 distance between the input and the first frame of the generated image. We do this so that the generator creates videos consistent with the input image. We train from scratch with the objective:

$$\begin{aligned}\min_{w_G}\max_{w_D} \mathbb{E}_{x\sim p_x(x)}\left[\log D(x;w_D)\right] + \mathbb{E}_{x_0\sim p_{x_0}(x_0)}\left[\log\left(1 - D(G(x_0;w_G);w_D)\right)\right] \\ + \mathbb{E}_{x_0\sim p_{x_0}(x_0)}\left[\lambda\|x_0 - G^0(x_0;w_G)\|_2^2\right]\end{aligned} \tag{3}$$

where x_0 is the first frame of the input, $G^0(\cdot)$ is the first frame of the generated video, and $\lambda \in \mathcal{R}$ is a hyperparameter. The discriminator will try to classify realistic frames and realistic motions as before, while the generator will try to produce a realistic video such that the first frame is reconstructed well.

Results: We qualitatively show a few examples of our approach in Figure 5 using held-out testing videos. Although the extrapolations are rarely correct, they often have fairly plausible motions. The

Figure 5: **Future Generation:** We show one application of generative video models where we predict videos given a single static image. The red arrows highlight regions of motion. Since this is an ambiguous task, our model usually does not generate the correct video, however the generation is often plausible. Please see `http://mit.edu/vondrick/tinyvideo` for animated movies.



(a) hidden unit that fires on "person" (b) hidden unit that fires on "train tracks"

Figure 6: **Visualizing Representation:** We visualize some hidden units in the encoder of the future generator, following the technique from [53]. We highlight regions of images that a particular convolutional hidden unit maximally activates on. While not at all units are semantic, some units activate on objects that are sources for motion, such as people and train tracks.

most common failure is that the generated video has a scene similar but not identical to the input image, such as by changing colors or dropping/hallucinating objects. The former could be solved by a color histogram normalization in post-processing (which we did not do for simplicity), while we suspect the latter will require building more powerful generative models. The generated videos are usually not the correct video, but we observe that often the motions are plausible. We are not aware of an existing approach that can directly generate multi-frame videos from a single static image. [33, 22] can generate video, but they require multiple input frames and empirically become blurry after extrapolating many frames. [44, 51] can predict optic flow from a single image, but they do not generate several frames of motion and may be susceptible to warping artifacts. We believe this experiment shows an important application of generative video models.

Visualizing Representation: Since generating the future requires understanding how objects move, the network may need learn to recognize some objects internally, even though it is not supervised to do so. Figure 6 visualizes some activations of hidden units in the third convolutional layer. While not at all units are semantic, some of the units tend to be selective for objects that are sources of motion, such as people or train tracks. These visualizations suggest that scaling up future generation might be a promising supervisory signal for object recognition and complementary to [27, 5, 47].

Conclusion: Understanding scene dynamics will be crucial for the next generation of computer vision systems. In this work, we explored how to learn some dynamics from large amounts of unlabeled video by capitalizing on adversarial learning methods. Since annotating dynamics is expensive, we believe learning from unlabeled data is a promising direction. While we are still a long way from fully harnessing the potential of unlabeled video, our experiments support that abundant unlabeled video can be lucrative for both learning to generate videos and learning visual representations.

Acknowledgements: We thank Yusuf Aytar for dataset discussions. We thank MIT TIG, especially Garrett Wollman, for troubleshooting issues on storing the 26 TB of video. We are grateful for the Torch7 community for answering many questions. NVidia donated GPUs used for this research. This work was supported by NSF grant #1524817 to AT, START program at UMBC to HP, and the Google PhD fellowship to CV.

References

[1] Yusuf Aytar, Carl Vondrick, and Antonio Torralba. Learning sound representations from unlabeled video. *NIPS*, 2016.

[2] Tali Basha, Yael Moses, and Shai Avidan. Photo sequencing. In *ECCV*. 2012.

[3] Chao-Yeh Chen and Kristen Grauman. Watching unlabeled video helps learn new human actions from very few labeled snapshots. In *CVPR*, 2013.

[4] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015.

[5] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015.

[6] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *arXiv*, 2016.

[7] József Fiser and Richard N Aslin. Statistical learning of higher-order temporal structure from visual shape sequences. *JEP*, 2002.

[8] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *ICCV*, 2015.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[10] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv*, 2015.

[12] Phillip Isola, Joseph J Lim, and Edward H Adelson. Discovering states and transformations in image collections. In *CVPR*, 2015.

[13] Dinesh Jayaraman and Kristen Grauman. Learning image representations tied to ego-motion. In *ICCV*, 2015.

[14] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *PAMI*, 2013.

[15] Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *arXiv*, 2016.

[16] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014.

[17] Kris Kitani, Brian Ziebart, James Bagnell, and Martial Hebert. Activity forecasting. *ECCV*, 2012.

[18] Quoc V Le. Building high-level features using large scale unsupervised learning. In *CASSP*, 2013.

[19] Yin Li, Manohar Paluri, James M Rehg, and Piotr Dollár. Unsupervised learning of edges. *arXiv*, 2015.

[20] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv*, 2016.

[21] David G Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[22] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv*, 2015.

[23] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv*, 2014.

[24] Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. Shuffle and Learn: Unsupervised Learning using Temporal Order Verification. In *ECCV*, 2016.

[25] Hossein Mobahi, Ronan Collobert, and Jason Weston. Deep learning from temporal coherence in video. In *ICML*, 2009.

[26] Phuc Xuan Nguyen, Gregory Rogez, Charless Fowlkes, and Deva Ramanan. The open world of micro-videos. *arXiv*, 2016.

[27] Andrew Owens, Jiajun Wu, Josh H McDermott, William T Freeman, and Antonio Torralba. Ambient sound provides supervision for visual learning. *arXiv*, 2016.

[28] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. *arXiv*, 2016.

[29] Nikola Petrovic, Aleksandar Ivanovic, and Nebojsa Jojic. Recursive estimation of generative models of video. In *CVPR*, 2006.

[30] Lyndsey Pickup, Zheng Pan, Donglai Wei, YiChang Shih, Changshui Zhang, Andrew Zisserman, Bernhard Scholkopf, and William Freeman. Seeing the arrow of time. In *CVPR*, 2014.

[31] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv*, 2015.
[32] Vignesh Ramanathan, Kevin Tang, Greg Mori, and Li Fei-Fei. Learning temporal embeddings for complex video analysis. In *CVPR*, 2015.
[33] MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv*, 2014.
[34] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.
[35] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv*, 2012.
[36] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.
[37] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. *arXiv*, 2015.
[38] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv*, 2015.
[39] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *ACM*, 2016.
[40] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. *arXiv*, 2014.
[41] Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently scaling up crowdsourced video annotation. *IJCV*, 2013.
[42] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating visual representations from unlabeled video. *CVPR*, 2015.
[43] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *ECCV*, 2016.
[44] Jacob Walker, Abhinav Gupta, and Martial Hebert. Patch to the future: Unsupervised visual prediction. In *CVPR*, 2014.
[45] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *ICCV*, 2013.
[46] Limin Wang, Yuanjun Xiong, Zhe Wang, and Yu Qiao. Towards good practices for very deep two-stream convnets. *arXiv*, 2015.
[47] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015.
[48] Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. *arXiv*, 2016.
[49] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv*, 2015.
[50] Tianfan Xue, Jiajun Wu, Katherine L Bouman, and William T Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. *arXiv*, 2016.
[51] Jenny Yuen and Antonio Torralba. A data-driven approach for event prediction. In *ECCV*. 2010.
[52] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *CVPR*, 2010.
[53] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv*, 2014.
[54] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.
[55] Yipin Zhou and Tamara L Berg. Temporal perception and prediction in ego-centric video. In *ICCV*, 2015.
[56] Yipin Zhou and Tamara L Berg. Learning temporal transformations from time-lapse videos. In *ECCV*, 2016.

Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling

Jiajun Wu*
MIT CSAIL

Chengkai Zhang*
MIT CSAIL

Tianfan Xue
MIT CSAIL

William T. Freeman
MIT CSAIL, Google Research

Joshua B. Tenenbaum
MIT CSAIL

Abstract

We study the problem of 3D object generation. We propose a novel framework, namely 3D Generative Adversarial Network (3D-GAN), which generates 3D objects from a probabilistic space by leveraging recent advances in volumetric convolutional networks and generative adversarial nets. The benefits of our model are three-fold: first, the use of an adversarial criterion, instead of traditional heuristic criteria, enables the generator to capture object structure implicitly and to synthesize high-quality 3D objects; second, the generator establishes a mapping from a low-dimensional probabilistic space to the space of 3D objects, so that we can sample objects without a reference image or CAD models, and explore the 3D object manifold; third, the adversarial discriminator provides a powerful 3D shape descriptor which, learned without supervision, has wide applications in 3D object recognition. Experiments demonstrate that our method generates high-quality 3D objects, and our unsupervisedly learned features achieve impressive performance on 3D object recognition, comparable with those of supervised learning methods.

1 Introduction

What makes a 3D generative model of object shapes appealing? We believe a good generative model should be able to synthesize 3D objects that are both highly varied and realistic. Specifically, for 3D objects to have variations, a generative model should be able to go beyond memorizing and recombining parts or pieces from a pre-defined repository to produce novel shapes; and for objects to be realistic, there need to be fine details in the generated examples.

In the past decades, researchers have made impressive progress on 3D object modeling and synthesis [Van Kaick et al., 2011, Tangelder and Veltkamp, 2008, Carlson, 1982], mostly based on meshes or skeletons. Many of these traditional methods synthesize new objects by borrowing parts from objects in existing CAD model libraries. Therefore, the synthesized objects look realistic, but not conceptually novel.

Recently, with the advances in deep representation learning and the introduction of large 3D CAD datasets like ShapeNet [Chang et al., 2015, Wu et al., 2015], there have been some inspiring attempts in learning deep object representations based on voxelized objects [Girdhar et al., 2016, Su et al., 2015a, Qi et al., 2016]. Different from part-based methods, many of these generative approaches do not explicitly model the concept of parts or retrieve them from an object repository; instead, they synthesize new objects based on learned object representations. This is a challenging problem because, compared to the space of 2D images, it is more difficult to model the space of 3D shapes due to its higher dimensionality. Their current results are encouraging, but often there still exist artifacts (*e.g.*, fragments or holes) in the generated objects.

In this paper, we demonstrate that modeling volumetric objects in a general-adversarial manner could be a promising solution to generate objects that are both novel and realistic. Our approach combines

∗ indicates equal contributions. Emails: {jiajunwu, ckzhang, tfxue, billf, jbt}@mit.edu

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

the merits of both general-adversarial modeling [Goodfellow et al., 2014, Radford et al., 2016] and volumetric convolutional networks [Maturana and Scherer, 2015, Wu et al., 2015]. Different from traditional heuristic criteria, generative-adversarial modeling introduces an adversarial discriminator to classify whether an object is synthesized or real. This could be a particularly favorable framework for 3D object modeling: as 3D objects are highly structured, a generative-adversarial criterion, but not a voxel-wise independent heuristic one, has the potential to capture the structural difference of two 3D objects. The use of a generative-adversarial loss may also avoid possible criterion-dependent overfitting (*e.g.*, generating mean-shape-like blurred objects when minimizing a mean squared error).

Modeling 3D objects in a generative-adversarial way offers additional distinctive advantages. First, it becomes possible to sample novel 3D objects from a probabilistic latent space such as a Gaussian or uniform distribution. Second, the discriminator in the generative-adversarial approach carries informative features for 3D object recognition, as demonstrated in experiments (Section 4). From a different perspective, instead of learning a single feature representation for both generating and recognizing objects [Girdhar et al., 2016, Sharma et al., 2016], our framework learns disentangled generative and discriminative representations for 3D objects without supervision, and applies them on generation and recognition tasks, respectively.

We show that our generative representation can be used to synthesize high-quality realistic objects, and our discriminative representation can be used for 3D object recognition, achieving comparable performance with recent supervised methods [Maturana and Scherer, 2015, Shi et al., 2015], and outperforming other unsupervised methods by a large margin. The learned generative and discriminative representations also have wide applications. For example, we show that our network can be combined with a variational autoencoder [Kingma and Welling, 2014, Larsen et al., 2016] to directly reconstruct a 3D object from a 2D input image. Further, we explore the space of object representations and demonstrate that both our generative and discriminative representations carry rich semantic information about 3D objects.

2 Related Work

Modeling and synthesizing 3D shapes 3D object understanding and generation is an important problem in the graphics and vision community, and the relevant literature is very rich [Carlson, 1982, Tangelder and Veltkamp, 2008, Van Kaick et al., 2011, Blanz and Vetter, 1999, Kalogerakis et al., 2012, Chaudhuri et al., 2011, Xue et al., 2012, Kar et al., 2015, Bansal et al., 2016, Wu et al., 2016]. Since decades ago, AI and vision researchers have made inspiring attempts to design or learn 3D object representations, mostly based on meshes and skeletons. Many of these shape synthesis algorithms are nonparametric and they synthesize new objects by retrieving and combining shapes and parts from a database. Recently, Huang et al. [2015] explored generating 3D shapes with pre-trained templates and producing both object structure and surface geometry. Our framework synthesizes objects without explicitly borrow parts from a repository, and requires no supervision during training.

Deep learning for 3D data The vision community have witnessed rapid development of deep networks for various tasks. In the field of 3D object recognition, Li et al. [2015], Su et al. [2015b], Girdhar et al. [2016] proposed to learn a joint embedding of 3D shapes and synthesized images, Su et al. [2015a], Qi et al. [2016] focused on learning discriminative representations for 3D object recognition, Wu et al. [2016], Xiang et al. [2015], Choy et al. [2016] discussed 3D object reconstruction from in-the-wild images, possibly with a recurrent network, and Girdhar et al. [2016], Sharma et al. [2016] explored autoencoder-based networks for learning voxel-based object representations. Wu et al. [2015], Rezende et al. [2016], Yan et al. [2016] attempted to generate 3D objects with deep networks, some using 2D images during training with a 3D to 2D projection layer. Many of these networks can be used for 3D shape classification [Su et al., 2015a, Sharma et al., 2016, Maturana and Scherer, 2015], 3D shape retrieval [Shi et al., 2015, Su et al., 2015a], and single image 3D reconstruction [Kar et al., 2015, Bansal et al., 2016, Girdhar et al., 2016], mostly with full supervision. In comparison, our framework requires no supervision for training, is able to generate objects from a probabilistic space, and comes with a rich discriminative 3D shape representation.

Learning with an adversarial net Generative Adversarial Nets (GAN) [Goodfellow et al., 2014] proposed to incorporate an adversarial discriminator into the procedure of generative modeling. More recently, LAPGAN [Denton et al., 2015] and DC-GAN [Radford et al., 2016] adopted GAN with convolutional networks for image synthesis, and achieved impressive performance. Researchers have also explored the use of GAN for other vision problems. To name a few, Wang and Gupta [2016] discussed how to model image style and structure with sequential GANs, Li and Wand [2016] and Zhu et al. [2016] used GAN for texture synthesis and image editing, respectively, and Im et al. [2016]

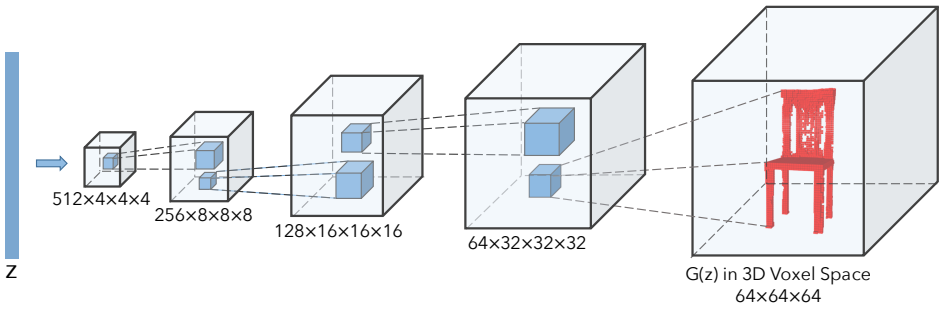


Figure 1: The generator in 3D-GAN. The discriminator mostly mirrors the generator.

developed a recurrent adversarial network for image generation. While previous approaches focus on modeling 2D images, we discuss the use of an adversarial component in modeling 3D objects.

3 Models

In this section we introduce our model for 3D object generation. We first discuss how we build our framework, 3D Generative Adversarial Network (3D-GAN), by leveraging previous advances on volumetric convolutional networks and generative adversarial nets. We then show how to train a variational autoencoder [Kingma and Welling, 2014] simultaneously so that our framework can capture a mapping from a 2D image to a 3D object.

3.1 3D Generative Adversarial Network (3D-GAN)

As proposed in Goodfellow et al. [2014], the Generative Adversarial Network (GAN) consists of a generator and a discriminator, where the discriminator tries to classify real objects and objects synthesized by the generator, and the generator attempts to confuse the discriminator. In our 3D Generative Adversarial Network (3D-GAN), the generator G maps a 200-dimensional latent vector z, randomly sampled from a probabilistic latent space, to a $64 \times 64 \times 64$ cube, representing an object $G(z)$ in 3D voxel space. The discriminator D outputs a confidence value $D(x)$ of whether a 3D object input x is real or synthetic.

Following Goodfellow et al. [2014], we use binary cross entropy as the classification loss, and present our overall adversarial loss function as

$$L_{\text{3D-GAN}} = \log D(x) + \log(1 - D(G(z))), \tag{1}$$

where x is a real object in a $64 \times 64 \times 64$ space, and z is a randomly sampled noise vector from a distribution $p(z)$. In this work, each dimension of z is an i.i.d. uniform distribution over $[0, 1]$.

Network structure Inspired by Radford et al. [2016], we design an all-convolutional neural network to generate 3D objects. As shown in Figure 1, the generator consists of five volumetric fully convolutional layers of kernel sizes $4 \times 4 \times 4$ and strides 2, with batch normalization and ReLU layers added in between and a Sigmoid layer at the end. The discriminator basically mirrors the generator, except that it uses Leaky ReLU [Maas et al., 2013] instead of ReLU layers. There are no pooling or linear layers in our network. More details can be found in the supplementary material.

Training details A straightforward training procedure is to update both the generator and the discriminator in every batch. However, the discriminator usually learns much faster than the generator, possibly because generating objects in a 3D voxel space is more difficult than differentiating between real and synthetic objects [Goodfellow et al., 2014, Radford et al., 2016]. It then becomes hard for the generator to extract signals for improvement from a discriminator that is way ahead, as all examples it generated would be correctly identified as synthetic with high confidence. Therefore, to keep the training of both networks in pace, we employ an adaptive training strategy: for each batch, the discriminator only gets updated if its accuracy in the last batch is not higher than 80%. We observe this helps to stabilize the training and to produce better results. We set the learning rate of G to 0.0025, D to 10^{-5}, and use a batch size of 100. We use ADAM [Kingma and Ba, 2015] for optimization, with $\beta = 0.5$.

3.2 3D-VAE-GAN

We have discussed how to generate 3D objects by sampling a latent vector z and mapping it to the object space. In practice, it would also be helpful to infer these latent vectors from observations. For example, if there exists a mapping from a 2D image to the latent representation, we can then recover the 3D object corresponding to that 2D image.

Following this idea, we introduce 3D-VAE-GAN as an extension to 3D-GAN. We add an additional image encoder E, which takes a 2D image x as input and outputs the latent representation vector z. This is inspired by VAE-GAN proposed by [Larsen et al., 2016], which combines VAE and GAN by sharing the decoder of VAE with the generator of GAN.

The 3D-VAE-GAN therefore consists of three components: an image encoder E, a decoder (the generator G in 3D-GAN), and a discriminator D. The image encoder consists of five spatial convolution layers with kernel size $\{11, 5, 5, 5, 8\}$ and strides $\{4, 2, 2, 2, 1\}$, respectively. There are batch normalization and ReLU layers in between, and a sampler at the end to sample a 200 dimensional vector used by the 3D-GAN. The structures of the generator and the discriminator are the same as those in Section 3.1.

Similar to VAE-GAN [Larsen et al., 2016], our loss function consists of three parts: an object reconstruction loss L_{recon}, a cross entropy loss $L_{\text{3D-GAN}}$ for 3D-GAN, and a KL divergence loss L_{KL} to restrict the distribution of the output of the encoder. Formally, these loss functions write as

$$L = L_{\text{3D-GAN}} + \alpha_1 L_{\text{KL}} + \alpha_2 L_{\text{recon}}, \tag{2}$$

where α_1 and α_2 are weights of the KL divergence loss and the reconstruction loss. We have

$$L_{\text{3D-GAN}} = \log D(x) + \log(1 - D(G(z))), \tag{3}$$
$$L_{\text{KL}} = D_{\text{KL}}(q(z|y) \,||\, p(z)), \tag{4}$$
$$L_{\text{recon}} = ||G(E(y)) - x||_2, \tag{5}$$

where x is a 3D shape from the training set, y is its corresponding 2D image, and $q(z|y)$ is the variational distribution of the latent representation z. The KL-divergence pushes this variational distribution towards to the prior distribution $p(z)$, so that the generator can sample the latent representation z from the same distribution $p(z)$. In this work, we choose $p(z)$ a multivariate Gaussian distribution with zero-mean and unit variance. For more details, please refer to Larsen et al. [2016].

Training 3D-VAE-GAN requires both 2D images and their corresponding 3D models. We render 3D shapes in front of background images ($16,913$ indoor images from the SUN database [Xiao et al., 2010]) in 72 views (from 24 angles and 3 elevations). We set $\alpha_1 = 5$, $\alpha_2 = 10^{-4}$, and use a similar training strategy as in Section 3.1. See our supplementary material for more details.

4 Evaluation

In this section, we evaluate our framework from various aspects. We first show qualitative results of generated 3D objects. We then evaluate the unsupervisedly learned representation from the discriminator by using them as features for 3D object classification. We show both qualitative and quantitative results on the popular benchmark ModelNet [Wu et al., 2015]. Further, we evaluate our 3D-VAE-GAN on 3D object reconstruction from a single image, and show both qualitative and quantitative results on the IKEA dataset [Lim et al., 2013].

4.1 3D Object Generation

Figure 2 shows 3D objects generated by our 3D-GAN. For this experiment, we train one 3D-GAN for each object category. For generation, we sample 200-dimensional vectors following an i.i.d. uniform distribution over $[0, 1]$, and render the largest connected component of each generated object. We compare 3D-GAN with Wu et al. [2015], the state-of-the-art in 3D object synthesis from a probabilistic space, and with a volumetric autoencoder, whose variants have been employed by multiple recent methods [Girdhar et al., 2016, Sharma et al., 2016]. Because an autoencoder does not restrict the distribution of its latent representation, we compute the empirical distribution $p_0(z)$ of the latent vector z of all training examples, fit a Gaussian distribution g_0 to p_0, and sample from g_0. Our algorithm produces 3D objects with much higher quality and more fine-grained details.

Compared with previous works, our 3D-GAN can synthesize high-resolution 3D objects with detailed geometries. Figure 3 shows both high-res voxels and down-sampled low-res voxels for comparison. Note that it is relatively easy to synthesize a low-res object, but is much harder to obtain a high-res one due to the rapid growth of 3D space. However, object details are only revealed in high resolution.

A natural concern to our generative model is whether it is simply memorizing objects from training data. To demonstrate that the network can generalize beyond the training set, we compare synthesized objects with their nearest neighbor in the training set. Since the retrieval objects based on ℓ^2 distance in the voxel space are visually very different from the queries, we use the output of the last convolutional

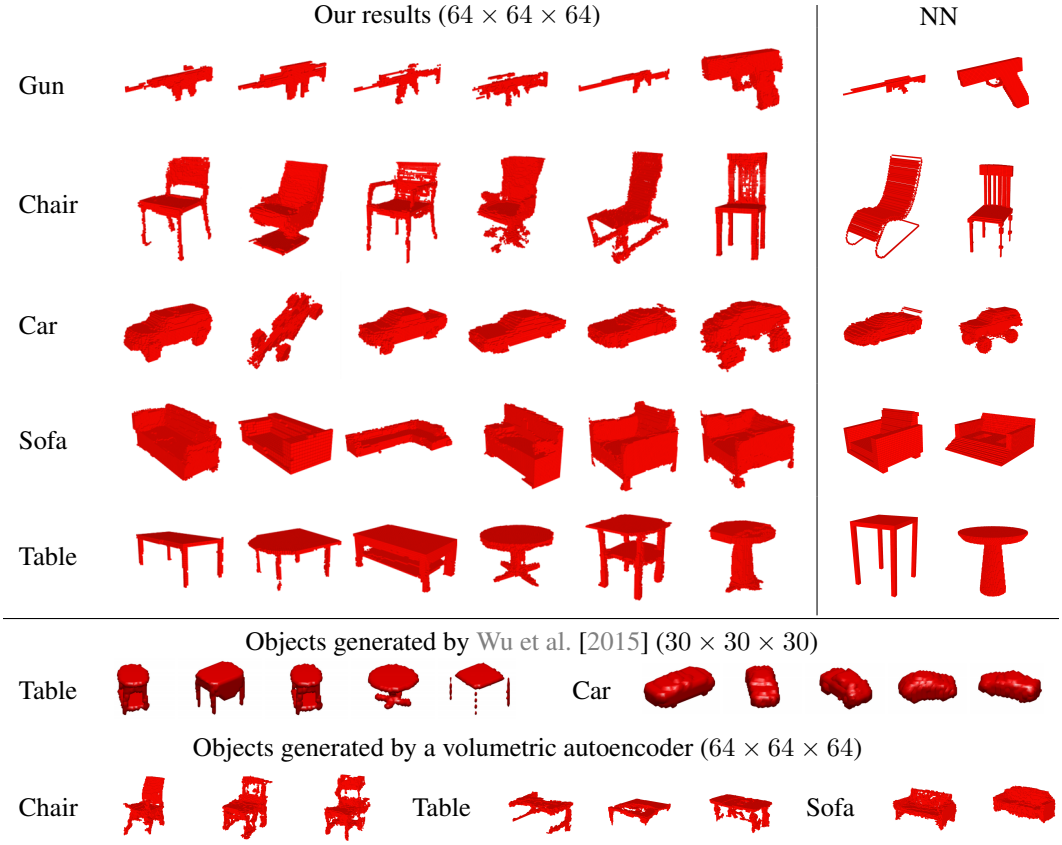


Figure 2: Objects generated by 3D-GAN from vectors, without a reference image/object. We show, for the last two objects in each row, the nearest neighbor retrieved from the training set. We see that the generated objects are similar, but not identical, to examples in the training set. For comparison, we show objects generated by the previous state-of-the-art [Wu et al., 2015] (results supplied by the authors). We also show objects generated by autoencoders trained on a single object category, with latent vectors sampled from empirical distribution. See text for details.

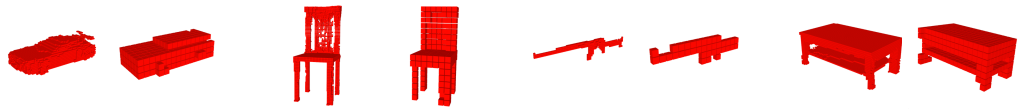


Figure 3: We present each object at high resolution ($64 \times 64 \times 64$) on the left and at low resolution (down-sampled to $16 \times 16 \times 16$) on the right. While humans can perceive object structure at a relatively low resolution, fine details and variations only appear in high-res objects.

layer in our discriminator (with a 2x pooling) as features for retrieval instead. Figure 2 shows that generated objects are similar, but not identical, to the nearest examples in the training set.

4.2 3D Object Classification

We then evaluate the representations learned by our discriminator. A typical way of evaluating representations learned without supervision is to use them as features for classification. To obtain features for an input 3D object, we concatenate the responses of the second, third, and fourth convolution layers in the discriminator, and apply max pooling of kernel sizes $\{8, 4, 2\}$, respectively. We use a linear SVM for classification.

Data We train a single 3D-GAN on the seven major object categories (chairs, sofas, tables, boats, airplanes, rifles, and cars) of ShapeNet [Chang et al., 2015]. We use ModelNet [Wu et al., 2015] for testing, following Sharma et al. [2016], Maturana and Scherer [2015], Qi et al. [2016].* Specifically, we evaluate our model on both ModelNet10 and ModelNet40, two subsets of ModelNet that are often

*For ModelNet, there are two train/test splits typically used. Qi et al. [2016], Shi et al. [2015], Maturana and Scherer [2015] used the train/test split included in the dataset, which we also follow; Wu et al. [2015], Su

Supervision	Pretraining	Method	Classification (Accuracy)	
			ModelNet40	ModelNet10
Category labels	ImageNet	MVCNN [Su et al., 2015a]	90.1%	-
		MVCNN-MultiRes [Qi et al., 2016]	**91.4%**	-
	None	3D ShapeNets [Wu et al., 2015]	77.3%	83.5%
		DeepPano [Shi et al., 2015]	77.6%	85.5%
		VoxNet [Maturana and Scherer, 2015]	83.0%	92.0%
		ORION [Sedaghat et al., 2016]	-	**93.8%**
Unsupervised	-	SPH [Kazhdan et al., 2003]	68.2%	79.8%
		LFD [Chen et al., 2003]	75.5%	79.9%
		T-L Network [Girdhar et al., 2016]	74.4%	-
		VConv-DAE [Sharma et al., 2016]	75.5%	80.5%
		3D-GAN (ours)	**83.3%**	**91.0%**

Table 1: Classification results on the ModelNet dataset. Our 3D-GAN outperforms other unsupervised learning methods by a large margin, and is comparable to some recent supervised learning frameworks.

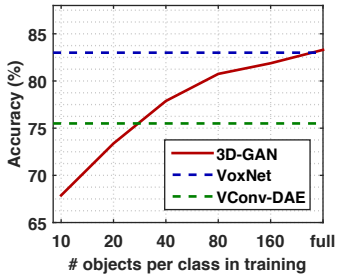


Figure 4: ModelNet40 classification with limited training data

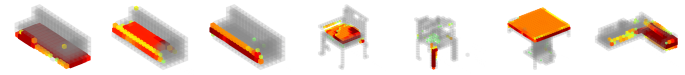


Figure 5: The effects of individual dimensions of the object vector

Figure 6: Intra/inter-class interpolation between object vectors

used as benchmarks for 3D object classification. Note that the training and test categories are not identical, which also shows the out-of-category generalization power of our 3D-GAN.

Results We compare with the state-of-the-art methods [Wu et al., 2015, Girdhar et al., 2016, Sharma et al., 2016, Sedaghat et al., 2016] and show per-class accuracy in Table 1. Our representation outperforms other features learned without supervision by a large margin (83.3% vs. 75.5% on ModelNet40, and 91.0% vs 80.5% on ModelNet10) [Girdhar et al., 2016, Sharma et al., 2016]. Further, our classification accuracy is also higher than some recent supervised methods [Shi et al., 2015], and is close to the state-of-the-art voxel-based supervised learning approaches [Maturana and Scherer, 2015, Sedaghat et al., 2016]. Multi-view CNNs [Su et al., 2015a, Qi et al., 2016] outperform us, though their methods are designed for classification, and require rendered multi-view images and an ImageNet-pretrained model.

3D-GAN also works well with limited training data. As shown in Figure 4, with roughly 25 training samples per class, 3D-GAN achieves comparable performance on ModelNet40 with other unsupervised learning methods trained with at least 80 samples per class.

4.3 Single Image 3D Reconstruction

As an application, our show that the 3D-VAE-GAN can perform well on single image 3D reconstruction. Following previous work [Girdhar et al., 2016], we test it on the IKEA dataset [Lim et al., 2013], and show both qualitative and quantitative results.

Data The IKEA dataset consists of images with IKEA objects. We crop the images so that the objects are centered in the images. Our test set consists of 1,039 objects cropped from 759 images (supplied by the author). The IKEA dataset is challenging because all images are captured in the wild, often with heavy occlusions. We test on all six categories of objects: bed, bookcase, chair, desk, sofa, and table.

Results We show our results in Figure 7 and Table 2, with performance of a single 3D-VAE-GAN jointly trained on all six categories, as well as the results of six 3D-VAE-GANs separately trained on

et al. [2015a], Sharma et al. [2016] used 80 training points and 20 test points in each category for experiments, possibly with viewpoint augmentation.

Method	Bed	Bookcase	Chair	Desk	Sofa	Table	Mean
AlexNet-fc8 [Girdhar et al., 2016]	29.5	17.3	20.4	19.7	38.8	16.0	23.6
AlexNet-conv4 [Girdhar et al., 2016]	38.2	26.6	31.4	26.6	69.3	19.1	35.2
T-L Network [Girdhar et al., 2016]	56.3	30.2	32.9	25.8	71.7	23.3	40.0
3D-VAE-GAN (jointly trained)	49.1	31.9	42.6	34.8	**79.8**	33.1	45.2
3D-VAE-GAN (separately trained)	**63.2**	**46.3**	**47.2**	**40.7**	78.8	**42.3**	**53.1**

Table 2: Average precision for voxel prediction on the IKEA dataset.[†]

Figure 7: Qualitative results of single image 3D reconstruction on the IKEA dataset

each class. Following Girdhar et al. [2016], we evaluate results at resolution $20 \times 20 \times 20$, use the average precision as our evaluation metric, and attempt to align each prediction with the ground-truth over permutations, flips, and translational alignments (up to 10%), as IKEA ground truth objects are not in a canonical viewpoint. In all categories, our model consistently outperforms previous state-of-the-art in voxel-level prediction and other baseline methods.[†]

5 Analyzing Learned Representations

In this section, we look deep into the representations learned by both the generator and the discriminator of 3D-GAN. We start with the 200-dimensional object vector, from which the generator produces various objects. We then visualize neurons in the discriminator, and demonstrate that these units capture informative semantic knowledge of the objects, which justifies its good performance on object classification presented in Section 4.

5.1 The Generative Representation

We explore three methods for understanding the latent space of vectors for object generation. We first visualize what an individual dimension of the vector represents; we then explore the possibility of interpolating between two object vectors and observe how the generated objects change; last, we present how we can apply shape arithmetic in the latent space.

Visualizing the object vector To visualize the semantic meaning of each dimension, we gradually increase its value, and observe how it affects the generated 3D object. In Figure 5, each column corresponds to one dimension of the object vector, where the red region marks the voxels affected by changing values of that dimension. We observe that some dimensions in the object vector carries semantic knowledge of the object, *e.g.*, the thickness or width of surfaces.

Interpolation We show results of interpolating between two object vectors in Figure 6. Earlier works demonstrated interpolation between two 2D images of the same category [Dosovitskiy et al., 2015, Radford et al., 2016]. Here we show interpolations both within and across object categories. We observe that for both cases walking over the latent space gives smooth transitions between objects.

Arithmetic Another way of exploring the learned representations is to show arithmetic in the latent space. Previously, Dosovitskiy et al. [2015], Radford et al. [2016] presented that their generative nets are able to encode semantic knowledge of chair or face images in its latent space; Girdhar et al. [2016] also showed that the learned representation for 3D objects behave similarly. We show our shape arithmetic in Figure 8. Different from Girdhar et al. [2016], all of our objects are randomly sampled, requiring no existing 3D CAD models as input.

5.2 The Discriminative Representation

We now visualize the neurons in the discriminator. Specifically, we would like to show what input objects, and which part of them produce the highest intensity values for each neuron. To do that,

[†]For methods from Girdhar et al. [2016], the mean values in the last column are higher than the originals in their paper, because we compute per-class accuracy instead of per-instance accuracy.

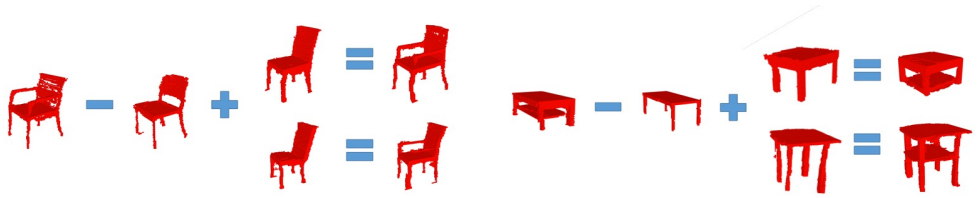


Figure 8: Shape arithmetic for chairs and tables. The left images show the obtained "arm" vector can be added to other chairs, and the right ones show the "layer" vector can be added to other tables.

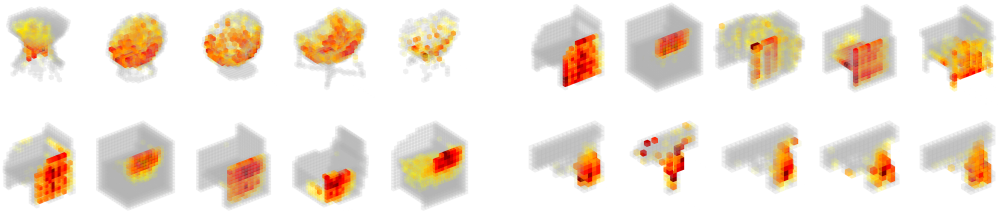


Figure 9: Objects and parts that activate specific neurons in the discriminator. For each neuron, we show five objects that activate it most strongly, with colors representing gradients of activations with respect to input voxels.

for each neuron in the second to last convolutional layer of the discriminator, we iterate through all training objects and exhibit the ones activating the unit most strongly. We further use guided back-propagation [Springenberg et al., 2015] to visualize the parts that produce the activation.

Figure 9 shows the results. There are two main observations: first, for a single neuron, the objects producing strongest activations have very similar shapes, showing the neuron is selective in terms of the overall object shape; second, the parts that activate the neuron, shown in red, are consistent across these objects, indicating the neuron is also learning semantic knowledge about object parts.

6 Conclusion

In this paper, we proposed 3D-GAN for 3D object generation, as well as 3D-VAE-GAN for learning an image to 3D model mapping. We demonstrated that our models are able to generate novel objects and to reconstruct 3D objects from images. We showed that the discriminator in GAN, learned without supervision, can be used as an informative feature representation for 3D objects, achieving impressive performance on shape classification. We also explored the latent space of object vectors, and presented results on object interpolation, shape arithmetic, and neuron visualization.

Acknowledgement This work is supported by NSF grants #1212849 and #1447476, ONR MURI N00014-16-1-2007, the Center for Brain, Minds and Machines (NSF STC award CCF-1231216), Toyota Research Institute, Adobe, Shell, IARPA MICrONS, and a hardware donation from Nvidia.

References

Aayush Bansal, Bryan Russell, and Abhinav Gupta. Marr revisited: 2d-3d alignment via surface normal prediction. In *CVPR*, 2016. 2

Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH*, 1999. 2

Wayne E Carlson. An algorithm and data structure for 3d object synthesis using surface patch intersections. In *SIGGRAPH*, 1982. 1, 2

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 1, 5

Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. Probabilistic reasoning for assembly-based 3d modeling. *ACM TOG*, 30(4):35, 2011. 2

Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. *CGF*, 2003. 6

Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *ECCV*, 2016. 2

Emily L Denton, Soumith Chintala, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015. 2

Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015. 7

Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *ECCV*, 2016. 1, 2, 4, 6, 7

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 2, 3

Haibin Huang, Evangelos Kalogerakis, and Benjamin Marlin. Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. *CGF*, 34(5):25–38, 2015. 2

Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*, 2016. 2

Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM TOG*, 31(4):55, 2012. 2

Abhishek Kar, Shubham Tulsiani, Joao Carreira, and Jitendra Malik. Category-specific object reconstruction from a single image. In *CVPR*, 2015. 2

Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3 d shape descriptors. In *SGP*, 2003. 6

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 3

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014. 2, 3

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *ICML*, 2016. 2, 4

Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. *arXiv preprint arXiv:1604.04382*, 2016. 2

Yangyan Li, Hao Su, Charles Ruizhongtai Qi, Noa Fish, Daniel Cohen-Or, and Leonidas J Guibas. Joint embeddings of shapes and images via cnn image purification. *ACM TOG*, 34(6):234, 2015. 2

Joseph J. Lim, Hamed Pirsiavash, and Antonio Torralba. Parsing ikea objects: Fine pose estimation. In *ICCV*, 2013. 4, 6

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013. 3

Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, 2015. 2, 5, 6

Charles R Qi, Hao Su, Matthias Niessner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *CVPR*, 2016. 1, 2, 5, 6

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 2, 3, 7

Danilo Jimenez Rezende, SM Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. In *NIPS*, 2016. 2

Nima Sedaghat, Mohammadreza Zolfaghari, and Thomas Brox. Orientation-boosted voxel nets for 3d object recognition. *arXiv preprint arXiv:1604.03351*, 2016. 6

Abhishek Sharma, Oliver Grau, and Mario Fritz. Vconv-dae: Deep volumetric shape learning without object labels. *arXiv preprint arXiv:1604.03755*, 2016. 2, 4, 5, 6

Baoguang Shi, Song Bai, Zhichao Zhou, and Xiang Bai. Deeppano: Deep panoramic representation for 3-d shape recognition. *IEEE SPL*, 22(12):2339–2343, 2015. 2, 5, 6

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR Workshop*, 2015. 8

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, 2015a. 1, 2, 5, 6

Hao Su, Charles R Qi, Yangyan Li, and Leonidas Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *ICCV*, 2015b. 2

Johan WH Tangelder and Remco C Veltkamp. A survey of content based 3d shape retrieval methods. *Multimedia tools and applications*, 39(3):441–471, 2008. 1, 2

Oliver Van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. A survey on shape correspondence. *CGF*, 2011. 1, 2

Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. In *ECCV*, 2016. 2

Jiajun Wu, Tianfan Xue, Joseph J Lim, Yuandong Tian, Joshua B Tenenbaum, Antonio Torralba, and William T Freeman. Single image 3d interpreter network. In *ECCV*, 2016. 2

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 1, 2, 4, 5, 6

Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Data-driven 3d voxel patterns for object category recognition. In *CVPR*, 2015. 2

Jianxiong Xiao, James Hays, Krista Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 4

Tianfan Xue, Jianzhuang Liu, and Xiaoou Tang. Example-based 3d object reconstruction from line drawings. In *CVPR*, 2012. 2

Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *NIPS*, 2016. 2

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, 2016. 2

A.1 Network Structure

Here we give the network structures of the generator, the discriminator, and the image encoder.

Generator The generator consists of five fully convolution layers with numbers of channels $\{512, 256, 128, 64, 1\}$, kernel sizes $\{4, 4, 4, 4, 4\}$, and strides $\{1, 2, 2, 2, 2\}$. We add ReLU and batch normalization layers between convolutional layers, and a Sigmoid layer at the end. The input is a 200-dimensional vector, and the output is a $64 \times 64 \times 64$ matrix with values in $[0, 1]$.

Discriminator As a mirrored version of the generator, the discriminator takes as input a $64 \times 64 \times 64$ matrix, and outputs a real number in $[0, 1]$. The discriminator consists of 5 volumetric convolution layers, with numbers of channels $\{64, 128, 256, 512, 1\}$, kernel sizes $\{4, 4, 4, 4, 4\}$, and strides $\{2, 2, 2, 2, 1\}$. There are leaky ReLU layers of parameter 0.2 and batch normalization layers in between, and a Sigmoid layer at the end.

Image encoder The image encoder in our 3D-VAE-GAN takes a $3 \times 256 \times 256$ image as input, and outputs a 200-dimensional vector. It consists of five spatial convolution layers with numbers of channels $\{64, 128, 256, 512, 400\}$, kernel sizes $\{11, 5, 5, 5, 8\}$, and strides $\{4, 2, 2, 2, 1\}$, respectively. There are ReLU and batch normalization layers in between. The output of the last convolution layer is a 400-dimensional vector representing a Gaussian distribution in the 200-dimensional space, where 200 dimensions are for the mean and the other 200 dimensions are for the diagonal variance. There is a sampling layer at the end to sample a 200-dimensional vector from the Gaussian distribution, which is later used by the 3D-GAN.

A.2 3D Shape Classification

Here we present the details of our shape classification experiments. For each object, we take the responses of the second, third, and fourth convolution layers of the discriminator, and apply max pooling of kernel sizes $\{8, 4, 2\}$, respectively. We then concatenate the outputs into a vector of length $7,168$, which is later used by a linear SVM for training and classification.

We use one-versus-all SVM classifiers. We use L2 penalty, balanced class weights, and intercept scaling during training. For ModelNet40, we train a linear SVM with penalty parameter $C = 0.07$, and for ModelNet10, we have $C = 0.01$. We also show the results with limited training data on both ModelNet40 and ModelNet10 in Figure A1.

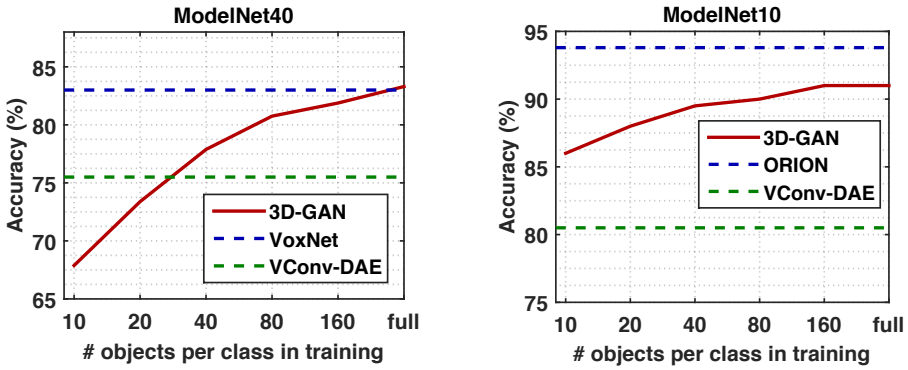


Figure A1: Classification accuracy with limited training data, on ModelNet40 and ModelNet10

A.3 3D-VAE-GAN Training

Let $\{x_i, y_i\}$ be the set of training pairs, where y_i is a 2D image and x_i is the corresponding 3D shape. In each iteration t of training, we first generate a random sample z_t from $N(\mathbf{0}, \mathbf{I})$[‡]. Then we update the discriminator D, the image encoder E, and the generator G sequentially. Specifically,

[‡] $\mathbf{I}$ is an identity matrix.

- Step 1: Update the discriminator D by minimizing the following loss function:

$$\log D(x_i) + \log(1 - D(G(z_t))). \tag{6}$$

- Step 2: Update the image encoder E by minimizing the following loss function:

$$D_{\text{KL}}\left(N(E_{\text{mean}}(y_i), E_{\text{var}}(y_i)) \;||\; N(\mathbf{0}, \mathbf{I})\right) + ||G(E(y_i)) - x_i||_2, \tag{7}$$

  where $E_{\text{mean}}(y_i)$ and $E_{\text{var}}(y_i)$ are the predicted mean and variance of the latent variable z, respectively.

- Step 3: Update the generator G by minimizing the following loss function:

$$\log(1 - D(G(z_t))) + ||G(E(y_i)) - x_i||_2. \tag{8}$$

Temporal Generative Adversarial Nets with Singular Value Clipping

Masaki Saito* Eiichi Matsumoto* Shunta Saito
Preferred Networks inc., Japan
{msaito, matsumoto, shunta}@preferred.jp

Abstract

In this paper, we propose a generative model, Temporal Generative Adversarial Nets (TGAN), which can learn a semantic representation of unlabeled videos, and is capable of generating videos. Unlike existing Generative Adversarial Nets (GAN)-based methods that generate videos with a single generator consisting of 3D deconvolutional layers, our model exploits two different types of generators: a temporal generator and an image generator. The temporal generator takes a single latent variable as input and outputs a set of latent variables, each of which corresponds to an image frame in a video. The image generator transforms a set of such latent variables into a video. To deal with instability in training of GAN with such advanced networks, we adopt a recently proposed model, Wasserstein GAN, and propose a novel method to train it stably in an end-to-end manner. The experimental results demonstrate the effectiveness of our methods.

1. Introduction

Unsupervised learning of feature representation from a large dataset is one of the most significant problems in computer vision. If good representation of data can be obtained from an unlabeled dataset, it could be of benefit to a variety of tasks such as classification, clustering, and generating new data points.

There have been many studies regarding unsupervised learning in the field of computer vision. Their targets are roughly two-fold; images and videos. As for unsupervised learning of images, Generative Adversarial Nets (GAN) [5] have shown impressive results and succeeded to generate plausible images with a dataset that contains plenty of natural images [2, 49]. In contrast, unsupervised learning of videos still has many difficulties compared to images. While recent studies have achieved remarkable progress [35, 25, 15] in a problem that predicts future frames from previous frames, video generation without any clues of data is still a highly challenging problem. Although the recent study tackled to address this problem by decomposing it into background generation and foreground generation, this approach has a drawback that it cannot generate a scene with dynamic background due to the static background assumption [44]. To the best of our knowledge, there is no study that tackles video generation without such assumption and generates diversified videos like natural videos.

Although a simple approach is to use 3D convolutional layers for representing the generating process of a video, it implies that images along x-t plane and y-t plane besides x-y plane are considered equally, where x and y denote the spatial dimensions and t denotes the time dimension. We believe that the nature of time dimension is essentially different from the spatial dimensions in the case of videos so that such approach has difficulty on the video generation problem. The relevance of this assumption has been also discussed in some recent studies [33, 24, 46] that have shown good performance on the video recognition task.

Based on the above discussion, in this paper, we extend an existing GAN model and propose Temporal Generative Adversarial Net (TGAN) that is capable of learning representation from an unlabeled video dataset and producing a new video. Unlike the existing video generator that generates videos with 3D deconvolutional layers [44], in our proposed model the generator consists of two sub networks called a *temporal generator* and an *image generator* (Fig.1). Specifically, the temporal generator first yields a set of latent variables, each of which corresponds to a latent variable for the image generator. Then, the image generator transforms these latent variables into a video which has the same number of frames as the variables. The model comprised of the temporal and image generators can not only enable to efficiently capture the time series, but also be easily extended to frame interpolation.

The typical problem that arises from such advanced networks is the instability of training of GANs. In this paper we adopt a recently proposed Wasserstein GAN (WGAN) which tackles the instability, however, we observed that our model still has sensitivity to a hyperparameter of WGAN. Therefore, to deal with this problem, we propose a novel method to remove the sensitive hyperparameter from WGAN and

*Authors contributed equally

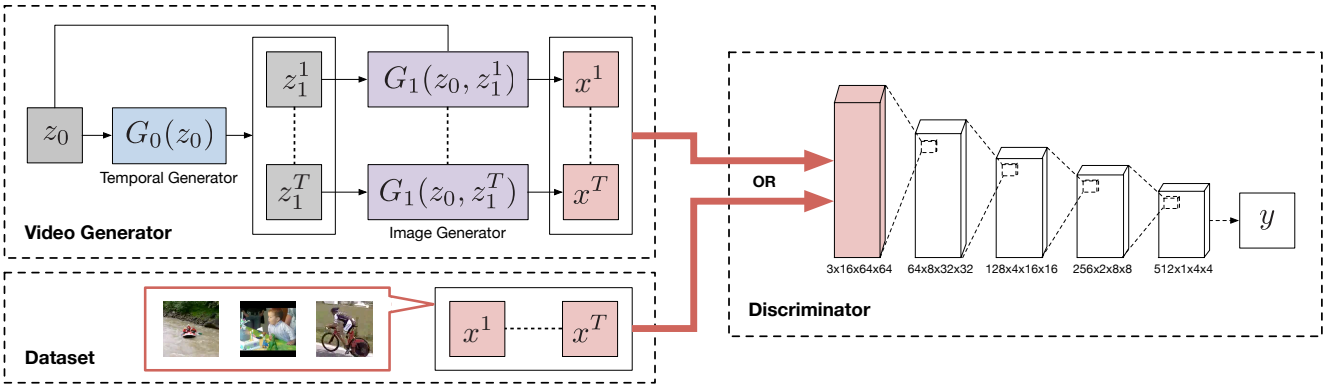


Figure 1. Illustration of TGAN. The video generator consists of two generators, the temporal generator and the image generator. The temporal generator G_0 yields a set of latent variables $z_1^t (t = 1, \ldots, T)$ from z_0. The image generator G_1 transforms those latent variables $z_1^t (t = 1, \ldots, T)$ and z_0 into a video data which has T frames. The discriminator consists of three-dimensional convolutional layers, and evaluates whether these frames are from the dataset or the video generator. The shape of a tensor in the discriminator is denoted as "(channels)×(time)×(height)×(width)".

stabilize the training further. The experiments show that our method is more stable than the conventional methods, and the model can be successfully trained even under the situation where the loss diverges with the conventional methods.

Our contributions are summarized as follows. (i) The generative model that can efficiently capture the latent space of the time dimension in videos. It also enables a natural extension to an application such as frame interpolation. (ii) The alternative parameter clipping method for WGAN that significantly stabilizes the training of the networks that have advanced structure.

2. Related work

2.1. Natural image generation

Supervised learning with Convolutional Neural Networks (CNNs) has recently shown outstanding performance in many tasks such as image classification [8, 9, 11] and action recognition [14, 16, 33, 43], whereas unsupervised learning with CNN has received relatively less attention. A common approach for generating images is the use of undirected graphical models such as Boltzmann machines [31, 18, 4]. However, due to the difficulty of approximating gradients, it has been empirically observed that such deep graphical models frequently fail to find good representation of natural images with sufficient diversity. Both Gregor *et al*. [7] and Dosvotiskiy *et al*. [3] have proposed models that respectively use recurrent and deconvolutional networks, and successfully generated natural images. However, both models make use of supervised learning and require additional information such as labels.

The Generative Adversarial Network (GAN), which we have mainly employed in this study, is a model for unsupervised learning that finds a good representation of samples by simultaneously training two different networks called the *generator* and the *discriminator*. Recently, many extensions for GANs have been proposed. Conditional GANs performs modeling of object attributes [22, 12]. Pathak *et al*. [26] adopted the adversarial network to generate the contents of an image region conditioned on its surroundings. Li and Wand [19] employed the GAN model in order to efficiently synthesize texture. Denton *et al*. [2] proposed a Laplacian GAN that outputs a high-resolution image by iteratively generating images in a coarse-to-fine manner. Arjovsky *et al*. [1] transformed the training of GAN into the minimization problem of Earth Mover's distance, and proposed a more robust method to train both the generator and the discriminator. Radford *et al*. [27] also proposed a simple yet powerful model called Deep Convolutional GAN (DCGAN) for generating realistic images with a pair of convolutional and deconvolutional networks. Based on these results, Wang *et al*. [49] extended DCGAN by factorizing the image generating process into two paths, and proposed a new model called a Style and Structure GAN (S^2-GAN) that exploits two types of generators.

2.2. Video recognition and unsupervised learning

As recognizing videos is a challenging task which has received a lot of attention, many researchers have tackled this problem in various ways. In supervised learning of videos, while a common approach is to use dense trajectories [45, 30, 29], recent methods have employed CNN and achieved state-of-the-art results [14, 16, 33, 43, 24, 46, 47]. Some studies are focused on extracting spatio-temporal feature vectors from a video in an unsupervised manner. Taylor *et al*. [39] proposed a method that extracts invariant features with Restricted Boltzmann Machines (RBMs). Temporal RBMs have also been proposed to explicitly capture the temporal correlations in videos [40, 38, 37]. Stavens and Thrun [36] dealt with this problem by using an optical flow and low-level features such as SIFT. Le *et al*. [17] use Independent Subspace Analysis (ISA) to extract spatio-temporal semantic

features. Deep neural networks have also been applied to feature extraction from videos [51, 6, 48] in the same way as supervised learning.

There also exist several studies focusing on predicting video sequences from an input sequence with Recurrent Neural Networks (RNNs) represented by Long Short-Term Memory (LSTM) [10]. In particular, Ranzato *et al*. [28] proposed a Recurrent Neural Network (RNN) model that can learn both spatial and temporal correlations. Srivastava *et al*. [35] also applied LSTMs and succeeded to predict the future sequence of a simple video. Zhou and Berg [50] proposed a network that creates depictions of objects at future times with LSTMs and DCGAN. Kalchbrenner *et al*. [15] also employed a convolutional LSTM model, and proposed Video Pixel Networks that directly learn the joint distribution of the raw pixel values. Oh *et al*. [25] proposed a deep auto-encoder model conditioned on actions, and predicted next sequences of Atari games from a single screen shot and an action sent by a game pad. In order to deal with the problem that generated sequences are "blurry" compared to natural images, Mithieu *et al*. [21] replaced a standard mean squared error loss and improved the quality of predicted images. However, the above studies cannot directly be applied to the task of generating entire sequences from scratch since they require an initial sequence as an input.

Vondrick *et al*. [44] recently proposed a generative model that yields a video sequence from scratch with DCGAN consisting of 3D deconvolutional layers. The main difference between their model and ours is *model representation*; while they simplified the video generation problem by assuming that a background in a video sequence is always static and generate the video with 3D deconvolutions, we do not use such assumption and decompose the generating process of video into the 1D and 2D deconvolutions.

3. Temporal Generative Adversarial Nets

3.1. Generative Adversarial Nets

Before we go into the details of TGAN, we briefly explain the existing GAN [5] and the Wasserstein GAN [1]. A GAN exploits two networks called the generator and the discriminator. The generator $G : \mathbb{R}^K \to \mathbb{R}^M$ is a function that generates samples $x \in \mathbb{R}^M$ which looks similar to a sample in the given dataset. The input is a latent variable $z \in \mathbb{R}^K$, where z is randomly drawn from a given distribution $p_G(z)$, e.g., a uniform distribution. The discriminator $D : \mathbb{R}^M \to [0, 1]$ is a classifier that discriminates whether a given sample is from the dataset or generated by G.

The GAN simultaneously trains the two networks by playing a non-cooperative game; the generator wins if it generates an image that the discriminator misclassifies, whereas the discriminator wins if it correctly classifies the input samples. Such minimax game can be represented as

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{x\sim p_{\text{data}}}[\ln D(x)] + \mathbb{E}_{z\sim p_G}[\ln(1 - D(G(z)))], \quad (1)$$

where θ_G and θ_D are the parameters of the generator and the discriminator, respectively. p_{data} denotes the empirical data distribution.

3.2. Wasserstein GAN

It is known that the GAN training is unstable and requires careful adjustment of the parameters. To overcome such instability of learning, Arjovsky *et al*. [1] focused on the property that the GAN training can also be interpreted as the minimization of the Jensen-Shannon (JS) divergence, and proposed Wasserstein GAN (WGAN) that trains the generator and the discriminator to minimize an Earth Mover's distance (EMD, a.k.a. first Wasserstein distance) instead of the JS divergence. Several experiments the authors conducted reported that WGANs are more robust than ordinal GANs, and tend to avoid mode dropping.

The significant property in the learning of WGAN is "K-Lipschitz" constraint with regard to the discriminator. Specifically, if the discriminator satisfies the K-Lipschitz constraint, i.e., $|D(x_1) - D(x_2)| \leq K|x_1 - x_2|$ for all x_1 and x_2, the minimax game of WGAN can be represented as

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{x\sim p_{\text{data}}}[D(x)] - \mathbb{E}_{z\sim p_G}[D(G(z))]. \quad (2)$$

Note that unlike the original GAN, the return value of D in Eq.(2) is an unbounded real value, i.e., $D : \mathbb{R}^M \to \mathbb{R}$. In this study we use Eq.(2) for training instead of Eq.(1).

In order to make the discriminator be the K-Lipschitz, the authors proposed a method that clamps all the weights in the discriminator to a fixed box denoted as $w \in [-c, c]$. Although this weight clipping is a simple and assures the discriminator satisfies the K-Lipschitz condition, it also implies we cannot know the relation of the parameters between c and K. As it is known that the objective of the discriminator of Eq.(2) is a good approximate expression of EMD in the case of $K = 1$, this could be a problem when we want to find the approximate value of EMD.

3.3. Temporal GAN

Here we introduce the proposed model based on the above discussion. Let $T > 0$ be the number of frames to be generated, and $G_0 : \mathbb{R}^{K_0} \to \mathbb{R}^{T\times K_1}$ be the temporal generator that gets another latent variable $z_0 \in \mathbb{R}^{K_0}$ as an argument and generates latent variables denoted as $[z_1^1, \ldots, z_1^T]$. In our model, z_0 is randomly drawn from a distribution $p_{G_0}(z_0)$.

Next, we introduce *image generator* $G_1 : \mathbb{R}^{K_0} \times \mathbb{R}^{K_1} \to \mathbb{R}^M$ that yields a video from these latent variables. Note that G_1 takes both the latent variables generated from G_0

Temporal generator	Image generator	
$z_0 \in \mathbb{R}^{1\times100}$	$z_0 \in \mathbb{R}^{1\times100}$	$z_1^t \in \mathbb{R}^{100}$
deconv (1, 512, 0, 1)	linear ($256 \cdot 4^2$)	linear ($256 \cdot 4^2$)
deconv (4, 256, 1, 2)	concat + deconv (4, 256, 1, 2)	
deconv (4, 128, 1, 2)	deconv (4, 128, 1, 2)	
deconv (4, 128, 1, 2)	deconv (4, 64, 1, 2)	
deconv (4, 100, 1, 2)	deconv (4, 32, 1, 2)	
tanh	deconv (3, 3, 1, 1) + tanh	

Table 1. Network configuration of the generator. The second row represents the input variables. "linear $(\cdot)$" is the number of output units in the linear layer. The parameters in the convolutional and the deconvolutional layer are denoted as "conv/deconv ((kernel size), (output channels), (padding), (strides))."

as well as original latent variable z_0 as arguments. While z_1 varies with time, z_0 is invariable regardless of the time, and we empirically observed that it has a significant role in suppressing a sudden change of the action of the generated video. That is, in our representation, the generated video is represented as $[G_1(z_0, z_1^1), \ldots, G_1(z_0, z_1^T)]$.

Using these notations, Eq.(2) can be rewritten as

$$\min_{\theta_{G_0},\theta_{G_1}} \max_{\theta_D} \mathbb{E}_{[x^1,\ldots,x^T]\sim p_{\text{data}}}[D([x^1,\ldots,x^T])] - \mathbb{E}_{z_0\sim p_{G_0}}[D([G_1(z_0,z_1^1),\ldots,G_1(z_0,z_1^T)])]\big), \quad (3)$$

where x^t is the t-th frame of a video in a dataset, and z_1^t is the latent variable corresponding to t-th frame generated by $G_0(z_0)$. θ_D, θ_{G_0}, and θ_{G_1} represent the parameter of D, G_0, and G_1, respectively.

3.4. Network configuration

This subsection describes the configuration of our three networks: the temporal generator, the image generator, and the discriminator. Table 1 shows a typical network setting.

Temporal generator Unlike typical CNNs that perform two-dimensional convolutions in the spatial direction, the deconvolutional layers in the temporal generator perform a one-dimensional deconvolution in the temporal direction. For convenience of computation, we first regard $z_0 \in \mathbb{R}^{K_0}$ as a one-dimensional activation map of $z_0 \in \mathbb{R}^{1\times K_0}$, where the length and the number of channels are one and K_0, respectively. A uniform distribution is used to sample z_0. Next, applying the deconvolutional layers we expand its length while reducing the number of channels. The settings for the deconvolutional layers are the same as those of the image generator except for the number of channels and one-dimensional deconvolution. Like the original image generator we insert a Batch Normalization (BN) layer [13] after deconvolution and use Rectified Linear Units (ReLU) [23] as activation functions.

Image generator The image generator takes two latent variables as arguments. After performing a linear transformation on each variable, we reshape them into the form shown in Table 1, concatenate them and perform five deconvolutions. These settings are almost the same as the existing DCGAN, i.e., we used ReLU [23] and Batch Normalization layer [13]. The kernel size, stride, and padding are respectively 4, 2, and 2 except for the last deconvolutional layer. Note that the number of output channels of the last deconvolutional layer depends on whether the dataset contains color information or not.

Discriminator We employ spatio-temporal 3D convolutional layers to model the discriminator. The layer settings are similar to the image generator. Specifically, we use four convolutional layers with $4 \times 4 \times 4$ kernel and a stride of 2. The number of output channels is 64 in the initial convolutional layer, and set to double when the layer goes deeper. As with the DCGAN, we used LeakyReLU [20] with $a = 0.2$ and Batch Normalization layer [13] after these convolutions. Note that we do not insert the batch normalization after the initial convolution. Finally, we use a fully-connected layer and summarize all of the units in a single scalar. Each shape of the tensor used in the discriminator is shown in Fig.1.

4. Singular Value Clipping

As we described before, WGAN requires the discriminator to fulfill the K-Lipschitz constraint, and the authors employed a parameter clipping method that clamps the weights in the discriminator to $[-c, c]$. However, we empirically observed that the tuning of hyper parameter c is severe, and it frequently fails in learning under a different situation like our proposed model. We assumed this problem would be caused by a property that the K-Lipschitz constraint widely varies depending the value of c, and propose an alternative method that can explicitly adjust the value of K.

Suppose that $D(x)$ is a composite function consisting of N primitive functions, and each function f_n is Lipschitz continuous with K_n. In this case D can be represented as $D = f_N \circ f_{N-1} \circ \cdots f_1$, and D is also Lipschitz continuous with $K = \prod_n K_n$. That is, what is important in our approach is to add constraints to all the functions such that f_n satisfies the condition of given K_n. Although in principle our method can derive operations that satisfy arbitrary K, in the case of $K = 1$ these operations are invariant regardless of the number of layers constituting the discriminator. For simplicity we focus on the case of $K = 1$.

To satisfy 1-Lipschitz constraint, we add a constraint to all linear layers in the discriminator that satisfies the spectral norm of weight parameter W is equal or less than one. This means that the singular values of weight matrix are all one or less. To this end, we perform singular value decomposition (SVD) after parameter update, replace all the singular values

Layer	Condition	Method
Linear	$\|W\| \leq 1$	SVC
Convolution	$\|\hat{W}\| \leq 1$	SVC
Batch normalization	$0 < \gamma \leq \sqrt{\sigma_B^2 + \epsilon}$	Clipping γ
LeakyReLU	$a \leq 1$	Do nothing

Table 2. Proposed methods to satisfy the 1-Lipschitz constraint. $\|\cdot\|$ denotes a spectral norm. a represents a fixed parameter of the LeakyReLU layer. γ and σ_B are a scaling parameter after the batch normalization and a running mean of a standard deviation of a batch, respectively.

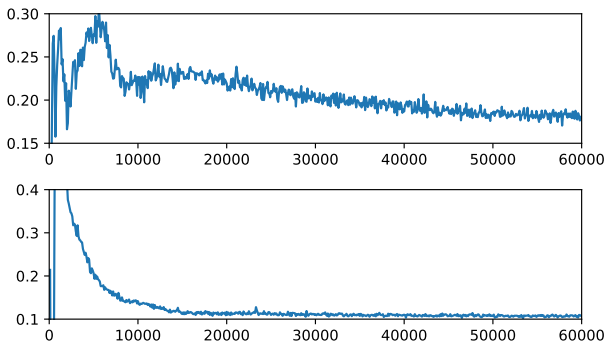


Figure 2. The difference of training curves in UCF-101 (see Section 6.1 for details). The upper row shows the loss of the generator per iteration in conventional clipping method, while the lower row shows the loss in our clipping method, Singular Value Clipping.

larger than one with one, and reconstruct the parameters with them. We also apply the same operation to convolutional layers by interpreting a higher order tensor in weight parameter as a matrix $\hat{W}$. We call these operations *Singular Value Clipping* (SVC).

As with the linear and the convolutional layer, we clamp the value of γ which represents a scaling parameter of the batch normalization layer in the same way. We summarize a clipping method of each layer in Table 2. Note that we do not perform any operations on ReLU and LeakyReLU layers because they always satisfy the condition unless a in the LeakyReLU is lower than 1.

The clear advantage of our alternative clipping method is that it does not require the careful tuning of hyperparameter c. Another advantage we have empirically observed is to stabilize the training of WGAN; in our experiments, our method can successfully train an advanced model even under the situation where the behavior of loss function becomes unstable with the conventional clipping. We show an example of such differences in Fig.2.

Although the problem of SVC is an increased computational cost, it can be mitigated by decreasing the frequency of performing the SVC. We show the summary of the algorithm of WGAN with the SVC in Algorithm 1. In our experiments, the computational time of SVD is almost the same as that of the forward-backward computation, but we observed the frequency of clipping is sufficient once every five iterations, i.e., $n_{\text{clip}} = 5$.

Algorithm 1 WGAN using Singular Value Clipping

Require: α: the learning rate. T: the number of iterations. n_D: the number of iterations of the discriminator per generator's iteration. n_{clip}: the number of intervals of the clipping.
for $t = 1$ to T **do**
  for $n = 1$ to n_D **do**
    Compute gradient of discriminator g_D
    $\theta_D \leftarrow \theta_D + \alpha \cdot \text{RMSProp}(\theta_D, g_D)$
  end for
  Compute gradient of generator g_G
  $\theta_G \leftarrow \theta_G - \alpha \cdot \text{RMSProp}(\theta_G, g_G)$
  if $t \bmod n_{\text{clip}} = 1$ **then**
    $\theta_D \leftarrow \text{SingularValueClipping}(\theta_D)$
  end if
end for

5. Applications

5.1. Frame interpolation

One of the advantages of our model is to be able to generate an intermediate frame between two adjacent frames. Since the video generation in our model is formulated as generating a trajectory in the latent image space represented by z_0 and z_1^t, our generator can easily yield long sequences by just interpolating the trajectory. Specifically, we add a bilinear filter to the last layer of the temporal generator, and interpolate the trajectory in the latent image space (see Section 3.4).

5.2. Conditional TGAN

In some cases, videos in a dataset contain some labels which correspond to a category of the video such as "IceDancing" or "Baseball". In order to exploit them and improve the quality of videos by the generator, we also develop a Conditional TGAN (CTGAN), in which the generator can take both label l and latent variable z_0.

The structure of CTGAN is similar with that of the original Conditional GAN. In temporal generator, after transforming l into one-hot vector v_l, we concatenate both this vector and z_0, and regard it as a new latent variable. That is, the temporal generator of the CTGAN is denoted as $G_0(z_0, v_l)$. The image generator of the CTGAN also takes the one-hot label vector as arguments, i.e., $G_1(z_0, z_1^t, v_l)$. As with the original image generator, we first perform linear transformation on each variable, reshape them, and operate five deconvolutions.

In the discriminator, we first broadcast the one-hot label vector to a voxel whose resolution is the same as that of the video. Thus, if the number of elements of v_l is V, the

number of channels of the voxel is equal to V. Next, we concatenate both the voxel and the input video, and send it into the convolutional layers.

6. Experiments

6.1. Datasets

We performed experiments with the following datasets.

Moving MNIST To investigate the properties of our models, we trained the models on the moving MNIST dataset [35], in which there are 10,000 clips each of which has 20 frames and consists of two digits moving inside a 64×64 patch. In these clips, two digits move linearly and the direction and magnitude of motion vectors are randomly chosen. If a digit approaches one of the edges in the patch, it bounces off the edge and its direction is changed while maintaining the speed. In our experiments, we randomly extracted 16 frames from these clips and used them as a training dataset.

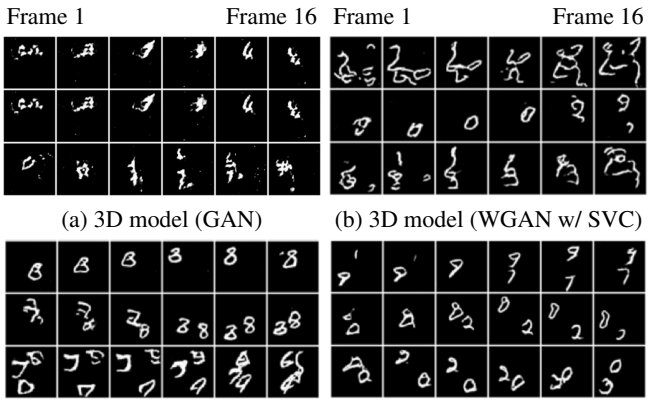
UCF-101 UCF-101 is a commonly used video dataset that consists of 13,320 videos belonging to 101 different categories such as *IceDancing* and *Baseball Pitch* [34]. Since the resolution of videos in the dataset is too large for the generative models, we resized all the videos to 85×64 pixels, randomly extracted 16 frames, and cropped a center square with 64 pixels.

Golf scene dataset Golf scene dataset is a large-scale video dataset made by Vondrick *et al*. [44], and contains 20,268 golf videos with 128×128 resolution. Since each video includes 29 short clips on average, it contains 583,508 short video clips in total. As with the UCF-101, we resized all the video clips with 64×64 pixels. To satisfy the assumption that the background is always fixed, they stabilized all of the videos with SIFT and RANSAC algorithms. As such assumption is not included in our method, this dataset is considered to be advantageous for existing methods.

6.2. Training configuration

All the parameters used in the optimizer are the same as those of the original WGAN. Specifically, we used the RMSProp optimizer [41] with the learning rate of 0.00005. All the weights in the temporal generator and the discriminator are initialized with HeNormal [8], and the weights in the image generator are initialized with the uniform distribution within a range of $[-0.01, 0.01]$. Chainer [42] was used to implement all models and for experiments.

For comparison, we employed the conventional clipping method and the SVC to train models with the WGAN. In the conventional clipping method, we carefully searched clipping parameter c and confirmed that the best value is $c = 0.01$. We set n_D to 1 for the both methods.



(a) 3D model (GAN) (b) 3D model (WGAN w/ SVC)

(c) TGAN (SVC, $G_1(z_1^t)$) (d) TGAN (SVC, $G_1(z_0, z_1^t)$)

Figure 3. Generated videos with four different models: (a) 3D model trained with the normal GAN, (b) 3D model trained with the WGAN and the SVC, (c) TGAN in which G_1 only uses z_1, and (d) TGAN in which G_1 uses both z_0 and z_1. Although these models generate 16 frames, for brevity we extract six frames from them at even intervals.

6.3. Comparative methods

For comparison, we implemented two models: (i) a simple model in which the generator has one linear layer and four 3D deconvolutional layers and the discriminator has five 3D convolutional layers, and (ii) a Video GAN proposed by [44]. We call the former *"3D model"*. In the generator of the 3D model, all the deconvolutional layers have $4 \times 4 \times 4$ kernel and the stride of 2. The number of channels in the initial deconvolutional layer is 512 and set to half when the layer goes deeper. We also used ReLU and batch normalization layers. The settings of the discriminator are exactly the same as those of our model. In the settings of the video GAN, we simply followed the settings in the original paper.

When we tried to train the 3D model and the video GAN model with the normal GAN loss, we observed that the discriminator easily wins against the generator and the training cannot proceed. To avoid this, we added Gaussian noise ($\sigma = 0.2$) to all layers of discriminators. In this case, all the scale parameters γ after the Batch Normalization layer are not used. Note that this noise addition is not used when we use the WGAN.

6.4. Qualitative evaluation

We trained our proposed model on the above datasets and visually confirmed the quality of the results. Fig.3 shows examples of generated videos by the generator trained on the moving MNIST dataset. It can be seen that the generated frames are quite different from those of the existing model proposed by Srivastava *et al*. [35]. While the predicted frames by the existing model tend to be blurry, our model is capable of producing consistent frames in which each image is sharp, clear and easy to discriminate two digits. We also observed that although our method can generate the frames

(e) 3D model (Normal GAN) (f) 3D model (SVC)

(g) Video GAN (SVC) (h) TGAN (SVC)

Figure 4. A comparison between four models: (e) 3D model trained with the normal GAN, (f) 3D model trained with the WGAN and the SVC, (g) Video GAN trained with the WGAN and the SVC, and (h) TGAN trained with the WGAN and the SVC. Only the first frame is shown.

in which each digit continues to move in a straight line, its shape sometimes slightly changes by time. Note that the existing models such as [35, 15] seem to generate frames in which each digit does not change, however, these methods can not be directly compared with our method because the qualitative results the authors have shown are for "video prediction" that predicts future frames from initial inputs, whereas our method generates them without such priors.

Fig.3 also shows that as for the quality of the generated videos, the 3D model using the normal GAN is the worst compared with the other methods. We considered that it is due to the high degree of freedom in the model caused by three-dimensional convolution, and explicitly dividing the spatio-temporal space could contribute to the improvement of the quality. We also confirmed that it is not the effect of selecting the normal GAN; although the quality of samples generated by the 3D model with the SVC outperforms that of the 3D model with the normal GAN, it is still lower than our proposed model (model (d) in Fig.3). In order to illustrate the effectiveness of z_0 in G_1, we further conducted the experiment with the TGAN in which G_1 does not take z_0 as an argument (model (c)). In this experiment, we observed that in the model (c) the problem of mode collapse tends to occur compared to our model.

We also compared the performance of our method with other existing methods when using practical data sets such as UCF-101. The qualitative experimental results are shown

Figure 5. Example of videos generated by the TGAN with WGAN and SVC. The golf scene dataset was used.

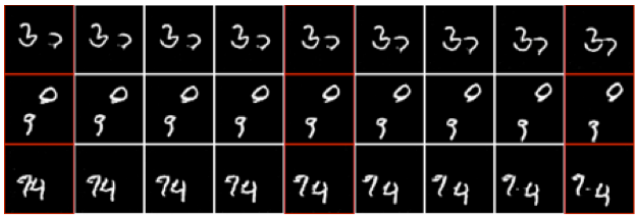


Figure 6. Examples of frame interpolation with our method. The red columns represent the adjacent frames generated by the temporal generator. The remaining columns show the intermediate frames.

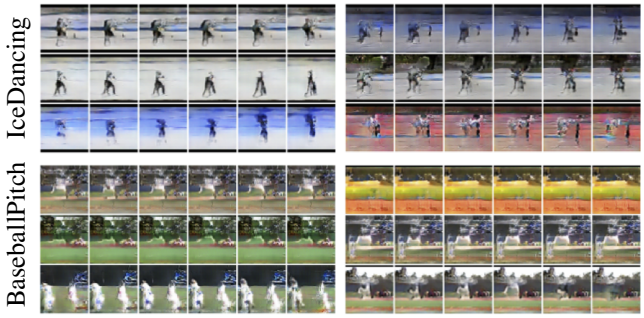


Figure 7. Generated videos by the conditional TGAN. The leftmost column shows the category in UCF-101 dataset, and the second and third columns show the generated samples given the category.

in Fig.4. We observed that the videos generated by the 3D model have the most artifacts compared with other models. The video GAN tends to avoid these artifacts because the background is relatively fixed in the UCF-101, however, the probability of generating unidentified videos is higher than that of the proposed model. We inferred that this problem is mainly due to the weakness of the existing method is vulnerable to videos with background movement.

Finally, in order to indicate that the quality of our model is comparable with that of the video GAN (these results can be seen in their project page), we conducted the experiment with the golf scene dataset. As we described before, it is considered that this dataset, in which the background is always fixed, is advantageous for the video GAN that exploits this assumption. Even under such unfavorable conditions, the quality of the videos generated by our model is almost the same as the existing method; both create a figure that seems likes a person's shadow, and it changes with time.

6.4.1 Applications

We performed the following experiments to illustrate the effectiveness of the applications described in Section 5.

Model A	Model B	GAM score	Winner
TGAN	3D model (GAN)	1.70	TGAN
TGAN	3D model (SVC)	1.27	TGAN
TGAN	TGAN ($G_1(z_1^t)$)	1.03	TGAN

Table 3. GAM scores for models of moving MNIST. "TGAN" denotes the model trained with the WGAN and the SVC. In "TGAN ($G_1(z_1^t)$)", G_1 has z_1 only (the SVC was used for training). "3D model (GAN)" and "3D model (SVC)" were trained with the normal GAN and the SVC, respectively.

To show our model can be applied to frame interpolation, we generated intermediate frames by interpolating two adjacent latent variables of the image space. These results are shown in Fig.6. It can be seen that the frame is not generated by a simple interpolation algorithm like dissolve, but semantically interpolating the adjacent frames.

We also experimentally confirmed that the proposed model is also extensible to the conditional GAN. These results are shown in Fig.7. We observed that the quality of the video generated by the conditional TGAN is significantly higher than that of the unsupervised ones. It is considered that adding semantic information of labels to the model contributed to the improvement of quality.

6.5. Quantitative evaluation

We performed the quantitative experiment to confirm the effectiveness of our method. As indicators of the quantitative evaluation, we adopted a *Generative Adversarial Metric (GAM)* [12] that compares adversarial models against each other, and an *inception score* [32] that has been commonly used to measure the quality of the generator.

For the comparison of two generative models, we used GAM scores in the moving MNIST dataset. Unlike the normal GAN in which the discriminator uses the binary cross entropy loss, the discriminator of the WGAN is learned to keep the fake samples and the real samples away, and we cannot choose zero as a threshold for discriminating real and fake samples. Therefore, we first generate a sufficient number of fake samples, and set a score that can classify fake and real samples well as the threshold.

Table 3 shows the results. In the GAM, a score higher than one means that the model A generates better fake samples that can fool the discriminator in the model B. It can be seen that our model can generate better samples that can deceive other existing methods. It can be seen that the TGAN beats the 3D models easily, but wins against the TGAN in which G_1 has z_1^t only. These results are the same as the results obtained by the aforementioned qualitative evaluation.

In order to compute the inception score, a dataset having label information and a good classifier for identifying the label are required. Thus, we used the UCF-101 dataset that has 101 action categories, and a pre-trained model of C3D [43], which was trained on Sports-1M dataset [16] and fine-tuned for the UCF-101, was employed as a classifier. We also calculated the inception scores by sampling 10,000 times from the latent random variable, and derived rough standard deviation by repeating this procedure four times. To compute the inception score when using the conditional TGAN, we added the prior distribution for the category to the generator, and transformed the conditional generator into the generator representing the model distribution. We also computed the inception score when using a real dataset to see an upper bound.

Method	Inception score
3D model (Weight clipping)	$4.32 \pm .01$
3D model (SVC)	$4.78 \pm .02$
Video GAN [44] (Normal GAN)	$8.18 \pm .05$
Video GAN (SVC)	$8.31 \pm .09$
TGAN (Normal GAN)	$9.18 \pm .11$
TGAN (Weight clipping)	$11.77 \pm .11$
TGAN (SVC)	$\mathbf{11.85 \pm .07}$
Conditional TGAN (SVC)	$\mathbf{15.83 \pm .18}$
UCF-101 dataset	$34.49 \pm .03$

Table 4. Inception scores for models of UCF-101.

Table 4 shows quantitative results. It can be seen that in the 3D model, the quality of the generated videos is worse than the video GAN and our proposed model. Although we observed that using the SVC slightly improves the inception score, its value is a little and still lower than that of the video GAN. We also confirmed that the SVC is effective in the case of the video GAN, however, its value is lower than our models. On the other hand, our models achieve the best scores compared with other existing methods. In addition to the video GAN, the TGAN using the SVC slightly outperformed the TGAN using the conventional weight clipping method. Although the quality of the SVC is almost indistinguishable compared with existing methods, we had to carefully change the value of c to achieve such quality. We believe that our clipping method is not a tool for dramatically improving the quality of the generator, but a convenient method to reduce the trouble of adjusting hyper parameters and significantly stabilize the training of the models.

7. Summary

We proposed a generative model that learns semantic representation of videos and can generate image sequences. We formulated the generating process of videos as a series of (i) a function that generates a set of latent variables, and (ii) a function that converts them into an image sequence. Using this representation, our model can generate videos with better quality and naturally achieves frame interpolation. We also proposed a novel parameter clipping method, Singular Value Clipping (SVC), that stabilizes the training of WGAN.

Acknowledgements We would like to thank Brian Vogel, Jethro Tan, Tommi Kerola, and Zornitsa Kostadinova for helpful discussions.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. In *arXiv preprint arXiv:1701.07875*, 2017. 2, 3

[2] E. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep Generative Image Models Using a Laplacian Pyramid of Adversarial Networks. In *NIPS*, 2015. 1, 2

[3] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox. Learning to Generate Chairs, Tables and Cars with Convolutional Networks. *arXiv preprint arXiv:1411.5928*, 2014. 2

[4] S. M. A. Eslami, N. Heess, and J. Winn. The Shape Boltzmann Machine : a Strong Model of Object Shape. In *CVPR*, 2012. 2

[5] 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, 3

[6] R. Goroshin, J. Bruna, J. Tompson, D. Eigen, and Y. LeCun. Unsupervised Learning of Spatiotemporally Coherent Metrics. In *ICCV*, 2015. 3

[7] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. DRAW: A Recurrent Neural Network For Image Generation. *arXiv preprint arXiv:1502.04623*, 2015. 2

[8] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *ICCV*, 2015. 2, 6

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 2

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735—-1780, 1997. 3

[11] G. Huang, Z. Liu, and K. Q. Weinberger. Densely Connected Convolutional Networks. In *arXiv preprint arXiv:1608.06993*, 2016. 2

[12] D. J. Im, C. D. Kim, H. Jiang, and R. Memisevic. Generating images with recurrent adversarial networks. In *arXiv preprint arXiv:1602.05110*, 2016. 2, 8

[13] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[14] S. Ji, W. Xu, M. Yang, and K. Yu. 3D Convolutional Neural Networks for Human Action Recognition. *PAMI*, 35(1):221–231, jan 2013. 2

[15] 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, 3, 7

[16] 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. 2, 8

[17] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011. 2

[18] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. In *ICML*. ACM Press, 2009. 2

[19] C. Li and M. Wand. Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks. In *arxiv preprint arXiv:1604.04382*, 2016. 2

[20] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *ICML*, 2013. 4

[21] M. Mathieu, C. Couprie, and Y. LeCun. Deep Multi-Scale Video Prediction beyond Mean Square Error. In *ICLR*, 2016. 3

[22] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. *arXiv preprint arXiv:1411.1784*, 2014. 2

[23] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *ICML*, (3):807–814, 2010. 4

[24] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond Short Snippets: Deep Networks for Video Classification. In *CVPR*, 2015. 1, 2

[25] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh. Action-Conditional Video Prediction using Deep Networks in Atari Games. In *NIPS*, 2015. 1, 3

[26] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros. Context Encoders: Feature Learning by Inpainting. In *CVPR*, 2016. 2

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

[28] 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. 3

[29] M. Rohrbach, S. Amin, M. Andriluka, and B. Schiele. A Database for Fine Grained Activity Detection of Cooking Activities. In *CVPR*, 2012. 2

[30] S. Sadanand and J. J. Corso. Action Bank: A High-Level Representation of Activity in Video. In *CVPR*, 2012. 2

[31] R. Salakhutdinov and G. Hinton. Deep Boltzmann Machines. In *AISTATS*, 2009. 2

[32] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved Techniques for Training GANs. In *NIPS*, 2016. 8

[33] K. Simonyan and A. Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. In *NIPS*, 2014. 1, 2

[34] 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. 6

[35] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised Learning of Video Representations using LSTMs. In *ICML*, 2015. 1, 3, 6, 7

[36] D. Stavens and S. Thrun. Unsupervised Learning of Invariant Features Using Video. In *CVPR*, 2010. 2

[37] I. Sutskever, G. Hinton, and G. Taylor. The Recurrent Temporal Restricted Boltzmann Machine. In *NIPS*, 2009. 2

[38] I. Sutskever and G. E. Hinton. Learning Multilevel Distributed Representations for High-Dimensional Sequences. In *AISTATS*, 2007. 2

[39] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional Learning of Spatio-temporal Features. In *ECCV*, 2010. 2

[40] G. W. Taylor, G. E. Hinton, and S. Roweis. Modeling Human Motion Using Binary Latent Variables. In *NIPS*, 2007. 2

[41] T. Tieleman and G. Hinton. Lecture 6.5 - RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. 6

[42] 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

[43] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning Spatiotemporal Features with 3D Convolutional Networks. In *ICCV*, 2015. 2, 8

[44] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating Videos with Scene Dynamics. In *NIPS*, 2016. 1, 3, 6, 8

[45] H. Wang, A. Klaser, C. Schmid, and L. Cheng-Lin. Action Recognition by Dense Trajectories. In *CVPR*, 2011. 2

[46] L. Wang, Y. Qiao, and X. Tang. Action Recognition with Trajectory-Pooled Deep-Convolutional Descriptors. In *CVPR*, 2015. 1, 2

[47] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool. Temporal Segment Networks: Towards Good Practices for Deep Action Recognition. In *ECCV*, 2016. 2

[48] X. Wang and A. Gupta. Unsupervised Learning of Visual Representations using Videos. In *ICCV*, 2015. 3

[49] X. Wang and A. Gupta. Generative Image Modeling using Style and Structure Adversarial Networks. *arXiv preprint arXiv:1603.05631*, 2016. 1, 2

[50] Y. Zhou and T. L. Berg. Learning Temporal Transformations From Time-Lapse Videos. In *ECCV*, 2016. 3

[51] W. Y. Zou, S. Zhu, A. Y. Ng, and K. Yu. Deep Learning of Invariant Features via Simulated Fixations in Video. In *NIPS*, 2012. 3

Wasserstein GAN

Martin Arjovsky[1], Soumith Chintala[2], and Léon Bottou[1,2]

[1]Courant Institute of Mathematical Sciences
[2]Facebook AI Research

1 Introduction

The problem this paper is concerned with is that of unsupervised learning. Mainly, what does it mean to learn a probability distribution? The classical answer to this is to learn a probability density. This is often done by defining a parametric family of densities $(P_\theta)_{\theta\in\mathbb{R}^d}$ and finding the one that maximized the likelihood on our data: if we have real data examples $\{x^{(i)}\}_{i=1}^m$, we would solve the problem

$$\max_{\theta\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \log P_\theta(x^{(i)})$$

If the real data distribution $\mathbb{P}_r$ admits a density and $\mathbb{P}_\theta$ is the distribution of the parametrized density P_θ, then, asymptotically, this amounts to minimizing the Kullback-Leibler divergence $KL(\mathbb{P}_r\|\mathbb{P}_\theta)$.

For this to make sense, we need the model density P_θ to exist. This is not the case in the rather common situation where we are dealing with distributions supported by low dimensional manifolds. It is then unlikely that the model manifold and the true distribution's support have a non-negligible intersection (see [1]), and this means that the KL distance is not defined (or simply infinite).

The typical remedy is to add a noise term to the model distribution. This is why virtually all generative models described in the classical machine learning literature include a noise component. In the simplest case, one assumes a Gaussian noise with relatively high bandwidth in order to cover all the examples. It is well known, for instance, that in the case of image generation models, this noise degrades the quality of the samples and makes them blurry. For example, we can see in the recent paper [23] that the optimal standard deviation of the noise added to the model when maximizing likelihood is around 0.1 to each pixel in a generated image, when the pixels were already normalized to be in the range $[0, 1]$. This is a very high amount of noise, so much that when papers report the samples of their models, they don't add the noise term on which they report likelihood numbers. In other words, the added noise term is clearly incorrect for the problem, but is needed to make the maximum likelihood approach work.

Rather than estimating the density of $\mathbb{P}_r$ which may not exist, we can define a random variable Z with a fixed distribution $p(z)$ and pass it through a parametric function $g_\theta : \mathcal{Z} \to \mathcal{X}$ (typically a neural network of some kind) that directly generates samples following a certain distribution $\mathbb{P}_\theta$. By varying θ, we can change this distribution and make it close to the real data distribution $\mathbb{P}_r$. This is useful in two ways. First of all, unlike densities, this approach can represent distributions confined to a low dimensional manifold. Second, the ability to easily generate samples is often more useful than knowing the numerical value of the density (for example in image superresolution or semantic segmentation when considering the conditional distribution of the output image given the input image). In general, it is computationally difficult to generate samples given an arbitrary high dimensional density [16].

Variational Auto-Encoders (VAEs) [9] and Generative Adversarial Networks (GANs) [4] are well known examples of this approach. Because VAEs focus on the approximate likelihood of the examples, they share the limitation of the standard models and need to fiddle with additional noise terms. GANs offer much more flexibility in the definition of the objective function, including Jensen-Shannon [4], and all f-divergences [17] as well as some exotic combinations [6]. On the other hand, training GANs is well known for being delicate and unstable, for reasons theoretically investigated in [1].

In this paper, we direct our attention on the various ways to measure how close the model distribution and the real distribution are, or equivalently, on the various ways to define a distance or divergence $\rho(\mathbb{P}_\theta, \mathbb{P}_r)$. The most fundamental difference between such distances is their impact on the convergence of sequences of probability distributions. A sequence of distributions $(\mathbb{P}_t)_{t\in\mathbb{N}}$ converges if and only if there is a distribution $\mathbb{P}_\infty$ such that $\rho(\mathbb{P}_t, \mathbb{P}_\infty)$ tends to zero, something that depends on how exactly the distance ρ is defined. Informally, a distance ρ induces a weaker topology when it makes it easier for a sequence of distribution to converge.[1] Section 2 clarifies how popular probability distances differ in that respect.

In order to optimize the parameter θ, it is of course desirable to define our model distribution $\mathbb{P}_\theta$ in a manner that makes the mapping $\theta \mapsto \mathbb{P}_\theta$ continuous. Continuity means that when a sequence of parameters θ_t converges to θ, the distributions $\mathbb{P}_{\theta_t}$ also converge to $\mathbb{P}_\theta$. However, it is essential to remember that the notion of the convergence of the distributions $\mathbb{P}_{\theta_t}$ depends on the way we compute the distance between distributions. The weaker this distance, the easier it is to define a continuous mapping from θ-space to $\mathbb{P}_\theta$-space, since it's easier for the distributions to converge. The main reason we care about the mapping $\theta \mapsto \mathbb{P}_\theta$ to be continuous is as follows. If ρ is our notion of distance between two distributions, we would like to have a loss function $\theta \mapsto \rho(\mathbb{P}_\theta, \mathbb{P}_r)$ that is continuous, and this is equivalent to having the mapping $\theta \mapsto \mathbb{P}_\theta$ be continuous when using the distance between distributions ρ.

[1] More exactly, the topology induced by ρ is weaker than that induced by ρ' when the set of convergent sequences under ρ is a superset of that under ρ'.

The contributions of this paper are:

- In Section 2, we provide a comprehensive theoretical analysis of how the Earth Mover (EM) distance behaves in comparison to popular probability distances and divergences used in the context of learning distributions.
- In Section 3, we define a form of GAN called Wasserstein-GAN that minimizes a reasonable and efficient approximation of the EM distance, and we theoretically show that the corresponding optimization problem is sound.
- In Section 4, we empirically show that WGANs cure the main training problems of GANs. In particular, training WGANs does not require maintaining a careful balance in training of the discriminator and the generator, and does not require a careful design of the network architecture either. The mode dropping phenomenon that is typical in GANs is also drastically reduced. One of the most compelling practical benefits of WGANs is the ability to continuously estimate the EM distance by training the discriminator to optimality. Plotting these learning curves is not only useful for debugging and hyperparameter searches, but also correlate remarkably well with the observed sample quality.

2 Different Distances

We now introduce our notation. Let $\mathcal{X}$ be a compact metric set (such as the space of images $[0,1]^d$) and let Σ denote the set of all the Borel subsets of $\mathcal{X}$. Let $\text{Prob}(\mathcal{X})$ denote the space of probability measures defined on $\mathcal{X}$. We can now define elementary distances and divergences between two distributions $\mathbb{P}_r, \mathbb{P}_g \in \text{Prob}(\mathcal{X})$:

- The *Total Variation* (TV) distance

$$\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)| \ .$$

- The *Kullback-Leibler* (KL) divergence

$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x) d\mu(x) \ ,$$

  where both $\mathbb{P}_r$ and $\mathbb{P}_g$ are assumed to be absolutely continuous, and therefore admit densities, with respect to a same measure μ defined on $\mathcal{X}$.[2] The KL divergence is famously assymetric and possibly infinite when there are points such that $P_g(x) = 0$ and $P_r(x) > 0$.

[2]Recall that a probability distribution $\mathbb{P}_r \in \text{Prob}(\mathcal{X})$ admits a density $p_r(x)$ with respect to μ, that is, $\forall A \in \Sigma$, $\mathbb{P}_r(A) = \int_A P_r(x) d\mu(x)$, if and only it is absolutely continuous with respect to μ, that is, $\forall A \in \Sigma$, $\mu(A) = 0 \Rightarrow \mathbb{P}_r(A) = 0$.

- The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m) \ ,$$

  where $\mathbb{P}_m$ is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$. This divergence is symmetrical and always defined because we can choose $\mu = \mathbb{P}_m$.

- The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \big[\ \|x - y\| \ \big] \ , \tag{1}$$

  where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. Intuitively, $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into the distribution $\mathbb{P}_g$. The EM distance then is the "cost" of the optimal transport plan.

The following example illustrates how apparently simple sequences of probability distributions converge under the EM distance but do not converge under the other distances and divergences defined above.

Example 1 (Learning parallel lines)**.** Let $Z \sim U[0, 1]$ the uniform distribution on the unit interval. Let $\mathbb{P}_0$ be the distribution of $(0, Z) \in \mathbb{R}^2$ (a 0 on the x-axis and the random variable Z on the y-axis), uniform on a straight vertical line passing through the origin. Now let $g_\theta(z) = (\theta, z)$ with θ a single real parameter. It is easy to see that in this case,

- $W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|$,
- $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0 \ , \\ 0 & \text{if } \theta = 0 \ , \end{cases}$
- $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0 \ , \\ 0 & \text{if } \theta = 0 \ , \end{cases}$
- and $\delta(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0 \ , \\ 0 & \text{if } \theta = 0 \ . \end{cases}$

When $\theta_t \to 0$, the sequence $(\mathbb{P}_{\theta_t})_{t\in\mathbb{N}}$ converges to $\mathbb{P}_0$ under the EM distance, but does not converge at all under either the JS, KL, reverse KL, or TV divergences. Figure 1 illustrates this for the case of the EM and JS distances.

Example 1 gives us a case where we can learn a probability distribution over a low dimensional manifold by doing gradient descent on the EM distance. This cannot be done with the other distances and divergences because the resulting loss function is not even continuous. Although this simple example features distributions with disjoint supports, the same conclusion holds when the supports have a non empty

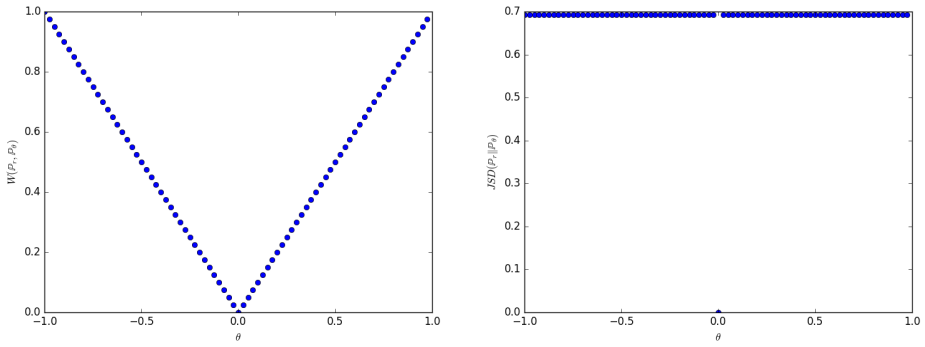


Figure 1: These plots show $\rho(\mathbb{P}_\theta, \mathbb{P}_0)$ as a function of θ when ρ is the EM distance (left plot) or the JS divergence (right plot). The EM plot is continuous and provides a usable gradient everywhere. The JS plot is not continuous and does not provide a usable gradient.

intersection contained in a set of measure zero. This happens to be the case when two low dimensional manifolds intersect in general position [1].

Since the Wasserstein distance is much weaker than the JS distance[3], we can now ask whether $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is a continuous loss function on θ under mild assumptions. This, and more, is true, as we now state and prove.

Theorem 1. *Let $\mathbb{P}_r$ be a fixed distribution over $\mathcal{X}$. Let Z be a random variable (e.g Gaussian) over another space $\mathcal{Z}$. Let $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ be a function, that will be denoted $g_\theta(z)$ with z the first coordinate and θ the second. Let $\mathbb{P}_\theta$ denote the distribution of $g_\theta(Z)$. Then,*

1. *If g is continuous in θ, so is $W(\mathbb{P}_r, \mathbb{P}_\theta)$.*
2. *If g is locally Lipschitz and satisfies regularity assumption 1, then $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is continuous everywhere, and differentiable almost everywhere.*
3. *Statements 1-2 are false for the Jensen-Shannon divergence $JS(\mathbb{P}_r, \mathbb{P}_\theta)$ and all the KLs.*

Proof. See Appendix C □

The following corollary tells us that learning by minimizing the EM distance makes sense (at least in theory) with neural networks.

Corollary 1. *Let g_θ be any feedforward neural network[4] parameterized by θ, and $p(z)$ a prior over z such that $\mathbb{E}_{z\sim p(z)}[\|z\|] < \infty$ (e.g. Gaussian, uniform, etc.).*

[3] The argument for *why* this happens, and indeed how we arrived to the idea that Wasserstein is what we should really be optimizing is displayed in Appendix A. We strongly encourage the interested reader who is not afraid of the mathematics to go through it.

[4] By a feedforward neural network we mean a function composed by affine transformations and pointwise nonlinearities which are smooth Lipschitz functions (such as the sigmoid, tanh, elu, softplus, etc). **Note**: the statement is also true for rectifier nonlinearities but the proof is more technical (even though very similar) so we omit it.

Then assumption 1 is satisfied and therefore $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is continuous everywhere and differentiable almost everywhere.

Proof. See Appendix C □

All this shows that EM is a much more sensible cost function for our problem than at least the Jensen-Shannon divergence. The following theorem describes the relative strength of the topologies induced by these distances and divergences, with KL the strongest, followed by JS and TV, and EM the weakest.

Theorem 2. *Let $\mathbb{P}$ be a distribution on a compact space $\mathcal{X}$ and $(\mathbb{P}_n)_{n\in\mathbb{N}}$ be a sequence of distributions on $\mathcal{X}$. Then, considering all limits as $n \to \infty$,*

1. *The following statements are equivalent*
   - *$\delta(\mathbb{P}_n, \mathbb{P}) \to 0$ with δ the total variation distance.*
   - *$JS(\mathbb{P}_n, \mathbb{P}) \to 0$ with JS the Jensen-Shannon divergence.*
2. *The following statements are equivalent*
   - *$W(\mathbb{P}_n, \mathbb{P}) \to 0$.*
   - *$\mathbb{P}_n \xrightarrow{\mathcal{D}} \mathbb{P}$ where $\xrightarrow{\mathcal{D}}$ represents convergence in distribution for random variables.*
3. *$KL(\mathbb{P}_n \| \mathbb{P}) \to 0$ or $KL(\mathbb{P} \| \mathbb{P}_n) \to 0$ imply the statements in (1).*
4. *The statements in (1) imply the statements in (2).*

Proof. See Appendix C □

This highlights the fact that the KL, JS, and TV distances are not sensible cost functions when learning distributions supported by low dimensional manifolds. However the EM distance is sensible in that setup. This obviously leads us to the next section where we introduce a practical approximation of optimizing the EM distance.

3 Wasserstein GAN

Again, Theorem 2 points to the fact that $W(\mathbb{P}_r, \mathbb{P}_\theta)$ might have nicer properties when optimized than $JS(\mathbb{P}_r, \mathbb{P}_\theta)$. However, the infimum in (1) is highly intractable. On the other hand, the Kantorovich-Rubinstein duality [22] tells us that

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \tag{2}$$

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$. Note that if we replace $\|f\|_L \leq 1$ for $\|f\|_L \leq K$ (consider K-Lipschitz for some constant K), then we end up with $K \cdot W(\mathbb{P}_r, \mathbb{P}_g)$. Therefore, if we have a parameterized family of

functions $\{f_w\}_{w \in \mathcal{W}}$ that are all K-Lipschitz for some K, we could consider solving the problem

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z)] \tag{3}$$

and if the supremum in (2) is attained for some $w \in \mathcal{W}$ (a pretty strong assumption akin to what's assumed when proving consistency of an estimator), this process would yield a calculation of $W(\mathbb{P}_r, \mathbb{P}_\theta)$ up to a multiplicative constant. Furthermore, we could consider differentiating $W(\mathbb{P}_r, \mathbb{P}_\theta)$ (again, up to a constant) by back-proping through equation (2) via estimating $\mathbb{E}_{z \sim p(z)}[\nabla_\theta f_w(g_\theta(z))]$. While this is all intuition, we now prove that this process is principled under the optimality assumption.

Theorem 3. *Let $\mathbb{P}_r$ be any distribution. Let $\mathbb{P}_\theta$ be the distribution of $g_\theta(Z)$ with Z a random variable with density p and g_θ a function satisfying assumption 1. Then, there is a solution $f : \mathcal{X} \to \mathbb{R}$ to the problem*

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

and we have

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z \sim p(z)}[\nabla_\theta f(g_\theta(z))]$$

when both terms are well-defined.

Proof. See Appendix C □

Now comes the question of finding the function f that solves the maximization problem in equation (2). To roughly approximate this, something that we can do is train a neural network parameterized with weights w lying in a compact space $\mathcal{W}$ and then backprop through $\mathbb{E}_{z \sim p(z)}[\nabla_\theta f_w(g_\theta(z))]$, as we would do with a typical GAN. Note that the fact that $\mathcal{W}$ is compact implies that all the functions f_w will be K-Lipschitz for some K that only depends on $\mathcal{W}$ and not the individual weights, therefore approximating (2) up to an irrelevant scaling factor and the capacity of the 'critic' f_w. In order to have parameters w lie in a compact space, something simple we can do is clamp the weights to a fixed box (say $\mathcal{W} = [-0.01, 0.01]^l$) after each gradient update. The Wasserstein Generative Adversarial Network (WGAN) procedure is described in Algorithm 1.

Weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used (such as in RNNs). We experimented with simple variants (such as projecting the weights to a sphere) with little difference, and we stuck with weight clipping due to its simplicity and already good performance. However, we do leave the topic of enforcing Lipschitz constraints in a neural network setting for further investigation, and we actively encourage interested researchers to improve on this method.

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

The fact that the EM distance is continuous and differentiable a.e. means that we can (and should) train the critic till optimality. The argument is simple, the more we train the critic, the more reliable gradient of the Wasserstein we get, which is actually useful by the fact that Wasserstein is differentiable almost everywhere. For the JS, as the discriminator gets better the gradients get more reliable but the true gradient is 0 since the JS is locally saturated and we get vanishing gradients, as can be seen in Figure 1 of this paper and Theorem 2.4 of [1]. In Figure 2 we show a proof of concept of this, where we train a GAN discriminator and a WGAN critic till optimality. The discriminator learns very quickly to distinguish between fake and real, and as expected provides no reliable gradient information. The critic, however, can't saturate, and converges to a linear function that gives remarkably clean gradients everywhere. The fact that we constrain the weights limits the possible growth of the function to be at most linear in different parts of the space, forcing the optimal critic to have this behaviour.

Perhaps more importantly, the fact that we can train the critic till optimality makes it impossible to collapse modes when we do. This is due to the fact that mode collapse comes from the fact that the optimal generator for a *fixed* discriminator is a sum of deltas on the points the discriminator assigns the highest values, as observed by [4] and highlighted in [11].

In the following section we display the practical benefits of our new algorithm, and we provide an in-depth comparison of its behaviour and that of traditional GANs.

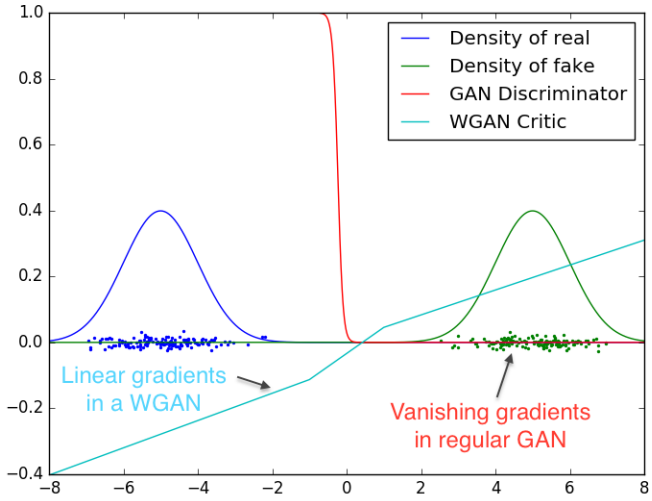


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

4 Empirical Results

We run experiments on image generation using our Wasserstein-GAN algorithm and show that there are significant practical benefits to using it over the formulation used in standard GANs.

We claim two main benefits:

- a meaningful loss metric that correlates with the generator's convergence and sample quality
- improved stability of the optimization process

4.1 Experimental Procedure

We run experiments on image generation. The target distribution to learn is the LSUN-Bedrooms dataset [24] – a collection of natural images of indoor bedrooms. Our baseline comparison is DCGAN [18], a GAN with a convolutional architecture trained with the standard GAN procedure using the $-\log D$ trick [4]. The generated samples are 3-channel images of 64x64 pixels in size. We use the hyper-parameters specified in Algorithm 1 for all of our experiments.

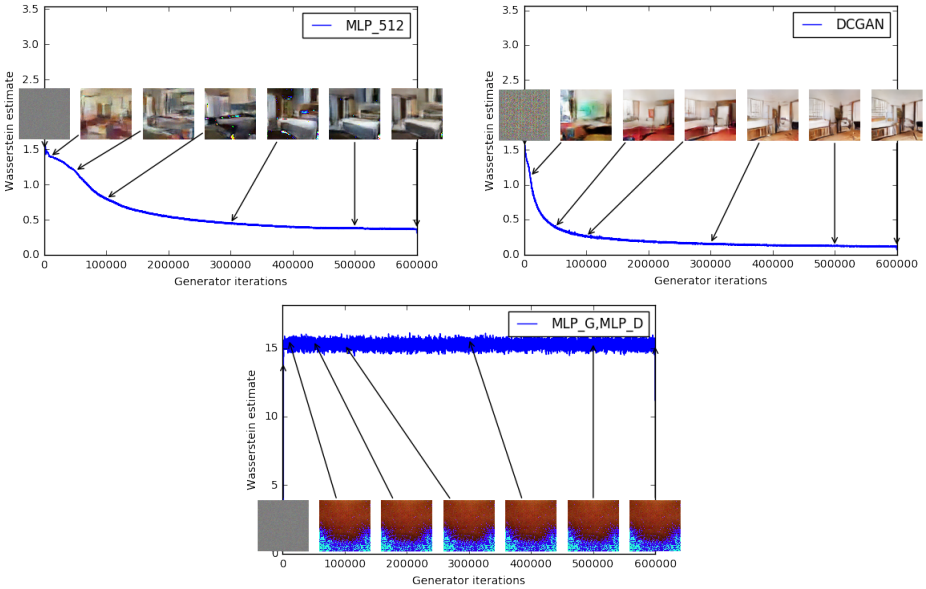


Figure 3: Training curves and samples at different stages of training. We can see a clear correlation between lower error and better sample quality. Upper left: the generator is an MLP with 4 hidden layers and 512 units at each layer. The loss decreases constistently as training progresses and sample quality increases. Upper right: the generator is a standard DCGAN. The loss decreases quickly and sample quality increases as well. In both upper plots the critic is a DCGAN without the sigmoid so losses can be subjected to comparison. Lower half: both the generator and the discriminator are MLPs with substantially high learning rates (so training failed). Loss is constant and samples are constant as well. The training curves were passed through a median filter for visualization purposes.

4.2 Meaningful loss metric

Because the WGAN algorithm attempts to train the critic f (lines 2–8 in Algorithm 1) relatively well before each generator update (line 10 in Algorithm 1), the loss function at this point is an estimate of the EM distance, up to constant factors related to the way we constrain the Lipschitz constant of f.

Our first experiment illustrates how this estimate correlates well with the quality of the generated samples. Besides the convolutional DCGAN architecture, we also ran experiments where we replace the generator or both the generator and the critic by 4-layer ReLU-MLP with 512 hidden units.

Figure 3 plots the evolution of the WGAN estimate (3) of the EM distance during WGAN training for all three architectures. The plots clearly show that these curves correlate well with the visual quality of the generated samples.

To our knowledge, this is the first time in GAN literature that such a property is shown, where the loss of the GAN shows properties of convergence. This property is extremely useful when doing research in adversarial networks as one does not need

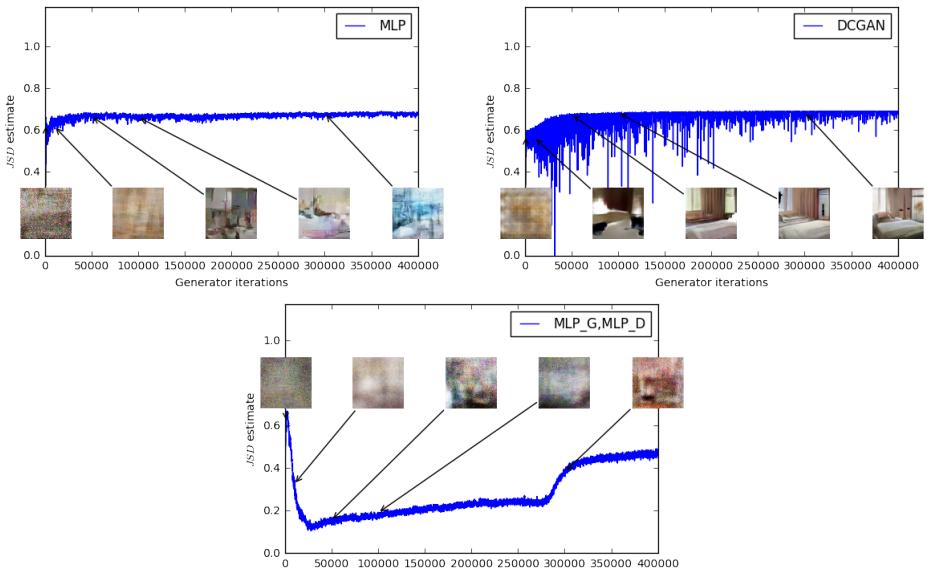


Figure 4: JS estimates for an MLP generator (upper left) and a DCGAN generator (upper right) trained with the standard GAN procedure. Both had a DCGAN discriminator. Both curves have increasing error. *Samples get better for the DCGAN but the JS estimate increases or stays constant, pointing towards no significant correlation between sample quality and loss. Bottom: MLP with both generator and discriminator. The curve goes up and down regardless of sample quality. All training curves were passed through the same median filter as in Figure 3.*

to stare at the generated samples to figure out failure modes and to gain information on which models are doing better over others.

However, we do not claim that this is a new method to quantitatively evaluate generative models yet. The constant scaling factor that depends on the critic's architecture means it's hard to compare models with different critics. Even more, in practice the fact that the critic doesn't have infinite capacity makes it hard to know just how close to the EM distance our estimate really is. This being said, we have succesfully used the loss metric to validate our experiments repeatedly and without failure, and we see this as a huge improvement in training GANs which previously had no such facility.

In contrast, Figure 4 plots the evolution of the GAN estimate of the JS distance during GAN training. More precisely, during GAN training, the discriminator is trained to maximize

$$L(D, g_\theta) = \mathbb{E}_{x\sim\mathbb{P}_r}[\log D(x)] + \mathbb{E}_{x\sim\mathbb{P}_\theta}[\log(1 - D(x))]$$

which is is a lower bound of $2JS(\mathbb{P}_r, \mathbb{P}_\theta) - 2\log 2$. In the figure, we plot the quantity $\frac{1}{2}L(D, g_\theta) + \log 2$, which is a lower bound of the JS distance.

This quantity clearly correlates poorly the sample quality. Note also that the

JS estimate usually stays constant or goes up instead of going down. In fact it often remains very close to $\log 2 \approx 0.69$ which is the highest value taken by the JS distance. In other words, the JS distance saturates, the discriminator has zero loss, and the generated samples are in some cases meaningful (DCGAN generator, top right plot) and in other cases collapse to a single nonsensical image [4]. This last phenomenon has been theoretically explained in [1] and highlighted in [11].

When using the $-\log D$ trick [4], the discriminator loss and the generator loss are different. Figure 8 in Appendix E reports the same plots for GAN training, but using the generator loss instead of the discriminator loss. This does not change the conclusions.

Finally, as a negative result, we report that WGAN training becomes unstable at times when one uses a momentum based optimizer such as Adam [8] (with $\beta_1 > 0$) on the critic, or when one uses high learning rates. Since the loss for the critic is nonstationary, momentum based methods seemed to perform worse. We identified momentum as a potential cause because, as the loss blew up and samples got worse, the cosine between the Adam step and the gradient usually turned negative. The only places where this cosine was negative was in these situations of instability. We therefore switched to RMSProp [21] which is known to perform well even on very nonstationary problems [13].

4.3 Improved stability

One of the benefits of WGAN is that it allows us to train the critic till optimality. When the critic is trained to completion, it simply provides a loss to the generator that we can train as any other neural network. This tells us that we no longer need to balance generator and discriminator's capacity properly. The better the critic, the higher quality the gradients we use to train the generator.

We observe that WGANs are much more robust than GANs when one varies the architectural choices for the generator. We illustrate this by running experiments on three generator architectures: (1) a convolutional DCGAN generator, (2) a convolutional DCGAN generator without batch normalization and with a constant number of filters, and (3) a 4-layer ReLU-MLP with 512 hidden units. The last two are known to perform very poorly with GANs. We keep the convolutional DCGAN architecture for the WGAN critic or the GAN discriminator.

Figures 5, 6, and 7 show samples generated for these three architectures using both the WGAN and GAN algorithms. We refer the reader to Appendix F for full sheets of generated samples. Samples were not cherry-picked.

In no experiment did we see evidence of mode collapse for the WGAN algorithm.

Figure 5: Algorithms trained with a DCGAN generator. Left: WGAN algorithm. Right: standard GAN formulation. Both algorithms produce high quality samples.

Figure 6: Algorithms trained with a generator without batch normalization and constant number of filters at every layer (as opposed to duplicating them every time as in [18]). Aside from taking out batch normalization, the number of parameters is therefore reduced by a bit more than an order of magnitude. Left: WGAN algorithm. Right: standard GAN formulation. As we can see the standard GAN failed to learn while the WGAN still was able to produce samples.

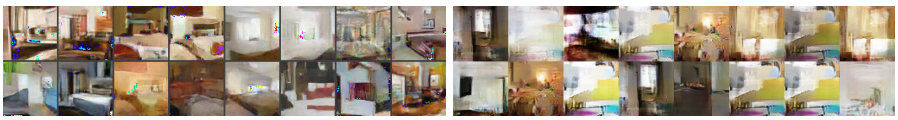


Figure 7: Algorithms trained with an MLP generator with 4 layers and 512 units with ReLU nonlinearities. The number of parameters is similar to that of a DCGAN, but it lacks a strong inductive bias for image generation. Left: WGAN algorithm. Right: standard GAN formulation. The WGAN method still was able to produce samples, lower quality than the DCGAN, and of higher quality than the MLP of the standard GAN. Note the significant degree of mode collapse in the GAN MLP.

5 Related Work

There's been a number of works on the so called Integral Probability Metrics (IPMs) [15]. Given $\mathcal{F}$ a set of functions from $\mathcal{X}$ to $\mathbb{R}$, we can define

$$d_{\mathcal{F}}(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{f \in \mathcal{F}} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)] \tag{4}$$

as an integral probability metric associated with the function class $\mathcal{F}$. It is easily verified that if for every $f \in \mathcal{F}$ we have $-f \in \mathcal{F}$ (such as all examples we'll consider), then $d_{\mathcal{F}}$ is nonnegative, satisfies the triangular inequality, and is symmetric. Thus, $d_{\mathcal{F}}$ is a pseudometric over $\text{Prob}(\mathcal{X})$.

While IPMs might seem to share a similar formula, as we will see different classes of functions can yeald to radically different metrics.

- By the Kantorovich-Rubinstein duality [22], we know that $W(\mathbb{P}_r, \mathbb{P}_\theta) = d_{\mathcal{F}}(\mathbb{P}_r, \mathbb{P}_\theta)$

when $\mathcal{F}$ is the set of 1-Lipschitz functions. Furthermore, if $\mathcal{F}$ is the set of K-Lipschitz functions, we get $K \cdot W(\mathbb{P}_r, \mathbb{P}_\theta) = d_{\mathcal{F}}(\mathbb{P}_r, \mathbb{P}_\theta)$.

- When $\mathcal{F}$ is the set of all measurable functions bounded between -1 and 1 (or all continuous functions between -1 and 1), we retrieve $d_{\mathcal{F}}(\mathbb{P}_r, \mathbb{P}_\theta) = \delta(\mathbb{P}_r, \mathbb{P}_\theta)$ the total variation distance [15]. This already tells us that going from 1-Lipschitz to 1-Bounded functions drastically changes the topology of the space, and the regularity of $d_{\mathcal{F}}(\mathbb{P}_r, \mathbb{P}_\theta)$ as a loss function (as by Theorems 1 and 2).

- Energy-based GANs (EBGANs) [25] can be thought of as the generative approach to the total variation distance. This connection is stated and proven in depth in Appendix D. At the core of the connection is that the discriminator will play the role of f maximizing equation (4) while its only restriction is being between 0 and m for some constant m. This will yeald the same behaviour as being restricted to be between -1 and 1 up to a constant scaling factor irrelevant to optimization. Thus, when the discriminator approaches optimality the cost for the generator will aproximate the total variation distance $\delta(\mathbb{P}_r, \mathbb{P}_\theta)$.

  Since the total variation distance displays the same regularity as the JS, it can be seen that EBGANs will suffer from the same problems of classical GANs regarding not being able to train the discriminator till optimality and thus limiting itself to very imperfect gradients.

- Maximum Mean Discrepancy (MMD) [5] is a specific case of integral probability metrics when $\mathcal{F} = \{f \in \mathcal{H} : \|f\|_\infty \leq 1\}$ for $\mathcal{H}$ some Reproducing Kernel Hilbert Space (RKHS) associated with a given kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. As proved on [5] we know that MMD is a proper metric and not only a pseudometric when the kernel is universal. In the specific case where $\mathcal{H} = L^2(\mathcal{X}, m)$ for m the normalized Lebesgue measure on $\mathcal{X}$, we know that $\{f \in C_b(\mathcal{X}), \|f\|_\infty \leq 1\}$ will be contained in $\mathcal{F}$, and therefore $d_{\mathcal{F}}(\mathbb{P}_r, \mathbb{P}_\theta) \leq \delta(\mathbb{P}_r, \mathbb{P}_\theta)$ so the regularity of the MMD distance as a loss function will be at least as bad as the one of the total variation. Nevertheless this is a very extreme case, since we would need a very powerful kernel to approximate the whole L^2. However, even Gaussian kernels are able to detect tiny noise patterns as recently evidenced by [20]. This points to the fact that especially with low bandwidth kernels, the distance might be close to a saturating regime similar as with total variation or the JS. This obviously doesn't need to be the case for every kernel, and figuring out how and which different MMDs are closer to Wasserstein or total variation distances is an interesting topic of research.

  The great aspect of MMD is that via the kernel trick there is no need to train a separate network to maximize equation (4) for the ball of a RKHS. However, this has the disadvantage that evaluating the MMD distance has computational cost that grows quadratically with the amount of samples used to estimate the expectations in (4). This last point makes MMD have limited scalability, and is sometimes inapplicable to many real life applications because of it. There are estimates with linear computational cost for the MMD

[5] which in a lot of cases makes MMD very useful, but they also have worse sample complexity.

- Generative Moment Matching Networks (GMMNs) [10, 2] are the generative counterpart of MMD. By backproping through the kernelized formula for equation (4), they directly optimize $d_{MMD}(\mathbb{P}_r, \mathbb{P}_\theta)$ (the IPM when $\mathcal{F}$ is as in the previous item). As mentioned, this has the advantage of not requiring a separate network to approximately maximize equation (4). However, GMMNs have enjoyed limited applicability. Partial explanations for their unsuccess are the quadratic cost as a function of the number of samples and vanishing gradients for low-bandwidth kernels. Furthermore, it may be possible that some kernels used in practice are unsuitable for capturing very complex distances in high dimensional sample spaces such as natural images. This is properly justified by the fact that [19] shows that for the typical Gaussian MMD test to be reliable (as in it's power as a statistical test approaching 1), we need the number of samples to grow linearly with the number of dimensions. Since the MMD computational cost grows quadratically with the number of samples in the batch used to estimate equation (4), this makes the cost of having a reliable estimator grow quadratically with the number of dimensions, which makes it very inapplicable for high dimensional problems. Indeed, for something as standard as 64x64 images, we would need minibatches of size at least 4096 (without taking into account the constants in the bounds of [19] which would make this number substantially larger) and a total cost per iteration of 4096^2, over 5 orders of magnitude more than a GAN iteration when using the standard batch size of 64.

  That being said, these numbers can be a bit unfair to the MMD, in the sense that we are comparing empirical sample complexity of GANs with the theoretical sample complexity of MMDs, which tends to be worse. However, in the original GMMN paper [10] they indeed used a minibatch of size 1000, much larger than the standard 32 or 64 (even when this incurred in quadratic computational cost). While estimates that have linear computational cost as a function of the number of samples exist [5], they have worse sample complexity, and to the best of our knowledge they haven't been yet applied in a generative context such as in GMMNs.

On another great line of research, the recent work of [14] has explored the use of Wasserstein distances in the context of learning for Restricted Boltzmann Machines for discrete spaces. The motivations at a first glance might seem quite different, since the manifold setting is restricted to continuous spaces and in finite discrete spaces the weak and strong topologies (the ones of W and JS respectively) coincide. However, in the end there is more in commmon than not about our motivations. We both want to compare distributions in a way that leverages the geometry of the underlying space, and Wasserstein allows us to do exactly that.

Finally, the work of [3] shows new algorithms for calculating Wasserstein distances between different distributions. We believe this direction is quite important, and perhaps could lead to new ways of evaluating generative models.

6 Conclusion

We introduced an algorithm that we deemed WGAN, an alternative to traditional GAN training. In this new model, we showed that we can improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyperparameter searches. Furthermore, we showed that the corresponding optimization problem is sound, and provided extensive theoretical work highlighting the deep connections to other distances between distributions.

Acknowledgments

We would like to thank Mohamed Ishmael Belghazi, Emily Denton, Ian Goodfellow, Ishaan Gulrajani, Alex Lamb, David Lopez-Paz, Eric Martin, Maxime Oquab, Aditya Ramesh, Ronan Riochet, Uri Shalit, Pablo Sprechmann, Arthur Szlam, Ruohan Wang, for helpful comments and advice.

References

[1] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations*, 2017. Under review.

[2] Gintare Karolina Dziugaite, Daniel M. Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *CoRR*, abs/1505.03906, 2015.

[3] Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3440–3448. Curran Associates, Inc., 2016.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[5] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773, 2012.

[6] Ferenc Huszar. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *CoRR*, abs/1511.05101, 2015.

[7] Shizuo Kakutani. Concrete representation of abstract (m)-spaces (a characterization of the space of continuous functions). *Annals of Mathematics*, 42(4):994–1024, 1941.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[9] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[10] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1718–1727. JMLR Workshop and Conference Proceedings, 2015.

[11] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *Corr*, abs/1611.02163, 2016.

[12] Paul Milgrom and Ilya Segal. Envelope theorems for arbitrary choice sets. *Econometrica*, 70(2):583–601, 2002.

[13] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1928–1937, 2016.

[14] Grégoire Montavon, Klaus-Robert Müller, and Marco Cuturi. Wasserstein training of restricted boltzmann machines. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3718–3726. Curran Associates, Inc., 2016.

[15] Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

[16] Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, April 2001.

[17] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. pages 271–279, 2016.

[18] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[19] Aaditya Ramdas, Sashank J. Reddi, Barnabas Poczos, Aarti Singh, and Larry Wasserman. On the high-dimensional power of linear-time kernel two-sample testing under mean-difference alternatives. *Corr*, abs/1411.6314, 2014.

[20] Dougal J Sutherland, Hsiao-Yu Tung, Heiko Strathmann, Soumyajit De, Aaditya Ramdas, Alex Smola, and Arthur Gretton. Generative models and model criticism via optimized maximum mean discrepancy. In *International Conference on Learning Representations*, 2017. Under review.

[21] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[22] Cédric Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer, Berlin, 2009.

[23] Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger B. Grosse. On the quantitative analysis of decoder-based generative models. *CoRR*, abs/1611.04273, 2016.

[24] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. *Corr*, abs/1506.03365, 2015.

[25] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *Corr*, abs/1609.03126, 2016.

A Why Wasserstein is indeed weak

We now introduce our notation. Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a compact set (such as $[0,1]^d$ the space of images). We define $\text{Prob}(\mathcal{X})$ to be the space of probability measures over $\mathcal{X}$. We note

$$C_b(\mathcal{X}) = \{f : \mathcal{X} \to \mathbb{R},\ f \text{ is continuous and bounded}\}$$

Note that if $f \in C_b(\mathcal{X})$, we can define $\|f\|_\infty = \max_{x\in\mathcal{X}} |f(x)|$, since f is bounded. With this norm, the space $(C_b(\mathcal{X}), \|\cdot\|_\infty)$ is a normed vector space. As for any normed vector space, we can define its dual

$$C_b(\mathcal{X})^* = \{\phi : C_b(\mathcal{X}) \to \mathbb{R},\ \phi \text{ is linear and continuous}\}$$

and give it the dual norm $\|\phi\| = \sup_{f\in C_b(\mathcal{X}), \|f\|_\infty \leq 1} |\phi(f)|$.

With this definitions, $(C_b(\mathcal{X})^*, \|\cdot\|)$ is another normed space. Now let μ be a signed measure over $\mathcal{X}$, and let us define the total variation distance

$$\|\mu\|_{TV} = \sup_{A\subseteq\mathcal{X}} |\mu(A)|$$

where the supremum is taken all Borel sets in $\mathcal{X}$. Since the total variation is a norm, then if we have $\mathbb{P}_r$ and $\mathbb{P}_\theta$ two probability distributions over $\mathcal{X}$,

$$\delta(\mathbb{P}_r, \mathbb{P}_\theta) := \|\mathbb{P}_r - \mathbb{P}_\theta\|_{TV}$$

is a distance in $\text{Prob}(\mathcal{X})$ (called the total variation distance).

We can consider

$$\Phi : (\text{Prob}(\mathcal{X}), \delta) \to (C_b(\mathcal{X})^*, \|\cdot\|)$$

where $\Phi(\mathbb{P})(f) := \mathbb{E}_{x\sim\mathbb{P}}[f(x)]$ is a linear function over $C_b(\mathcal{X})$. The Riesz Representation theorem ([7], Theorem 10) tells us that Φ is an isometric immersion. This tells us that we can effectively consider $\text{Prob}(\mathcal{X})$ with the total variation distance as a subset of $C_b(\mathcal{X})^*$ with the norm distance. Thus, just to accentuate it one more time, the total variation over $\text{Prob}(\mathcal{X})$ is exactly the norm distance over $C_b(\mathcal{X})^*$.

Let us stop for a second and analyze what all this technicality meant. The main thing to carry is that we introduced a distance δ over probability distributions. When looked as a distance over a subset of $C_b(\mathcal{X})^*$, this distance gives the norm topology. The norm topology is very strong. Therefore, we can expect that not many functions $\theta \mapsto \mathbb{P}_\theta$ will be continuous when measuring distances between distributions with δ. As we will show later in Theorem 2, δ gives the same topology as the Jensen-Shannon divergence, pointing to the fact that the JS is a very strong distance, and is thus more propense to give a discontinuous loss function.

Now, all dual spaces (such as $C_b(\mathcal{X})^*$ and thus $\text{Prob}(\mathcal{X})$) have a strong topology (induced by the norm), and a weak* topology. As the name suggests, the weak* topology is much weaker than the strong topology. In the case of $\text{Prob}(\mathcal{X})$, the strong topology is given by the total variation distance, and the weak* topology is given by the Wasserstein distance (among others) [22].

B Assumption definitions

Assumption 1. *Let $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ be locally Lipschitz between finite dimensional vector spaces. We will denote $g_\theta(z)$ it's evaluation on coordinates (z, θ). We say that g satisfies assumption 1 for a certain probability distribution p over $\mathcal{Z}$ if there are local Lipschitz constants $L(\theta, z)$ such that*

$$\mathbb{E}_{z\sim p}[L(\theta, z)] < +\infty$$

C Proofs of things

Proof of Theorem 1. Let θ and θ' be two parameter vectors in $\mathbb{R}^d$. Then, we will first attempt to bound $W(\mathbb{P}_\theta, \mathbb{P}_{\theta'})$, from where the theorem will come easily. The main element of the proof is the use of the coupling γ, the distribution of the joint $(g_\theta(Z), g_{\theta'}(Z))$, which clearly has $\gamma \in \Pi(\mathbb{P}_\theta, \mathbb{P}_{\theta'})$.

By the definition of the Wasserstein distance, we have

$$\begin{aligned} W(\mathbb{P}_\theta, \mathbb{P}_{\theta'}) &\leq \int_{\mathcal{X}\times\mathcal{X}} \|x - y\| \, \mathrm{d}\gamma \\ &= \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \\ &= \mathbb{E}_z[\|g_\theta(z) - g_{\theta'}(z)\|] \end{aligned}$$

If g is continuous in θ, then $g_\theta(z) \to_{\theta\to\theta'} g_{\theta'}(z)$, so $\|g_\theta - g_{\theta'}\| \to 0$ pointwise as functions of z. Since $\mathcal{X}$ is compact, the distance of any two elements in it has to be uniformly bounded by some constant M, and therefore $\|g_\theta(z) - g_{\theta'}(z)\| \leq M$ for all θ and z uniformly. By the bounded convergence theorem, we therefore have

$$W(\mathbb{P}_\theta, \mathbb{P}_{\theta'}) \leq \mathbb{E}_z[\|g_\theta(z) - g_{\theta'}(z)\|] \to_{\theta\to\theta'} 0$$

Finally, we have that

$$|W(\mathbb{P}_r, \mathbb{P}_\theta) - W(\mathbb{P}_r, \mathbb{P}_{\theta'})| \leq W(\mathbb{P}_\theta, \mathbb{P}_{\theta'}) \to_{\theta\to\theta'} 0$$

proving the continuity of $W(\mathbb{P}_r, \mathbb{P}_\theta)$.

Now let g be locally Lipschitz. Then, for a given pair (θ, z) there is a constant $L(\theta, z)$ and an open set U such that $(\theta, z) \in U$, such that for every $(\theta', z') \in U$ we have

$$\|g_\theta(z) - g'_\theta(z')\| \leq L(\theta, z)(\|\theta - \theta'\| + \|z - z'\|)$$

By taking expectations and $z' = z$ we

$$\mathbb{E}_z[\|g_\theta(z) - g_{\theta'}(z)\|] \leq \|\theta - \theta'\|\mathbb{E}_z[L(\theta, z)]$$

whenever $(\theta', z) \in U$. Therefore, we can define $U_\theta = \{\theta' | (\theta', z) \in U\}$. It's easy to see that since U was open, U_θ is as well. Furthermore, by assumption 1, we can define $L(\theta) = \mathbb{E}_z[L(\theta, z)]$ and achieve

$$|W(\mathbb{P}_r, \mathbb{P}_\theta) - W(\mathbb{P}_r, \mathbb{P}_{\theta'})| \leq W(\mathbb{P}_\theta, \mathbb{P}_{\theta'}) \leq L(\theta)\|\theta - \theta'\|$$

for all $\theta' \in U_\theta$, meaning that $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is locally Lipschitz. This obviously implies that $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is everywhere continuous, and by Radamacher's theorem we know it has to be differentiable almost everywhere.

The counterexample for item 3 of the Theorem is indeed Example 1. □

Proof of Corollary 1. We begin with the case of smooth nonlinearities. Since g is C^1 as a function of (θ, z) then for any fixed (θ, z) we have $L(\theta, Z) \leq \|\nabla_{\theta,x} g_\theta(z)\| + \epsilon$ is an acceptable local Lipschitz constant for all $\epsilon > 0$. Therefore, it suffices to prove

$$\mathbb{E}_{z\sim p(z)}[\|\nabla_{\theta,z} g_\theta(z)\|] < +\infty$$

If H is the number of layers we know that $\nabla_z g_\theta(z) = \prod_{k=1}^H W_k D_k$ where W_k are the weight matrices and D_k is are the diagonal Jacobians of the nonlinearities. Let $f_{i:j}$ be the application of layers i to j inclusively (e.g. $g_\theta = f_{1:H}$). Then, $\nabla_{W_k} g_\theta(z) = \left(\left(\prod_{i=k+1}^H W_i D_i\right) D_k\right) f_{1:k-1}(z)$. We recall that if L is the Lipschitz constant of the nonlinearity, then $\|D_i\| \leq L$ and $\|f_{1:k-1}(z)\| \leq \|z\| L^{k-1} \prod_{i=1}^{k-1} W_i$. Putting this together,

$$\begin{aligned}
\|\nabla_{z,\theta} g_\theta(z)\| &\leq \|\prod_{i=1}^H W_i D_i\| + \sum_{k=1}^H \| \left(\left(\prod_{i=k+1}^H W_i D_i\right) D_k\right) f_{1:k-1}(z)\| \\
&\leq L^H \prod_{i=H}^K \|W_i\| + \sum_{k=1}^H \|z\| L^H \left(\prod_{i=1}^{k-1} \|W_i\|\right) \left(\prod_{i=k+1}^H \|W_i\|\right)
\end{aligned}$$

If $C_1(\theta) = L^H \left(\prod_{i=1}^H \|W_i\|\right)$ and $C_2(\theta) = \sum_{k=1}^H L^H \left(\prod_{i=1}^{k-1} \|W_i\|\right) \left(\prod_{i=k+1}^H \|W_i\|\right)$ then

$$\mathbb{E}_{z\sim p(z)}[\|\nabla_{\theta,z} g_\theta(z)\|] \leq C_1(\theta) + C_2(\theta) \mathbb{E}_{z\sim p(z)}[\|z\|] < +\infty$$

finishing the proof □

Proof of Theorem 2.

1. • $(\delta(\mathbb{P}_n, \mathbb{P}) \to 0 \Rightarrow JS(\mathbb{P}_n, \mathbb{P}) \to 0)$ — Let $\mathbb{P}_m$ be the mixture distribution $\mathbb{P}_m = \frac{1}{2}\mathbb{P}_n + \frac{1}{2}\mathbb{P}$ (note that $\mathbb{P}_m$ depends on n). It is easily verified that $\delta(\mathbb{P}_m, \mathbb{P}_n) \leq \delta(\mathbb{P}_n, \mathbb{P})$, and in particular this tends to 0 (as does $\delta(\mathbb{P}_m, \mathbb{P})$). We now show this for completeness. Let μ be a signed measure, we define $\|\mu\|_{TV} = \sup_{A\subseteq\mathcal{X}} |\mu(A)|$. for all Borel sets A. In this case,

$$\begin{aligned}
\delta(\mathbb{P}_m, \mathbb{P}_n) &= \|\mathbb{P}_m - \mathbb{P}_n\|_{TV} \\
&= \|\frac{1}{2}\mathbb{P} + \frac{1}{2}\mathbb{P}_n - \mathbb{P}_n\|_{TV} \\
&= \frac{1}{2}\|\mathbb{P} - \mathbb{P}_n\|_{TV} \\
&= \frac{1}{2}\delta(\mathbb{P}_n, \mathbb{P}) \leq \delta(\mathbb{P}_n, \mathbb{P})
\end{aligned}$$

Let $f_n = \frac{d\mathbb{P}_n}{d\mathbb{P}_m}$ be the Radon-Nykodim derivative between $\mathbb{P}_n$ and the mixture. Note that by construction for every Borel set A we have $\mathbb{P}_n(A) \leq 2\mathbb{P}_m(A)$. If $A = \{f_n > 3\}$ then we get

$$\mathbb{P}_n(A) = \int_A f_n \, \mathrm{d}\mathbb{P}_m \geq 3\mathbb{P}_m(A)$$

which implies $\mathbb{P}_m(A) = 0$. This means that f_n is bounded by 3 $\mathbb{P}_m$(and therefore $\mathbb{P}_n$ and $\mathbb{P}$)-almost everywhere. We could have done this for any constant larger than 2 but for our purposes 3 will sufice.
Let $\epsilon > 0$ fixed, and $A_n = \{f_n > 1 + \epsilon\}$. Then,

$$\mathbb{P}_n(A_n) = \int_{A_n} f_n \, \mathrm{d}\mathbb{P}_m \geq (1 + \epsilon)\mathbb{P}_m(A_n)$$

Therefore,

$$\begin{aligned}
\epsilon\mathbb{P}_m(A_n) &\leq \mathbb{P}_n(A_n) - \mathbb{P}_m(A_n) \\
&\leq |\mathbb{P}_n(A_n) - \mathbb{P}_m(A_n)| \\
&\leq \delta(\mathbb{P}_n, \mathbb{P}_m) \\
&\leq \delta(\mathbb{P}_n, \mathbb{P}).
\end{aligned}$$

Which implies $\mathbb{P}_m(A_m) \leq \frac{1}{\epsilon}\delta(\mathbb{P}_n, \mathbb{P})$. Furthermore,

$$\begin{aligned}
\mathbb{P}_n(A_n) &\leq \mathbb{P}_m(A_n) + |\mathbb{P}_n(A_n) - \mathbb{P}_m(A_n)| \\
&\leq \frac{1}{\epsilon}\delta(\mathbb{P}_n, \mathbb{P}) + \delta(\mathbb{P}_n, \mathbb{P}_m) \\
&\leq \frac{1}{\epsilon}\delta(\mathbb{P}_n, \mathbb{P}) + \delta(\mathbb{P}_n, \mathbb{P}) \\
&\leq \left(\frac{1}{\epsilon} + 1\right)\delta(\mathbb{P}_n, \mathbb{P})
\end{aligned}$$

We now can see that

$$\begin{aligned}
KL(\mathbb{P}_n \| \mathbb{P}_m) &= \int \log(f_n) \, \mathrm{d}\mathbb{P}_n \\
&\leq \log(1 + \epsilon) + \int_{A_n} \log(f_n) \, \mathrm{d}\mathbb{P}_n \\
&\leq \log(1 + \epsilon) + \log(3)\mathbb{P}_n(A_n) \\
&\leq \log(1 + \epsilon) + \log(3)\left(\frac{1}{\epsilon} + 1\right)\delta(\mathbb{P}_n, \mathbb{P})
\end{aligned}$$

Taking limsup we get $0 \leq \limsup KL(\mathbb{P}_n \| \mathbb{P}_m) \leq \log(1 + \epsilon)$ for all $\epsilon > 0$, which means $KL(\mathbb{P}_n \| \mathbb{P}_m) \to 0$.
In the same way, we can define $g_n = \frac{d\mathbb{P}}{d\mathbb{P}_m}$, and

$$2\mathbb{P}_m(\{g_n > 3\}) \geq \mathbb{P}(\{g_n > 3\}) \geq 3\mathbb{P}_m(\{g_n > 3\})$$

meaning that $\mathbb{P}_m(\{g_n > 3\}) = 0$ and therefore g_n is bounded by 3 almost everywhere for $\mathbb{P}_n, \mathbb{P}_m$ and $\mathbb{P}$. With the same calculation, $B_n = \{g_n > 1 + \epsilon\}$ and

$$\mathbb{P}(B_n) = \int_{B_n} g_n \, \mathrm{d}\mathbb{P}_m \geq (1+\epsilon)\mathbb{P}_m(B_n)$$

so $\mathbb{P}_m(B_n) \leq \frac{1}{\epsilon}\delta(\mathbb{P}, \mathbb{P}_m) \to 0$, and therefore $\mathbb{P}(B_n) \to 0$. We can now show

$$\begin{aligned} KL(\mathbb{P}\|\mathbb{P}_m) &= \int \log(g_n) \, \mathrm{d}\mathbb{P} \\ &\leq \log(1+\epsilon) + \int_{B_n} \log(g_n) \, \mathrm{d}\mathbb{P} \\ &\leq \log(1+\epsilon) + \log(3)\mathbb{P}(B_n) \end{aligned}$$

so we achieve $0 \leq \limsup KL(\mathbb{P}\|\mathbb{P}_m) \leq \log(1+\epsilon)$ and then $KL(\mathbb{P}\|\mathbb{P}_m) \to 0$. Finally, we conclude

$$JS(\mathbb{P}_n, \mathbb{P}) = \frac{1}{2}KL(\mathbb{P}_n\|\mathbb{P}_m) + \frac{1}{2}KL(\mathbb{P}\|\mathbb{P}_m) \to 0$$

- $(JS(\mathbb{P}_n, \mathbb{P}) \to 0 \Rightarrow \delta(\mathbb{P}_n, \mathbb{P}) \to 0)$ — by a simple application of the triangular and Pinsker's inequalities we get

$$\begin{aligned} \delta(\mathbb{P}_n, \mathbb{P}) &\leq \delta(\mathbb{P}_n, \mathbb{P}_m) + \delta(\mathbb{P}, \mathbb{P}_m) \\ &\leq \sqrt{\frac{1}{2}KL(\mathbb{P}_n\|\mathbb{P}_m)} + \sqrt{\frac{1}{2}KL(\mathbb{P}\|\mathbb{P}_m)} \\ &\leq 2\sqrt{JS(\mathbb{P}_n, \mathbb{P})} \to 0 \end{aligned}$$

2. This is a long known fact that W metrizes the weak* topology of $(C(\mathcal{X}), \|\cdot\|_\infty)$ on $\mathrm{Prob}(\mathcal{X})$, and by definition this is the topology of convergence in distribution. A proof of this can be found (for example) in [22].

3. This is a straightforward application of Pinsker's inequality

$$\delta(\mathbb{P}_n, \mathbb{P}) \leq \sqrt{\frac{1}{2}KL(\mathbb{P}_n\|\mathbb{P})} \to 0$$
$$\delta(\mathbb{P}, \mathbb{P}_n) \leq \sqrt{\frac{1}{2}KL(\mathbb{P}\|\mathbb{P}_n)} \to 0$$

4. This is trivial by recalling the fact that δ and W give the strong and weak* topologies on the dual of $(C(\mathcal{X}), \|\cdot\|_\infty)$ when restricted to $\mathrm{Prob}(\mathcal{X})$.

□

Proof of Theorem 3. Let us define

$$\begin{aligned} V(\tilde{f}, \theta) &= \mathbb{E}_{x \sim \mathbb{P}_r}[\tilde{f}(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[\tilde{f}(x)] \\ &= \mathbb{E}_{x \sim \mathbb{P}_r}[\tilde{f}(x)] - \mathbb{E}_{z \sim p(z)}[\tilde{f}(g_\theta(z))] \end{aligned}$$

where $\tilde{f}$ lies in $\mathcal{F} = \{\tilde{f} : \mathcal{X} \to \mathbb{R} \text{ , } \tilde{f} \in C_b(\mathcal{X}), \|\tilde{f}\|_L \leq 1\}$ and $\theta \in \mathbb{R}^d$.

Since $\mathcal{X}$ is compact, we know by the Kantorovich-Rubenstein duality [22] that there is an $f \in \mathcal{F}$ that attains the value

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\tilde{f} \in \mathcal{F}} V(\tilde{f}, \theta) = V(f, \theta)$$

Let us define $X^*(\theta) = \{f \in \mathcal{F} : V(f, \theta) = W(\mathbb{P}_r, \mathbb{P}_\theta)\}$. By the above point we know then that $X^*(\theta)$ is non-empty. We know that by a simple envelope theorem ([12], Theorem 1) that

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = \nabla_\theta V(f, \theta)$$

for any $f \in X^*(\theta)$ when both terms are well-defined.

Let $f \in X^*(\theta)$, which we knows exists since $X^*(\theta)$ is non-empty for all θ. Then, we get

$$\begin{aligned} \nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) &= \nabla_\theta V(f, \theta) \\ &= \nabla_\theta [\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{z \sim p(z)}[f(g_\theta(z))] \\ &= -\nabla_\theta \mathbb{E}_{z \sim p(z)}[f(g_\theta(z))] \end{aligned}$$

under the condition that the first and last terms are well-defined. The rest of the proof will be dedicated to show that

$$-\nabla_\theta \mathbb{E}_{z \sim p(z)}[f(g_\theta(z))] = -\mathbb{E}_{z \sim p(z)}[\nabla_\theta f(g_\theta(z))] \tag{5}$$

when the right hand side is defined. For the reader who is not interested in such technicalities, he or she can skip the rest of the proof.

Since $f \in \mathcal{F}$, we know that it is 1-Lipschitz. Furthermore, $g_\theta(z)$ is locally Lipschitz as a function of (θ, z). Therefore, $f(g_\theta(z))$ is locally Lipschitz on (θ, z) with constants $L(\theta, z)$ (the same ones as g). By Radamacher's Theorem, $f(g_\theta(z))$ has to be differentiable almost everywhere for (θ, z) jointly. Rewriting this, the set $A = \{(\theta, z) : f \circ g \text{ is not differentiable}\}$ has measure 0. By Fubini's Theorem, this implies that for almost every θ the section $A_\theta = \{z : (\theta, z) \in A\}$ has measure 0. Let's now fix a θ_0 such that the measure of A_{θ_0} is null (**such as when the right hand side of equation** (5) **is well defined**). For this θ_0 we have $\nabla_\theta f(g_\theta(z))|_{\theta_0}$ is well-defined for almost any z, and since $p(z)$ has a density, it is defined $p(z)$-a.e. By assumption 1 we know that

$$\mathbb{E}_{z \sim p(z)}[\|\nabla_\theta f(g_\theta(z))|_{\theta_0}\|] \leq \mathbb{E}_{z \sim p(z)}[L(\theta_0, z)] < +\infty$$

so $\mathbb{E}_{z \sim p(z)}[\nabla_\theta f(g_\theta(z))|_{\theta_0}]$ is well-defined for almost every θ_0. Now, we can see

$$\frac{\mathbb{E}_{z \sim p(z)}[f(g_\theta(z))] - \mathbb{E}_{z \sim p(z)}[f(g_{\theta_0}(z))] - \langle (\theta - \theta_0), \mathbb{E}_{z \sim p(z)}[\nabla_\theta f(g_\theta(z))|_{\theta_0}] \rangle}{\|\theta - \theta_0\|} \tag{6}$$

$$= \mathbb{E}_{z\sim p(z)}\left[\frac{f(g_\theta(z)) - f(g_{\theta_0}(z)) - \langle(\theta-\theta_0), \nabla_\theta f(g_\theta(z))|_{\theta_0}\rangle}{\|\theta-\theta_0\|}\right]$$

By differentiability, the term inside the integral converges $p(z)$-a.e. to 0 as $\theta \to \theta_0$. Furthermore,

$$\begin{aligned}\|\frac{f(g_\theta(z)) - f(g_{\theta_0}(z)) - \langle(\theta-\theta_0), \nabla_\theta f(g_\theta(z))|_{\theta_0}\rangle}{\|\theta-\theta_0\|}\| & \\ \leq \frac{\|\theta-\theta_0\|L(\theta_0,z) + \|\theta-\theta_0\|\|\nabla_\theta f(g_\theta(z))|_{\theta_0}\|}{\|\theta-\theta_0\|} & \\ \leq 2L(\theta_0,z) &\end{aligned}$$

and since $\mathbb{E}_{z\sim p(z)}[2L(\theta_0,z)] < +\infty$ by assumption 1, we get by dominated convergence that Equation 6 converges to 0 as $\theta \to \theta_0$ so

$$\nabla_\theta \mathbb{E}_{z\sim p(z)}[f(g_\theta(z))] = \mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z))]$$

for almost every θ, and in particular when the right hand side is well defined. Note that the mere existance of the left hand side (meaning the differentiability a.e. of $\mathbb{E}_{z\sim p(z)}[f(g_\theta(z))]$) had to be proven, which we just did. □

D Energy-based GANs optimize total variation

In this appendix we show that under an optimal discriminator, energy-based GANs (EBGANs) [25] optimize the total variation distance between the real and generated distributions.

Energy-based GANs are trained in a similar fashion to GANs, only under a different loss function. They have a discriminator D who tries to minimize

$$L_D(D, g_\theta) = \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \mathbb{E}_{z\sim p(z)}[[m - D(g_\theta(z))]^+]$$

for some $m > 0$ and $[x]^+ = \max(0, x)$ and a generator network g_θ that's trained to minimize

$$L_G(D, g_\theta) = \mathbb{E}_{z\sim p(z)}[D(g_\theta(z))] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]$$

Very importantly, D is constrained to be non-negative, since otherwise the trivial solution for D would be to set everything to arbitrarily low values. The original EBGAN paper used only $\mathbb{E}_{z\sim p(z)}[D(g_\theta(z))]$ for the loss of the generator, but this is obviously equivalent to our definition since the term $\mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]$ does not dependent on θ for a fixed discriminator (such as when backproping to the generator in EBGAN training) and thus minimizing one or the other is equivalent.

We say that a measurable function $D^* : \mathcal{X} \to [0, +\infty)$ is optimal for g_θ (or $\mathbb{P}_\theta$) if $L_D(D^*, g_\theta) \leq L_D(D, g_\theta)$ for all other measurable functions D. We show that such a discriminator always exists for any two distributions $\mathbb{P}_r$ and $\mathbb{P}_\theta$, and that under such a discriminator, $L_G(D^*, g_\theta)$ is proportional to $\delta(\mathbb{P}_r, \mathbb{P}_\theta)$. As a simple corollary, we get the fact that $L_G(D^*, g_\theta)$ attains its minimum value if and only if $\delta(\mathbb{P}_r, \mathbb{P}_\theta)$ is at its minimum value, which is 0, and $\mathbb{P}_r = \mathbb{P}_\theta$ (Theorems 1-2 of [25]).

Theorem 4. *Let $\mathbb{P}_r$ be a the real data distribution over a compact space $\mathcal{X}$. Let $g_\theta : \mathcal{Z} \to \mathcal{X}$ be a measurable function (such as any neural network). Then, an optimal discriminator D^* exists for $\mathbb{P}_r$ and $\mathbb{P}_\theta$, and*

$$L_G(D^*, g_\theta) = \frac{m}{2}\delta(\mathbb{P}_r, \mathbb{P}_\theta)$$

Proof. First, we prove that there exists an optimal discriminator. Let $D : \mathcal{X} \to [0, +\infty)$ be a measurable function, then $D'(x) := \min(D(x), m)$ is also a measurable function, and $L_D(D', g_\theta) \leq L_D(D, g_\theta)$. Therefore, a function $D^* : \mathcal{X} \to [0, +\infty)$ is optimal if and only if $D^{*\prime}$ is. Furthermore, it is optimal if and only if $L_D(D^*, g_\theta) \leq L_D(D, g_\theta)$ for all $D : \mathcal{X} \to [0, m]$. We are then interested to see if there's an optimal discriminator for the problem $\min_{0\leq D(x)\leq m} L_D(D, g_\theta)$.

Note now that if $0 \leq D(x) \leq m$ we have

$$\begin{aligned}
L_D(D, g_\theta) &= \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \mathbb{E}_{z\sim p(z)}[[m - D(g_\theta(z))]^+] \\
&= \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \mathbb{E}_{z\sim p(z)}[m - D(g_\theta(z))] \\
&= m + \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{z\sim p(z)}[D(g_\theta(z))] \\
&= m + \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[D(x)]
\end{aligned}$$

Therefore, we know that

$$\begin{aligned}\inf_{0\leq D(x)\leq m} L_D(D, g_\theta) &= m + \inf_{0\leq D(x)\leq m} \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[D(x)] \\ &= m + \inf_{-\frac{m}{2}\leq D(x)\leq \frac{m}{2}} \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[D(x)] \\ &= m + \frac{m}{2} \inf_{-1\leq f(x)\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]\end{aligned}$$

The interesting part is that

$$\inf_{-1\leq f(x)\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] = -\delta(\mathbb{P}_r, \mathbb{P}_\theta) \tag{7}$$

and there is an $f^* : \mathcal{X} \to [-1, 1]$ such that $\mathbb{E}_{x\sim\mathbb{P}_r}[f^*(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f^*(x)] = -\delta(\mathbb{P}_r, \mathbb{P}_\theta)$. This is a long known fact, found for example in [22], but we prove it later for completeness. In that case, we define $D^*(x) = \frac{m}{2} f^*(x) + \frac{m}{2}$. We then have $0 \leq D(x) \leq m$ and

$$\begin{aligned}L_D(D^*, g_\theta) &= m + \mathbb{E}_{x\sim\mathbb{P}_r}[D^*(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[D^*(x)] \\ &= m + \frac{m}{2}\mathbb{E}_{x\sim\mathbb{P}_r}[D^*(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f^*(x)] \\ &= m - \frac{m}{2}\delta(\mathbb{P}_r, \mathbb{P}_\theta) \\ &= \inf_{0\leq D(x)\leq m} L_D(D, g_\theta)\end{aligned}$$

This shows that D^* is optimal and $L_D(D^*, g_\theta) = m - \frac{m}{2}\delta(\mathbb{P}_r, \mathbb{P}_\theta)$. Furthermore,

$$\begin{aligned}L_G(D^*, g_\theta) &= \mathbb{E}_{z\sim p(z)}[D^*(g_\theta(z))] - \mathbb{E}_{x\sim\mathbb{P}_r}[D^*(x)] \\ &= -L_D(D^*, g_\theta) + m \\ &= \frac{m}{2}\delta(\mathbb{P}_r, \mathbb{P}_g)\end{aligned}$$

concluding the proof.

For completeness, we now show a proof for equation (7) and the existence of said f^* that attains the value of the infimum. Take $\mu = \mathbb{P}_r - \mathbb{P}_\theta$, which is a signed measure, and (P, Q) its Hahn decomposition. Then, we can define $f^* := \mathbb{1}_Q - \mathbb{1}_P$. By construction, then

$$\begin{aligned}EE_{x\sim\mathbb{P}_r}[f^*(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f^*(x)] &= \int f^* \,\mathrm{d}\mu = \mu(Q) - \mu(P) \\ &= -(\mu(P) - \mu(Q)) = -\|\mu\|_{TV} \\ &= -\|\mathbb{P}_r - \mathbb{P}_\theta\|_{TV} \\ &= -\delta(\mathbb{P}_r, \mathbb{P}_\theta)\end{aligned}$$

Furthermore, if f is bounded between -1 and 1, we get

$$\begin{aligned}
|\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]| &= |\int f \,\mathrm{d}\mathbb{P}_r - \int f \,\mathrm{d}\mathbb{P}_\theta| \\
&= |\int f \,\mathrm{d}\mu| \\
&\leq \int |f| \,\mathrm{d}|\mu| \leq \int 1 \,\mathrm{d}|\mu| \\
&= |\mu|(\mathcal{X}) = \|\mu\|_{TV} = \delta(\mathbb{P}_r, \mathbb{P}_\theta)
\end{aligned}$$

Since δ is positive, we can conclude $\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \geq -\delta(\mathbb{P}_r, \mathbb{P}_\theta)$. $\square$

E Generator's cost during normal GAN training

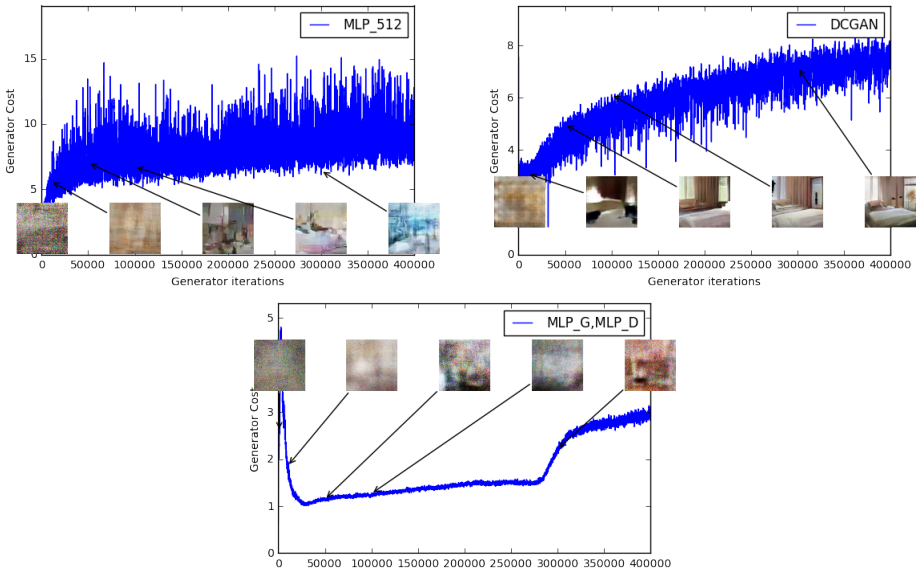


Figure 8: Cost of the generator during normal GAN training, for an MLP generator (upper left) and a DCGAN generator (upper right). Both had a DCGAN discriminator. ***Both curves have increasing error****. Samples get better for the DCGAN but the cost of the generator increases, pointing towards no significant correlation between sample quality and loss. Bottom: MLP with both generator and discriminator. The curve goes up and down regardless of sample quality. All training curves were passed through the same median filter as in Figure 3.*

F Sheets of samples

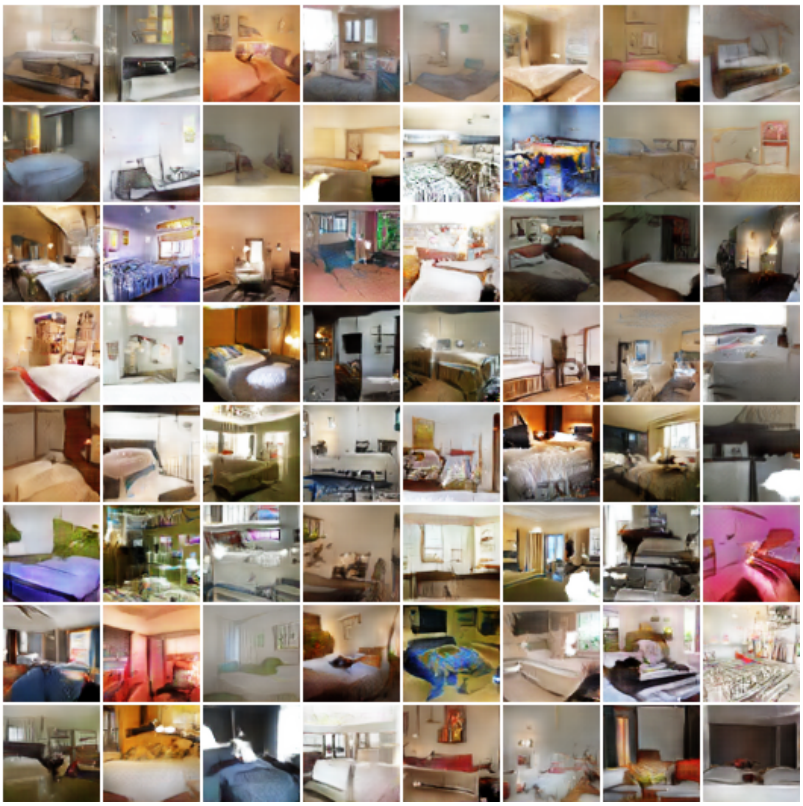


Figure 9: WGAN algorithm: generator and critic are DCGANs.

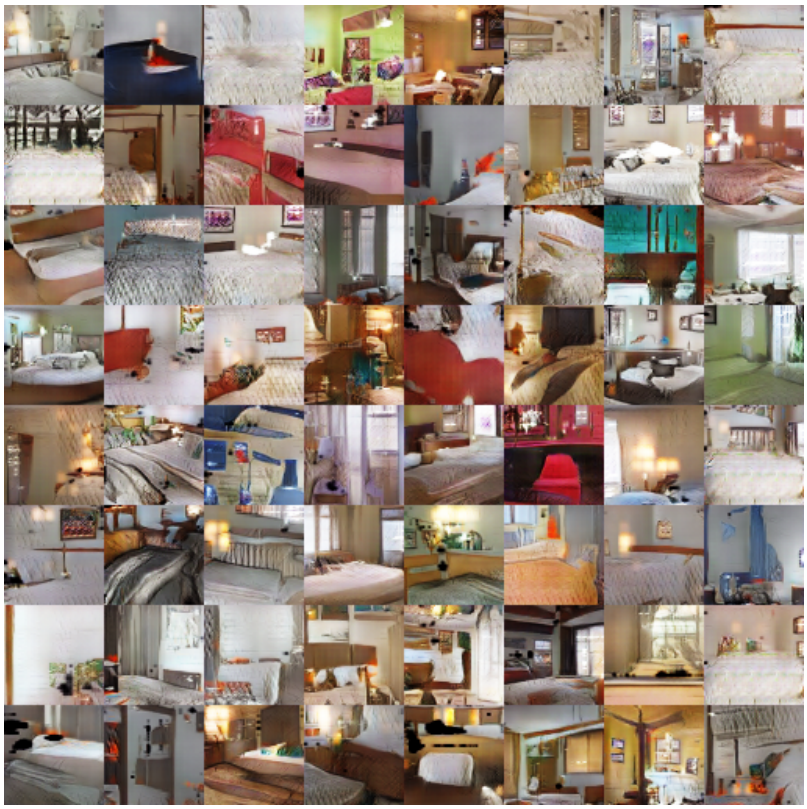


Figure 10: Standard GAN procedure: generator and discriminator are DCGANs.

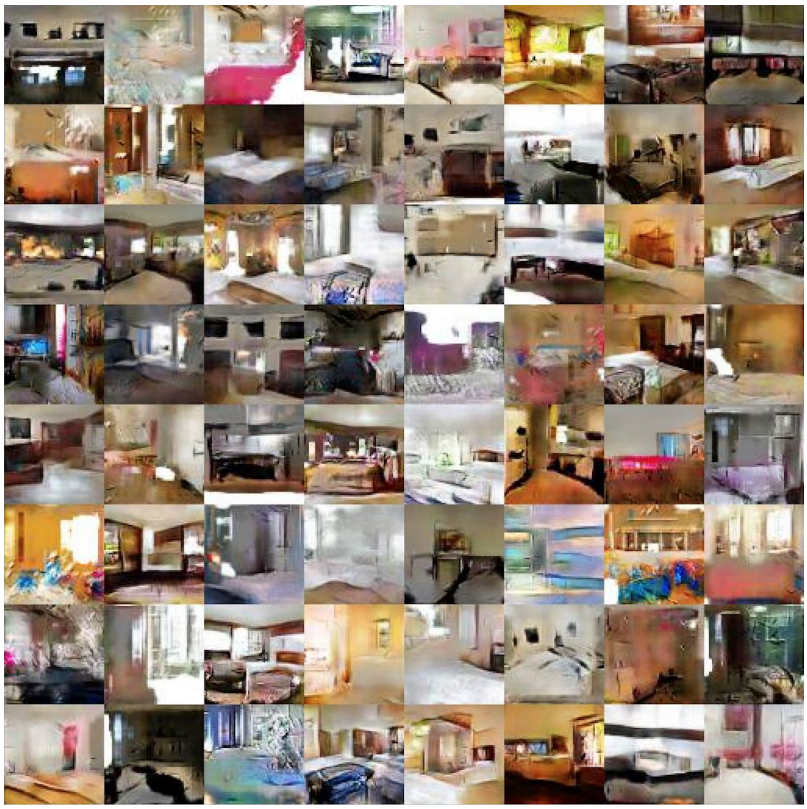


Figure 11: WGAN algorithm: generator is a DCGAN without batchnorm and constant filter size. Critic is a DCGAN.

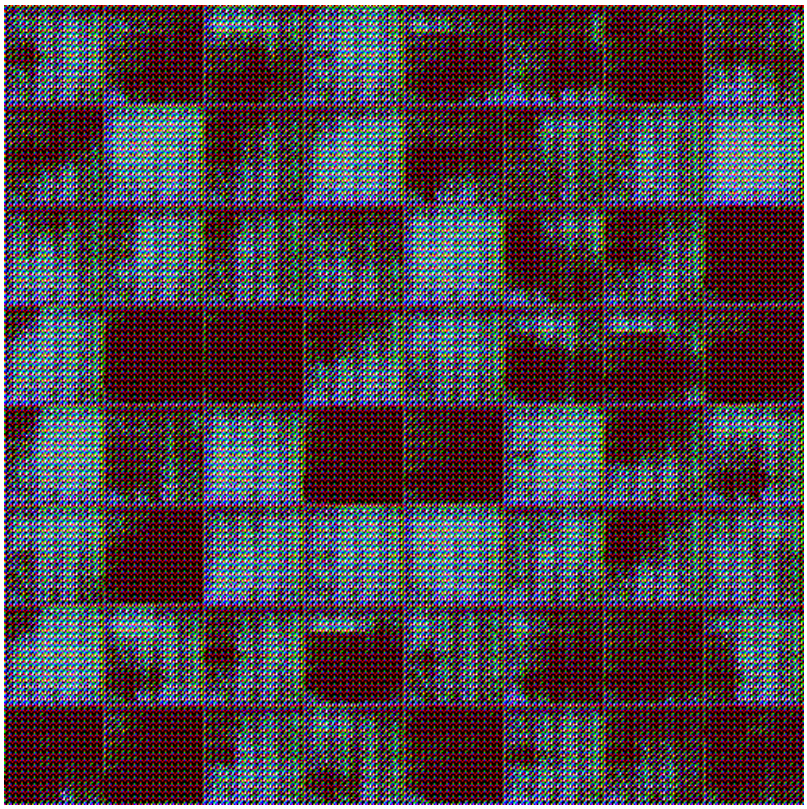


Figure 12: Standard GAN procedure: generator is a DCGAN without batchnorm and constant filter size. Discriminator is a DCGAN.

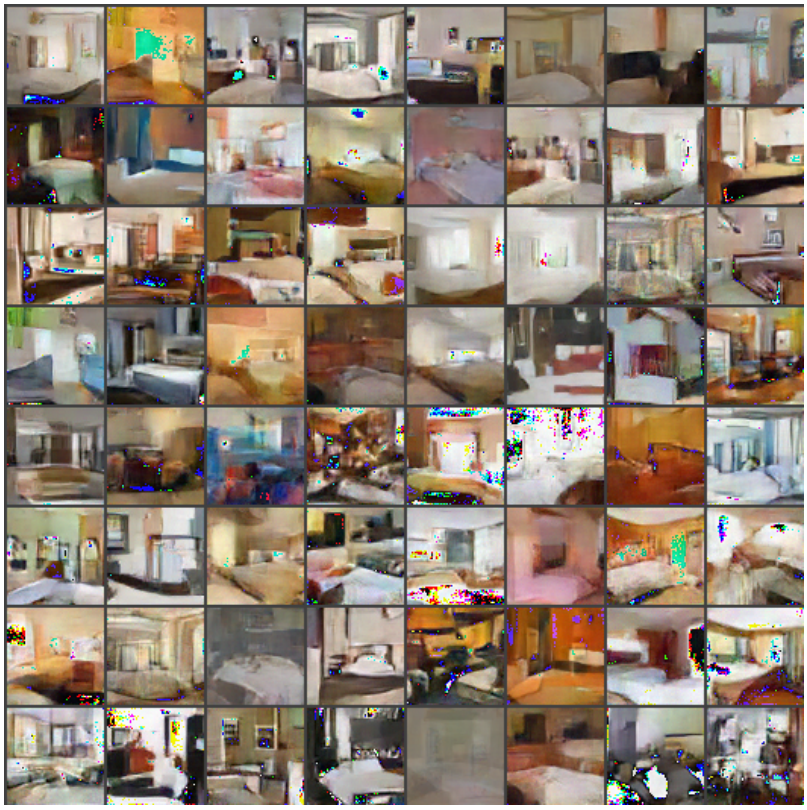


Figure 13: WGAN algorithm: generator is an MLP with 4 hidden layers of 512 units, critic is a DCGAN.

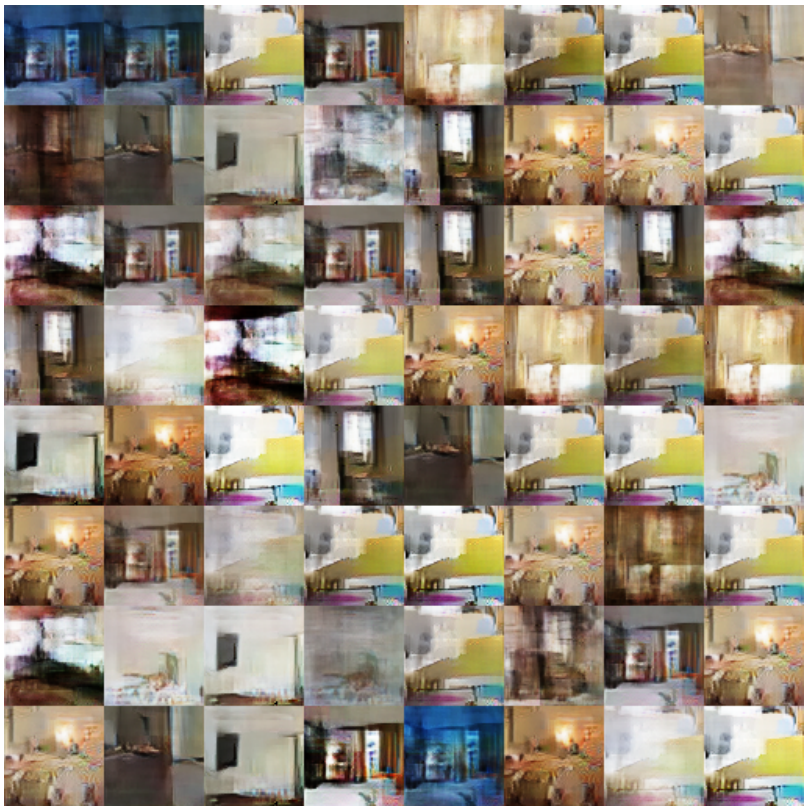


Figure 14: Standard GAN procedure: generator is an MLP with 4 hidden layers of 512 units, discriminator is a DCGAN.

Improved Training of Wasserstein GANs

Ishaan Gulrajani[1]*, **Faruk Ahmed**[1], **Martin Arjovsky**[2], **Vincent Dumoulin**[1], **Aaron Courville**[1,3]
[1] Montreal Institute for Learning Algorithms
[2] Courant Institute of Mathematical Sciences
[3] CIFAR Fellow
igul222@gmail.com
{faruk.ahmed,vincent.dumoulin,aaron.courville}@umontreal.ca
ma4371@nyu.edu

Abstract

Generative Adversarial Networks (GANs) are powerful generative models, but suffer from training instability. The recently proposed Wasserstein GAN (WGAN) makes progress toward stable training of GANs, but sometimes can still generate only poor samples or fail to converge. We find that these problems are often due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to undesired behavior. We propose an alternative to clipping weights: penalize the norm of gradient of the critic with respect to its input. Our proposed method performs better than standard WGAN and enables stable training of a wide variety of GAN architectures with almost no hyperparameter tuning, including 101-layer ResNets and language models with continuous generators. We also achieve high quality generations on CIFAR-10 and LSUN bedrooms. [†]

1 Introduction

Generative Adversarial Networks (GANs) [9] are a powerful class of generative models that cast generative modeling as a game between two networks: a generator network produces synthetic data given some noise source and a discriminator network discriminates between the generator's output and true data. GANs can produce very visually appealing samples, but are often hard to train, and much of the recent work on the subject [23, 19, 2, 21] has been devoted to finding ways of stabilizing training. Despite this, consistently stable training of GANs remains an open problem.

In particular, [1] provides an analysis of the convergence properties of the value function being optimized by GANs. Their proposed alternative, named Wasserstein GAN (WGAN) [2], leverages the Wasserstein distance to produce a value function which has better theoretical properties than the original. WGAN requires that the discriminator (called the *critic* in that work) must lie within the space of 1-Lipschitz functions, which the authors enforce through weight clipping.

Our contributions are as follows:

1. On toy datasets, we demonstrate how critic weight clipping can lead to undesired behavior.
2. We propose *gradient penalty (WGAN-GP)*, which does not suffer from the same problems.
3. We demonstrate stable training of varied GAN architectures, performance improvements over weight clipping, high-quality image generation, and a character-level GAN language model without any discrete sampling.

*Now at Google Brain
[†]Code for our models is available at https://github.com/igul222/improved_wgan_training.

2 Background

2.1 Generative adversarial networks

The GAN training strategy is to define a game between two competing networks. The *generator* network maps a source of noise to the input space. The *discriminator* network receives either a generated sample or a true data sample and must distinguish between the two. The generator is trained to fool the discriminator.

Formally, the game between the generator G and the discriminator D is the minimax objective:

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(1 - D(\tilde{\boldsymbol{x}}))], \tag{1}$$

where $\mathbb{P}_r$ is the data distribution and $\mathbb{P}_g$ is the model distribution implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z}),\ \boldsymbol{z} \sim p(\boldsymbol{z})$ (the input $\boldsymbol{z}$ to the generator is sampled from some simple noise distribution p, such as the uniform distribution or a spherical Gaussian distribution).

If the discriminator is trained to optimality before each generator parameter update, then minimizing the value function amounts to minimizing the Jensen-Shannon divergence between $\mathbb{P}_r$ and $\mathbb{P}_g$ [9], but doing so often leads to vanishing gradients as the discriminator saturates. In practice, [9] advocates that the generator be instead trained to maximize $\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(D(\tilde{\boldsymbol{x}}))]$, which goes some way to circumvent this difficulty. However, even this modified loss function can misbehave in the presence of a good discriminator [1].

2.2 Wasserstein GANs

[2] argues that the divergences which GANs typically minimize are potentially not continuous with respect to the generator's parameters, leading to training difficulty. They propose instead using the *Earth-Mover* (also called Wasserstein-1) distance $W(q, p)$, which is informally defined as the minimum cost of transporting mass in order to transform the distribution q into the distribution p (where the cost is mass times transport distance). Under mild assumptions, $W(q, p)$ is continuous everywhere and differentiable almost everywhere.

The WGAN value function is constructed using the Kantorovich-Rubinstein duality [25] to obtain

$$\min_G \max_{D\in\mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\big[D(\tilde{\boldsymbol{x}})\big] \tag{2}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions and $\mathbb{P}_g$ is once again the model distribution implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z}),\ \boldsymbol{z} \sim p(\boldsymbol{z})$. In that case, under an optimal discriminator (called a *critic* in the paper, since it's not trained to classify), minimizing the value function with respect to the generator parameters minimizes $W(\mathbb{P}_r, \mathbb{P}_g)$.

The WGAN value function results in a critic function whose gradient with respect to its input is better behaved than its GAN counterpart, making optimization of the generator easier. Empirically, it was also observed that the WGAN value function appears to correlate with sample quality, which is not the case for GANs [2].

To enforce the Lipschitz constraint on the critic, [2] propose to clip the weights of the critic to lie within a compact space $[-c, c]$. The set of functions satisfying this constraint is a subset of the k-Lipschitz functions for some k which depends on c and the critic architecture. In the following sections, we demonstrate some of the issues with this approach and propose an alternative.

2.3 Properties of the optimal WGAN critic

In order to understand why weight clipping is problematic in a WGAN critic, as well as to motivate our approach, we highlight some properties of the optimal critic in the WGAN framework. We prove these in the Appendix.

Proposition 1. *Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions in $\mathcal{X}$, a compact metric space. Then, there is a 1-Lipschitz function f^* which is the optimal solution of* $\max_{\|f\|_L \leq 1} \mathbb{E}_{y \sim \mathbb{P}_r}[f(y)] - \mathbb{E}_{x \sim \mathbb{P}_g}[f(x)]$. *Let π be the optimal coupling between $\mathbb{P}_r$ and $\mathbb{P}_g$, defined as the minimizer of:* $W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\pi \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\pi}[\|x - y\|]$ *where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of joint distributions $\pi(x, y)$ whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively. Then, if f^* is differentiable*[‡], *$\pi(x = y) = 0$*[§], *and $x_t = tx + (1-t)y$ with $0 \leq t \leq 1$, it holds that* $\mathbb{P}_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}\right] = 1$.

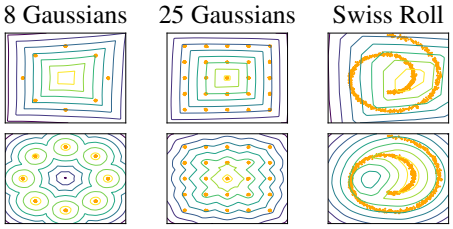
Corollary 1. *f^* has gradient norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$.*

3 Difficulties with weight constraints

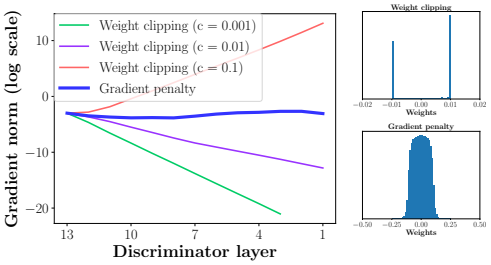
We find that weight clipping in WGAN leads to optimization difficulties, and that even when optimization succeeds the resulting critic can have a pathological value surface. We explain these problems below and demonstrate their effects; however we do not claim that each one always occurs in practice, nor that they are the only such mechanisms.

Our experiments use the specific form of weight constraint from [2] (hard clipping of the magnitude of each weight), but we also tried other weight constraints (L2 norm clipping, weight normalization), as well as soft constraints (L1 and L2 weight decay) and found that they exhibit similar problems.

To some extent these problems can be mitigated with batch normalization in the critic, which [2] use in all of their experiments. However even with batch normalization, we observe that very deep WGAN critics often fail to converge.



(a) Value surfaces of WGAN critics trained to optimality on toy datasets using (top) weight clipping and (bottom) gradient penalty. Critics trained with weight clipping fail to capture higher moments of the data distribution. The 'generator' is held fixed at the real data plus Gaussian noise.



(b) (left) Gradient norms of deep WGAN critics during training on the Swiss Roll dataset either explode or vanish when using weight clipping, but not when using a gradient penalty. (right) Weight clipping (top) pushes weights towards two values (the extremes of the clipping range), unlike gradient penalty (bottom).

Figure 1: Gradient penalty in WGANs does not exhibit undesired behavior like weight clipping.

3.1 Capacity underuse

Implementing a k-Lipshitz constraint via weight clipping biases the critic towards much simpler functions. As stated previously in Corollary 1, the optimal WGAN critic has unit gradient norm almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$; under a weight-clipping constraint, we observe that our neural network architectures which try to attain their maximum gradient norm k end up learning extremely simple functions.

To demonstrate this, we train WGAN critics with weight clipping to optimality on several toy distributions, holding the generator distribution $\mathbb{P}_g$ fixed at the real distribution plus unit-variance Gaussian noise. We plot value surfaces of the critics in Figure 1a. We omit batch normalization in the

[‡]We can actually assume much less, and talk only about directional derivatives on the direction of the line; which we show in the proof always exist. This would imply that in every point where f^* is differentiable (and thus we can take gradients in a neural network setting) the statement holds.

[§]This assumption is in order to exclude the case when the matching point of sample x is x itself. It is satisfied in the case that $\mathbb{P}_r$ and $\mathbb{P}_g$ have supports that intersect in a set of measure 0, such as when they are supported by two low dimensional manifolds that don't perfectly align [1].

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

critic. In each case, the critic trained with weight clipping ignores higher moments of the data distribution and instead models very simple approximations to the optimal functions. In contrast, our approach does not suffer from this behavior.

3.2 Exploding and vanishing gradients

We observe that the WGAN optimization process is difficult because of interactions between the weight constraint and the cost function, which result in either vanishing or exploding gradients without careful tuning of the clipping threshold c.

To demonstrate this, we train WGAN on the Swiss Roll toy dataset, varying the clipping threshold c in $[10^{-1}, 10^{-2}, 10^{-3}]$, and plot the norm of the gradient of the critic loss with respect to successive layers of activations. Both generator and critic are 12-layer ReLU MLPs without batch normalization. Figure 1b shows that for each of these values, the gradient either grows or decays exponentially as we move farther back in the network. We find our method results in more stable gradients that neither vanish nor explode, allowing training of more complicated networks.

4 Gradient penalty

We now propose an alternative way to enforce the Lipschitz constraint. A differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere, so we consider directly constraining the gradient norm of the critic's output with respect to its input. To circumvent tractability issues, we enforce a soft version of the constraint with a penalty on the gradient norm for random samples $\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}$. Our new objective is

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Our gradient penalty}}. \tag{3}$$

Sampling distribution We implicitly define $\mathbb{P}_{\hat{\boldsymbol{x}}}$ sampling uniformly along straight lines between pairs of points sampled from the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$. This is motivated by the fact that the optimal critic contains straight lines with gradient norm 1 connecting coupled points from $\mathbb{P}_r$ and $\mathbb{P}_g$ (see Proposition 1). Given that enforcing the unit gradient norm constraint everywhere is intractable, enforcing it only along these straight lines seems sufficient and experimentally results in good performance.

Penalty coefficient All experiments in this paper use $\lambda = 10$, which we found to work well across a variety of architectures and datasets ranging from toy tasks to large ImageNet CNNs.

No critic batch normalization Most prior GAN implementations [22, 23, 2] use batch normalization in both the generator and the discriminator to help stabilize training, but batch normalization changes the form of the discriminator's problem from mapping a single input to a single output to mapping from an entire batch of inputs to a batch of outputs [23]. Our penalized training objective is no longer valid in this setting, since we penalize the norm of the critic's gradient with respect to each input independently, and not the entire batch. To resolve this, we simply omit batch normalization in the critic in our models, finding that they perform well without it. Our method works with normalization schemes which don't introduce correlations between examples. In particular, we recommend layer normalization [3] as a drop-in replacement for batch normalization.

Two-sided penalty We encourage the norm of the gradient to go towards 1 (two-sided penalty) instead of just staying below 1 (one-sided penalty). Empirically this seems not to constrain the critic too much, likely because the optimal WGAN critic anyway has gradients with norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$ and in large portions of the region in between (see subsection 2.3). In our early observations we found this to perform slightly better, but we don't investigate this fully. We describe experiments on the one-sided penalty in the appendix.

5 Experiments

5.1 Training random architectures within a set

We experimentally demonstrate our model's ability to train a large number of architectures which we think are useful to be able to train. Starting from the DCGAN architecture, we define a set of architecture variants by changing model settings to random corresponding values in Table 1. We believe that reliable training of many of the architectures in this set is a useful goal, but we do not claim that our set is an unbiased or representative sample of the whole space of useful architectures: it is designed to demonstrate a successful regime of our method, and readers should evaluate whether it contains architectures similar to their intended application.

Table 1: We evaluate WGAN-GP's ability to train the architectures in this set.

Nonlinearity (G)	[ReLU, LeakyReLU, $\frac{\text{softplus}(2x+2)}{2} - 1$, $\tanh$]
Nonlinearity (D)	[ReLU, LeakyReLU, $\frac{\text{softplus}(2x+2)}{2} - 1$, $\tanh$]
Depth (G)	[4, 8, 12, 20]
Depth (D)	[4, 8, 12, 20]
Batch norm (G)	[True, False]
Batch norm (D; layer norm for WGAN-GP)	[True, False]
Base filter count (G)	[32, 64, 128]
Base filter count (D)	[32, 64, 128]

From this set, we sample 200 architectures and train each on 32×32 ImageNet with both WGAN-GP and the standard GAN objectives. Table 2 lists the number of instances where either: only the standard GAN succeeded, only WGAN-GP succeeded, both succeeded, or both failed, where success is defined as `inception_score > min_score`. For most choices of score threshold, WGAN-GP successfully trains many architectures from this set which we were unable to train with the standard GAN objective. We give more experimental details in the appendix.

Table 2: Outcomes of training 200 random architectures, for different success thresholds. For comparison, our standard DCGAN scored 7.24.

Min. score	Only GAN	Only WGAN-GP	Both succeeded	Both failed
1.0	0	8	192	0
3.0	1	88	110	1
5.0	0	147	42	11
7.0	1	104	5	90
9.0	0	0	0	200

Figure 2: Different GAN architectures trained with different methods. We only succeeded in training every architecture with a shared set of hyperparameters using WGAN-GP.

5.2 Training varied architectures on LSUN bedrooms

To demonstrate our model's ability to train many architectures with its default settings, we train six different GAN architectures on the LSUN bedrooms dataset [31]. In addition to the baseline DCGAN architecture from [22], we choose six architectures whose successful training we demonstrate: *(1)* no BN and a constant number of filters in the generator, as in [2], *(2)* 4-layer 512-dim ReLU MLP generator, as in [2], *(3)* no normalization in either the discriminator or generator *(4)* gated multiplicative nonlinearities, as in [24], *(5)* tanh nonlinearities, and *(6)* 101-layer ResNet generator and discriminator.

Although we do not claim it is impossible without our method, to the best of our knowledge this is the first time very deep residual networks were successfully trained in a GAN setting. For each architecture, we train models using four different GAN methods: WGAN-GP, WGAN with weight clipping, DCGAN [22], and Least-Squares GAN [18]. For each objective, we used the default set of optimizer hyperparameters recommended in that work (except LSGAN, where we searched over learning rates).

For WGAN-GP, we replace any batch normalization in the discriminator with layer normalization (see section 4). We train each model for 200K iterations and present samples in Figure 2. We only succeeded in training every architecture with a shared set of hyperparameters using WGAN-GP. For every other training method, some of these architectures were unstable or suffered from mode collapse.

5.3 Improved performance over weight clipping

One advantage of our method over weight clipping is improved training speed and sample quality. To demonstrate this, we train WGANs with weight clipping and our gradient penalty on CIFAR-10 [13] and plot Inception scores [23] over the course of training in Figure 3. For WGAN-GP,

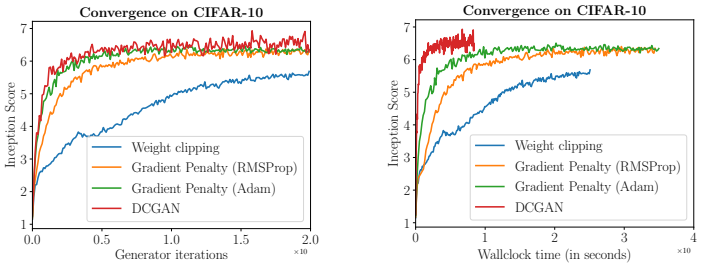


Figure 3: CIFAR-10 Inception score over generator iterations (left) or wall-clock time (right) for four models: WGAN with weight clipping, WGAN-GP with RMSProp and Adam (to control for the optimizer), and DCGAN. WGAN-GP significantly outperforms weight clipping and performs comparably to DCGAN.

we train one model with the same optimizer (RMSProp) and learning rate as WGAN with weight clipping, and another model with Adam and a higher learning rate. Even with the same optimizer, our method converges faster and to a better score than weight clipping. Using Adam further improves performance. We also plot the performance of DCGAN [22] and find that our method converges more slowly (in wall-clock time) than DCGAN, but its score is more stable at convergence.

5.4 Sample quality on CIFAR-10 and LSUN bedrooms

For equivalent architectures, our method achieves comparable sample quality to the standard GAN objective. However the increased stability allows us to improve sample quality by exploring a wider range of architectures. To demonstrate this, we find an architecture which establishes a new state of the art Inception score on unsupervised CIFAR-10 (Table 3). When we add label information (using the method in [20]), the same architecture outperforms all other published models except for SGAN.

Table 3: Inception scores on CIFAR-10. Our unsupervised model achieves state-of-the-art performance, and our conditional model outperforms all others except SGAN.

Unsupervised		Supervised	
Method	Score	Method	Score
ALI [8] (in [27])	$5.34 \pm .05$	SteinGAN [26]	6.35
BEGAN [4]	5.62	DCGAN (with labels, in [26])	6.58
DCGAN [22] (in [11])	$6.16 \pm .07$	Improved GAN [23]	$8.09 \pm .07$
Improved GAN (-L+HA) [23]	$6.86 \pm .06$	AC-GAN [20]	$8.25 \pm .07$
EGAN-Ent-VI [7]	$7.07 \pm .10$	SGAN-no-joint [11]	$8.37 \pm .08$
DFM [27]	$7.72 \pm .13$	WGAN-GP ResNet (ours)	$8.42 \pm .10$
WGAN-GP ResNet (ours)	$7.86 \pm .07$	**SGAN** [11]	$8.59 \pm .12$

We also train a deep ResNet on 128×128 LSUN bedrooms and show samples in Figure 4. We believe these samples are at least competitive with the best reported so far on any resolution for this dataset.

5.5 Modeling discrete data with a continuous generator

To demonstrate our method's ability to model degenerate distributions, we consider the problem of modeling a complex discrete distribution with a GAN whose generator is defined over a continuous space. As an instance of this problem, we train a character-level GAN language model on the Google Billion Word dataset [6]. Our generator is a simple 1D CNN which deterministically transforms a latent vector into a sequence of 32 one-hot character vectors through 1D convolutions. We apply a softmax nonlinearity at the output, but use no sampling step: during training, the softmax output is

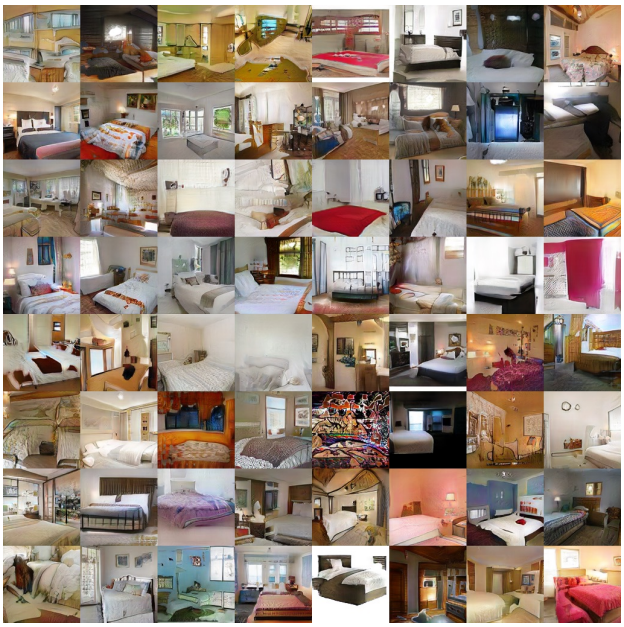


Figure 4: Samples of 128×128 LSUN bedrooms. We believe these samples are at least comparable to the best published results so far.

passed directly into the critic (which, likewise, is a simple 1D CNN). When decoding samples, we just take the argmax of each output vector.

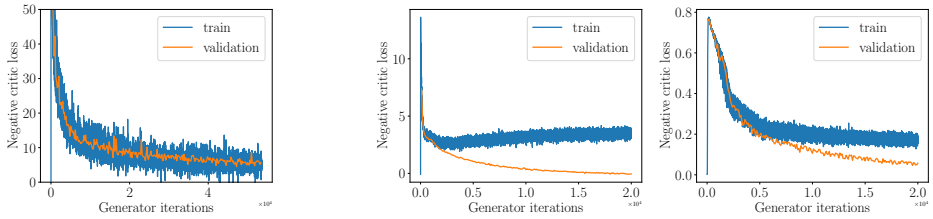
We present samples from the model in Table 4. Our model makes frequent spelling errors (likely because it has to output each character independently) but nonetheless manages to learn quite a lot about the statistics of language. We were unable to produce comparable results with the standard GAN objective, though we do not claim that doing so is impossible.

Table 4: Samples from a WGAN-GP character-level language model trained on sentences from the Billion Word dataset, truncated to 32 characters. The model learns to directly output one-hot character embeddings from a latent vector without any discrete sampling step. We were unable to achieve comparable results with the standard GAN objective and a continuous generator.

`Busino game camperate spent odea`	`Solice Norkedin pring in since`
`In the bankaway of smarling the`	`ThiS record ( 31. ) UBS ) and Ch`
`SingersMay , who kill that imvic`	`It was not the annuas were plogr`
`Keray Pents of the same Reagun D`	`This will be us , the ect of DAN`
`Manging include a tudancs shat "`	`These leaded as most-worsd p2 a0`
`His Zuith Dudget , the Denmbern`	`The time I paidOa South Cubry i`
`In during the Uitational questio`	`Dour Fraps higs it was these del`
`Divos from The ' noth ronkies of`	`This year out howneed allowed lo`
`She like Monday , of macunsuer S`	`Kaulna Seto consficutes to repor`

The difference in performance between WGAN and other GANs can be explained as follows. Consider the simplex $\Delta_n = \{p \in \mathbb{R}^n : p_i \geq 0, \sum_i p_i = 1\}$, and the set of vertices on the simplex (or one-hot vectors) $V_n = \{p \in \mathbb{R}^n : p_i \in \{0, 1\}, \sum_i p_i = 1\} \subseteq \Delta_n$. If we have a vocabulary of size n and we have a distribution $\mathbb{P}_r$ over sequences of size T, we have that $\mathbb{P}_r$ is a distribution on $V_n^T = V_n \times \cdots \times V_n$. Since V_n^T is a subset of Δ_n^T, we can also treat $\mathbb{P}_r$ as a distribution on Δ_n^T (by assigning zero probability mass to all points not in V_n^T).

$\mathbb{P}_r$ is discrete (or supported on a finite number of elements, namely V_n^T) on Δ_n^T, but $\mathbb{P}_g$ can easily be a continuous distribution over Δ_n^T. The KL divergences between two such distributions are infinite,



(a) (b)

Figure 5: (a) The negative critic loss of our model on LSUN bedrooms converges toward a minimum as the network trains. (b) WGAN training and validation losses on a random 1000-digit subset of MNIST show overfitting when using either our method (left) or weight clipping (right). In particular, with our method, the critic overfits faster than the generator, causing the training loss to increase gradually over time even as the validation loss drops.

and so the JS divergence is saturated. Although GANs do not literally minimize these divergences [16], in practice this means a discriminator might quickly learn to reject all samples that don't lie on V_n^T (sequences of one-hot vectors) and give meaningless gradients to the generator. However, it is easily seen that the conditions of Theorem 1 and Corollary 1 of [2] are satisfied even on this non-standard learning scenario with $\mathcal{X} = \Delta_n^T$. This means that $W(\mathbb{P}_r, \mathbb{P}_g)$ is still well defined, continuous everywhere and differentiable almost everywhere, and we can optimize it just like in any other continuous variable setting. The way this manifests is that in WGANs, the Lipschitz constraint forces the critic to provide a linear gradient from all Δ_n^T towards towards the real points in V_n^T.

Other attempts at language modeling with GANs [32, 14, 30, 5, 15, 10] typically use discrete models and gradient estimators [28, 12, 17]. Our approach is simpler to implement, though whether it scales beyond a toy language model is unclear.

5.6 Meaningful loss curves and detecting overfitting

An important benefit of weight-clipped WGANs is that their loss correlates with sample quality and converges toward a minimum. To show that our method preserves this property, we train a WGAN-GP on the LSUN bedrooms dataset [31] and plot the negative of the critic's loss in Figure 5a. We see that the loss converges as the generator minimizes $W(\mathbb{P}_r, \mathbb{P}_g)$.

Given enough capacity and too little training data, GANs will overfit. To explore the loss curve's behavior when the network overfits, we train large unregularized WGANs on a random 1000-image subset of MNIST and plot the negative critic loss on both the training and validation sets in Figure 5b. In both WGAN and WGAN-GP, the two losses diverge, suggesting that the critic overfits and provides an inaccurate estimate of $W(\mathbb{P}_r, \mathbb{P}_g)$, at which point all bets are off regarding correlation with sample quality. However in WGAN-GP, the training loss gradually increases even while the validation loss drops.

[29] also measure overfitting in GANs by estimating the generator's log-likelihood. Compared to that work, our method detects overfitting in the critic (rather than the generator) and measures overfitting against the same loss that the network minimizes.

6 Conclusion

In this work, we demonstrated problems with weight clipping in WGAN and introduced an alternative in the form of a penalty term in the critic loss which does not exhibit the same problems. Using our method, we demonstrated strong modeling performance and stability across a variety of architectures. Now that we have a more stable algorithm for training GANs, we hope our work opens the path for stronger modeling performance on large-scale image datasets and language. Another interesting direction is adapting our penalty term to the standard GAN objective function, where it might stabilize training by encouraging the discriminator to learn smoother decision boundaries.

Acknowledgements

We would like to thank Mohamed Ishmael Belghazi, Léon Bottou, Zihang Dai, Stefan Doerr, Ian Goodfellow, Kyle Kastner, Kundan Kumar, Luke Metz, Alec Radford, Colin Raffel, Sai Rajeshwar, Aditya Ramesh, Tom Sercu, Zain Shah and Jake Zhao for insightful comments.

References

[1] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. 2017.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[3] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] D. Berthelot, T. Schumm, and L. Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[5] T. Che, Y. Li, R. Zhang, R. D. Hjelm, W. Li, Y. Song, and Y. Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017.

[6] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[7] Z. Dai, A. Almahairi, P. Bachman, E. Hovy, and A. Courville. Calibrating energy-based generative adversarial networks. *arXiv preprint arXiv:1702.01691*, 2017.

[8] V. Dumoulin, M. I. D. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville. Adversarially learned inference. 2017.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[10] R. D. Hjelm, A. P. Jacob, T. Che, K. Cho, and Y. Bengio. Boundary-seeking generative adversarial networks. *arXiv preprint arXiv:1702.08431*, 2017.

[11] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie. Stacked generative adversarial networks. *arXiv preprint arXiv:1612.04357*, 2016.

[12] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[13] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[14] J. Li, W. Monroe, T. Shi, A. Ritter, and D. Jurafsky. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017.

[15] X. Liang, Z. Hu, H. Zhang, C. Gan, and E. P. Xing. Recurrent topic-transition gan for visual paragraph generation. *arXiv preprint arXiv:1703.07022*, 2017.

[16] S. Liu, O. Bousquet, and K. Chaudhuri. Approximation and convergence properties of generative adversarial learning. *arXiv preprint arXiv:1705.08991*, 2017.

[17] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[18] X. Mao, Q. Li, H. Xie, R. Y. Lau, and Z. Wang. Least squares generative adversarial networks. *arXiv preprint arXiv:1611.04076*, 2016.

[19] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[20] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016.

[21] B. Poole, A. A. Alemi, J. Sohl-Dickstein, and A. Angelova. Improved generator objectives for gans. *arXiv preprint arXiv:1612.02780*, 2016.

[22] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[23] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

[24] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.

[25] C. Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[26] D. Wang and Q. Liu. Learning to draw samples: With application to amortized mle for generative adversarial learning. *arXiv preprint arXiv:1611.01722*, 2016.

[27] D. Warde-Farley and Y. Bengio. Improving generative adversarial networks with denoising feature matching. 2017.

[28] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[29] Y. Wu, Y. Burda, R. Salakhutdinov, and R. Grosse. On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273*, 2016.

[30] Z. Yang, W. Chen, F. Wang, and B. Xu. Improving neural machine translation with conditional sequence generative adversarial nets. *arXiv preprint arXiv:1703.04887*, 2017.

[31] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[32] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473*, 2016.

A Proof of Proposition 1

Proof. Since $\mathcal{X}$ is a compact space, by Theorem 5.10 of [25], part (iii), we know that there is an optimal f^*. By Theorem 5.10 of [25], part (ii) we know that if π is an optimal coupling,

$$\mathbb{P}_{(x,y)\sim\pi}\left[f^*(y) - f^*(x) = \|y - x\|\right] = 1$$

Let (x, y) be such that $f^*(y) - f^*(x) = \|y - x\|$. We can safely assume that $x \neq y$ as well, since this happens under π with probability 1. Let $\psi(t) = f^*(x_t) - f^*(x)$. We claim that $\psi(t) = \|x_t - x\| = t\|y - x\|$.

Let $t, t' \in [0, 1]$, then

$$\begin{aligned}
|\psi(t) - \psi(t')| &= \|f^*(x_t) - f^*(x_{t'})\| \\
&\leq \|x_t - x_{t'}\| \\
&= |t - t'|\|x - y\|
\end{aligned}$$

Therefore, ψ is $\|x - y\|$-Lipschitz. This in turn implies

$$\begin{aligned}
\psi(1) - \psi(0) &= \psi(1) - \psi(t) + \psi(t) - \psi(0) \\
&\leq (1 - t)\|x - y\| + \psi(t) - \psi(0) \\
&\leq (1 - t)\|x - y\| + t\|x - y\| \\
&= \|x - y\|
\end{aligned}$$

However, $|\psi(1) - \psi(0)| = |f^*(y) - f^*(x)| = \|y - x\|$ so the inequalities have to actually be equalities. In particular, $\psi(t) - \psi(0) = t\|x - y\|$, and $\psi(0) = f^*(x) - f^*(x) = 0$. Therefore, $\psi(t) = t\|x - y\|$ and we finish our claim.

Let

$$\begin{aligned}
v &= \frac{y - x_t}{\|y - x_t\|} \\
&= \frac{y - ((1 - t)x - ty)}{\|y - ((1 - t)x - ty)\|} \\
&= \frac{(1 - t)(y - x)}{\|(1 - t)\|y - x\|} \\
&= \frac{y - x}{\|y - x\|}
\end{aligned}$$

Now we know that $f^*(x_t) - f^*(x) = \psi(t) = t\|x - y\|$, so $f^*(x_t) = f^*(x) + t\|x - y\|$. Then, we have the partial derivative

$$\begin{aligned}
\frac{\partial}{\partial v} f^*(x_t) &= \lim_{h\to 0} \frac{f^*(x_t + hv) - f^*(x_t)}{h} \\
&= \lim_{h\to 0} \frac{f^*\left(x + t(y - x) + \frac{h}{\|y-x\|}(y - x)\right) - f^*(x_t)}{h} \\
&= \lim_{h\to 0} \frac{f^*\left(x_{t+\frac{h}{\|y-x\|}}\right) - f^*(x_t)}{h} \\
&= \lim_{h\to 0} \frac{f^*(x) + \left(t + \frac{h}{\|y-x\|}\right)\|x - y\| - (f^*(x) + t\|x - y\|)}{h} \\
&= \lim_{h\to 0} \frac{h}{h} \\
&= 1
\end{aligned}$$

If f^* is differentiable at x_t, we know that $\|\nabla f^*(x_t)\| \leq 1$ since it is a 1-Lipschitz function. Therefore, by simple Pythagoras and using that v is a unit vector

$$\begin{aligned}
1 &\leq \|\nabla f^*(x)\|^2 \\
&= \langle v, \nabla f^*(x_t)\rangle^2 + \|\nabla f^*(x_t) - \langle v, \nabla f^*(x_t)\rangle v\|^2 \\
&= |\frac{\partial}{\partial v} f^*(x_t)|^2 + \|\nabla f^*(x_t) - v\frac{\partial}{\partial v} f^*(x_t)\|^2 \\
&= 1 + \|\nabla f^*(x_t) - v\|^2 \\
&\leq 1
\end{aligned}$$

The fact that both extremes of the inequality coincide means that it was all an equality and $1 = 1 + \|\nabla f^*(x_t) - v\|^2$ so $\|\nabla f^*(x_t) - v\| = 0$ and therefore $\nabla f^*(x_t) = v$. This shows that $\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}$.

To conclude, we showed that if (x, y) have the property that $f^*(y) - f^*(x) = \|y - x\|$, then $\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}$. Since this happens with probability 1 under π, we know that

$$\mathbb{P}_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}\right] = 1$$

and we finished the proof.

□

B More details for training random architectures within a set

All models were trained on 32×32 ImageNet for 100K generator iterations using Adam with hyperparameters as recommended in [22] ($\alpha = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999$) for the standard GAN objective and our recommended settings ($\alpha = 0.0001, \beta_1 = 0, \beta_2 = 0.9$) for WGAN-GP. In the discriminator, if we use batch normalization (or layer normalization) we also apply a small weight decay ($\lambda = 10^{-3}$), finding that this helps both algorithms slightly.

Table 5: Outcomes of training 200 random architectures, for different success thresholds. For comparison, our standard DCGAN achieved a score of 7.24.

Min. score	**Only GAN**	**Only WGAN-GP**	**Both succeeded**	**Both failed**
1.0	0	8	192	0
1.5	0	50	150	0
2.0	0	60	140	0
2.5	0	74	125	1
3.0	1	88	110	1
3.5	0	111	86	3
4.0	1	126	67	6
4.5	0	136	55	9
5.0	0	147	42	11
5.5	0	148	32	20
6.0	0	145	21	34
6.5	1	131	11	57
7.0	1	104	5	90
7.5	2	67	3	128
8.0	1	34	0	165
8.5	0	6	0	194
9.0	0	0	0	200

C Experiments with one-sided penalty

We considered a one-sided penalty of the form $\lambda\,\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[\max(0, \|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]$ which would penalize gradients larger than 1 but not gradients smaller than 1, but we observe that the two-sided

version seems to perform slightly better. We sample 174 architectures from the set specified in Table 1 and train each architecture with the one-sided and two-sided penalty terms. The two-sided penalty achieved a higher Inception score in 100 of the trials, compared to 77 for the one-sided penalty. We note that this result is not statistically significant at $p < 0.05$ and further is with respect to only one (somewhat arbitrary) metric and distribution of architectures, and it is entirely possible (likely, in fact) that there are settings where the one-sided penalty performs better, but we leave a thorough comparison for future work. Other training details are the same as in Appendix B.

D Nonsmooth activation functions

The gradient of our objective with respect to the discriminator's parameters contains terms which involve second derivatives of the network's activation functions. In the case of networks with ReLU or other common nonsmooth activation functions, this means the gradient is undefined at some points (albeit a measure zero set) and the gradient penalty objective might not be continuous with respect to the parameters. Gradient descent is not guaranteed to succeed in this setting, but empirically this seems not to be a problem for some common activation functions: in our random architecture and LSUN architecture experiments we find that we are able to train networks with piecewise linear activation functions (ReLU, leaky ReLU) as well as smooth activation functions. We do note that we were unable to train networks with ELU activations, whose derivative is continuous but not smooth. Replacing ELU with a very similar nonlinearity which is smooth ($\frac{\text{softplus}(2x+2)}{2} - 1$) fixed the issue.

E Hyperparameters used for LSUN robustness experiments

For each method we used the hyperparameters recommended in that method's paper. For LSGAN, we additionally searched over learning rate (because the paper did not make a specific recommendation).

- WGAN with gradient penalty: Adam ($\alpha = .0001, \beta_1 = .5, \beta_2 = .9$)
- WGAN with weight clipping: RMSProp ($\alpha = .00005$)
- DCGAN: Adam ($\alpha = .0002, \beta_1 = .5$)
- LSGAN: RMSProp ($\alpha = .0001$) [chosen by search over $\alpha = .001, .0002, .0001$]

F CIFAR-10 ResNet architecture

The generator and critic are residual networks; we use pre-activation residual blocks with two 3×3 convolutional layers each and ReLU nonlinearity. Some residual blocks perform downsampling (in the critic) using mean pooling after the second convolution, or nearest-neighbor upsampling (in the generator) before the second convolution. We use batch normalization in the generator but not the critic. We optimize using Adam with learning rate 2×10^{-4}, decayed linearly to 0 over 100K generator iterations, and batch size 64.

For further architectural details, please refer to our open-source implementation.

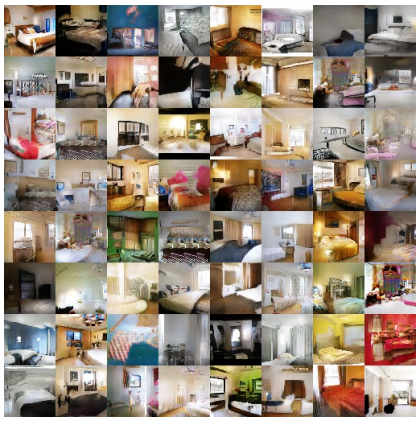
Generator $G(z)$			
	Kernel size	Resample	Output shape
z	-	-	128
Linear	-	-	$128 \times 4 \times 4$
Residual block	[3×3] × 2	Up	$128 \times 8 \times 8$
Residual block	[3×3] × 2	Up	$128 \times 16 \times 16$
Residual block	[3×3] × 2	Up	$128 \times 32 \times 32$
Conv, tanh	3×3	-	$3 \times 32 \times 32$

Critic $D(x)$			
	Kernel size	Resample	Output shape
Residual block	[3×3] × 2	Down	$128 \times 16 \times 16$
Residual block	[3×3] × 2	Down	$128 \times 8 \times 8$
Residual block	[3×3] × 2	-	$128 \times 8 \times 8$
Residual block	[3×3] × 2	-	$128 \times 8 \times 8$
ReLU, mean pool	-	-	128
Linear	-	-	1

G CIFAR-10 ResNet samples

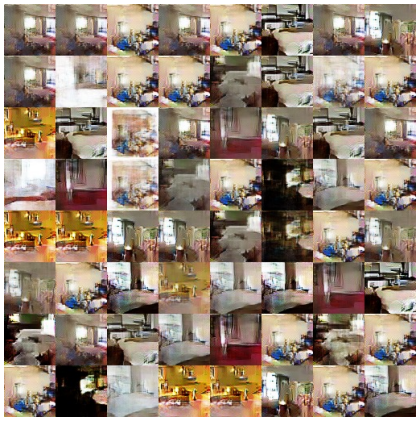
Figure 6: *(left)* CIFAR-10 samples generated by our unsupervised model. *(right)* Conditional CIFAR-10 samples, from adding AC-GAN conditioning to our unconditional model. Samples from the same class are displayed in the same column.

H More LSUN samples

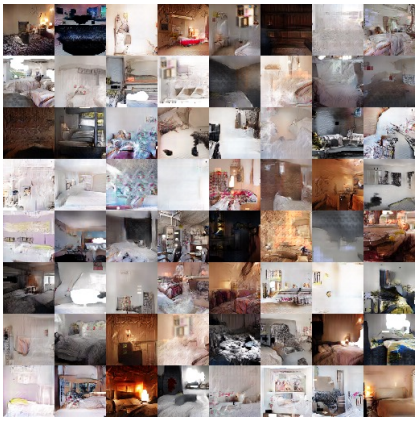


Method: DCGAN
G: DCGAN, D: DCGAN

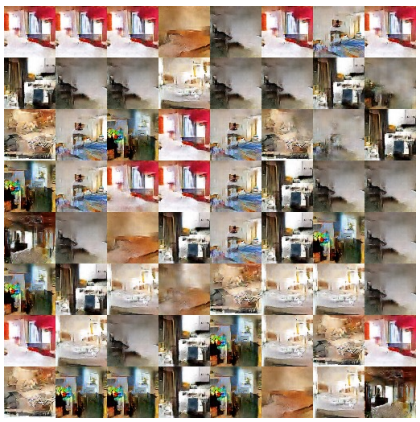
Method: DCGAN
G: No BN and const. filter count



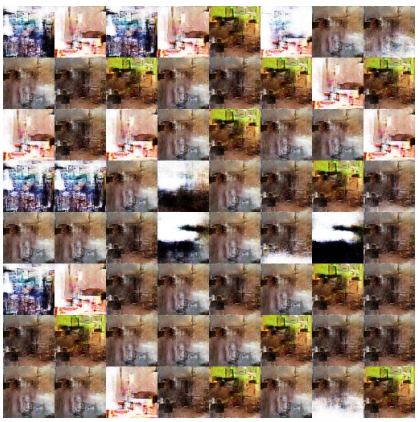
Method: DCGAN
G: 4-layer 512-dim ReLU MLP



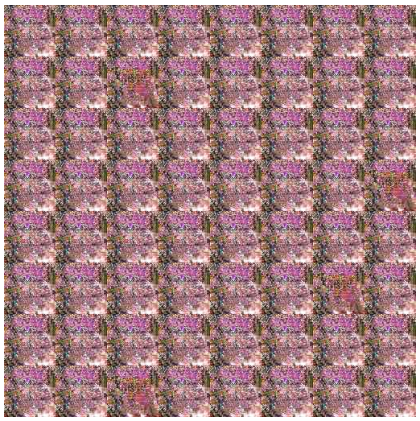
Method: DCGAN
No normalization in either G or D



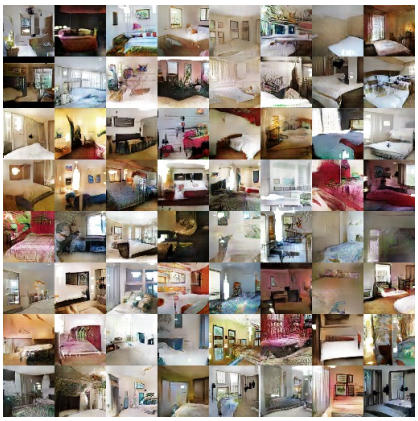
Method: DCGAN
Gated multiplicative nonlinearities



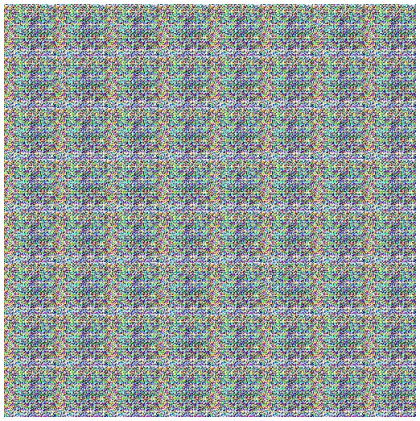
Method: DCGAN
$\tanh$ nonlinearities



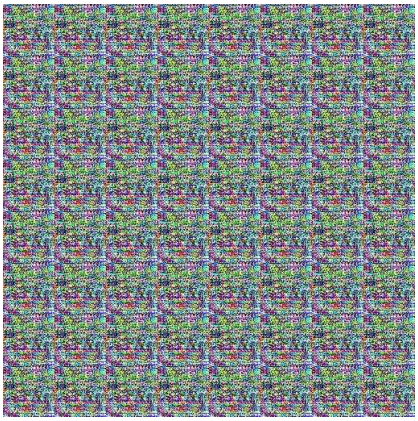
Method: DCGAN
101-layer ResNet G and D



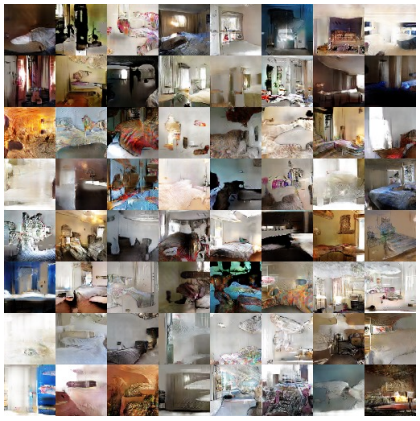
Method: LSGAN
G: DCGAN, D: DCGAN



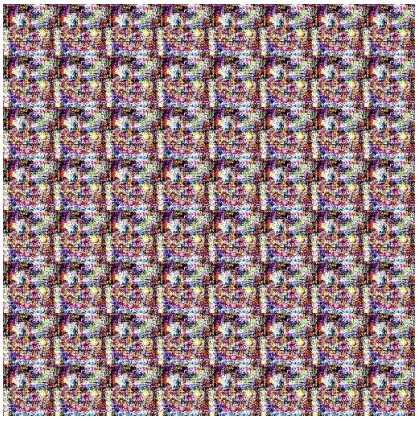
Method: LSGAN
G: No BN and const. filter count



Method: LSGAN
G: 4-layer 512-dim ReLU MLP

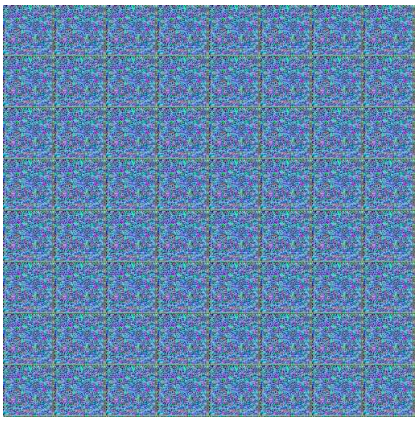


Method: LSGAN
No normalization in either G or D

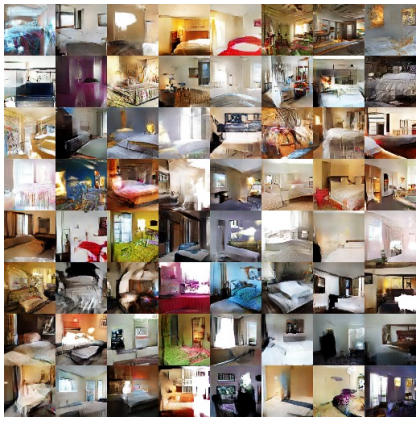


Method: LSGAN
Gated multiplicative nonlinearities

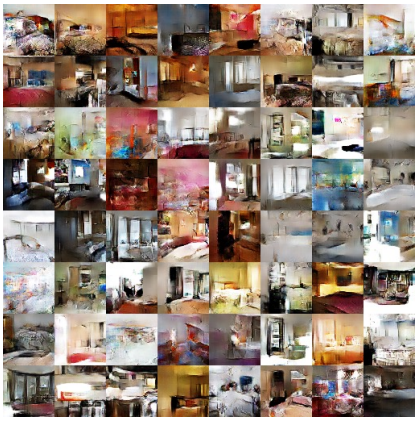
Method: LSGAN
$\tanh$ nonlinearities



Method: LSGAN
101-layer ResNet G and D

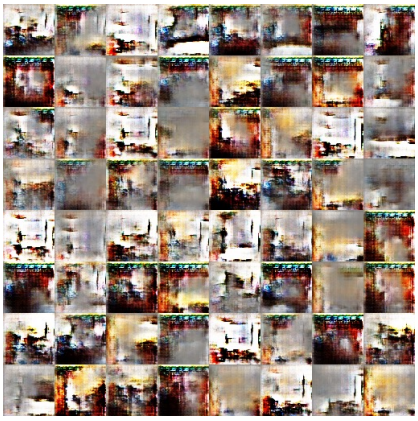


Method: WGAN with clipping
G: DCGAN, D: DCGAN



Method: WGAN with clipping
G: No BN and const. filter count

Method: WGAN with clipping
G: 4-layer 512-dim ReLU MLP

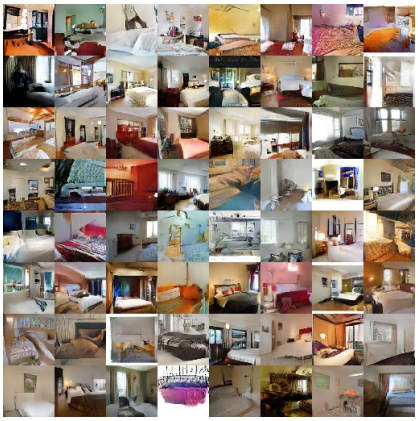


Method: WGAN with clipping
No normalization in either G or D

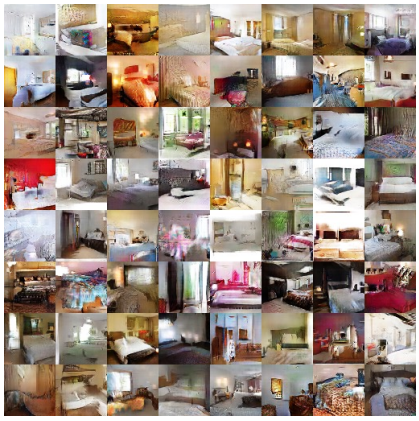
Method: WGAN with clipping
Gated multiplicative nonlinearities

Method: WGAN with clipping
$\tanh$ nonlinearities

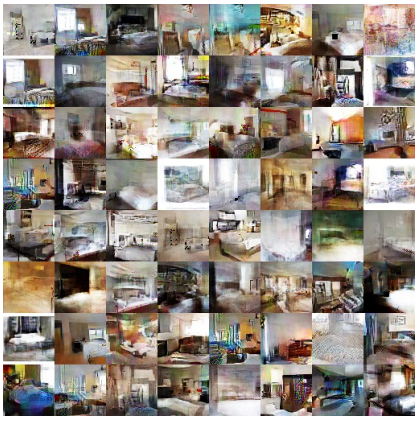
Method: WGAN with clipping
101-layer ResNet G and D



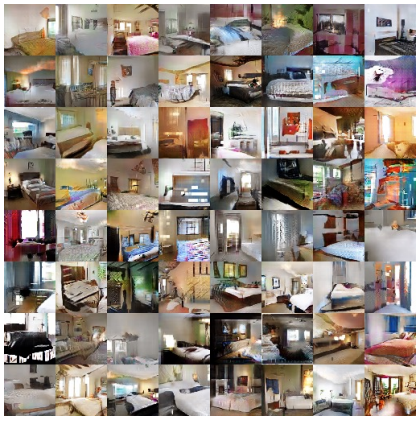
Method: WGAN-GP (ours)
G: DCGAN, D: DCGAN



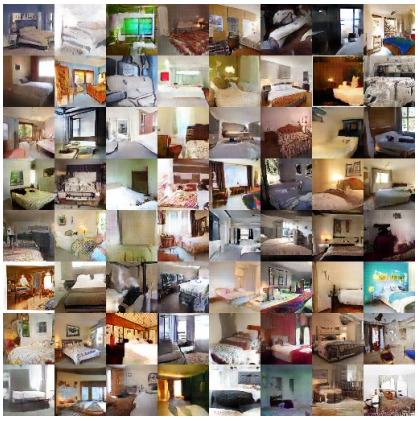
Method: WGAN-GP (ours)
G: No BN and const. filter count



Method: WGAN-GP (ours)
G: 4-layer 512-dim ReLU MLP

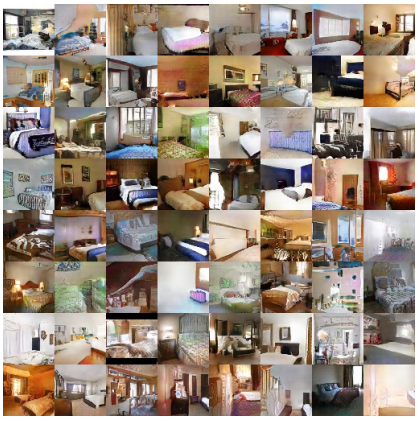


Method: WGAN-GP (ours)
No normalization in either G or D



Method: WGAN-GP (ours)
Gated multiplicative nonlinearities

Method: WGAN-GP (ours)
$\tanh$ nonlinearities



Method: WGAN-GP (ours)
101-layer ResNet G and D

MoCoGAN: Decomposing Motion and Content for Video Generation

Sergey Tulyakov,
Snap Research
stulyakov@snap.com

Ming-Yu Liu, Xiaodong Yang, Jan Kautz
NVIDIA
{mingyul,xiaodongy,jkautz}@nvidia.com

Abstract

Visual signals in a video can be divided into content and motion. While content specifies which objects are in the video, motion describes their dynamics. Based on this prior, we propose the Motion and Content decomposed Generative Adversarial Network (MoCoGAN) framework for video generation. The proposed framework generates a video by mapping a sequence of random vectors to a sequence of video frames. Each random vector consists of a content part and a motion part. While the content part is kept fixed, the motion part is realized as a stochastic process. To learn motion and content decomposition in an unsupervised manner, we introduce a novel adversarial learning scheme utilizing both image and video discriminators. Extensive experimental results on several challenging datasets with qualitative and quantitative comparison to the state-of-the-art approaches, verify effectiveness of the proposed framework. In addition, we show that MoCoGAN allows one to generate videos with same content but different motion as well as videos with different content and same motion.

1. Introduction

Deep generative models have recently received an increasing amount of attention, not only because they provide a means to learn deep feature representations in an unsupervised manner that can potentially leverage all the unlabeled images on the Internet for training, but also because they can be used to generate novel images necessary for various vision applications. As steady progress toward better image generation is made, it is also important to study the video generation problem. However, the extension from generating images to generating videos turns out to be a highly challenging task, although the generated data has just one more dimension – the time dimension.

We argue video generation is much harder for the following reasons. First, since a video is a spatio-temporal recording of visual information of objects performing various actions, a generative model needs to learn the plausible physical motion models of objects in addition to learning their appearance models. If the learned object motion model is incorrect, the generated video may contain objects performing physically impossible motion. Second, the time dimension brings in a huge amount of variations. Consider the amount of speed variations that a person can have when performing a squat movement. Each speed pattern results in a different video, although the appearances of the human in the videos are the same. Third, as human beings have evolved to be sensitive to motion, motion artifacts are particularly perceptible.

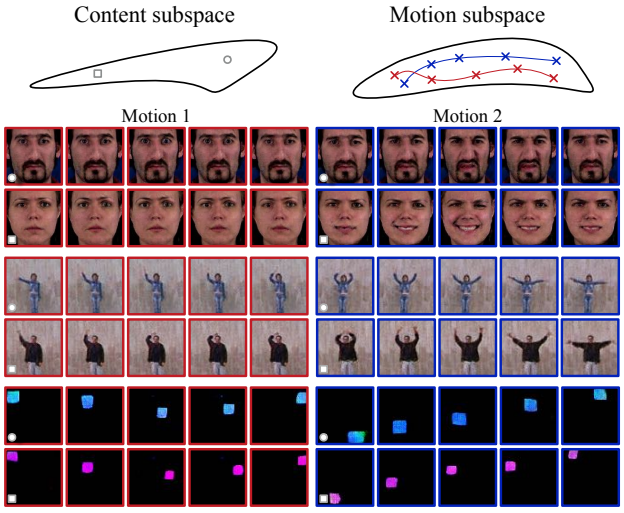


Figure 1: MoCoGAN adopts a motion and content decomposed representation for video generation. It uses an image latent space (each latent code represents an image) and divides the latent space into content and motion subspaces. By sampling a point in the content subspace and sampling different trajectories in the motion subspace, it generates videos of the same object performing different motion. By sampling different points in the content subspace and the same motion trajectory in the motion subspace, it generates videos of different objects performing the same motion.

Recently, a few attempts to approach the video generation problem were made through generative adversarial networks (GANs) [12]. Vondrick *et al*. [40] hypothesize that a video clip is a point in a latent space and proposed a VGAN framework for learning a mapping from the latent space to

video clips. A similar approach was proposed in the TGAN work [30]. We argue that assuming a video clip is a point in the latent space unnecessarily increases the complexity of the problem, because videos of the same action with different execution speed are represented by different points in the latent space. Moreover, this assumption forces every generated video clip to have the same length, while the length of real-world video clips varies. An alternative (and likely more intuitive and efficient) approach would assume a latent space of images and consider that a video clip is generated by traversing the points in the latent space. Video clips of different lengths correspond to latent space trajectories of different lengths.

In addition, as videos are about objects (content) performing actions (motion), the latent space of images should be further decomposed into two subspaces, where the deviation of a point in the first subspace (the content subspace) leads content changes in a video clip and the deviation in the second subspace (the motion subspace) results in temporal motions. Through this modeling, videos of an action with different execution speeds will only result in different traversal speeds of a trajectory in the motion space. Decomposing motion and content allows a more controlled video generation process. By changing the content representation while fixing the motion trajectory, we have videos of different objects performing the same motion. By changing motion trajectories while fixing the content representation, we have videos of the same object performing different motion as illustrated in Fig. 1.

In this paper, we propose the Motion and Content decomposed Generative Adversarial Network (MoCoGAN) framework for video generation. It generates a video clip by sequentially generating video frames. At each time step, an image generative network maps a random vector to an image. The random vector consists of two parts where the first is sampled from a content subspace and the second is sampled from a motion subspace. Since content in a short video clip usually remains the same, we model the content space using a Gaussian distribution and use the same realization to generate each frame in a video clip. On the other hand, sampling from the motion space is achieved through a recurrent neural network where the network parameters are learned during training. Despite lacking supervision regarding the decomposition of motion and content in natural videos, we show that MoCoGAN can learn to disentangle these two factors through a novel adversarial training scheme. Through extensive qualitative and quantitative experimental validations with comparison to the state-of-the-art approaches including VGAN [40] and TGAN [30], as well as the future frame prediction methods including Conditional-VGAN (C-VGAN) [40] and Motion and Content Network (MCNET) [39], we verify the effectiveness of MoCoGAN.

1.1. Related Work

Video generation is not a new problem. Due to limitations in computation, data, and modeling tools, early video generation works focused on generating dynamic texture patterns [34, 41, 9]. In the recent years, with the availability of GPUs, Internet videos, and deep neural networks, we are now better positioned to tackle this intriguing problem.

Various deep generative models were recently proposed for image generation including GANs [12], variational autoencoders (VAEs) [20, 28, 36], and PixelCNNs [38]. In this paper, we propose the MoCoGAN framework for video generation, which is based on GANs.

Multiple GAN-based image generation frameworks were proposed. Denton *et al*. [8] showed a Laplacian pyramid implementation. Radford *et al*. [27] used a deeper convolution network. Zhang *et al*. [43] stacked two generative networks to progressively render realistic images. Coupled GANs [22] learned to generate corresponding images in different domains, later extended to translate an image from one domain to a different domain in an unsupervised fashion [21]. InfoGAN [5] learned a more interpretable latent representation. Salimans *et al*. [31] proposed several GAN training tricks. The WGAN [3] and LSGAN [23] frameworks adopted alternative distribution distance metrics for more stable adversarial training. Roth *et al*. [29] proposed a special gradient penalty to further stabilize training. Karras *et al*. [18] used progressive growing of the discriminator and the generator to generate high resolution images. The proposed MoCoGAN framework generates a video clip by sequentially generating images using an image generator. The framework can easily leverage advances in image generation in the GAN framework for improving the quality of the generated videos. As discussed in Section 1, [40, 30] extended the GAN framework to the video generation problem by assuming a latent space of video clips where all the clips have the same length.

Recurrent neural networks for image generation were previously explored in [14, 16]. Specifically, some works used recurrent mechanisms to iteratively refine a generated image. Our work is different to [14, 16] in that we use the recurrent mechanism to generate motion embeddings of video frames in a video clip. The image generation is achieved through a convolutional neural network.

The future frame prediction problem studied in [33, 26, 24, 17, 10, 37, 42, 39, 7] is different to the video generation problem. In future frame prediction, the goal is to predict future frames in a video given the observed frames in the video. Previous works on future frame prediction can be roughly divided into two categories where one focuses on generating raw pixel values in future frames based on the observed ones [33, 26, 24, 17, 42, 39], while the other focuses on generating transformations for reshuffling the pixels in the previous frames to construct fu-

ture frames [10, 37]. The availability of previous frames makes future frame prediction a conditional image generation problem, which is different to the video generation problem where the input to the generative network is only a vector drawn from a latent space. We note that [39] used a convolutional LSTM [15] encoder to encode temporal differences between consecutive previous frames for extracting motion information and a convolutional encoder to extract content information from the current image. The concatenation of the motion and content information was then fed to a decoder to predict future frames.

1.2. Contributions

Our contributions are as follows:

1. We propose a novel GAN framework for unconditional video generation, mapping noise vectors to videos.
2. We show the proposed framework provides a means to control content and motion in video generation, which is absent in the existing video generation frameworks.
3. We conduct extensive experimental validation on benchmark datasets with both quantitative and subjective comparison to the state-of-the-art video generation algorithms including VGAN[40] and TGAN [30] to verify the effectiveness of the proposed algorithm.

2. Generative Adversarial Networks

GANs [12] consist of a generator and a discriminator. The objective of the generator is to generate images resembling real images, while the objective of the discriminator is to distinguish real images from generated ones.

Let $\mathbf{x}$ be a real image drawn from an image distribution, p_X, and $\mathbf{z}$ be a random vector in $Z_{\mathrm{I}} \equiv \mathbb{R}^d$. Let G_{I} and D_{I} be the image generator and the image discriminator. The generator takes $\mathbf{z}$ as input and outputs an image, $\tilde{\mathbf{x}} = G_{\mathrm{I}}(\mathbf{z})$, that has the same support as $\mathbf{x}$. We denote the distribution of $G_{\mathrm{I}}(\mathbf{z})$ as $p_{G_{\mathrm{I}}}$. The discriminator estimates the probability that an input image is drawn from p_X. Ideally, $D_{\mathrm{I}}(\mathbf{x}) = 1$ if $\mathbf{x} \sim p_X$ and $D_{\mathrm{I}}(\tilde{\mathbf{x}}) = 0$ if $\tilde{\mathbf{x}} \sim p_{G_{\mathrm{I}}}$. Training of G_{I} and D_{I} is achieved via solving a minimax problem given by

$$\max_{G_{\mathrm{I}}} \min_{D_{\mathrm{I}}} \mathcal{F}_{\mathrm{I}}(D_{\mathrm{I}}, G_{\mathrm{I}}) \tag{1}$$

where the functional $\mathcal{F}_{\mathrm{I}}$ is given by

$$\begin{aligned} \mathcal{F}_{\mathrm{I}}(D_{\mathrm{I}}, G_{\mathrm{I}}) \;=\; & \mathbb{E}_{\mathbf{x}\sim p_X}[-\log D_{\mathrm{I}}(\mathbf{x})] + \\ & \mathbb{E}_{\mathbf{z}\sim p_{Z_{\mathrm{I}}}}[-\log(1 - D_{\mathrm{I}}(G_{\mathrm{I}}(\mathbf{z})))]. \end{aligned} \tag{2}$$

In practice, (1) is solved by alternating gradient update.

Goodfellow *et al*. [12] show that, given enough capacity to D_{I} and G_{I} and sufficient training iterations, the distribution $p_{G_{\mathrm{I}}}$ converges to p_X. As a result, from a random vector input $\mathbf{z}$, the network G_{I} can synthesize an image that resembles one drawn from the true distribution, p_X.

2.1. Extension to Fixed-length Video Generation

Recently, [40] extended the GAN framework to video generation by proposing a Video GAN (VGAN) framework. Let $\mathbf{v}^{\mathrm{L}} = [\mathbf{x}^{(1)}, ..., \mathbf{x}^{(\mathrm{L})}]$ be a video clip with L frames. The video generation in VGAN is achieved by replacing the vanilla CNN-based image generator and discriminator, G_{I} and D_{I}, with a spatio-temporal CNN-based video generator and discriminator, $G_{\mathrm{V^L}}$ and $D_{\mathrm{V^L}}$. The video generator $G_{\mathrm{V^L}}$ maps a random vector $\mathbf{z} \in Z_{\mathrm{V^L}} \equiv \mathbb{R}^d$ to a fixed-length video clip, $\tilde{\mathbf{v}}^{\mathrm{L}} = [\tilde{\mathbf{x}}^{(1)}, ..., \tilde{\mathbf{x}}^{(\mathrm{L})}] = G_{\mathrm{V^L}}(\mathbf{z})$ and the video discriminator $D_{\mathrm{V^L}}$ differentiates real video clips from generated ones. Ideally, $D_{\mathrm{V^L}}(\mathbf{v}^{\mathrm{L}}) = 1$ if $\mathbf{v}^{\mathrm{L}}$ is sampled from $p_{V^{\mathrm{L}}}$ and $D_{\mathrm{V^L}}(\tilde{\mathbf{v}}^{\mathrm{L}}) = 0$ if $\tilde{\mathbf{v}}^{\mathrm{L}}$ is sampled from the video generator distribution $p_{G_{\mathrm{V^L}}}$. The TGAN framework [30] also maps a random vector to a fixed length clip. The difference is that TGAN maps the random vector, representing a fixed-length video, to a fixed number of random vectors, representing individual frames in the video clip and uses an image generator for generation. Instead of using the vanilla GAN framework for minimizing the Jensen-Shannon divergence, the TGAN training is based on the WGAN framework [3] and minimizes the earth mover distance.

3. Motion and Content Decomposed GAN

In MoCoGAN, we assume a latent space of images $Z_{\mathrm{I}} \equiv \mathbb{R}^d$ where each point $\mathbf{z} \in Z_{\mathrm{I}}$ represents an image, and a video of K frames is represented by a path of length K in the latent space, $[\mathbf{z}^{(1)}, ..., \mathbf{z}^{(\mathrm{K})}]$. By adopting this formulation, videos of different lengths can be generated by paths of different lengths. Moreover, videos of the same action executed with different speeds can be generated by traversing the same path in the latent space with different speeds.

We further assume Z_{I} is decomposed into the content Z_{C} and motion Z_{M} subspaces: $Z_{\mathrm{I}} = Z_{\mathrm{C}} \times Z_{\mathrm{M}}$ where $Z_{\mathrm{C}} = \mathbb{R}^{d_{\mathrm{C}}}$, $Z_{\mathrm{M}} = \mathbb{R}^{d_{\mathrm{M}}}$, and $d = d_{\mathrm{C}} + d_{\mathrm{M}}$. The content subspace models motion-independent appearance in videos, while the motion subspace models motion-dependent appearance in videos. For example, in a video of a person smiling, content represents the identity of the person, while motion represents the changes of facial muscle configurations of the person. A pair of the person's identity and the facial muscle configuration represents a face image of the person. A sequence of these pairs represents a video clip of the person smiling. By swapping the look of the person with the look of another person, a video of a different person smiling is represented.

We model the content subspace using a Gaussian distribution: $\mathbf{z}_{\mathrm{C}} \sim p_{Z_{\mathrm{C}}} \equiv \mathcal{N}(\mathbf{z}|0, I_{d_{\mathrm{C}}})$ where $I_{d_{\mathrm{C}}}$ is an identity matrix of size $d_{\mathrm{C}} \times d_{\mathrm{C}}$. Based on the observation that the content remains largely the same in a short video clip, we use the same realization, $\mathbf{z}_{\mathrm{C}}$, for generating different frames in a video clip. Motion in the video clip is modeled by a

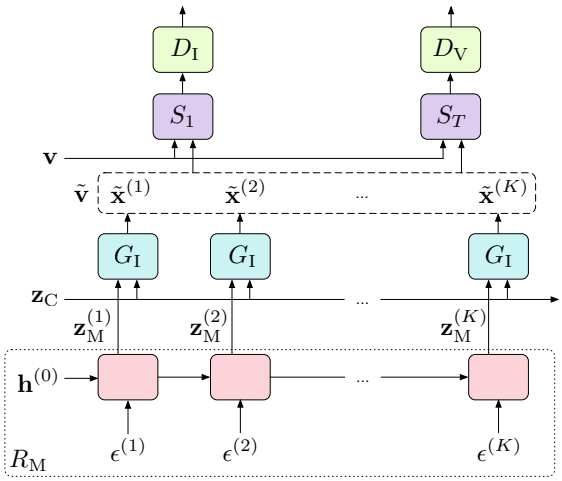


Figure 2: The MoCoGAN framework for video generation. For a video, the content vector, $\mathbf{z}_{\mathrm{C}}$, is sampled once and fixed. Then, a series of random variables $[\epsilon^{(1)}, ..., \epsilon^{(K)}]$ is sampled and mapped to a series of motion codes $[\mathbf{z}_{\mathrm{M}}^{(1)}, ..., \mathbf{z}_{\mathrm{M}}^{(K)}]$ via the recurrent neural network R_{M}. A generator G_{I} produces a frame, $\tilde{\mathbf{x}}^{(k)}$, using the content and the motion vectors $\{\mathbf{z}_{\mathrm{C}}, \mathbf{z}_{\mathrm{M}}^{(k)}\}$. The discriminators, D_{I} and D_{V}, are trained on real and fake images and videos, respectively, sampled from the training set $\mathbf{v}$ and the generated set $\tilde{\mathbf{v}}$. The function S_1 samples a single frame from a video, S_{T} samples T consequtive frames.

path in the motion subspace Z_{M}. The sequence of vectors for generating a video is represented by

$$[\mathbf{z}^{(1)}, ..., \mathbf{z}^{(K)}] = \left[\begin{bmatrix} \mathbf{z}_{\mathrm{C}} \\ \mathbf{z}_{\mathrm{M}}^{(1)} \end{bmatrix}, ..., \begin{bmatrix} \mathbf{z}_{\mathrm{C}} \\ \mathbf{z}_{\mathrm{M}}^{(K)} \end{bmatrix}\right] \tag{3}$$

where $\mathbf{z}_{\mathrm{C}} \in Z_{\mathrm{C}}$ and $\mathbf{z}_{\mathrm{M}}^{(k)} \in Z_{\mathrm{M}}$ for all k's. Since not all paths in Z_{M} correspond to physically plausible motion, we need to learn to generate valid paths. We model the path generation process using a recurrent neural network.

Let R_{M} to be a recurrent neural network. At each time step, it takes a vector sampled from a Gaussian distribution as input: $\epsilon^{(k)} \sim p_{\mathrm{E}} \equiv \mathcal{N}(\epsilon|0, I_{d_{\mathrm{E}}})$ and outputs a vector in Z_{M}, which is used as the motion representation. Let $R_{\mathrm{M}}(k)$ be the output of the recurrent neural network at time k. Then, $\mathbf{z}_{\mathrm{M}}^{(k)} = R_{\mathrm{M}}(k)$. Intuitively, the function of the recurrent neural network is to map a sequence of independent and identically distributed (i.i.d.) random variables $[\epsilon^{(1)}, ..., \epsilon^{(K)}]$ to a sequence of correlated random variables $[R_{\mathrm{M}}(1), ..., R_{\mathrm{M}}(K)]$ representing the dynamics in a video. Injecting noise at every iteration models uncertainty of the future motion at each timestep. We implement R_{M} using a one-layer GRU network [6].

Networks. MoCoGAN consists of 4 sub-networks, which are the recurrent neural network, R_{M}, the image generator, G_{I}, the image discriminator, D_{I}, and the video discriminator, D_{V}. The image generator generates a video clip by sequentially mapping vectors in Z_{I} to images, from a sequence of vectors $[\begin{bmatrix} \mathbf{z}_{\mathrm{C}} \\ \mathbf{z}_{\mathrm{M}}^{(1)} \end{bmatrix}, ..., \begin{bmatrix} \mathbf{z}_{\mathrm{C}} \\ \mathbf{z}_{\mathrm{M}}^{(K)} \end{bmatrix}]$ to a sequence of images, $\tilde{\mathbf{v}} = [\tilde{\mathbf{x}}^{(1)}, ..., \tilde{\mathbf{x}}^{(K)}]$, where $\tilde{\mathbf{x}}^{(k)} = G_{\mathrm{I}}(\begin{bmatrix} \mathbf{z}_{\mathrm{C}} \\ \mathbf{z}_{\mathrm{M}}^{(k)} \end{bmatrix})$ and $\mathbf{z}_{\mathrm{M}}^{(k)}$'s are from the recurrent neural network, R_{M}. We note that the video length K can vary for each video generation.

Both D_{I} and D_{V} play the judge role, providing criticisms to G_{I} and R_{M}. The image discriminator D_{I} is specialized in criticizing G_{I} based on individual images. It is trained to determine if a frame is sampled from a real video clip, $\mathbf{v}$, or from $\tilde{\mathbf{v}}$. On the other hand, D_{V} provides criticisms to G_{I} based on the generated video clip. D_{V} takes a fixed length video clip, say T frames, and decides if a video clip was sampled from a real video or from $\tilde{\mathrm{v}}$. Different from D_{I}, which is based on vanilla CNN architecture, D_{V} is based on a spatio-temporal CNN architecture. We note that the clip length T is a hyperparameter, which is set to 16 throughout our experiments. We also note that T can be smaller than the generated video length K. A video of length K can be divided into $K - \mathrm{T} + 1$ clips in a sliding-window fashion, and each of the clips can be fed into D_{V}.

The video discriminator D_{V} also evaluates the generated motion. Since G_{I} has no concept of motion, the criticisms on the motion part go directly to the recurrent neural network, R_{M}. In order to generate a video with realistic dynamics that fools D_{V}, R_{M} has to learn to generate a sequence of motion codes $[\mathbf{z}_{\mathrm{M}}^{(1)}, ..., \mathbf{z}_{\mathrm{M}}^{(K)}]$ from a sequence of i.i.d. noise inputs $[\epsilon^{(1)}, ..., \epsilon^{(K)}]$ in a way such that G_{I} can map $\mathbf{z}^{(k)} = [\mathbf{z}_{\mathrm{C}}, \mathbf{z}_{\mathrm{M}}^{(k)}]$ to consecutive frames in a video.

Ideally, D_{V} alone should be sufficient for training G_{I} and R_{M}, because D_{V} provides feedback on both static image appearance and video dynamics. However, we found that using D_{I} significantly improves the convergence of the adversarial training. This is because training D_{I} is simpler, as it only needs to focus on static appearances. Fig. 2 shows visual representation of the MoCoGAN framework.

Learning. Let p_{V} be the distribution of video clips of variable lengths. Let κ be a discrete random variable denoting the length of a video clip sampled from p_{V}. (In practice, we can estimate the distribution of κ, termed p_{K}, by computing a histogram of video clip length from training data). To generate a video, we first sample a content vector, $\mathbf{z}_{\mathrm{C}}$, and a length, κ. We then run R_{M} for κ steps and, at each time step, R_{M} takes a random variable ϵ as the input. A generated video is then given by

$$\tilde{\mathbf{v}} = \left[G_{\mathrm{I}}(\begin{bmatrix} z_{\mathrm{C}} \\ R_{\mathrm{M}}(1) \end{bmatrix}), ..., G_{\mathrm{I}}(\begin{bmatrix} z_{\mathrm{C}} \\ R_{\mathrm{M}}(\kappa) \end{bmatrix})\right]. \tag{4}$$

Recall that our D_{I} and D_{V} take one frame and T consecutive frames in a video as input, respectively. In order

to represent these sampling mechanisms, we introduce two random access functions S_1 and S_T. The function S_1 takes a video clip (either $\mathbf{v} \sim p_V$ or $\tilde{\mathbf{v}} \sim p_{\tilde{V}}$) and outputs a random frame from the clip, while the function S_T takes a video clip and randomly returns T consecutive frames from the clip. With this notation, the MoCoGAN learning problem is

$$\max_{G_I, R_M} \min_{D_I, D_V} \mathcal{F}_V(D_I, D_V, G_I, R_M) \tag{5}$$

where the objective function $\mathcal{F}_V(D_I, D_V, G_I, R_M)$ is

$$\mathbb{E}_{\mathbf{v}}[-\log D_I(S_1(\mathbf{v}))] + \mathbb{E}_{\tilde{\mathbf{v}}}[-\log(1 - D_I(S_1(\tilde{\mathbf{v}})))] + \mathbb{E}_{\mathbf{v}}[-\log D_V(S_T(\mathbf{v}))] + \mathbb{E}_{\tilde{\mathbf{v}}}[-\log(1 - D_V(S_T(\tilde{\mathbf{v}})))], \tag{6}$$

where $\mathbb{E}_{\mathbf{v}}$ is a shorthand for $\mathbb{E}_{\mathbf{v}\sim p_V}$, and $\mathbb{E}_{\tilde{\mathbf{v}}}$ for $\mathbb{E}_{\tilde{\mathbf{v}}\sim p_{\tilde{V}}}$. In (6), the first and second terms encourage D_I to output 1 for a video frame from a real video clip $\mathbf{v}$ and 0 for a video frame from a generated one $\tilde{\mathbf{v}}$. Similarly, the third and fourth terms encourage D_V to output 1 for T consecutive frames in a real video clip $\mathbf{v}$ and 0 for T consecutive frames in a generated one $\tilde{\mathbf{v}}$. The second and fourth terms encourage the image generator and the recurrent neural network to produce realistic images and video sequences of T-consecutive frames, such that no discriminator can distinguish them from real images and videos.

We train MoCoGAN using the alternating gradient update algorithm as in [11]. Specifically, in one step, we update D_I and D_V while fixing G_I and R_M. In the alternating step, we update G_I and R_M while fixing D_I and D_V.

3.1. Categorical Dynamics

Dynamics in videos are often categorical (e.g., discrete action categories: walking, running, jumping, etc.). To model this categorical signal, we augment the input to R_M with a categorical random variable, $\mathbf{z}_A$, where each realization is a one-hot vector. We keep the realization fixed since the action category in a short video remains the same. The input to R_M is then given by

$$\left[\begin{bmatrix} \mathbf{z}_A \\ \epsilon^{(1)} \end{bmatrix}, ..., \begin{bmatrix} \mathbf{z}_A \\ \epsilon^{(K)} \end{bmatrix}\right], \tag{7}$$

To relate $\mathbf{z}_A$ to the true action category, we adopt the InfoGAN learning [5] and augment the objective function in (6) to $\mathcal{F}_V(D_I, D_V, G_I, R_M) + \lambda L_I(G_I, Q)$ where L_I is a lower bound of the mutual information between the generated video clip and $\mathbf{z}_A$, λ is a hyperparameter, and the auxiliary distribution Q (which approximates the distribution of the action category variable conditioning on the video clip) is implemented by adding a softmax layer to the last feature layer of D_V. We use $\lambda = 1$. We note that when the labeled training data are available, we can train Q to output the category label for a real input video clip to further improve the performance [25].

4. Experiments

We conducted extensive experimental validation to evaluate MoCoGAN. In addition to comparing to VGAN [40] and TGAN [30], both quantitatively and qualitatively, we evaluated the ability of MoCoGAN to generate 1) videos of the same object performing different motions by using a fixed content vector and varying motion trajectories and 2) videos of different objects performing the same motion by using different content vectors and the same motion trajectory. We then compared a variant of the MoCoGAN framework with state-of-the-art next frame prediction methods: VGAN and MCNET [39]. Evaluating generative models is known to be a challenging task [35]. Hence, we report experimental results on several datasets, where we can obtain reliable performance metrics:

- **Shape motion.** The dataset contained two types of shapes (circles and squares) with varying sizes and colors, performing two types of motion: one moving from left to right, and the other moving from top to bottom. The motion trajectories were sampled from Bezier curves. There were $4,000$ videos in the dataset; the image resolution was 64×64 and video length was 16.
- **Facial expression.** We used the MUG Facial Expression Database [1] for this experiment. The dataset consisted of 86 subjects. Each video consisted of 50 to 160 frames. We cropped the face regions and scaled to 96×96. We discarded videos containing fewer than 64 frames and used only the sequences representing one of the six facial expressions: anger, fear, disgust, happiness, sadness, and surprise. In total, we trained on $1,254$ videos.
- **Tai-Chi.** We downloaded $4,500$ Tai Chi video clips from YouTube. For each clip, we applied a human pose estimator [4] and cropped the clip so that the performer is in the center. Videos were scaled to 64×64 pixels.
- **Human actions.** We used the Weizmann Action database [13], containing 81 videos of 9 people performing 9 actions, including jumping-jack and waving-hands. We scaled the videos to 96×96. Due to the small size, we did not conduct a quantitative evaluation using the dataset. Instead, we provide visual results in Fig. 1 and Fig. 4a.
- **UCF101** [32]**.** The database is commonly used for video action recognition. It includes $13,220$ videos of 101 different action categories. Similarly to the TGAN work [30], we scaled each frame to 85×64 and cropped the central 64×64 regions for learning.

Implementation. The details of the network designs are given in the supplementary materials. We used ADAM [19] for training, with a learning rate of 0.0002 and momentums of 0.5 and 0.999. Our code will be made public.

(a) Generated by MoCoGAN (b) Generated by VGAN [40] (c) Generated by TGAN [30]

Figure 3: Generated video clips used in the user study. The video clips were randomly selected. The figure is best viewed via the Acrobat Reader on a desktop. Click each image to play the video clip.

Table 1: Video generation content consistency comparison. A smaller ACD means the generated frames in a video are perceptually more similar. We also compute the ACD for the training set, which is the reference.

ACD	Shape Motion	Facial Expressions
Reference	0	0.116
VGAN [40]	5.02	0.322
TGAN [30]	2.08	0.305
MoCoGAN	**1.79**	**0.201**

Table 2: Inception score for models trained on UCF101. All values except MoCoGAN's are taken from [30].

	VGAN	TGAN	MoCoGAN
UCF101	$8.18 \pm .05$	$11.85 \pm .07$	$\mathbf{12.42 \pm .03}$

Table 3: User preference score on video generation quality.

User preference, %	Facial Exp.	Tai-Chi
MoCoGAN / VGAN	**84.2** / 15.8	**75.4** / 24.6
MoCoGAN / TGAN	**54.7** / 45.3	**68.0** / 32.0

4.1. Video Generation Performance

Quantitative comparison. We compared MoCoGAN to VGAN and TGAN[1] using the shape motion and facial expression datasets. For each dataset, we trained a video generation model and generated 256 videos for evaluation. The VGAN and TGAN implementations can only generate fixed-length videos (32 frames and 16 frames correspondingly). For a fair comparison, we generated 16 frames using MoCoGAN, and selected every second frame from the videos generated by VGAN, such that each video has 16 frames in total.

For quantitative comparison, we measured content consistency of a generated video using the Average Content Distance (ACD) metric. For shape motion, we first computed the average color of the generated shape in each frame. Each frame was then represented by a 3-dimensional vector. The ACD is then given by the average pairwise L2 distance of the per-frame average color vectors. For facial expression videos, we employed OpenFace [2], which outperforms human performance in the face recognition task, for measuring video content consistency. OpenFace produced a feature vector for each frame in a face video. The ACD was then computed using the average pairwise L2 distance of the per-frame feature vectors.

We computed the average ACD scores for the 256 videos generated by the competing algorithms for comparison. The results are given in Table 1. From the table, we found that the content of the videos generated by MoCoGAN was more consistent, especially for the facial expression video generation task: MoCoGAN achieved an ACD score of 0.201, which was almost 40% better than 0.322 of VGAN and 34% better than 0.305 of TGAN. Fig. 3 shows examples of facial expression videos for competing algorithms.

Furthermore, we compared with TGAN and VGAN by training on the UCF101 database and computing the inception score similarly to [30]. Table 2 shows comparison results. In this experiment we used the same MoCoGAN model as in all other experiments. We observed that MoCoGAN was able to learn temporal dynamics better, due to the decomposed representation, as it generated more realistic temporal sequences. We also noted that TGAN reached the inception score of 11.85 with WGAN training procedure and Singular Value Clipping (SVC), while MoCoGAN showed a higher inception score 12.42 without such tricks, supporting that the proposed framework is more stable than and superior to the TGAN approach.

[1]The VGAN and TGAN implementations are provided by their authors.

(a) Samples from the model trained on the Weizmann database.

(b) Examples of changing the motion code while fixing the content code. Every row has fixed content, every column has fixed motion.

(c) Image-to-video translation. In each block: input image (left), video generated by MoCoGAN (right).

Figure 4: The figure is best viewed with Acrobat Reader on a desktop. Click each image to play the video clip.

User study. We conducted a user study to quantitatively compare MoCoGAN to VGAN and TGAN using the facial expression and Tai-Chi datasets. For each algorithm, we used the trained model to randomly generate 80 videos for each task. We then randomly paired the videos generated by the MoCoGAN with the videos from one of the competing algorithms to form 80 questions. These questions were sent to the workers on Amazon Mechanical Turk (AMT) for evaluation. The videos from different algorithms were shown in random order for a fair comparison. Each question was answered by 3 different workers. The workers were instructed to choose the video that looks more realistic. Only the workers with a lifetime HIT (Human Intelligent Task) approval rate greater than 95% participated in the user study.

We report the average preference scores (the average number of times, a worker prefers an algorithm) in Table 3. From the table, we find that the workers considered the videos generated by MoCoGAN more realistic most of the times. Compared to VGAN, MoCoGAN achieved a preference score of 84.2% and 75.4% for the facial expression and Tai-Chi datasets, respectively. Compared to TGAN, MoCoGAN achieved a preference score of 54.7% and 68.0% for the facial expression and Tai-Chi datasets, respectively. In Fig. 3, we visualize the facial expression and Tai-Chi videos generated by the competing algorithms. We find that the videos generated by MoCoGAN are more realistic and contained less content and motion artifacts.

Qualitative evaluation. We conducted a qualitative experiment to demonstrate our motion and content decomposed representation. We sampled two content codes and seven motion codes, giving us 14 videos. Fig. 4b shows an example randomly selected from this experiment. Each row has the same content code, and each column has the same motion code. We observed that MoCoGAN generated the same motion sequences for two different content samples.

Table 4: Performance on categorical facial expression video generation with various MoCoGAN settings.

Settings		MCS	ACD
~~D_I~~	$\mathbf{z}_\mathrm{A} \to G_\mathrm{I}$	0.472	1.115
~~D_I~~	$\mathbf{z}_\mathrm{A} \to R_\mathrm{M}$	0.491	1.073
D_I	$\mathbf{z}_\mathrm{A} \to G_\mathrm{I}$	0.355	0.738
D_I	$\mathbf{z}_\mathrm{A} \to R_\mathrm{M}$	**0.581**	**0.606**

4.2. Categorical Video Generation

We augmented MoCoGAN with categorical variables and trained it for facial expression video generation as described in Section 3.1. The MUG dataset contains 6 different facial expressions and hence $\mathbf{z}_\mathrm{A}$ is realized as a 6 dimensional one-hot vector. We then generated 96 frames of facial expression videos. During generation, we changed the action category, $\mathbf{z}_\mathrm{A}$, every 16 frames to cover all 6 expressions. Hence, a generated video corresponded to a person performing 6 different facial expressions, one after another.

To evaluate the performance, we computed the ACD of the generated videos. A smaller ACD means the generated faces over the 96 frames were more likely to be from the same person. Note that the ACD reported in this subsection are generally larger than the ACD reported in Table 1, because the generated videos in this experiment are 6 times longer and contain 6 facial expressions versus 1. We also used the motion control score (MCS) to evaluate MoCoGAN's capability in motion generation control. To compute MCS, we first trained a spatio-temporal CNN classifier for action recognition using the labeled training dataset. During test time, we used the classifier to verify whether the generated video contained the action. The MCS is then given by testing accuracy of the classifier. A model with larger MCS offers better control over the action category.

In this experiment, we also evaluated the impact of different conditioning schemes to the categorical video generation performance. The first scheme is our default scheme where $\mathbf{z}_\mathrm{A} \to R_\mathrm{M}$. The second scheme, termed $\mathbf{z}_\mathrm{A} \to G_\mathrm{I}$, was to feed the category variable directly to the image generator. In addition, to show the impact of the image discriminative network D_I, we considered training the MoCoGAN framework without D_I.

Table 4 shows experimental results. We find that the models trained with D_I consistently yield better performances on various metrics. We also find that $\mathbf{z}_\mathrm{A} \to R_\mathrm{M}$

Figure 5: Generated videos of changing facial expressions. We changed the expression from smile to fear through surprise.

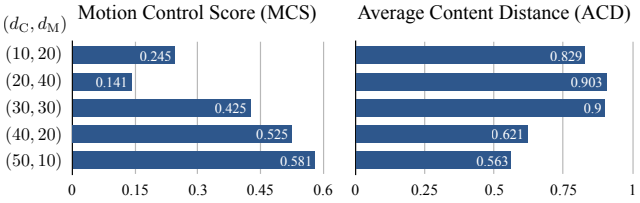


Figure 6: MoCoGAN models with varying (d_C, d_M) settings on facial expression generation.

yields better performance. Fig. 5 shows two videos from the best model in Table 4. We observe that by fixing the content vector but changing the expression label, it generates videos of the same person performing different expressions. And similarly, by changing the content vector and providing the same motion trajectory, we generate videos of different people showing the same expression sequence.

We conducted an experiment to empirically analyze the impact of the dimensions of the content and motion vectors $\mathbf{z}_C$ and $\mathbf{z}_M^{(t)}$ (referred to as d_C and d_M) to the categorical video generation performance. In the experiment, we fixed the sum of the dimensions to 60 (i.e., $d_C + d_M = 60$) and changed the value of d_C from 10 to 50, with a step size of 10. Fig. 6 shows the results.

We found when d_C was large, MoCoGAN had a small ACD. This meant a video generated by the MoCoGAN resembled the same person performing different expressions. We were expecting a larger $\mathbf{z}_M$ would lead to a larger MCS but found the contrary. Inspecting the generated videos, we found when d_M was large (i.e. d_C was small), MoCoGAN failed to generate recognizable faces, resulting in a poor MCS. In this case, given poor image quality, the facial expression recognition network could only perform a random guess on the expression and scored poorly. Based on this, we set $d_C = 50$ and $d_M = 10$ in all the experiments.

4.3. Image-to-video Translation

We trained a variant of the MoCoGAN framework, in which the generator is realized as an encoder-decoder architecture [21], where the encoder produced the content code $\mathbf{z}_C$ and the initial motion code $\mathbf{z}_m^{(0)}$. Subsequent motion codes were produced by R_M and concatenated with the content code to generate each frame. That is the input was an image and the output was a video. We trained a MoCoGAN model using the Tai-Chi dataset. In test time, we sampled random images from a withheld test set to generate video sequences. In addition to the MoCoGAN loss, we have added the L_1 reconstruction loss for training the encoder-decoder architecture similar to [21]. Under this setting, MoCoGAN generated a video sequence starting from the first frame (see Fig. 4c). We compared with two state-of-the-art approaches on this task: a Conditional-VGAN (C-VGAN) and Motion Content Network (MCNET) [39] and performed a user study to compare the competing methods. We note that MCNET used 4 frames to predict a video, while C-VGAN and MoCoGAN required a single frame only. Table 5 shows the user preference scores. The results support that MoCoGAN was able to generate more realistic videos than C-VGAN and MCNET.

Table 5: User preference score on the quality of the image-to-video-translation results.

User preference, %	Tai-Chi
MoCoGAN / C-VGAN	**66.9** / 33.1
MoCoGAN / MCNET	**65.6** / 34.4

5. Conclusion

We presented the MoCoGAN framework for motion and content decomposed video generation. Given sufficient video training data, MoCoGAN automatically learns to disentangle motion from content in an unsupervised manner. For instance, given videos of people performing different facial expressions, MoCoGAN learns to separate a person's identity from their expression, thus allowing us to synthesize a new video of a person performing different expressions, or fixing the expression and generating various identities. This is enabled by a new generative adversarial network, which generates a video clip by sequentially generating video frames. Each video frame is generated from a random vector, which consists of two parts, one signifying content and one signifying motion. The content subspace is modeled with a Gaussian distribution, whereas the motion subspace is modeled with a recurrent neural network. We sample this space in order to synthesize each video frame. Our experimental evaluation supports that the proposed framework is superior to current state-of-the-art video generation and next frame prediction methods.

References

[1] N. Aifanti, C. Papachristou, and A. Delopoulos. The mug facial expression database. In *Image Analysis for Multimedia Interactive Services (WIAMIS), 2010 11th International Workshop on*, pages 1–4, 2010.

[2] B. Amos, B. Ludwiczuk, and M. Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science, 2016.

[3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[4] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[5] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Advances in Neural Information Processing Systems (NIPS) Workshop*, 2014.

[7] E. Denton and V. Birodkar. Unsupervised learning of disentangled representations from video. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[8] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[9] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto. Dynamic textures. *International Journal of Computer Vision (IJCV)*, 2003.

[10] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[11] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[13] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 29(12):2247–2253, 2007.

[14] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning (ICML)*, 2015.

[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[16] D. J. Im, C. D. Kim, H. Jiang, and R. Memisevic. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*, 2016.

[17] N. Kalchbrenner, A. v. d. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

[18] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[19] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[20] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[21] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[22] M.-Y. Liu and O. Tuzel. Coupled generative adversarial networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[23] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[24] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations (ICLR)*, 2016.

[25] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier gans. In *International Conference on Machine Learning (ICML)*, 2017.

[26] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[27] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[28] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and variational inference in deep latent gaussian models. In *International Conference on Machine Learning (ICML)*, 2014.

[29] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[30] M. Saito, E. Matsumoto, and S. Saito. Temporal generative adversarial nets with singular value clipping. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[31] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[32] 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.

[33] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning (ICML)*, 2015.

[34] M. Szummer and R. W. Picard. Temporal texture modeling. In *International Conference on Image Processing (ICIP)*, 1996.

[35] L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR)*, 2016.

[36] S. Tulyakov, A. Fitzgibbon, and S. Nowozin. Hybrid vae: Improving deep generative models using partial observations. *Advances in Neural Information Processing Systems (NIPS) Workshop*, 2017.

[37] J. van Amersfoort, A. Kannan, M. Ranzato, A. Szlam, D. Tran, and S. Chintala. Transformation-based models of video sequences. *arXiv preprint arXiv:1701.08435*, 2017.

[38] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[39] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee. Decomposing motion and content for natural video sequence prediction. In *International Conference on Learning Representations (ICLR)*, 2017.

[40] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[41] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *ACM SIGGRAPH*, 2000.

[42] T. Xue, J. Wu, K. Bouman, and B. Freeman. Probabilistic modeling of future frames from a single image. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[43] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *IEEE International Conference on Computer Vision (ICCV)*, 2016.

A. Network Architecture

Table 6: Network architectures of the image generative network G_{I}, the image discriminative network D_{I}, and the video generative network D_{V} used in the experiments.

G_{I}	Configuration
Input	$[\mathbf{z}_{\mathrm{a}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \mathbf{z}_{\mathrm{m}} \sim R_{\mathrm{M}}]$
0	DCONV-(N512, K6, S0, P0), BN, LeakyReLU
1	DCONV-(N256, K4, S2, P1), BN, LeakyReLU
2	DCONV-(N128, K4, S2, P1), BN, LeakyReLU
3	DCONV-(N64, K4, S2, P1), BN, LeakyReLU
4	DCONV-(N3, K4, S2, P1), BN, LeakyReLU

D_{I}	Configuration
Input	height $\times$ width $\times$ 3
0	CONV-(N64, K4, S2, P1), BN, LeakyReLU
1	CONV-(N128, K4, S2, P1), BN, LeakyReLU
2	CONV-(N256, K4, S2, P1), BN, LeakyReLU
3	CONV-(N1, K4, S2, P1), Sigmoid

D_{V}	Configuration
Input	16 $\times$ height $\times$ width $\times$ 3
0	CONV3D-(N64, K4, S1, P0), BN, LeakyReLU
1	CONV3D-(N128, K4, S1, P0), BN, LeakyReLU
2	CONV3D-(N256, K4, S1, P0), BN, LeakyReLU
3	CONV3D-(N1, K4, S1, P0), Sigmoid

Table 6 detailed the network architecture used in the main paper. We used several different type of layers. A convolutional layer with N output channels, kernel size K, stride S, and padding P is denoted in the table as CONV-(N, K, S, P) and similarly for 3D convolutional layers CONV3D-(N, K, S, P). Kernel size, padding, and stride are equal for all the dimensions in each layer. Batch normalization layers are followed by the LeakyReLU nonlinearity in our case. The R_{M} consisted of a single GRU module.

B. Additional Qualitative Results

Figures 7 and 8 show categorical facial expression and categorical human actions video generation results, respectively. In each figure, every group of three rows was generated with a fixed content vector $\mathbf{z}_{\mathrm{C}}$, but random action vector $\mathbf{z}_{\mathrm{A}}$ and motion vectors $\mathbf{z}_{\mathrm{M}}^{(t)}$'s. We found that the identity was kept fixed throughout a video. We noted that the length of the generated videos were longer than those in the training data. This showed that the MoCoGAN can generalize the video generation along the time dimension.

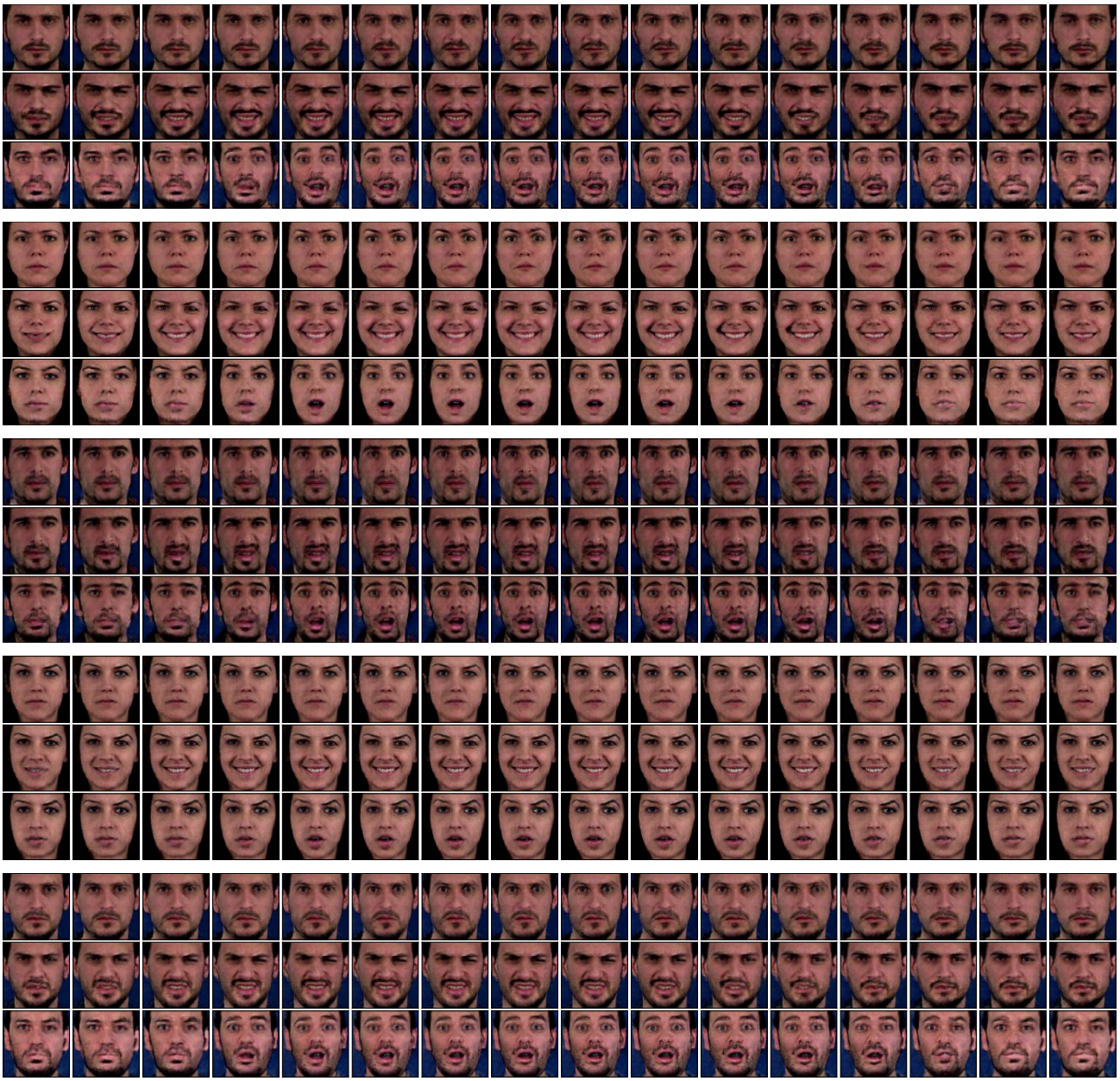


Figure 7: Facial expression video generation results. Every three rows have the same identity but different $\mathbf{z}_{\mathrm{A}}$.

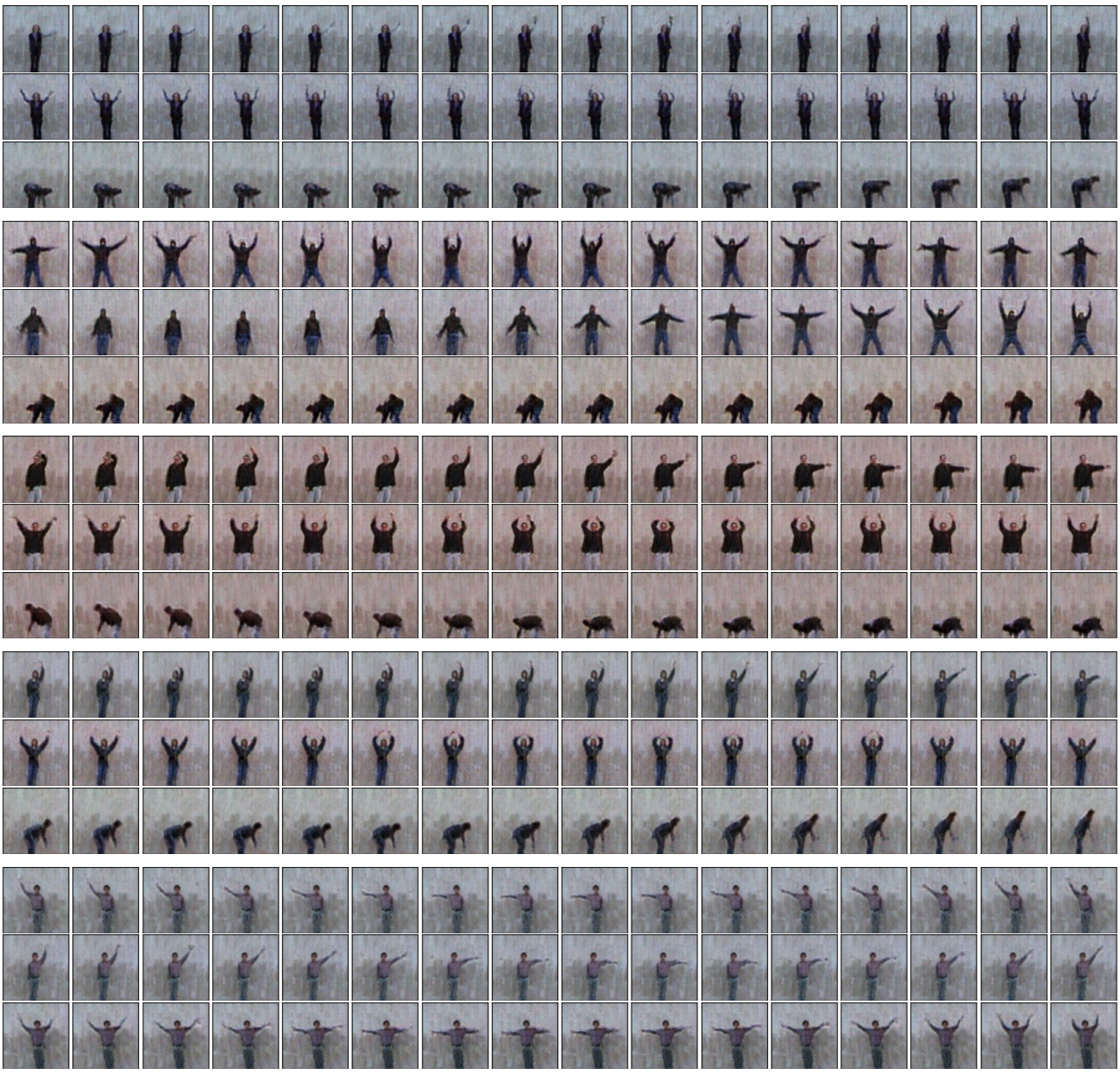


Figure 8: Human action video generation results. Every three rows have the same identity but different $\mathbf{z}_{\mathrm{A}}$.

LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS

Andrew Brock[*†]
Heriot-Watt University
ajb5@hw.ac.uk

Jeff Donahue[†]
DeepMind
jeffdonahue@google.com

Karen Simonyan[†]
DeepMind
simonyan@google.com

ABSTRACT

Despite recent progress in generative image modeling, successfully generating high-resolution, diverse samples from complex datasets such as ImageNet remains an elusive goal. To this end, we train Generative Adversarial Networks at the largest scale yet attempted, and study the instabilities specific to such scale. We find that applying orthogonal regularization to the generator renders it amenable to a simple "truncation trick," allowing fine control over the trade-off between sample fidelity and variety by reducing the variance of the Generator's input. Our modifications lead to models which set the new state of the art in class-conditional image synthesis. When trained on ImageNet at 128×128 resolution, our models (BigGANs) achieve an Inception Score (IS) of 166.5 and Fréchet Inception Distance (FID) of 7.4, improving over the previous best IS of 52.52 and FID of 18.65.

1 INTRODUCTION

Figure 1: Class-conditional samples generated by our model.

The state of generative image modeling has advanced dramatically in recent years, with Generative Adversarial Networks (GANs, Goodfellow et al. (2014)) at the forefront of efforts to generate high-fidelity, diverse images with models learned directly from data. GAN training is dynamic, and sensitive to nearly every aspect of its setup (from optimization parameters to model architecture), but a torrent of research has yielded empirical and theoretical insights enabling stable training in a variety of settings. Despite this progress, the current state of the art in conditional ImageNet modeling (Zhang et al., 2018) achieves an Inception Score (Salimans et al., 2016) of 52.5, compared to 233 for real data.

In this work, we set out to close the gap in fidelity and variety between images generated by GANs and real-world images from the ImageNet dataset. We make the following three contributions towards this goal:

- We demonstrate that GANs benefit dramatically from scaling, and train models with two to four times as many parameters and eight times the batch size compared to prior art. We introduce two simple, general architectural changes that improve scalability, and modify a regularization scheme to improve conditioning, demonstrably boosting performance.

[*]Work done at DeepMind
[†]Equal contribution

- As a side effect of our modifications, our models become amenable to the "truncation trick," a simple sampling technique that allows explicit, fine-grained control of the trade-off between sample variety and fidelity.
- We discover instabilities specific to large scale GANs, and characterize them empirically. Leveraging insights from this analysis, we demonstrate that a combination of novel and existing techniques can reduce these instabilities, but complete training stability can only be achieved at a dramatic cost to performance.

Our modifications substantially improve class-conditional GANs. When trained on ImageNet at 128×128 resolution, our models (BigGANs) improve the state-of-the-art Inception Score (IS) and Fréchet Inception Distance (FID) from 52.52 and 18.65 to 166.5 and 7.4 respectively. We also successfully train BigGANs on ImageNet at 256×256 and 512×512 resolution, and achieve IS and FID of 232.5 and 8.1 at 256×256 and IS and FID of 241.5 and 11.5 at 512×512. Finally, we train our models on an even larger dataset – JFT-300M – and demonstrate that our design choices transfer well from ImageNet. Code and weights for our pretrained generators are publicly available [1].

2 BACKGROUND

A Generative Adversarial Network (GAN) involves Generator (**G**) and Discriminator (**D**) networks whose purpose, respectively, is to map random noise to samples and discriminate real and generated samples. Formally, the GAN objective, in its original form (Goodfellow et al., 2014) involves finding a Nash equilibrium to the following two player min-max problem:

$$\min_G \max_D \mathbb{E}_{x\sim q_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))], \tag{1}$$

where $\boldsymbol{z} \in \mathbb{R}^{d_z}$ is a latent variable drawn from distribution $p(\boldsymbol{z})$ such as $\mathcal{N}(0, I)$ or $\mathcal{U}[-1, 1]$. When applied to images, **G** and **D** are usually convolutional neural networks (Radford et al., 2016). Without auxiliary stabilization techniques, this training procedure is notoriously brittle, requiring finely-tuned hyperparameters and architectural choices to work at all.

Much recent research has accordingly focused on modifications to the vanilla GAN procedure to impart stability, drawing on a growing body of empirical and theoretical insights (Nowozin et al., 2016; Sønderby et al., 2017; Fedus et al., 2018). One line of work is focused on changing the objective function (Arjovsky et al., 2017; Mao et al., 2016; Lim & Ye, 2017; Bellemare et al., 2017; Salimans et al., 2018) to encourage convergence. Another line is focused on constraining **D** through gradient penalties (Gulrajani et al., 2017; Kodali et al., 2017; Mescheder et al., 2018) or normalization (Miyato et al., 2018), both to counteract the use of unbounded loss functions and ensure **D** provides gradients everywhere to **G**.

Of particular relevance to our work is Spectral Normalization (Miyato et al., 2018), which enforces Lipschitz continuity on **D** by normalizing its parameters with running estimates of their first singular values, inducing backwards dynamics that adaptively regularize the top singular direction. Relatedly Odena et al. (2018) analyze the condition number of the Jacobian of **G** and find that performance is dependent on **G**'s conditioning. Zhang et al. (2018) find that employing Spectral Normalization in **G** improves stability, allowing for fewer **D** steps per iteration. We extend on these analyses to gain further insight into the pathology of GAN training.

Other works focus on the choice of architecture, such as SA-GAN (Zhang et al., 2018) which adds the self-attention block from (Wang et al., 2018) to improve the ability of both **G** and **D** to model global structure. ProGAN (Karras et al., 2018) trains high-resolution GANs in the single-class setting by training a single model across a sequence of increasing resolutions.

In conditional GANs (Mirza & Osindero, 2014) class information can be fed into the model in various ways. In (Odena et al., 2017) it is provided to **G** by concatenating a 1-hot class vector to the noise vector, and the objective is modified to encourage conditional samples to maximize the corresponding class probability predicted by an auxiliary classifier. de Vries et al. (2017) and

[1] https://tfhub.dev/s?q=biggan

Batch	Ch.	Param (M)	Shared	Skip-z	Ortho.	Itr $\times 10^3$	FID	IS
256	64	81.5	SA-GAN Baseline			1000	18.65	52.52
512	64	81.5	✗	✗	✗	1000	15.30	58.77($\pm$1.18)
1024	64	81.5	✗	✗	✗	1000	14.88	63.03($\pm$1.42)
2048	64	81.5	✗	✗	✗	732	12.39	76.85($\pm$3.83)
2048	96	173.5	✗	✗	✗	295($\pm$18)	9.54($\pm$0.62)	92.98($\pm$4.27)
2048	96	160.6	✓	✗	✗	185($\pm$11)	9.18($\pm$0.13)	94.94($\pm$1.32)
2048	96	158.3	✓	✓	✗	152($\pm$7)	8.73($\pm$0.45)	98.76($\pm$2.84)
2048	96	158.3	✓	✓	✓	165($\pm$13)	8.51($\pm$0.32)	99.31($\pm$2.10)
2048	64	71.3	✓	✓	✓	371($\pm$7)	10.48($\pm$0.10)	86.90($\pm$0.61)

Table 1: Fréchet Inception Distance (FID, lower is better) and Inception Score (IS, higher is better) for ablations of our proposed modifications. *Batch* is batch size, *Param* is total number of parameters, *Ch.* is the channel multiplier representing the number of units in each layer, *Shared* is using shared embeddings, *Skip-z* is using skip connections from the latent to multiple layers, *Ortho.* is Orthogonal Regularization, and *Itr* indicates if the setting is stable to 10^6 iterations, or it collapses at the given iteration. Other than rows 1-4, results are computed across 8 random initializations.

Dumoulin et al. (2017) modify the way class conditioning is passed to **G** by supplying it with class-conditional gains and biases in BatchNorm (Ioffe & Szegedy, 2015) layers. In Miyato & Koyama (2018), **D** is conditioned by using the cosine similarity between its features and a set of learned class embeddings as additional evidence for distinguishing real and generated samples, effectively encouraging generation of samples whose features match a learned class prototype.

Objectively evaluating implicit generative models is difficult (Theis et al., 2015). A variety of works have proposed heuristics for measuring the sample quality of models without tractable likelihoods (Salimans et al., 2016; Heusel et al., 2017; Bińkowski et al., 2018; Wu et al., 2017). Of these, the Inception Score (IS, Salimans et al. (2016)) and Fréchet Inception Distance (FID, Heusel et al. (2017)) have become popular despite their notable flaws (Barratt & Sharma, 2018). We employ them as approximate measures of sample quality, and to enable comparison against previous work.

3 Scaling Up GANs

In this section, we explore methods for scaling up GAN training to reap the performance benefits of larger models and larger batches. As a baseline, we employ the SA-GAN architecture of Zhang et al. (2018), which uses the hinge loss (Lim & Ye, 2017; Tran et al., 2017) GAN objective. We provide class information to **G** with class-conditional BatchNorm (Dumoulin et al., 2017; de Vries et al., 2017) and to **D** with projection (Miyato & Koyama, 2018). The optimization settings follow Zhang et al. (2018) (notably employing Spectral Norm in **G**) with the modification that we halve the learning rates and take two **D** steps per **G** step. For evaluation, we employ moving averages of **G**'s weights following Karras et al. (2018); Mescheder et al. (2018); Yazc et al. (2018), with a decay of 0.9999. We use Orthogonal Initialization (Saxe et al., 2014), whereas previous works used $\mathcal{N}(0, 0.02I)$ (Radford et al., 2016) or Xavier initialization (Glorot & Bengio, 2010). Each model is trained on 128 to 512 cores of a Google TPUv3 Pod (Google, 2018), and computes BatchNorm statistics in **G** across all devices, rather than per-device as is typical. We find progressive growing (Karras et al., 2018) unnecessary even for our 512×512 models. Additional details are in Appendix C.

We begin by increasing the batch size for the baseline model, and immediately find tremendous benefits in doing so. Rows 1-4 of Table 1 show that simply increasing the batch size by a factor of 8 improves the state-of-the-art IS by 46%. We conjecture that this is a result of each batch covering more modes, providing better gradients for both networks. One notable side effect of this scaling is that our models reach better final performance in fewer iterations, but become unstable and undergo complete training collapse. We discuss the causes and ramifications of this in Section 4. For these experiments, we report scores from checkpoints saved just before collapse.

We then increase the width (number of channels) in each layer by 50%, approximately doubling the number of parameters in both models. This leads to a further IS improvement of 21%, which we posit is due to the increased capacity of the model relative to the complexity of the dataset. Doubling

(a) (b)

Figure 2: (a) The effects of increasing truncation. From left to right, the threshold is set to 2, 1, 0.5, 0.04. (b) Saturation artifacts from applying truncation to a poorly conditioned model.

the depth did not initially lead to improvement – we addressed this later in the BigGAN-deep model, which uses a different residual block structure.

We note that class embeddings c used for the conditional BatchNorm layers in **G** contain a large number of weights. Instead of having a separate layer for each embedding (Miyato et al., 2018; Zhang et al., 2018), we opt to use a shared embedding, which is linearly projected to each layer's gains and biases (Perez et al., 2018). This reduces computation and memory costs, and improves training speed (in number of iterations required to reach a given performance) by 37%. Next, we add direct skip connections (skip-z) from the noise vector z to multiple layers of **G** rather than just the initial layer. The intuition behind this design is to allow **G** to use the latent space to directly influence features at different resolutions and levels of hierarchy. In BigGAN, this is accomplished by splitting z into one chunk per resolution, and concatenating each chunk to the conditional vector c which gets projected to the BatchNorm gains and biases. In BigGAN-deep, we use an even simpler design, concatenating the entire z with the conditional vector without splitting it into chunks. Previous works (Goodfellow et al., 2014; Denton et al., 2015) have considered variants of this concept; our implementation is a minor modification of this design. Skip-z provides a modest performance improvement of around 4%, and improves training speed by a further 18%.

3.1 Trading off variety and fidelity with the Truncation Trick

Unlike models which need to backpropagate through their latents, GANs can employ an arbitrary prior $p(z)$, yet the vast majority of previous works have chosen to draw z from either $\mathcal{N}(0, I)$ or $\mathcal{U}[-1, 1]$. We question the optimality of this choice and explore alternatives in Appendix E.

Remarkably, our best results come from using a different latent distribution for sampling than was used in training. Taking a model trained with $z \sim \mathcal{N}(0, I)$ and sampling z from a *truncated normal* (where values which fall outside a range are resampled to fall inside that range) immediately provides a boost to IS and FID. We call this the *Truncation Trick*: truncating a z vector by resampling the values with magnitude above a chosen threshold leads to improvement in individual sample quality at the cost of reduction in overall sample variety. Figure 2(a) demonstrates this: as the threshold is reduced, and elements of z are truncated towards zero (the mode of the latent distribution), individual samples approach the mode of **G**'s output distribution. Related observations about this trade-off were made in (Marchesi, 2016; Pieters & Wiering, 2014).

This technique allows fine-grained, post-hoc selection of the trade-off between sample quality and variety for a given **G**. Notably, we can compute FID and IS for a range of thresholds, obtaining the variety-fidelity curve reminiscent of the precision-recall curve (Figure 17). As IS does not penalize lack of variety in class-conditional models, reducing the truncation threshold leads to a direct increase in IS (analogous to precision). FID penalizes lack of variety (analogous to recall) but also rewards precision, so we initially see a moderate improvement in FID, but as truncation approaches zero and variety diminishes, the FID sharply drops. The distribution shift caused by sampling with different latents than those seen in training is problematic for many models. Some of our larger models are not amenable to truncation, producing saturation artifacts (Figure 2(b)) when fed truncated noise. To counteract this, we seek to enforce amenability to truncation by conditioning **G** to be smooth, so that the full space of z will map to good output samples. For this, we turn to Orthogonal Regularization (Brock et al., 2017), which directly enforces the orthogonality condition:

$$R_\beta(W) = \beta \|W^\top W - I\|_{\mathrm{F}}^2, \tag{2}$$

where W is a weight matrix and β a hyperparameter. This regularization is known to often be too limiting (Miyato et al., 2018), so we explore several variants designed to relax the constraint while still imparting the desired smoothness to our models. The version we find to work best removes the diagonal terms from the regularization, and aims to minimize the pairwise cosine similarity between filters but does not constrain their norm:

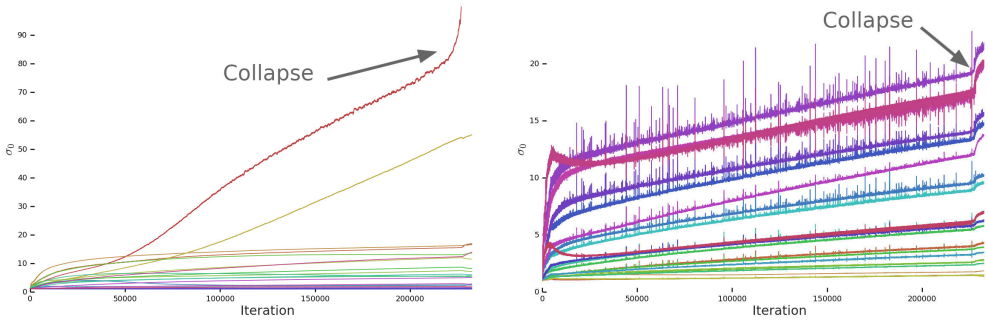
$$R_\beta(W) = \beta \|W^\top W \odot (\mathbf{1} - I)\|_{\mathrm{F}}^2, \tag{3}$$

where $\mathbf{1}$ denotes a matrix with all elements set to 1. We sweep β values and select 10^{-4}, finding this small added penalty sufficient to improve the likelihood that our models will be amenable to truncation. Across runs in Table 1, we observe that without Orthogonal Regularization, only 16% of models are amenable to truncation, compared to 60% when trained with Orthogonal Regularization.

3.2 Summary

We find that current GAN techniques are sufficient to enable scaling to large models and distributed, large-batch training. We find that we can dramatically improve the state of the art and train models up to 512×512 resolution without need for explicit multiscale methods like Karras et al. (2018). Despite these improvements, our models undergo training collapse, necessitating early stopping in practice. In the next two sections we investigate why settings which were stable in previous works become unstable when applied at scale.

4 Analysis



(a) **G** (b) **D**

Figure 3: A typical plot of the first singular value σ_0 in the layers of **G** (a) and **D** (b) before Spectral Normalization. Most layers in **G** have well-behaved spectra, but without constraints a small subset grow throughout training and explode at collapse. **D**'s spectra are noisier but otherwise better-behaved. Colors from red to violet indicate increasing depth.

4.1 Characterizing Instability: The Generator

Much previous work has investigated GAN stability from a variety of analytical angles and on toy problems, but the instabilities we observe occur for settings which are stable at small scale, necessitating direct analysis at large scale. We monitor a range of weight, gradient, and loss statistics during training, in search of a metric which might presage the onset of training collapse, similar to (Odena et al., 2018). We found the top three singular values $\sigma_0, \sigma_1, \sigma_2$ of each weight matrix to be the most informative. They can be efficiently computed using the Alrnoldi iteration method (Golub & der Vorst, 2000), which extends the power iteration method, used in Miyato et al. (2018), to estimation of additional singular vectors and values. A clear pattern emerges, as can be seen in Figure 3(a) and Appendix F: most **G** layers have well-behaved spectral norms, but some layers

(typically the first layer in **G**, which is over-complete and not convolutional) are ill-behaved, with spectral norms that grow throughout training and explode at collapse.

To ascertain if this pathology is a cause of collapse or merely a symptom, we study the effects of imposing additional conditioning on **G** to explicitly counteract spectral explosion. First, we directly regularize the top singular values σ_0 of each weight, either towards a fixed value σ_{reg} or towards some ratio r of the second singular value, $r \cdot sg(\sigma_1)$ (with sg the stop-gradient operation to prevent the regularization from increasing σ_1). Alternatively, we employ a partial singular value decomposition to instead clamp σ_0. Given a weight W, its first singular vectors u_0 and v_0, and σ_{clamp} the value to which the σ_0 will be clamped, our weights become:

$$W = W - \max(0, \sigma_0 - \sigma_{clamp}) v_0 u_0^\top, \tag{4}$$

where σ_{clamp} is set to either σ_{reg} or $r \cdot sg(\sigma_1)$. We observe that both with and without Spectral Normalization these techniques have the effect of preventing the gradual increase and explosion of either σ_0 or $\frac{\sigma_0}{\sigma_1}$, but even though in some cases they mildly improve performance, no combination prevents training collapse. This evidence suggests that while conditioning **G** might improve stability, it is insufficient to ensure stability. We accordingly turn our attention to **D**.

4.2 Characterizing Instability: The Discriminator

As with **G**, we analyze the spectra of **D**'s weights to gain insight into its behavior, then seek to stabilize training by imposing additional constraints. Figure 3(b) displays a typical plot of σ_0 for **D** (with further plots in Appendix F). Unlike **G**, we see that the spectra are noisy, $\frac{\sigma_0}{\sigma_1}$ is well-behaved, and the singular values grow throughout training but only jump at collapse, instead of exploding.

The spikes in **D**'s spectra might suggest that it periodically receives very large gradients, but we observe that the Frobenius norms are smooth (Appendix F), suggesting that this effect is primarily concentrated on the top few singular directions. We posit that this noise is a result of optimization through the adversarial training process, where **G** periodically produces batches which strongly perturb **D** . If this spectral noise is causally related to instability, a natural counter is to employ gradient penalties, which explicitly regularize changes in **D**'s Jacobian. We explore the R_1 zero-centered gradient penalty from Mescheder et al. (2018):

$$R_1 := \frac{\gamma}{2} \mathbb{E}_{p_{\mathcal{D}}(x)} \left[\|\nabla D(x)\|_F^2 \right]. \tag{5}$$

With the default suggested γ strength of 10, training becomes stable and improves the smoothness and boundedness of spectra in both **G** and **D**, but performance severely degrades, resulting in a 45% reduction in IS. Reducing the penalty partially alleviates this degradation, but results in increasingly ill-behaved spectra; even with the penalty strength reduced to 1 (the lowest strength for which sudden collapse does not occur) the IS is reduced by 20%. Repeating this experiment with various strengths of Orthogonal Regularization, DropOut (Srivastava et al., 2014), and L2 (See Appendix I for details), reveals similar behaviors for these regularization strategies: with high enough penalties on **D**, training stability can be achieved, but at a substantial cost to performance.

We also observe that **D**'s loss approaches zero during training, but undergoes a sharp upward jump at collapse (Appendix F). One possible explanation for this behavior is that **D** is overfitting to the training set, memorizing training examples rather than learning some meaningful boundary between real and generated images. As a simple test for **D**'s memorization (related to Gulrajani et al. (2017)), we evaluate uncollapsed discriminators on the ImageNet training and validation sets, and measure what percentage of samples are classified as real or generated. While the training accuracy is consistently above 98%, the validation accuracy falls in the range of 50-55%, no better than random guessing (regardless of regularization strategy). This confirms that **D** is indeed memorizing the training set; we deem this in line with **D**'s role, which is not explicitly to generalize, but to distill the training data and provide a useful learning signal for **G**. Additional experiments and discussion are provided in Appendix G.

4.3 Summary

We find that stability does not come solely from **G** or **D**, but from their interaction through the adversarial training process. While the symptoms of their poor conditioning can be used to track and

Model	Res.	FID/IS	(min FID) / IS	FID / (valid IS)	FID / (max IS)
SN-GAN	128	27.62/36.80	N/A	N/A	N/A
SA-GAN	128	18.65/52.52	N/A	N/A	N/A
BigGAN	128	$8.7 \pm .6/98.8 \pm 3$	$7.7 \pm .2/126.5 \pm 0$	$9.6 \pm .4/166.3 \pm 1$	$25 \pm 2/206 \pm 2$
BigGAN	256	$8.7 \pm .1/142.3 \pm 2$	$7.7 \pm .1/178.0 \pm 5$	$9.3 \pm .3/233.1 \pm 1$	$25 \pm 5/291 \pm 4$
BigGAN	512	8.1/144.2	7.6/170.3	11.8/241.4	27.0/275
BigGAN-deep	128	$5.7 \pm .3/124.5 \pm 2$	$6.3 \pm .3/148.1 \pm 4$	$7.4 \pm .6/166.5 \pm 1$	$25 \pm 2/253 \pm 11$
BigGAN-deep	256	$6.9 \pm .2/171.4 \pm 2$	$7.0 \pm .1/202.6 \pm 2$	$8.1 \pm .1/232.5 \pm 2$	$27 \pm 8/317 \pm 6$
BigGAN-deep	512	7.5/152.8	7.7/181.4	11.5/241.5	39.7/298

Table 2: Evaluation of models at different resolutions. We report scores without truncation (Column 3), scores at the best FID (Column 4), scores at the IS of validation data (Column 5), and scores at the max IS (Column 6). Standard deviations are computed over at least three random initializations.

identify instability, ensuring reasonable conditioning proves necessary for training but insufficient to prevent eventual training collapse. It is possible to enforce stability by strongly constraining **D**, but doing so incurs a dramatic cost in performance. With current techniques, better final performance can be achieved by relaxing this conditioning and allowing collapse to occur at the later stages of training, by which time a model is sufficiently trained to achieve good results.

5 EXPERIMENTS

(a) 128×128 (b) 256×256 (c) 512×512 (d)

Figure 4: Samples from our BigGAN model with truncation threshold 0.5 (a-c) and an example of class leakage in a partially trained model (d).

5.1 EVALUATION ON IMAGENET

We evaluate our models on ImageNet ILSVRC 2012 (Russakovsky et al., 2015) at 128×128, 256×256, and 512×512 resolutions, employing the settings from Table 1, row 8. The samples generated by our models are presented in Figure 4, with additional samples in Appendix A, and online [2]. We report IS and FID in Table 2. As our models are able to trade sample variety for quality, it is unclear how best to compare against prior art; we accordingly report values at three settings, with complete curves in Appendix D. First, we report the FID/IS values at the truncation setting which attains the best FID. Second, we report the FID at the truncation setting for which our model's IS is the same as that attained by the real validation data, reasoning that this is a passable measure of maximum sample variety achieved while still achieving a good level of "objectness." Third, we report FID at the maximum IS achieved by each model, to demonstrate how much variety must be traded off to maximize quality. In all three cases, our models outperform the previous state-of-the-art IS and FID scores achieved by Miyato et al. (2018) and Zhang et al. (2018).

In addition to the BigGAN model introduced in the first version of the paper and used in the majority of experiments (unless otherwise stated), we also present a 4x deeper model (BigGAN-deep) which uses a different configuration of residual blocks. As can be seen from Table 2, BigGAN-deep substantially outperforms BigGAN across all resolutions and metrics. This confirms that our findings

[2] https://drive.google.com/drive/folders/1lWC6XEPD0LT5KUnPXeve_kWeY-FxH002

Ch.	Param (M)	Shared	Skip-z	Ortho.	FID	IS	(min FID) / IS	FID / (max IS)
64	317.1	✗	✗	✗	48.38	23.27	48.6/23.1	49.1/23.9
64	99.4	✓	✓	✓	23.48	24.78	22.4/21.0	60.9/35.8
96	207.9	✓	✓	✓	18.84	27.86	17.1/23.3	51.6/38.1
128	355.7	✓	✓	✓	13.75	30.61	13.0/28.0	46.2/47.8

Table 3: BigGAN results on JFT-300M at 256×256 resolution. The *FID* and *IS* columns report these scores given by the JFT-300M-trained Inception v2 classifier with noise distributed as $z \sim \mathcal{N}(0, I)$ (non-truncated). The *(min FID) / IS* and *FID / (max IS)* columns report scores at the best FID and IS from a sweep across truncated noise distributions ranging from $\sigma = 0$ to $\sigma = 2$. Images from the JFT-300M validation set have an IS of 50.88 and FID of 1.94.

extend to other architectures, and that increased depth leads to improvement in sample quality. Both BigGAN and BigGAN-deep architectures are described in Appendix B.

Our observation that **D** overfits to the training set, coupled with our model's sample quality, raises the obvious question of whether or not **G** simply memorizes training points. To test this, we perform class-wise nearest neighbors analysis in pixel space and the feature space of pre-trained classifier networks (Appendix A). In addition, we present both interpolations between samples and class-wise interpolations (where z is held constant) in Figures 8 and 9. Our model convincingly interpolates between disparate samples, and the nearest neighbors for its samples are visually distinct, suggesting that our model does not simply memorize training data.

We note that some failure modes of our partially-trained models are distinct from those previously observed. Most previous failures involve local artifacts (Odena et al., 2016), images consisting of texture blobs instead of objects (Salimans et al., 2016), or the canonical mode collapse. We observe *class leakage*, where images from one class contain properties of another, as exemplified by Figure 4(d). We also find that many classes on ImageNet are more difficult than others for our model; our model is more successful at generating dogs (which make up a large portion of the dataset, and are mostly distinguished by their texture) than crowds (which comprise a small portion of the dataset and have more large-scale structure). Further discussion is available in Appendix A.

5.2 Additional evaluation on JFT-300M

To confirm that our design choices are effective for even larger and more complex and diverse datasets, we also present results of our system on a subset of JFT-300M (Sun et al., 2017). The full JFT-300M dataset contains 300M real-world images labeled with 18K categories. Since the category distribution is heavily long-tailed, we subsample the dataset to keep only images with the 8.5K most common labels. The resulting dataset contains 292M images – two orders of magnitude larger than ImageNet. For images with multiple labels, we sample a single label randomly and independently whenever an image is sampled. To compute IS and FID for the GANs trained on this dataset, we use an Inception v2 classifier (Szegedy et al., 2016) trained on this dataset. Quantitative results are presented in Table 3. All models are trained with batch size 2048. We compare an ablated version of our model – comparable to SA-GAN (Zhang et al., 2018) but with the larger batch size – against a "full" BigGAN model that makes uses of all of the techniques applied to obtain the best results on ImageNet (shared embedding, skip-z, and orthogonal regularization). Our results show that these techniques substantially improve performance even in the setting of this much larger dataset at the same model capacity (64 base channels). We further show that for a dataset of this scale, we see significant additional improvements from expanding the capacity of our models to 128 base channels, while for ImageNet GANs that additional capacity was not beneficial.

In Figure 19 (Appendix D), we present truncation plots for models trained on this dataset. Unlike for ImageNet, where truncation limits of $\sigma \approx 0$ tend to produce the highest fidelity scores, IS is typically maximized for our JFT-300M models when the truncation value σ ranges from 0.5 to 1. We suspect that this is at least partially due to the intra-class variability of JFT-300M labels, as well as the relative complexity of the image distribution, which includes images with multiple objects at a variety of scales. Interestingly, unlike models trained on ImageNet, where training tends to collapse without heavy regularization (Section 4), the models trained on JFT-300M remain stable over many

hundreds of thousands of iterations. This suggests that moving beyond ImageNet to larger datasets may partially alleviate GAN stability issues.

The improvement over the baseline GAN model that we achieve on this dataset without changes to the underlying models or training and regularization techniques (beyond expanded capacity) demonstrates that our findings extend from ImageNet to datasets with scale and complexity thus far unprecedented for generative models of images.

6 Conclusion

We have demonstrated that Generative Adversarial Networks trained to model natural images of multiple categories highly benefit from scaling up, both in terms of fidelity and variety of the generated samples. As a result, our models set a new level of performance among ImageNet GAN models, improving on the state of the art by a large margin. We have also presented an analysis of the training behavior of large scale GANs, characterized their stability in terms of the singular values of their weights, and discussed the interplay between stability and performance.

Acknowledgments

We would like to thank Kai Arulkumaran, Matthias Bauer, Peter Buchlovsky, Jeffrey Defauw, Sander Dieleman, Ian Goodfellow, Ariel Gordon, Karol Gregor, Dominik Grewe, Chris Jones, Jacob Menick, Augustus Odena, Suman Ravuri, Ali Razavi, Mihaela Rosca, and Jeff Stanway.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

Shane Barratt and Rishi Sharma. A note on the Inception Score. In *arXiv preprint arXiv:1801.01973*, 2018.

Marc G. Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The Cramer distance as a solution to biased Wasserstein gradients. In *arXiv preprint arXiv:1705.10743*, 2017.

Mikolaj Bińkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *ICLR*, 2018.

Andrew Brock, Theodore Lim, J.M. Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. In *ICLR*, 2017.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2016.

Harm de Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron Courville. Modulating early visual processing by language. In *NIPS*, 2017.

Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015.

Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. In *ICLR*, 2017.

William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M. Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: GANs do not need to decrease a divergence at every step. In *ICLR*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Gene Golub and Henk Van der Vorst. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics*, 123:35–65, 2000.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, and Aaron Courville Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

Google. Cloud TPUs. `https://cloud.google.com/tpu/`, 2018.

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of Wasserstein GANs. In *NIPS*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *ICLR*, 2018.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On convergence and stability of GANs. In *arXiv preprint arXiv:1705.07215*, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

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

Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Least squares generative adversarial networks. In *arXiv preprint arXiv:1611.04076*, 2016.

Marco Marchesi. Megapixel size image creation using generative adversarial networks. In *arXiv preprint arXiv:1706.00082*, 2016.

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? In *ICML*, 2018.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. In *arXiv preprint arXiv:1411.1784*, 2014.

Takeru Miyato and Masanori Koyama. cGANs with projection discriminator. In *ICLR*, 2018.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. In *NIPS*, 2016.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In *ICML*, 2017.

Augustus Odena, Jacob Buckman, Catherine Olsson, Tom B. Brown, Christopher Olah, Colin Raffel, and Ian Goodfellow. Is generator conditioning causally related to GAN performance? In *ICML*, 2018.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

Mathijs Pieters and Marco Wiering. Comparing generative adversarial network techniques for image creation and modificatio. In *arXiv preprint arXiv:1803.09093*, 2014.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, and Michael Bernstein. ImageNet large scale visual recognition challenge. *IJCV*, 115:211–252, 2015.

Tim Salimans and Diederik Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *NIPS*, 2016.

Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving GANs using optimal transport. In *ICLR*, 2018.

Andrew Saxe, James McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszr. Amortised map inference for image super-resolution. In *ICLR*, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15:1929–1958, 2014.

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *arXiv preprint arXiv:1511.01844*, 2015.

Dustin Tran, Rajesh Ranganath, and David M. Blei. Hierarchical implicit models and likelihood-free variational inference. In *NIPS*, 2017.

Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018.

Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger B. Grosse. On the quantitative analysis of decoder-based generative models. In *ICLR*, 2017.

Yasin Yazc, Chuan-Sheng Foo, Stefan Winkler, Kim-Hui Yap, Georgios Piliouras, and Vijay Chandrasekhar. The unusual effectiveness of averaging in gan training. In *arXiv preprint arXiv:1806.04498*, 2018.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *arXiv preprint arXiv:1805.08318*, 2018.

APPENDIX A ADDITIONAL SAMPLES, INTERPOLATIONS, AND NEAREST NEIGHBORS FROM IMAGENET MODELS

Figure 5: Samples generated by our BigGAN model at 256×256 resolution.

Figure 6: Samples generated by our BigGAN model at 512×512 resolution.

(a) (b)

Figure 7: Comparing easy classes (a) with difficult classes (b) at 512×512. Classes such as dogs which are largely textural, and common in the dataset, are far easier to model than classes involving unaligned human faces or crowds. Such classes are more dynamic and structured, and often have details to which human observers are more sensitive. The difficulty of modeling global structure is further exacerbated when producing high-resolution images, even with non-local blocks.

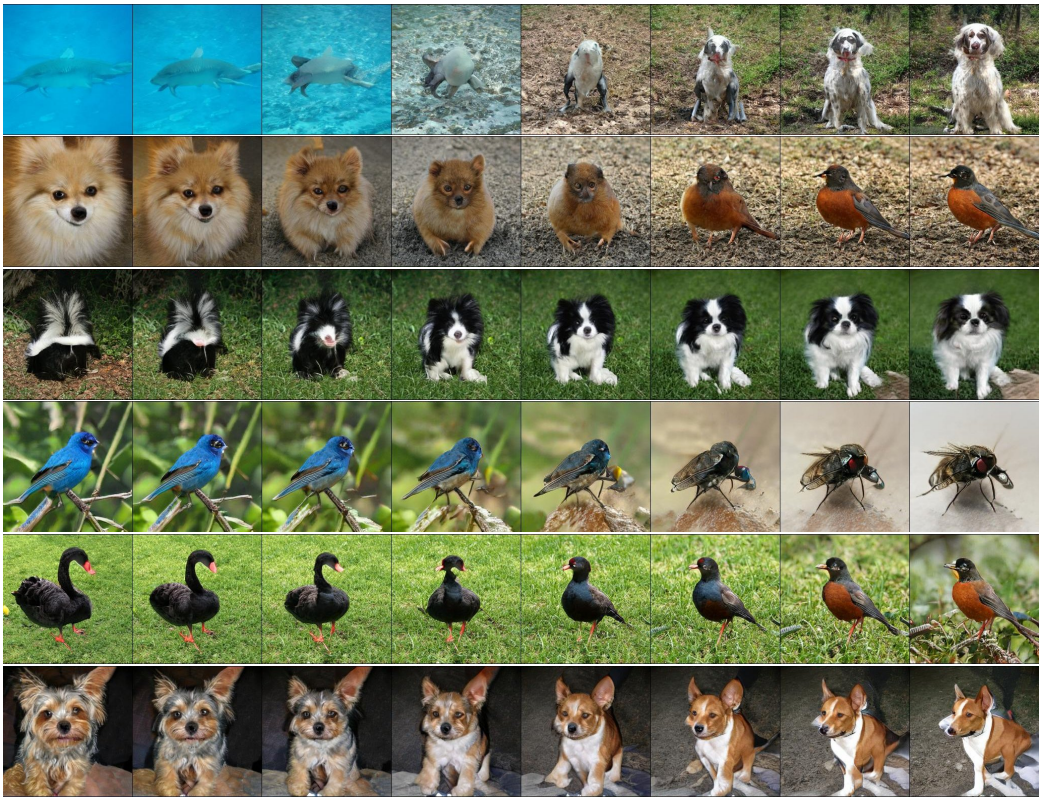


Figure 8: Interpolations between z, c pairs.

Figure 9: Interpolations between c with z held constant. Pose semantics are frequently maintained between endpoints (particularly in the final row). Row 2 demonstrates that grayscale is encoded in the joint z, c space, rather than in z.

Figure 10: Nearest neighbors in VGG-16-fc7 (Simonyan & Zisserman, 2015) feature space. The generated image is in the top left.

Figure 11: Nearest neighbors in ResNet-50-avgpool (He et al., 2016) feature space. The generated image is in the top left.

Figure 12: Nearest neighbors in pixel space. The generated image is in the top left.

Figure 13: Nearest neighbors in VGG-16-fc7 (Simonyan & Zisserman, 2015) feature space. The generated image is in the top left.

Figure 14: Nearest neighbors in ResNet-50-avgpool (He et al., 2016) feature space. The generated image is in the top left.

APPENDIX B ARCHITECTURAL DETAILS

In the BigGAN model (Figure 15), we use the ResNet (He et al., 2016) GAN architecture of (Zhang et al., 2018), which is identical to that used by (Miyato et al., 2018), but with the channel pattern in **D** modified so that the number of filters in the first convolutional layer of each block is equal to the number of output filters (rather than the number of input filters, as in Miyato et al. (2018); Gulrajani et al. (2017)). We use a single shared class embedding in **G**, and skip connections for the latent vector z (skip-z). In particular, we employ hierarchical latent spaces, so that the latent vector z is split along its channel dimension into chunks of equal size (20-D in our case), and each chunk is concatenated to the shared class embedding and passed to a corresponding residual block as a conditioning vector. The conditioning of each block is linearly projected to produce per-sample gains and biases for the BatchNorm layers of the block. The bias projections are zero-centered, while the gain projections are centered at 1. Since the number of residual blocks depends on the image resolution, the full dimensionality of z is 120 for 128×128, 140 for 256×256, and 160 for 512×512 images.

The BigGAN-deep model (Figure 16) differs from BigGAN in several aspects. It uses a simpler variant of skip-z conditioning: instead of first splitting z into chunks, we concatenate the entire z with the class embedding, and pass the resulting vector to each residual block through skip connections. BigGAN-deep is based on residual blocks with bottlenecks (He et al., 2016), which incorporate two additional 1×1 convolutions: the first reduces the number of channels by a factor of 4 before the more expensive 3×3 convolutions; the second produces the required number of output channels. While BigGAN relies on 1×1 convolutions in the skip connections whenever the number of channels needs to change, in BigGAN-deep we use a different strategy aimed at preserving identity throughout the skip connections. In **G**, where the number of channels needs to be reduced, we simply retain the first group of channels and drop the rest to produce the required number of channels. In **D**, where the number of channels should be increased, we pass the input channels unperturbed, and concatenate them with the remaining channels produced by a 1×1 convolution. As far as the network configuration is concerned, the discriminator is an exact reflection of the generator. There are two blocks at each resolution (BigGAN uses one), and as a result BigGAN-deep is four times deeper than BigGAN. Despite their increased depth, the BigGAN-deep models have significantly fewer parameters mainly due to the bottleneck structure of their residual blocks. For example, the 128×128 BigGAN-deep **G** and **D** have 50.4M and 34.6M parameters respectively, while the corresponding original BigGAN models have 70.4M and 88.0M parameters. All BigGAN-deep models use attention at 64×64 resolution, channel width multiplier $ch = 128$, and $z \in \mathbb{R}^{128}$.

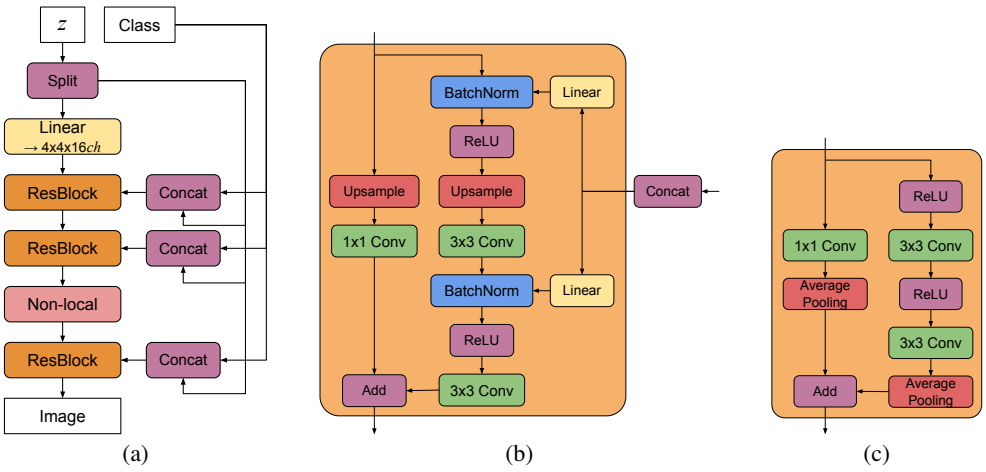


Figure 15: (a) A typical architectural layout for BigGAN's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

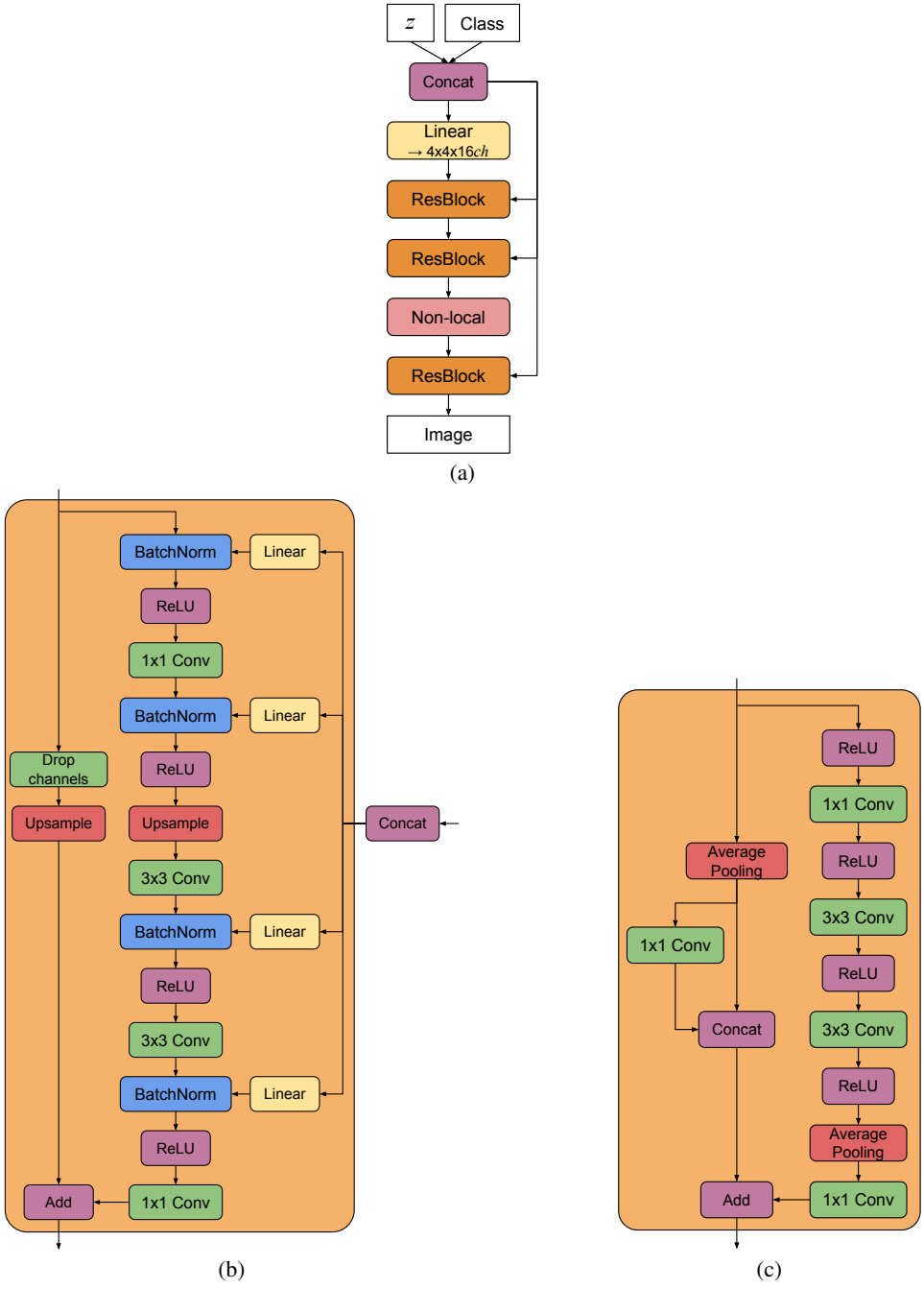


Figure 16: (a) A typical architectural layout for BigGAN-deep's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN-deep's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN-deep's **D**. A *ResBlock* (without *up* or *down*) in BigGAN-deep does not include the *Upsample* or *Average Pooling* layers, and has identity skip connections.

Table 4: BigGAN architecture for 128×128 images. ch represents the channel width multiplier in each network from Table 1.

$z \in \mathbb{R}^{120} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(20 + 128) \rightarrow 4 \times 4 \times 16ch$
ResBlock up $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 4ch$
ResBlock up $4ch \rightarrow 2ch$
Non-Local Block (64×64)
ResBlock up $2ch \rightarrow ch$
BN, ReLU, 3×3 Conv $ch \rightarrow 3$
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{128 \times 128 \times 3}$
ResBlock down $ch \rightarrow 2ch$
Non-Local Block (64×64)
ResBlock down $2ch \rightarrow 4ch$
ResBlock down $4ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 16ch$
ResBlock down $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\rightarrow$ 1)

(b) Discriminator

Table 5: BigGAN architecture for 256×256 images. Relative to the 128×128 architecture, we add an additional ResBlock in each network at 16×16 resolution, and move the non-local block in **G** to 128×128 resolution. Memory constraints prevent us from moving the non-local block in **D**.

$z \in \mathbb{R}^{140} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(20 + 128) \rightarrow 4 \times 4 \times 16ch$
ResBlock up $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 4ch$
ResBlock up $4ch \rightarrow 2ch$
Non-Local Block (128×128)
ResBlock up $2ch \rightarrow ch$
BN, ReLU, 3×3 Conv $ch \rightarrow 3$
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{256 \times 256 \times 3}$
ResBlock down $ch \rightarrow 2ch$
ResBlock down $2ch \rightarrow 4ch$
Non-Local Block (64×64)
ResBlock down $4ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 16ch$
ResBlock down $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\rightarrow$ 1)

(b) Discriminator

Table 6: BigGAN architecture for 512×512 images. Relative to the 256×256 architecture, we add an additional ResBlock at the 512×512 resolution. Memory constraints force us to move the non-local block in both networks back to 64×64 resolution as in the 128×128 pixel setting.

$z \in \mathbb{R}^{160} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(20 + 128) \to 4 \times 4 \times 16ch$
ResBlock up $16ch \to 16ch$
ResBlock up $16ch \to 8ch$
ResBlock up $8ch \to 8ch$
ResBlock up $8ch \to 4ch$
Non-Local Block (64×64)
ResBlock up $4ch \to 2ch$
ResBlock up $2ch \to ch$
ResBlock up $ch \to ch$
BN, ReLU, 3×3 Conv $ch \to 3$
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{512 \times 512 \times 3}$
ResBlock down $ch \to ch$
ResBlock down $ch \to 2ch$
ResBlock down $2ch \to 4ch$
Non-Local Block (64×64)
ResBlock down $4ch \to 8ch$
ResBlock down $8ch \to 8ch$
ResBlock down $8ch \to 16ch$
ResBlock down $16ch \to 16ch$
ResBlock $16ch \to 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\to 1$)

(b) Discriminator

Table 7: BigGAN-deep architecture for 128×128 images.

$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(128 + 128) \to 4 \times 4 \times 16ch$
ResBlock $16ch \to 16ch$
ResBlock up $16ch \to 16ch$
ResBlock $16ch \to 16ch$
ResBlock up $16ch \to 8ch$
ResBlock $8ch \to 8ch$
ResBlock up $8ch \to 4ch$
ResBlock $4ch \to 4ch$
ResBlock up $4ch \to 2ch$
Non-Local Block (64×64)
ResBlock $2ch \to 2ch$
ResBlock up $2ch \to ch$
BN, ReLU, 3×3 Conv $ch \to 3$
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{128 \times 128 \times 3}$
3×3 Conv $3 \to ch$
ResBlock down $ch \to 2ch$
ResBlock $2ch \to 2ch$
Non-Local Block (64×64)
ResBlock down $2ch \to 4ch$
ResBlock $4ch \to 4ch$
ResBlock down $4ch \to 8ch$
ResBlock $8ch \to 8ch$
ResBlock down $8ch \to 16ch$
ResBlock $16ch \to 16ch$
ResBlock down $16ch \to 16ch$
ResBlock $16ch \to 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\to 1$)

(b) Discriminator

Table 8: BigGAN-deep architecture for 256×256 images.

$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$ Embed(y) $\in \mathbb{R}^{128}$
Linear $(128 + 128) \rightarrow 4 \times 4 \times 16ch$
ResBlock $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 4ch$
Non-Local Block (64×64)
ResBlock $4ch \rightarrow 4ch$
ResBlock up $4ch \rightarrow 2ch$
ResBlock $2ch \rightarrow 2ch$
ResBlock up $2ch \rightarrow ch$
BN, ReLU, 3×3 Conv $ch \rightarrow 3$
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{256 \times 256 \times 3}$
3×3 Conv $3 \rightarrow ch$
ResBlock down $ch \rightarrow 2ch$
ResBlock $2ch \rightarrow 2ch$
ResBlock down $2ch \rightarrow 4ch$
ResBlock $4ch \rightarrow 4ch$
Non-Local Block (64×64)
ResBlock down $4ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ResBlock down $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ReLU, Global sum pooling
Embed(y)$\cdot \boldsymbol{h}$ + (linear $\rightarrow$ 1)

(b) Discriminator

Table 9: BigGAN-deep architecture for 512×512 images.

$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(128 + 128) \rightarrow 4 \times 4 \times 16ch$
ResBlock $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 4ch$
Non-Local Block (64×64)
ResBlock $4ch \rightarrow 4ch$
ResBlock up $4ch \rightarrow 2ch$
ResBlock $2ch \rightarrow 2ch$
ResBlock up $2ch \rightarrow ch$
ResBlock $ch \rightarrow ch$
ResBlock up $ch \rightarrow ch$
BN, ReLU, 3×3 Conv $ch \rightarrow 3$
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{512 \times 512 \times 3}$
3×3 Conv $3 \rightarrow ch$
ResBlock down $ch \rightarrow ch$
ResBlock $ch \rightarrow ch$
ResBlock down $ch \rightarrow 2ch$
ResBlock $2ch \rightarrow 2ch$
ResBlock down $2ch \rightarrow 4ch$
ResBlock $4ch \rightarrow 4ch$
Non-Local Block (64×64)
ResBlock down $4ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 8ch$
ResBlock $8ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ResBlock down $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\rightarrow$ 1)

(b) Discriminator

APPENDIX C EXPERIMENTAL DETAILS

Our basic setup follows SA-GAN (Zhang et al., 2018), and is implemented in TensorFlow (Abadi et al., 2016). We employ the architectures detailed in Appendix B, with non-local blocks inserted at a single stage in each network. Both **G** and **D** networks are initialized with Orthogonal Initialization (Saxe et al., 2014). We use Adam optimizer (Kingma & Ba, 2014) with $\beta_1 = 0$ and $\beta_2 = 0.999$ and a constant learning rate. For BigGAN models at all resolutions, we use $2 \cdot 10^{-4}$ in **D** and $5 \cdot 10^{-5}$ in **G**. For BigGAN-deep, we use the learning rate of $2 \cdot 10^{-4}$ in **D** and $5 \cdot 10^{-5}$ in **G** for 128×128 models, and $2.5 \cdot 10^{-5}$ in both **D** and **G** for 256×256 and 512×512 models. We experimented with the number of **D** steps per **G** step (varying it from 1 to 6) and found that two **D** steps per **G** step gave the best results.

We use an exponential moving average of the weights of **G** at sampling time, with a decay rate set to 0.9999. We employ cross-replica BatchNorm (Ioffe & Szegedy, 2015) in **G**, where batch statistics are aggregated across all devices, rather than a single device as in standard implementations. Spectral Normalization (Miyato et al., 2018) is used in both **G** and **D**, following SA-GAN (Zhang et al., 2018). We train on a Google TPU v3 Pod, with the number of cores proportional to the resolution: 128 for 128×128, 256 for 256×256, and 512 for 512×512. Training takes between 24 and 48 hours for most models. We increase ϵ from the default 10^{-8} to 10^{-4} in BatchNorm and Spectral Norm to mollify low-precision numerical issues. We preprocess data by cropping along the long edge and rescaling to a given resolution with area resampling.

C.1 BATCHNORM STATISTICS AND SAMPLING

The default behavior with batch normalized classifier networks is to use a running average of the activation moments at test time. Previous works (Radford et al., 2016) have instead used batch statistics when sampling images. While this is not technically an invalid way to sample, it means that results are dependent on the test batch size (and how many devices it is split across), and further complicates reproducibility.

We find that this detail is extremely important, with changes in test batch size producing drastic changes in performance. This is further exacerbated when one uses exponential moving averages of **G**'s weights for sampling, as the BatchNorm running averages are computed with non-averaged weights and are poor estimates of the activation statistics for the averaged weights.

To counteract both these issues, we employ "standing statistics," where we compute activation statistics at sampling time by running the **G** through multiple forward passes (typically 100) each with different batches of random noise, and storing means and variances aggregated across all forward passes. Analogous to using running statistics, this results in **G**'s outputs becoming invariant to batch size and the number of devices, even when producing a single sample.

C.2 CIFAR-10

We run our networks on CIFAR-10 (Krizhevsky & Hinton, 2009) using the settings from Table 1, row 8, and achieve an IS of 9.22 and an FID of 14.73 without truncation.

C.3 INCEPTION SCORES OF IMAGENET IMAGES

We compute the IS for both the training and validation sets of ImageNet. At 128×128 the training data has an IS of 233, and the validation data has an IS of 166. At 256×256 the training data has an IS of 377, and the validation data has an IS of 234. At 512×512 the training data has an IS of 348, and the validation data has an IS of 241. The discrepancy between training and validation scores is due to the Inception classifier having been trained on the training data, resulting in high-confidence outputs that are preferred by the Inception Score.

APPENDIX D ADDITIONAL PLOTS

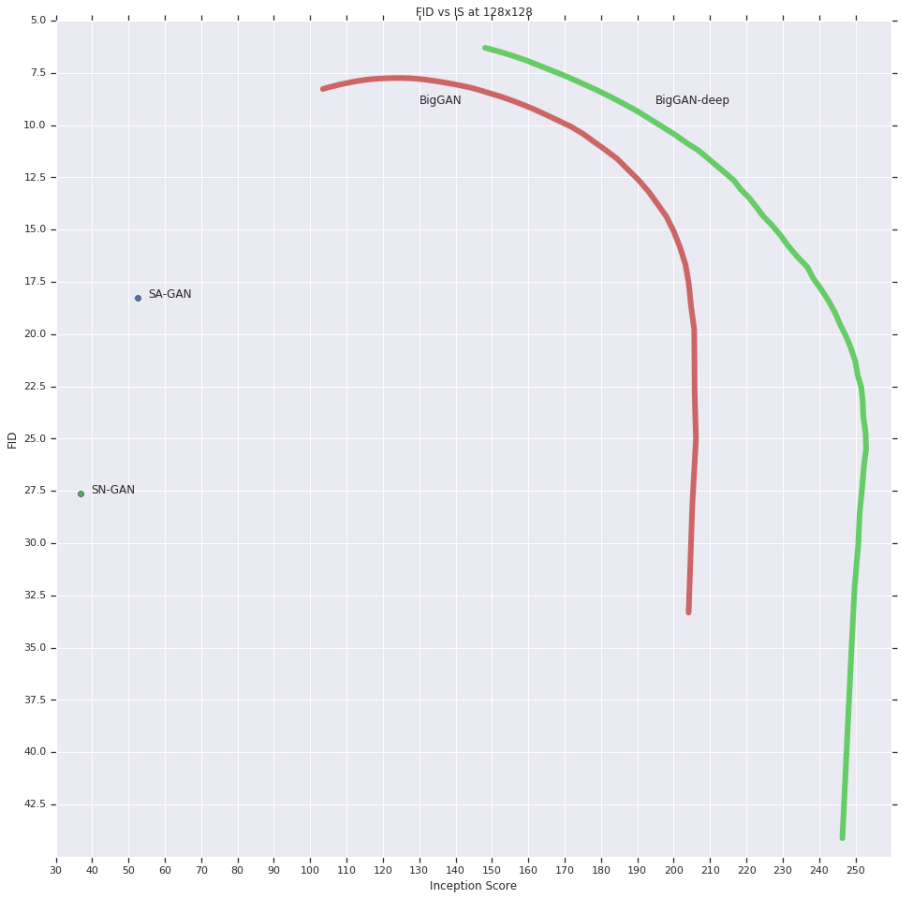


Figure 17: IS vs. FID at 128×128. Scores are averaged across three random seeds.

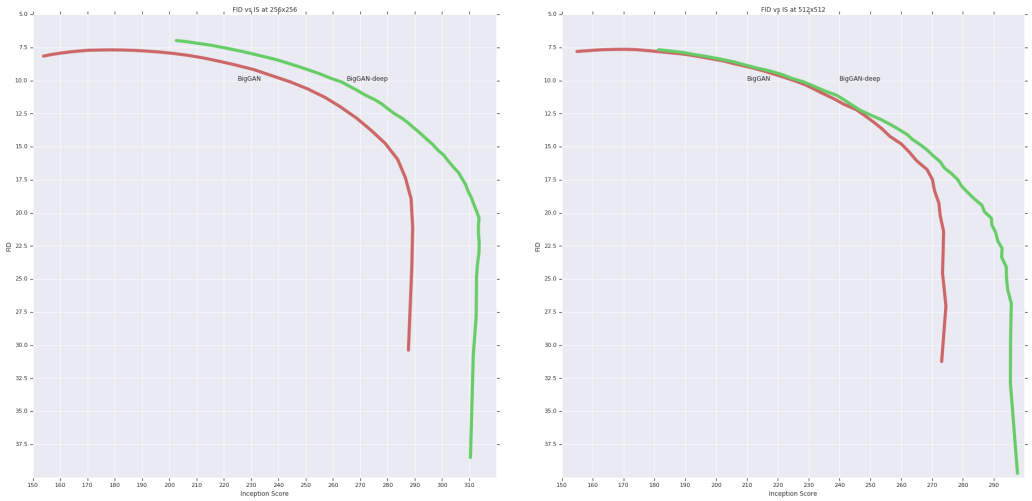


Figure 18: IS vs. FID at 256 and 512 pixels. Scores are averaged across three random seeds for 256.

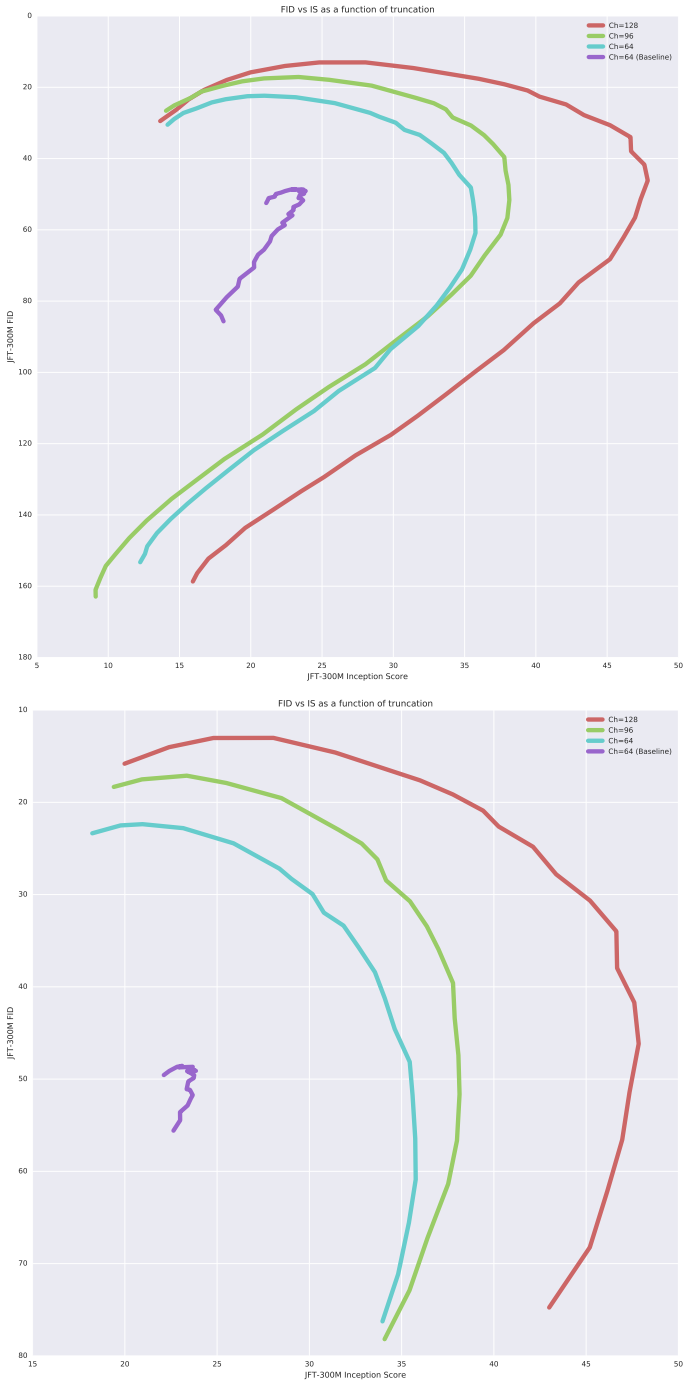


Figure 19: JFT-300M IS vs. FID at 256×256. We show truncation values from $\sigma = 0$ to $\sigma = 2$ (top) and from $\sigma = 0.5$ to $\sigma = 1.5$ (bottom). Each curve corresponds to a row in Table 3. The curve labeled with *baseline* corresponds to the first row (with orthogonal regularization and other techniques disabled), while the rest correspond to rows 2-4 – the same architecture at different capacities (*Ch*).

APPENDIX E CHOOSING LATENT SPACES

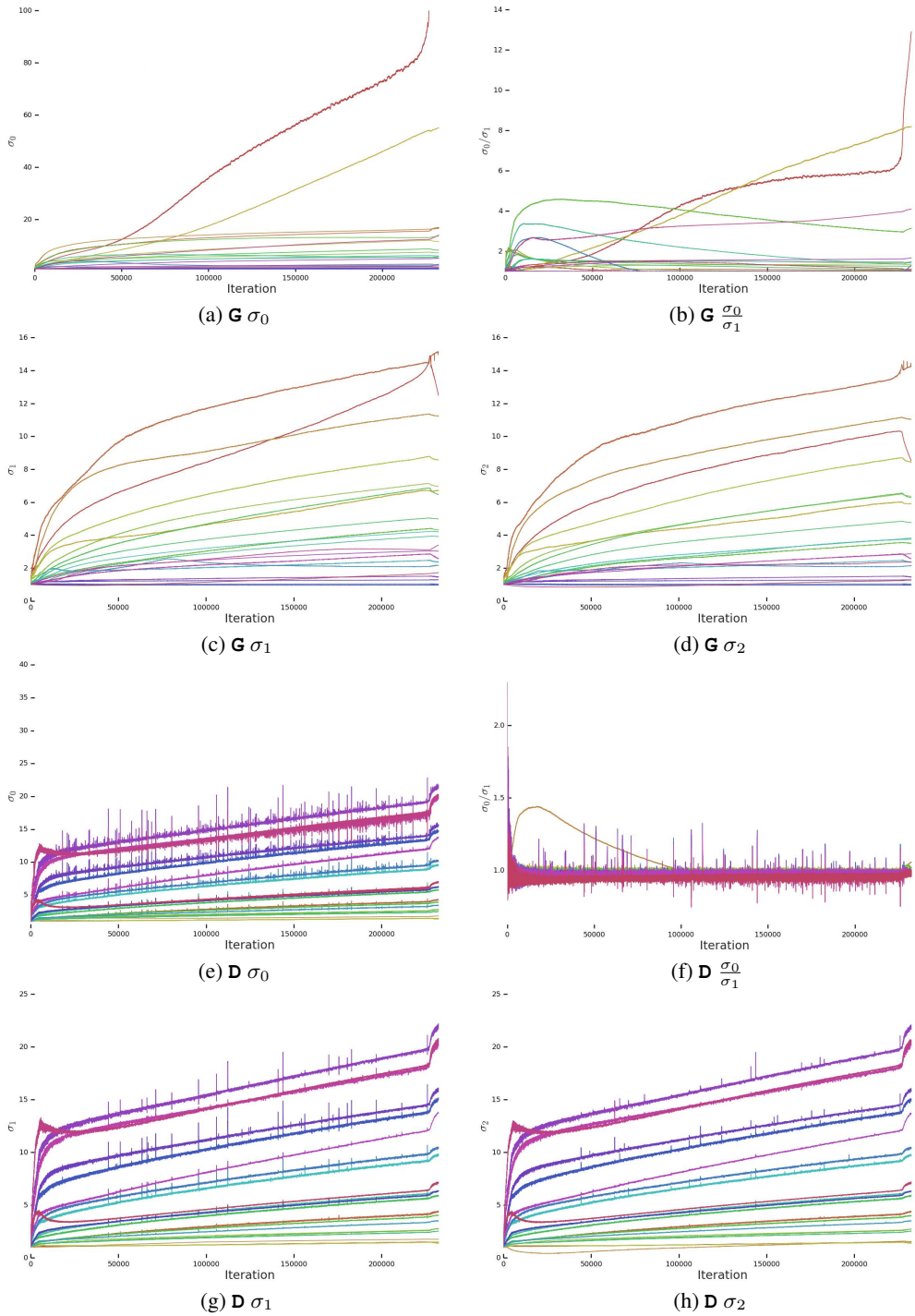
While most previous work has employed $\mathcal{N}(0, I)$ or $\mathcal{U}[-1, 1]$ as the prior for z (the noise input to $\mathbf{G}$), we are free to choose any latent distribution from which we can sample. We explore the choice of latents by considering an array of possible designs, described below. For each latent, we provide the intuition behind its design and briefly describe how it performs when used as a drop-in replacement for $z \sim \mathcal{N}(0, I)$ in an SA-GAN baseline. As the Truncation Trick proved more beneficial than switching to any of these latents, we do not perform a full ablation study, and employ $z \sim \mathcal{N}(0, I)$ for our main results to take full advantage of truncation. The two latents which we find to work best without truncation are Bernoulli $\{0, 1\}$ and Censored Normal $\max(\mathcal{N}(0, I), 0)$, both of which improve speed of training and lightly improve final performance, but are less amenable to truncation.

We also ablate the choice of latent space dimensonality (which by default is $z \in \mathbb{R}^{128}$), finding that we are able to successfully train with latent dimensions as low as $z \in \mathbb{R}^8$, and that with $z \in \mathbb{R}^{32}$ we see a minimal drop in performance. While this is substantially smaller than many previous works, direct comparison to single-class networks (such as those in Karras et al. (2018), which employ a $z \in \mathbb{R}^{512}$ latent space on a highly constrained dataset with 30,000 images) is improper, as our networks have additional class information provided as input.

LATENTS

- $\mathcal{N}(0, I)$. A standard choice of the latent space which we use in the main experiments.
- $\mathcal{U}[-1, 1]$. Another standard choice; we find that it performs similarly to $\mathcal{N}(0, I)$.
- Bernoulli $\{0, 1\}$. A discrete latent might reflect our prior that underlying factors of variation in natural images are not continuous, but discrete (one feature is present, another is not). This latent outperforms $\mathcal{N}(0, I)$ (in terms of IS) by 8% and requires 60% fewer iterations.
- $\max(\mathcal{N}(0, I), 0)$, also called Censored Normal. This latent is designed to introduce sparsity in the latent space (reflecting our prior that certain latent features are sometimes present and sometimes not), but also allow those latents to vary continuously, expressing different degrees of intensity for latents which are active. This latent outperforms $\mathcal{N}(0, I)$ (in terms of IS) by 15-20% and tends to require fewer iterations.
- Bernoulli $\{-1, 1\}$. This latent is designed to be discrete, but not sparse (as the network can learn to activate in response to negative inputs). This latent performs near-identically to $\mathcal{N}(0, I)$.
- Independent Categorical in $\{-1, 0, 1\}$, with equal probability. This distribution is chosen to be discrete and have sparsity, but also to allow latents to take on both positive and negative values. This latent performs near-identically to $\mathcal{N}(0, I)$.
- $\mathcal{N}(0, I)$ multiplied by Bernoulli $\{0, 1\}$. This distribution is chosen to have continuous latent factors which are also sparse (with a peak at zero), similar to Censored Normal but not constrained to be positive. This latent performs near-identically to $\mathcal{N}(0, I)$.
- Concatenating $\mathcal{N}(0, I)$ and Bernoulli $\{0, 1\}$, each taking half of the latent dimensions. This is inspired by Chen et al. (2016), and is chosen to allow some factors of variation to be discrete, while others are continuous. This latent outperforms $\mathcal{N}(0, I)$ by around 5%.
- Variance annealing: we sample from $\mathcal{N}(0, \sigma I)$, where σ is allowed to vary over training. We compared a variety of piecewise schedules and found that starting with $\sigma = 2$ and annealing towards $\sigma = 1$ over the course of training mildly improved performance. The space of possible variance schedules is large, and we did not explore it in depth – we suspect that a more principled or better-tuned schedule could more strongly impact performance.
- Per-sample variable variance: $\mathcal{N}(0, \sigma_i I)$, where $\sigma_i \sim \mathcal{U}[\sigma_l, \sigma_h]$ independently for each sample i in a batch, and (σ_l, σ_h) are hyperparameters. This distribution was chosen to try and improve amenability to the Truncation Trick by feeding the network noise samples with non-constant variance. This did not appear to affect performance, but we did not explore it in depth. One might also consider scheduling (σ_l, σ_h), similar to variance annealing.

APPENDIX F MONITORED TRAINING STATISTICS



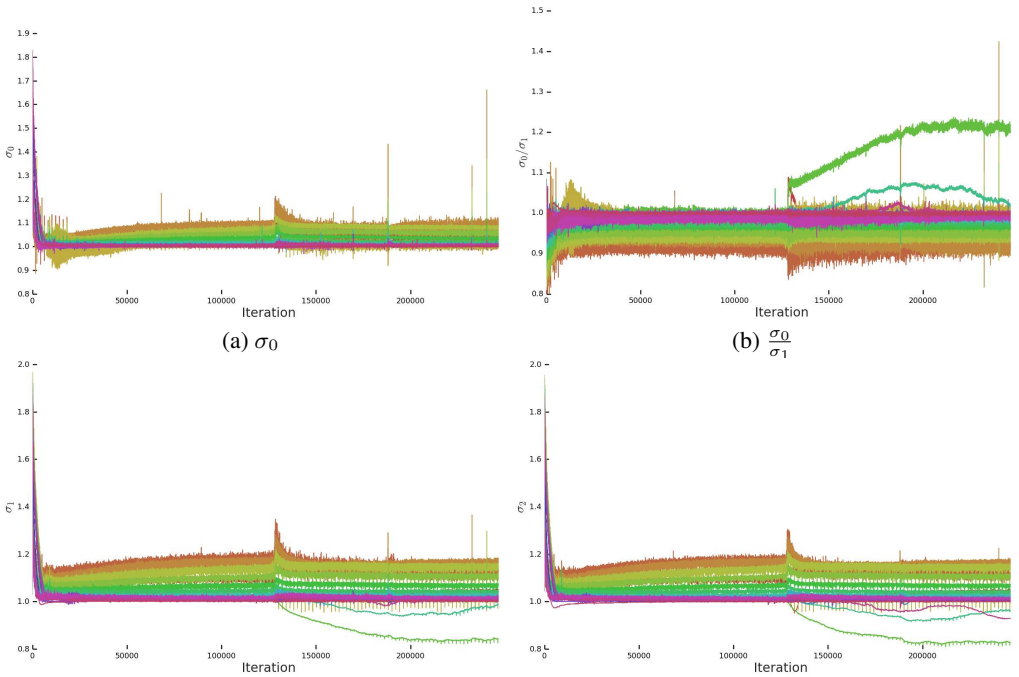
(a) **G** σ_0 (b) **G** $\frac{\sigma_0}{\sigma_1}$

(c) **G** σ_1 (d) **G** σ_2

(e) **D** σ_0 (f) **D** $\frac{\sigma_0}{\sigma_1}$

(g) **D** σ_1 (h) **D** σ_2

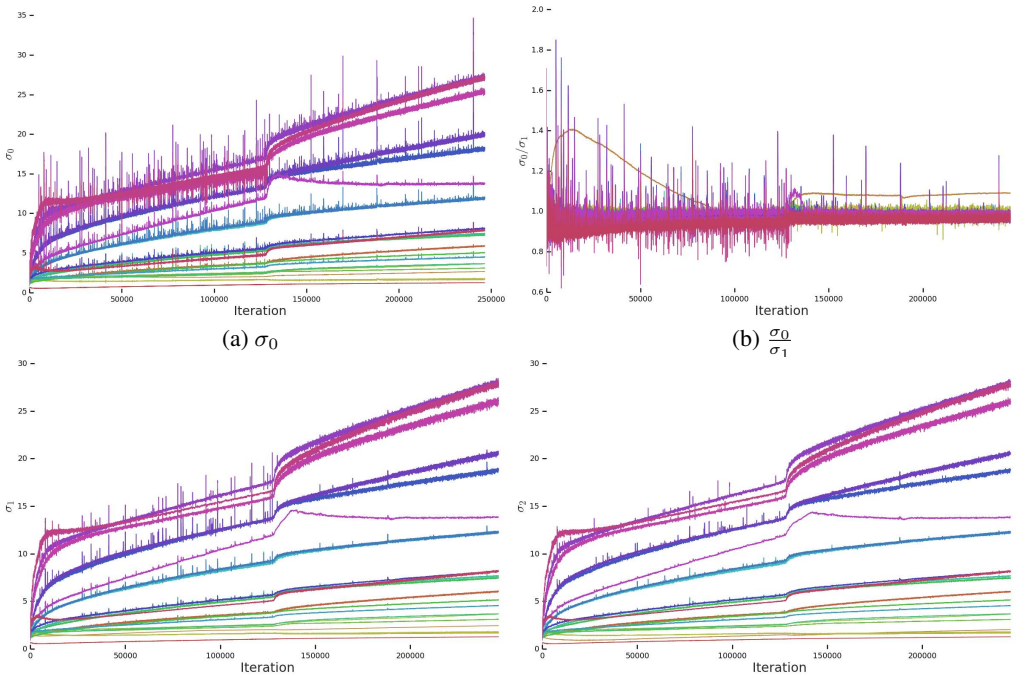
Figure 20: Training statistics for a typical model without special modifications. Collapse occurs after 200000 iterations.



(a) σ_0 (b) $\frac{\sigma_0}{\sigma_1}$

(c) σ_1 (d) σ_2

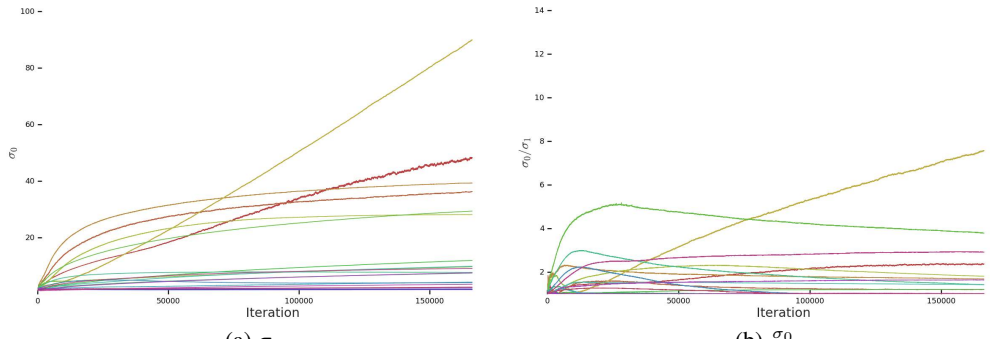
Figure 21: **G** training statistics with σ_0 in **G** regularized towards 1. Collapse occurs after 125000 iterations.



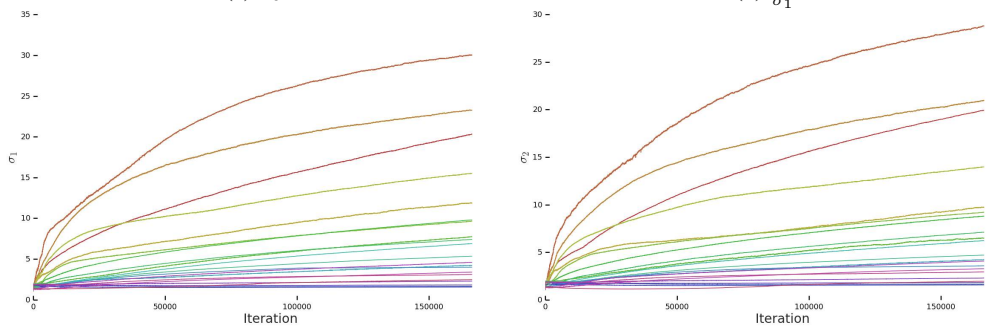
(a) σ_0 (b) $\frac{\sigma_0}{\sigma_1}$

(c) σ_1 (d) σ_2

Figure 22: **D** training statistics with σ_0 in **G** regularized towards 1. Collapse occurs after 125000 iterations.

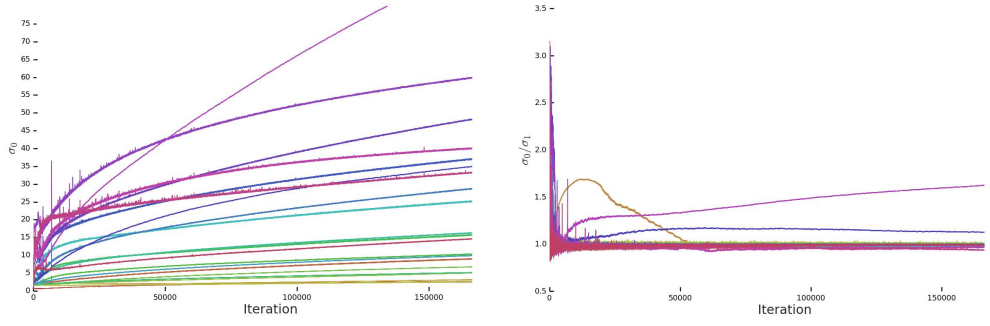


(a) σ_0 (b) $\frac{\sigma_0}{\sigma_1}$

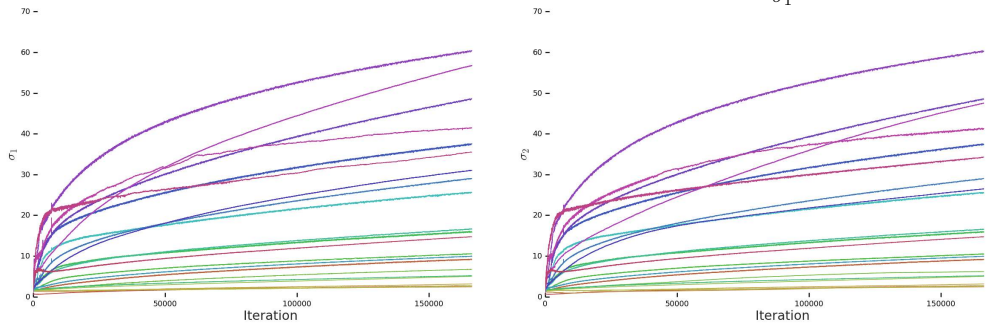


(c) σ_1 (d) σ_2

Figure 23: **G** training statistics with an R1 Gradient Penalty of strength 10 on **D**. This model does not collapse, but only reaches a maximum IS of 55.

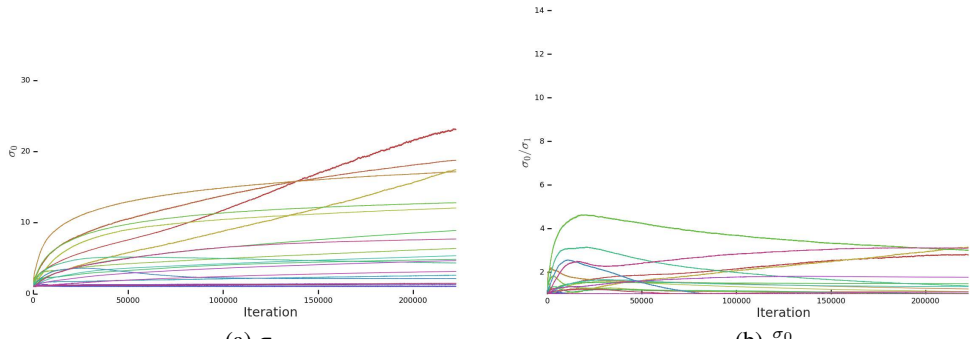


(a) σ_0 (b) $\frac{\sigma_0}{\sigma_1}$

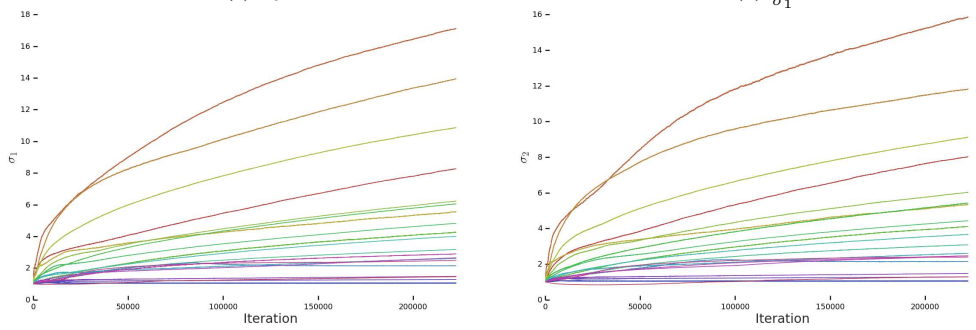


(c) σ_1 (d) σ_2

Figure 24: **D** training statistics with an R1 Gradient Penalty of strength 10 on **D**. This model does not collapse, but only reaches a maximum IS of 55.

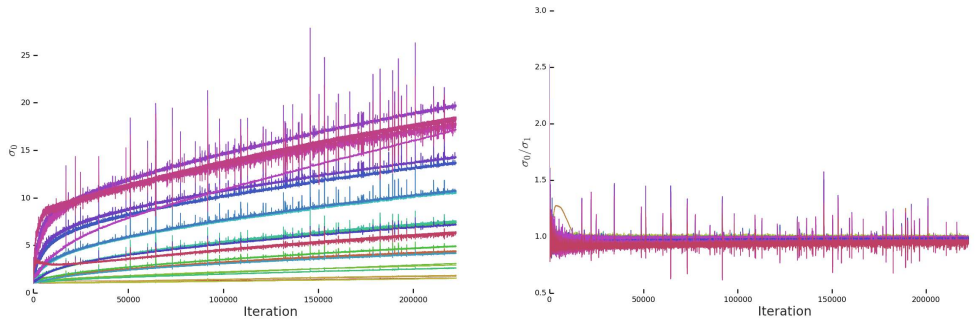


(a) σ_0 (b) $\frac{\sigma_0}{\sigma_1}$

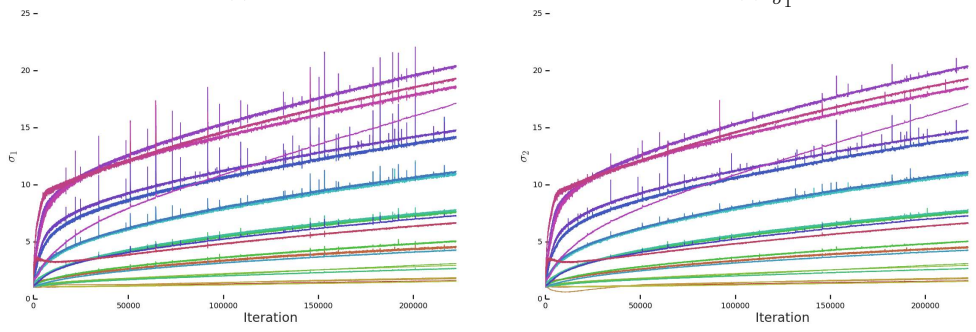


(c) σ_1 (d) σ_2

Figure 25: **G** training statistics with Dropout (keep probability 0.8) applied to the last feature layer of **D**. This model does not collapse, but only reaches a maximum IS of 70.

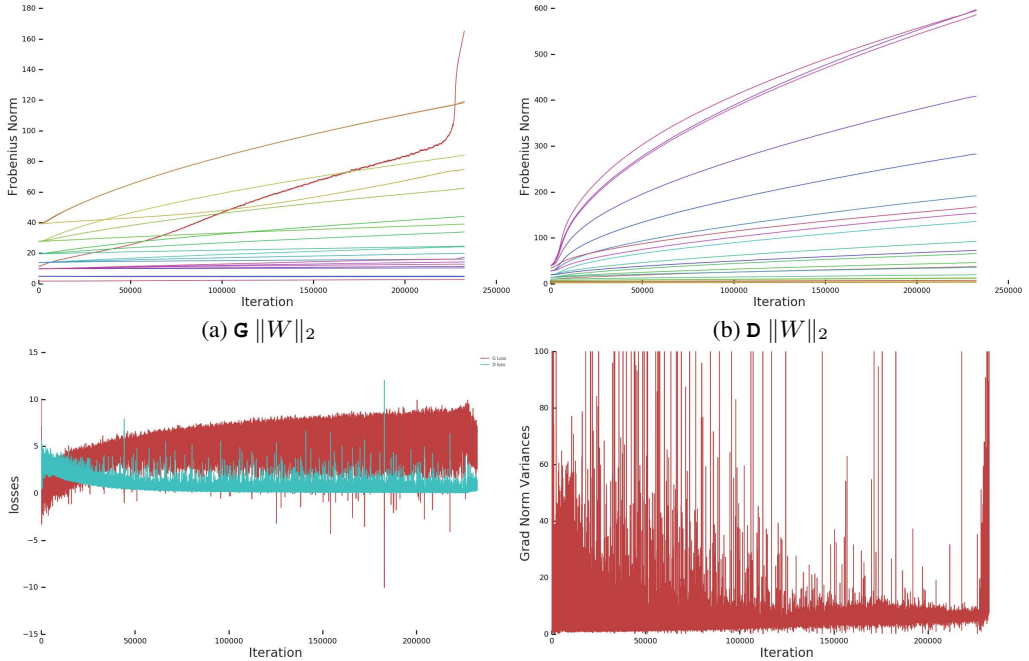


(a) σ_0 (b) $\frac{\sigma_0}{\sigma_1}$



(c) σ_1 (d) σ_2

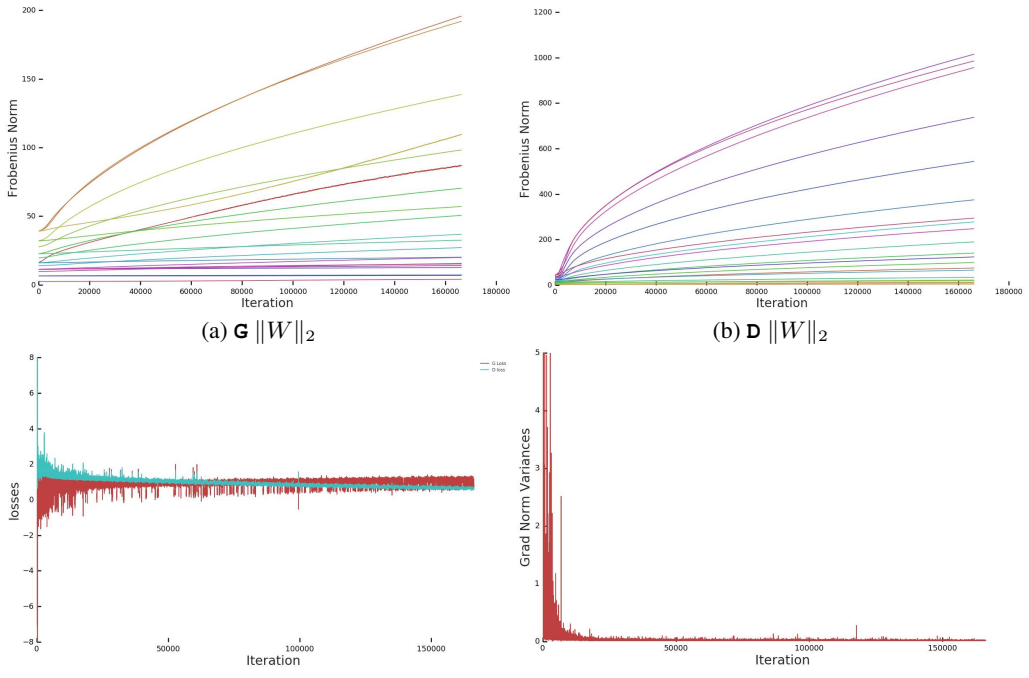
Figure 26: **D** training statistics with Dropout (keep probability 0.8) applied to the last feature layer of **D**. This model does not collapse, but only reaches a maximum IS of 70.



(a) **G** $\|W\|_2$ (b) **D** $\|W\|_2$

(c) losses (d) Variance of all gradient norms in **G** and **D**

Figure 27: Additional training statistics for a typical model without special modifications. Collapse occurs after 200000 iterations.



(a) **G** $\|W\|_2$ (b) **D** $\|W\|_2$

(c) losses (d) Variance of all gradient norms in **G** and **D**

Figure 28: Additional training statistics with an R1 Gradient Penalty of strength 10 on **D**. This model does not collapse, but only reaches a maximum IS of 55.

APPENDIX G ADDITIONAL DISCUSSION: STABILITY AND COLLAPSE

In this section, we present and discuss additional investigations into the stability of our models, expanding upon the discussion in Section 4.

G.1 INTERVENING BEFORE COLLAPSE

The symptoms of collapse are sharp and sudden, with sample quality dropping from its peak to its lowest value over the course of a few hundred iterations. We can detect this collapse when the singular values in **G** explode, but while the (unnormalized) singular values grow throughout training, there is no consistent threshold at which collapse occurs. This raises the question of whether it is possible to prevent or delay collapse by taking a model checkpoint several thousand iterations before collapse, and continuing training with some hyperparameters modified (e.g., the learning rate).

We conducted a range of intervention experiments wherein we took checkpoints of a collapsed model ten or twenty thousand iterations before collapse, changed some aspect of the training setup, then observed whether collapse occurred, when it occurred relative to the original collapse, and the final performance attained at collapse.

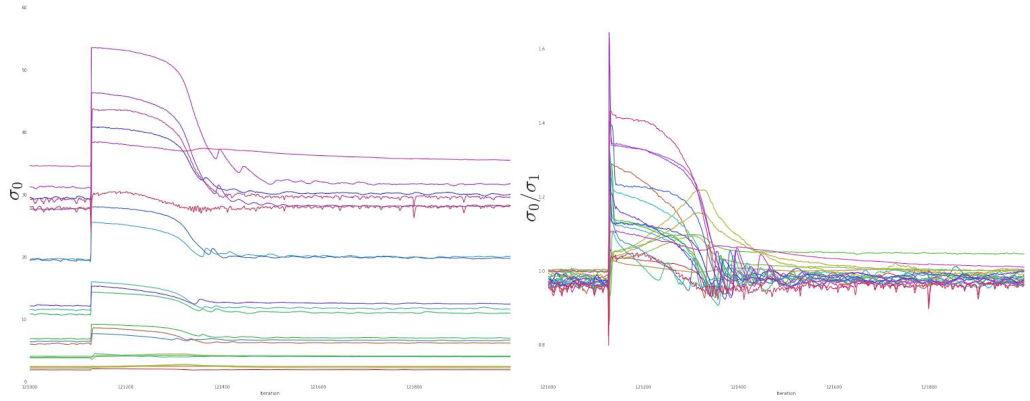
We found that increasing the learning rates (relative to their initial values) in either **G** or **D**, or both **G** and **D**, led to immediate collapse. This occurred even when doubling the learning rates from $2 \cdot 10^{-4}$ in **D** and $5 \cdot 10^{-5}$ in **G**, to $4 \cdot 10^{-4}$ in **D** and $1 \cdot 10^{-4}$ in **G**, a setting which is not normally unstable when used as the initial learning rates. We also tried changing the momentum terms (Adam's β_1 and β_2), or resetting the momentum vectors to zero, but this tended to either make no difference or, when increasing the momentum, cause immediate collapse.

We found that decreasing the learning rate in **G**, but keeping the learning rate in **D** unchanged could delay collapse (in some cases by over one hundred thousand iterations), but also crippled training—once the learning rate in **G** was decayed, performance either stayed constant or slowly decayed. Conversely, reducing the learning rate in **D** while keeping **G**'s learning rate led to immediate collapse. We hypothesize that this is because of the need for **D** to remain optimal throughout training—if its learning rate is reduced, it can no longer "keep up" with **G**, and training collapses. With this in mind, we also tried increasing the number of **D** steps per **G** step, but this either had no effect, or delayed collapse at the cost of crippling training (similar to decaying **G**'s learning rate).

To further illuminate these dynamics, we construct two additional intervention experiments, one where we freeze **G** before collapse (by ceasing all parameter updates) and observe whether **D** remains stable, and the reverse, where we freeze **D** before collapse and observe whether **G** remains stable. We find that when **G** is frozen, **D** remains stable, and slowly reduces both components of its loss towards zero. However, when **D** is frozen, **G** immediately and dramatically collapses, maxing out **D**'s loss to values upwards of 300, compared to the normal range of 0 to 3.

This leads to two conclusions: first, as has been noted in previous works (Miyato et al., 2018; Gulrajani et al., 2017; Zhang et al., 2018), **D** must remain optimal with respect to **G** both for stability and to provide useful gradient information. The consequence of **G** being allowed to win the game is a complete breakdown of the training process, regardless of **G**'s conditioning or optimization settings. Second, favoring **D** over **G** (either by training it with a larger learning rate, or for more steps) is insufficient to ensure stability even if **D** is well-conditioned. This suggests either that in practice, an optimal **D** is necessary but insufficient for training stability, or that some aspect of the system results in **D** not being trained towards optimality. With the latter possibility in mind, we take a closer look at the noise in **D**'s spectra in the following section.

G.2 SPIKES IN THE DISCRIMINATOR'S SPECTRA



(a) **D** σ_0 (b) **D** $\frac{\sigma_0}{\sigma_1}$

Figure 29: A closeup of **D**'s spectra at a noise spike.

If some element of **D**'s training process results in undesirable dynamics, it follows that the behavior of **D**'s spectra may hold clues as to what that element is. The top three singular values of **D** differ from **G**'s in that they have a large noise component, tend to grow throughout training but only show a small response to collapse, and the ratio of the first two singular values tends to be centered around one, suggesting that the spectra of **D** have a slow decay. When viewed up close (Figure 29), the noise spikes resemble an impulse response: at each spike, the spectra jump upwards, then slowly decrease, with some oscillation.

One possible explanation is that this behavior is a consequence of **D** memorizing the training data, as suggested by experiments in Section 4.2. As it approaches perfect memorization, it receives less and less signal from real data, as both the original GAN loss and the hinge loss provide zero gradients when **D** outputs a confident and correct prediction for a given example. If the gradient signal from real data attenuates to zero, this can result in **D** eventually becoming biased due to exclusively received gradients that encourage its outputs to be negative. If this bias passes a certain threshold, **D** will eventually misclassify a large number of real examples and receive a large gradient encouraging positive outputs, resulting in the observed impulse responses.

This argument suggests several fixes. First, one might consider an unbounded loss (such as the Wasserstein loss (Arjovsky et al., 2017)) which would not suffer this gradient attentuation. We found that even with gradient penalties and brief re-tuning of optimizer hyperparameters, our models did not stably train for more than a few thousand iterations with this loss. We instead explored changing the margin of the hinge loss as a partial compromise: for a given model and minibatch of data, increasing the margin will result in more examples falling within the margin, and thus contributing to the loss.[3]. Training with a smaller margin (by a factor of 2) measurably reduces performance, but training with a larger margin (by up to a factor of 3) does not prevent collapse or reduce the noise in **D**'s spectra. Increasing the margin beyond 3 results in unstable training similar to using the Wasserstein loss. Finally, the memorization argument might suggest that using a smaller **D** or using dropout in **D** would improve training by reducing its capacity to memorize, but in practice this degrades training.

[3]Unconstrained models could easily learn a different output scale to account for this margin, but the use of Spectral Normalization constrains our models and makes the specific selection of the margin meaningful.

APPENDIX H NEGATIVE RESULTS

We explored a range of novel and existing techniques which ended up degrading or otherwise not affecting performance in our setting. We report them here; our evaluations for this section are not as thorough as those for the main architectural choices.

Our intention in reporting these results is to save time for future work, and to give a more complete picture of our attempts to improve performance or stability. We note, however, that these results must be understood to be specific to the particular setup we used. A pitfall of reporting negative results is that one might report that a particular technique doesn't work, when the reality is that this technique did not have the desired effect when applied in a particular way to a particular problem. Drawing overly general conclusions might close off potentially fruitful avenues of research.

- We found that doubling the depth (by inserting an additional Residual block after every up- or down-sampling block) hampered performance.
- We experimented with sharing class embeddings between both **G** and **D** (as opposed to just within **G**). This is accomplished by replacing **D**'s class embedding with a projection from **G**'s embeddings, as is done in **G**'s BatchNorm layers. In our initial experiments this seemed to help and accelerate training, but we found this trick scaled poorly and was sensitive to optimization hyperparameters, particularly the choice of number of **D** steps per **G** step.
- We tried replacing BatchNorm in **G** with WeightNorm (Salimans & Kingma, 2016), but this crippled training. We also tried removing BatchNorm and only having Spectral Normalization, but this also crippled training.
- We tried adding BatchNorm to **D** (both class-conditional and unconditional) in addition to Spectral Normalization, but this crippled training.
- We tried varying the choice of location of the attention block in **G** and **D** (and inserting multiple attention blocks at different resolutions) but found that at 128×128 there was no noticeable benefit to doing so, and compute and memory costs increased substantially. We found a benefit to moving the attention block up one stage when moving to 256×256, which is in line with our expectations given the increased resolution.
- We tried using filter sizes of 5 or 7 instead of 3 in either **G** or **D** or both. We found that having a filter size of 5 in **G** only provided a small improvement over the baseline but came at an unjustifiable compute cost. All other settings degraded performance.
- We tried varying the dilation for convolutional filters in both **G** and **D** at 128×128, but found that even a small amount of dilation in either network degraded performance.
- We tried bilinear upsampling in **G** in place of nearest-neighbors upsampling, but this degraded performance.
- In some of our models, we observed class-conditional mode collapse, where the model would only output one or two samples for a subset of classes but was still able to generate samples for all other classes. We noticed that the collapsed classes had embedings which had become very large relative to the other embeddings, and attempted to ameliorate this issue by applying weight decay to the shared embedding only. We found that small amounts of weight decay (10^{-6}) instead degraded performance, and that only even smaller values (10^{-8}) did not degrade performance, but these values were also too small to prevent the class vectors from exploding. Higher-resolution models appear to be more resilient to this problem, and none of our final models appear to suffer from this type of collapse.
- We experimented with using MLPs instead of linear projections from **G**'s class embeddings to its BatchNorm gains and biases, but did not find any benefit to doing so. We also experimented with Spectrally Normalizing these MLPs, and with providing these (and the linear projections) with a bias at their output, but did not notice any benefit.
- We tried gradient norm clipping (both the global variant typically used in recurrent networks, and a local version where the clipping value is determined on a per-parameter basis) but found this did not alleviate instability.

APPENDIX I HYPERPARAMETERS

We performed various hyperparameter sweeps in this work:

- We swept the Cartesian product of the learning rates for each network through [10^{-5}, $5 \cdot 10^{-5}$, 10^{-4}, $2 \cdot 10^{-4}$, $4 \cdot 10^{-4}$, $8 \cdot 10^{-4}$, 10^{-3}], and initially found that the SA-GAN settings (**G**'s learning rate 10^{-4}, **D**'s learning rate $4 \cdot 10^{-4}$) were optimal at lower batch sizes; we did not repeat this sweep at higher batch sizes but did try halving and doubling the learning rate, arriving at the halved settings used for our experiments.
- We swept the R1 gradient penalty strength through [10^{-3}, 10^{-2}, 10^{-1}, 0.5, 1, 2, 3, 5, 10]. We find that the strength of the penalty correlates negatively with performance, but that settings above 0.5 impart training stability.
- We swept the keep probabilities for DropOut in the final layer of **D** through [0.5, 0.6, 0.7, 0.8, 0.9, 0.95]. We find that DropOut has a similar stabilizing effect to R1 but also degrades performance.
- We swept **D**'s Adam β_1 parameter through [0.1, 0.2, 0.3, 0.4, 0.5] and found it to have a light regularization effect similar to DropOut, but not to significantly improve results. Higher β_1 terms in either network crippled training.
- We swept the strength of the modified Orthogonal Regularization penalty in **G** through [10^{-5}, $5 \cdot 10^{-5}$, 10^{-4}, $5 \cdot 10^{-4}$, 10^{-3}, 10^{-2}], and selected 10^{-4}.

Train Sparsely, Generate Densely: Memory-efficient Unsupervised Training of High-resolution Temporal GAN

Masaki Saito · Shunta Saito · Masanori Koyama · Sosuke Kobayashi

Abstract Training of Generative Adversarial Network (GAN) on a video dataset is a challenge because of the sheer size of the dataset and the complexity of each observation. In general, the computational cost of training GAN scales exponentially with the resolution. In this study, we present a novel memory efficient method of unsupervised learning of high-resolution video dataset whose computational cost scales only linearly with the resolution. We achieve this by designing the generator model as a stack of small sub-generators and training the model in a specific way. We train each sub-generator with its own specific discriminator. At the time of the training, we introduce between each pair of consecutive sub-generators an auxiliary subsampling layer that reduces the frame-rate by a certain ratio. This procedure can allow each sub-generator to learn the distribution of the video at different levels of resolution. We also need only a few GPUs to train a highly complex generator that far outperforms the predecessor in terms of inception scores. The source code is available at `https://github.com/pfnet-research/tgan2`.

Keywords Generative Adversarial Network · Video generation · Subsampling layer

Masaki Saito
E-mail: msaito@preferred.jp

Shunta Saito
E-mail: shunta@preferred.jp

Masanori Koyama
E-mail: masomatics@preferred.jp

Sosuke Kobayashi
E-mail: sosk@preferred.jp

1 Introduction

Generative Adversarial Network (GAN) is a powerful family of unsupervised learning, and various versions of GANs have been developed to date for different types of datasets, including image and audio dataset. GANs have been particularly successful for its application to image dataset [39, 34, 6].

In this study, we present a novel method of unsupervised learning for video dataset, an important type of dataset with numerous applications such as autonomous vehicles, creative tasks, video compression, and frame interpolation.

There are two major challenges in training a generative model for video dataset. The first challenge comes from the sheer complexity of each observation. Video has a time dimension in addition to width and height, and the correlation between each pair of time frames are usually governed by complex dynamics underlying the system. Also, many applications of video generation methods–including those pertaining to industrial projects–require every frame of the generated video to be photo-realistic. This is a challenging problem on its own because photo-realistic image generation has been made possible only recently with the invention of techniques to stabilize the training of GANs on large dataset [24, 34, 33]. One must not only prepare a model that is sophisticated enough to produce a photo-realistic video but also to come up with an appropriate strategy to train the model stably within a reasonable time frame and computational resource.

The second challenge is the size of the dataset to be used in the training process. Unsupervised learning of generative model usually requires large dataset to guarantee high generalization ability. This can be a particular problem for the learners of video dataset because each observation is *large* on its own. Moreover,

the required computational resource grows exponentially with the resolution. A naive approach is bound to fail because one must reserve massive resource for both *high-resolution dataset* and the set of *model parameters*.

Most previous studies have only focused on improving the photo-realism of the output [51,42,49] while using the dataset of low-resolution videos in the range of 64×64 for the training.

In general, batch-size is limited by the memory capacity of GPU, and under a situation where the number of usable GPU is limited, it is unrealistic in practice to use a sufficiently large batch size for the training of currently available models on 256×256 resolution videos with few GPUs. These challenges warrant a clever mechanism to train a massive model fast in a memory-efficient manner.

We resolve these two problems by introducing a separate architecture into the model at the time of the training. Our generator is a stack of multiple sub-generators, in which each sub-generator takes a video of intermediate feature maps as an input and outputs a video of more complex intermediate features with the same number of frames (Figure 1). At the training process, we introduce a subsampling layer between each sub-generator and force the intermediate features from each sub-generator to make a detour to a subsampling layer that reduces its frame-rate by a certain ratio. For example, if the video of intermediate features at $k-1$th layer has m frames, the sub-generator at k-th level will receive a sparse, subsampled version of the video with m/s_t frames at the time of the training, where $s_t \geq 1$ denotes the subsampling rate. This way, the sub-generators at low level are made to receive high-frame-rate, low-resolution videos of abstract and global information, and the sub-generators at a high level are made to receive low-frame-rate, high-resolution videos of local information. This procedure can significantly reduce the computational cost and the memory requirement during the training process. Also, by preparing a separate discriminator for each sub-generator, we evaluate the fidelity of the generated video at various levels of resolution.

Because local information tends to be slow in changing (low frame-rate), our design of the *division of roles* allows us to produce high-resolution videos with high fidelity while keeping the memory consumption low. Our method can efficiently train a generator that can generate videos with significantly higher inception score (~ 26) than all predecessors.

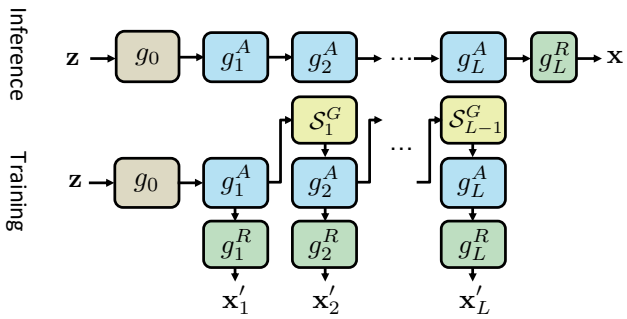


Fig. 1 The generator using multiple subsampling layers.

2 Related work

2.1 Image generation

Many applications of the Generative Adversarial Network [14] mainly focus on the image generation problem; this is primarily because that the Deep Convolutional Generative Adversarial Networks (DCGAN) [39] demonstrated that the GAN is quite effective in image generation. After that, the DCGAN was extended to a network called Self-Attention Generative Adversarial Networks (SAGAN) [56] that includes spectral normalization [34] and self-attention blocks [54]. BigGANs [6] succeeded in generating high fidelity images by introducing several tricks such as orthogonal regularization that stabilizes the training with large batch size. In the image-to-image translation problem, which transfers an input image from its domain to another domain, there exist many studies for transforming a high-resolution image to another high-resolution image. [21,53,61,29, 20].

2.2 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 [61], and solves a video retargeting problem with two video datasets for which correspondences are not given. Vid2Vid [52] 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 (e.g., one or two), image and video generation GAN requires a large batch size for the training, which makes the training of GAN models for video generation more difficult.

2.3 Multi-scale GANs

Our approach is related to methods in which a generator produces multi-scale images. LAPGAN [9] first generates a coarse image and updates it by using a difference of an initial image. StackGAN [58,57] and HDGAN [59] directly generates multi-scale images, and the corresponding discriminator returns a score from each image. Although our approach itself is similar to StackGAN and HDGAN, 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.

ProgressiveGAN [24] is another model that uses multi-resolution images and generates high-resolution images by growing both the generator and the discriminator gradually. Although the ProgressiveGAN also saves computational cost by using low-resolution images in the early stages of the training, our method has two advantages compared to ProgressiveGAN. The first advantage is that our method does not require hyperparameters to grow the 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 [43] using a pre-trained model accepting only a sample with a fixed shape.

2.4 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 [40,36,46,22,12,31,11,3,7,10,26]. Another well-known approach is to predict intermediate features of videos such as optical flow [28,30,16,27]. Some studies introduce adversarial training to avoid generating a blurred image caused by the image reconstruction loss [32,28, 26].

2.5 Video generation

There are two major approaches in the field of video generation using GANs. One is a study for generating videos by incorporating dataset specific prior knowledge into the method (e.g., limiting the problem to human action generation and giving the number of parts as prior knowledge) [8,60,55], and the other is a study that can handle any datasets without such restriction [51,42,37,49,1]. Our study is related to the latter.

We describe several models for video generation in the following. To the best of our knowledge, VGAN [51] is the first model to generate videos using GANs and consists of a 2D network to generate a still background and 3D convolutional networks to generate foreground videos. After that, Saito et al. [42] found that it is better to separate a spatiotemporal generator into time-series and space models to generate videos and proposed TGAN that first generates a set of latent vectors corresponding to each frame and then transforms them into actual images. MoCoGAN [49] was proposed to produce videos more efficiently by decomposing the latent space into the motion and the content subspaces. Unlike TGAN where the discriminator consists of a stack of 3D convolutional layers, MoCoGAN uses two different sub-discriminators to improve the quality of videos; the first is a three-dimensional discriminator that aims to extract global motion in the video, and another is two-dimensional one that identifies from a still frame in the video.

Although these models illustrate that models using GAN are also useful in video generation, they have a problem of requiring an enormous computational cost and GPU memory. It is particularly problematic when dealing with high-resolution video. Our proposed model aims to solve it. Besides, our model can be regarded as a further extension of the MoCoGAN, where the discriminator uses two different discriminators. Although our model also uses multiple discriminators, its computational cost and memory consumption are quite lower than MoCoGAN, which directly takes the generated video as an input. It is due to the multiple subsampling layers described later.

3 Method

In this section, we first describe the conventional design of GANs for video dataset. We will describe the challenges posed by these models, and explain how our models can resolve them.

3.1 GAN models for video generation

Generative Adversarial Network is a framework of unsupervised learning in which a generator model strives to mimic the target distribution while a discriminator model strives to distinguish the samples from the target distribution from the samples synthesized by the generator. The generator synthesizes an artificial sample

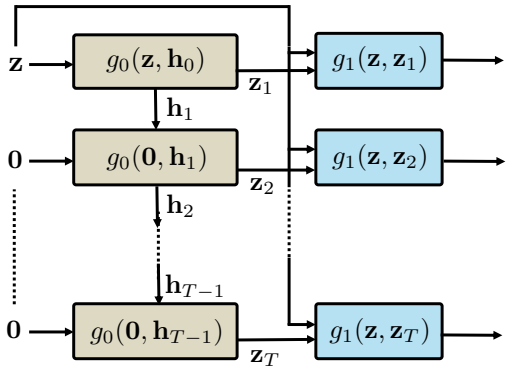


Fig. 2 The overview of relationship between a temporal generator and an image generator used in conventional temporal GANs. To produce a set of T latent vectors, the temporal generator typically is a recurrent network. For the proposed method, we used a convolutional LSTM for this part as detailed in Section 4.1.

by applying a network function G to a sample from an user-specified prior distribution p_z. The discriminator network D classifies an input as *real* or *synthetic*. Let us denote a sample from p_z by $\mathbf{z}$, and denote a sample from p_d (target distribution) by $\mathbf{x}$. The training of these networks is performed by alternately maximizing and minimizing the following objective:

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

Conventional temporal GAN (i.e., TGAN and MoCoGAN) uses a generator consisting of two sub-networks: *temporal generator* g_0 and *image generator* g_1. For the generation of a T-frame video, *temporal generator* generates a size-T set of latent vectors (or a T-frame video in the latent space) $\{\mathbf{z}_1, \ldots, \mathbf{z}_T\}$ from noise vector $\mathbf{z}$, and *image generator* transforms the *video* of latent vectors and the noise vector into the *video* of images as shown in Figure 2. The synthesized video is then classified as *synthetic* or *real* by a discriminator network consisting of three-dimensional convolutional layers.

3.2 Subsampling layer

As mentioned in the introduction, the conventional architecture introduced above is memory inefficient. A memory that must be reserved for the three dimensional convolutional layer in the discriminator is especially heavy, and the number of parameters in the model can be literally on order 100 million. Because GPU must reserve a large space for massive G and D, currently available lines of GPU cannot process a sufficient batch when a video is larger than 64×64 pixels with 16 frames.

In order to resolve this resource shortage, we introduce an architecture called *subsampling layer*s at the time of training.

For simplicity, we first describe the training of GAN with a single subsampling layer. Suppose a generator that outputs video $\mathbf{x}$ from noise vector $\mathbf{z}$ and consists of two blocks: *abstract block* g^A and *rendering block* g^R. The abstract block computes the latent feature map (we call it an *abstract map*) from noise vector, and the rendering block transforms it into a video. In inference time, the generation process of samples by this generator is equivalent to that of 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 discriminator in conventional temporal GANs computes a score from a high-frame rate video of original resolution. This can be especially costly when the original resolution is high. To cope with this problem, in the training stage, we modify G into G' as follows by introducing between g^R and g^A a *subsampling layer* $\mathcal{S}^G$ that reduces the frame rate of the abstract map produced from g^A.

With G', the subsampled video $\mathbf{x}'$ is produced by applying g^R to $\mathcal{S}^G(g^A(\mathbf{z}))$. That is,

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

The discriminator D' then evaluates the score for $\mathbf{x}'$.

In the training process of D, one must also prepare a set of *real* data to compare against the synthetic data. We prepare this by applying S^D to the real videos, an architecture that downscales the resolution and reduces the frame-rate so that the dimension of the output tensor matches the output of G'.

In other words, in the training stage of our method, the objective of Equation (1) can be rewritten by

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

This way, we can significantly reduce the burden on the discriminator. The computational cost for D' is much smaller than the original D.

Indeed, this benefit comes at the cost of making the domain of the video at the inference-time differ from that of the video at the training time. We introduce two tricks to deal with this problem. The first trick is to statically change the position of the first frame (that is, we sample video at times $ts_t + b_t$ with random b_t; cf. Equation (9)) so that there will be no frames that are not sampled. The second trick is to use multiple subsampling layers that drop the frame rate by different scales. We describe the detail of this method in the next subsection.

3.3 Multiple subsampling layers

Unfortunately, however, we experimentally observed that the scores of the samples generated by the architecture in the previous section are significantly lower than those generated by the naive implementation without the subsampling layer.

By using a model made of multiple sub-generators and multiple subsampling layers, however, we can drastically reduce the overall computational cost without compromising the quality of the video. In fact, the quality of the video generated by our stacked sub-generator is significantly better than that of the video generated by the conventional architecture. Below we elaborate the construction of the model (Figure 1).

We describe the model consisting of L sub-generators, or L abstract blocks. At the time of the training, this architecture is trained with L corresponding rendering blocks and $L-1$ subsampling layers.

Let us denote the abstract block, rendering block, and the subsampling layer at level l by g_l^A, g_l^R, and $\mathcal{S}_l^G$ respectively. In the inference time, $\mathbf{x}$ can be evaluated simply by sequentially applying abstract blocks and rendering with a single g_L^R, i.e.,

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

Here and also in the following explanation, g_1^A includes the temporal generator g_0, but we abuse the notation g_1^A as $g_1^A \circ g_0$ for simplicity (see Figure 1). Note that the noise vector is concatenated with each latent vector from the temporal generator only when it is input to the first image generator g_1^A. In the training time, we use L-set of sub-generators consisting of $G'_l(\mathbf{z})$, each of which recursively applies g_m^A and $\mathcal{S}_m^G$ $(m = 1, \ldots, l-1)$ to abstract maps and converting the output of the final abstract block g_l^A to the video by g_l^R:

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

Note that in our implementation, all rendering blocks $(g_i^R (i = 1, \ldots, L))$ do not have any shared parameters, i.e., every block has its own parameters. G'_1 is a model that generates a video of lowest resolution and highest-frame-rate. It applies g_1^A to the high-frame-rate video of latent vectors and applies g_1^R to the result to produce a high-frame-rate video of low-resolution images $\mathbf{x}'_1$.

G'_2 on the other hand reduces the frame-rate of the video output of g_1^A with $\mathcal{S}_1^G$, feed the subsampled video to g_2^A, and converts the output of g_2^A to a video of higher resolution image $\mathbf{x}'_2$ by g_2^R. Defined recursively, G'_L generates a video of highest resolution and lowest-frame-rate. Because the increase in the resolution is countered by the decrease in frame-rate, the computational cost for G_L does not increase exponentially with the resolution of the video.

3.4 Multiple discriminators

In order to train the above generator consisting of multiple sub-generators $G'_1, \ldots, G'_L$, we need a discriminator that evaluates a score for the set of videos produced by them. Our discriminator consists of multiple sub-discriminators. Let D'_l be the l-th sub-discriminator that takes a sample x'_l from l-th sub-generator G'_l and returns a scalar value. Our discriminator D' evaluates a score for a set of x'_ls by the following formula:

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

where $\sigma(\cdot)$ is a sigmoid function. To take a sample from the raw dataset and evaluate a score for it, we apply $\mathcal{S}_l^D$ to each raw video, which downscales the resolution and reduces the frame-rate, so that the output will match the synthetic video produced by G'_l.

3.5 Role of each discriminator

The essence of our method is the division of roles; instead of feeding the dense raw dataset of a single massive model, we train each sub-generator with select parts of the dataset that can help the sub-generator improve at playing the given role.

The role of the sub-generator with low indices is to mimic the original video dataset at an abstract level; that is, to produce a low-resolution video that flows naturally with time. The discriminators with low indices are responsible for evaluating the quality of high-frame-rate videos of low resolution. This allows the generators with low indices to capture global motion in the video that is independent from high resolution details.

On the other hand, the role of the sub-generator with high indices is to mimic the original video dataset in visual quality. That is, they only require a low frame-rate video of high resolution to train. The discriminator with the highest index in our model may in fact be designed to evaluate the fidelity of the still image (one frame).

Our model can also be regarded as an extension of MoCoGAN, a model that uses two different discriminators.

Our method also aims to reduce both computational cost and memory consumption simultaneously. We will further discuss this topic in Section 4.3.

4 Network architecture

4.1 Generator

We describe our design of a generator consisting of four sub-generators that synthesizes a video with T frames and $W \times H$ pixels. As with the TGAN [42] and MoCoGAN [49], our generator first produces T latent feature maps from a noise vector and then transforms each map into a corresponding frame (see Figure 2). The specific network structure is shown in Figure 3 and Figure 4.

We first 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 $(W/64)\times(H/64)$ pixel resolution through a fully-connected layer. A recurrent block consisting of a Convolutional LSTM [44] (CLSTM) 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 feature map derived from $\mathbf{z}$ as input, but at $t \geq 1$ it always receives a feature map derived 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 (As we mentioned before, all $g_i^R (i = 1, \ldots, L)$ do not share any parameters and have their own parameters). That is, the upsampling block is a function that outputs a feature map whose resolution is double that of the input feature map, and the rendering block converts it to the image while maintaining the resolution.

In the training phase, the generator progressively reduces the size of abstract maps with the three subsampling layers placed between each g_i^A.

Each layer is given the role of reducing the number of frames.

Consider for example an abstract map $\mathbf{h}$ with C_h channels, T_h frames, width W_h and height H_h. We represent each element of this tensor by $h_{c,t,h,w}$. The subsampling layer with reduction rate s_t reduces the shape of $\mathbf{h}$ to $(C_h \times \lceil T_h/s_t \rceil \times H_h \times W_h)$; The output $\mathbf{h}'$ produced from subsampling layer is the following probabilistic mixture of subsampled versions of $h_{c,t,h,w}$:

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

where B is a Boolean function given by

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

with b_t being a sample from discrete uniform distribution $\mathcal{U}\{0, \min(s_t, T_h) - 1\}$. Using the slicing notation of NumPy [38], the above equation can be evaluated easily by invoking `hd = h[:, bt::st]`. In experiments, we set s_t to the same value for all subsampling layers.

The advantage of this strategy is that it can prevent the memory consumption of the discriminator from growing exponentially with the resolution.

For this example model, the amount of GPU memory required by the discriminator will be constant regardless of the resolution if $s_t \geq 4$ and T is sufficiently large.

At the same time, if s_t is too large relative to T, the number of frames will reduce to 1 at a block close to the input layer, and there will be no computational gain from the subsampling layer from that point onward. The number of original frames required to guarantee the *constant-order computational cost* at every block grows exponentially with s_t. It is therefore important to choose the appropriate pair of s_t and the number of blocks that suits the purpose and the memory capacity of GPU.

In Section 5 we will investigate the effect of the choice of s_t on the quality of the generated video.

4.2 Discriminator

As we described in Section 3.3, our discriminator consists of several sub-discriminators. In our implementation, we used four 3D ResNet models, each containing several spatiotemporal three-dimensional residual blocks [17,18] and one fully-connected layer. The network configuration of each 3D ResNet was almost the same as the discriminator used in Miyato et al. [34] except that the kernel of all convolutional layers was replaced with (3×3) to $(3 \times 3 \times 3)$. For more details, see Figure 4. As for the initializers, we used the GlorotUniform initializer [13] 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.

We shall emphasize that, even though all sub- discriminators are given the same network structure, they all play different roles in the model as a whole. The sub-discriminator with the lowest index evaluates the fidelity of the global flow of a given video, and the sub-discriminator with the highest index evaluates the photorealism of randomly selected several frames.

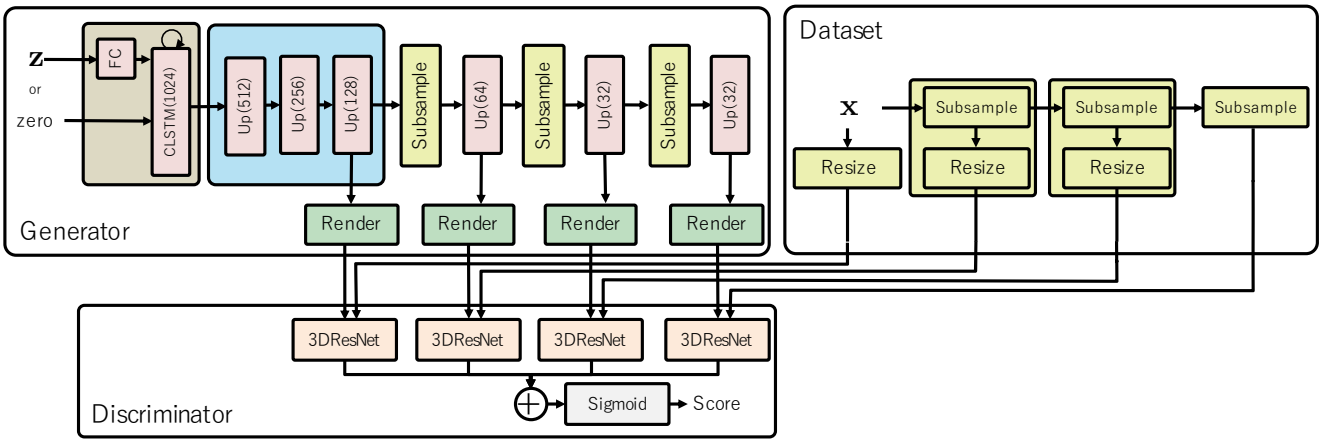


Fig. 3 Network configuration of our model. "CLSTM(C)" represents the convolutional LSTM with C channels and 3×3 kernel. "Up(C)" means the upsampling block that returns a feature map with C channels and twice the resolution of the input.

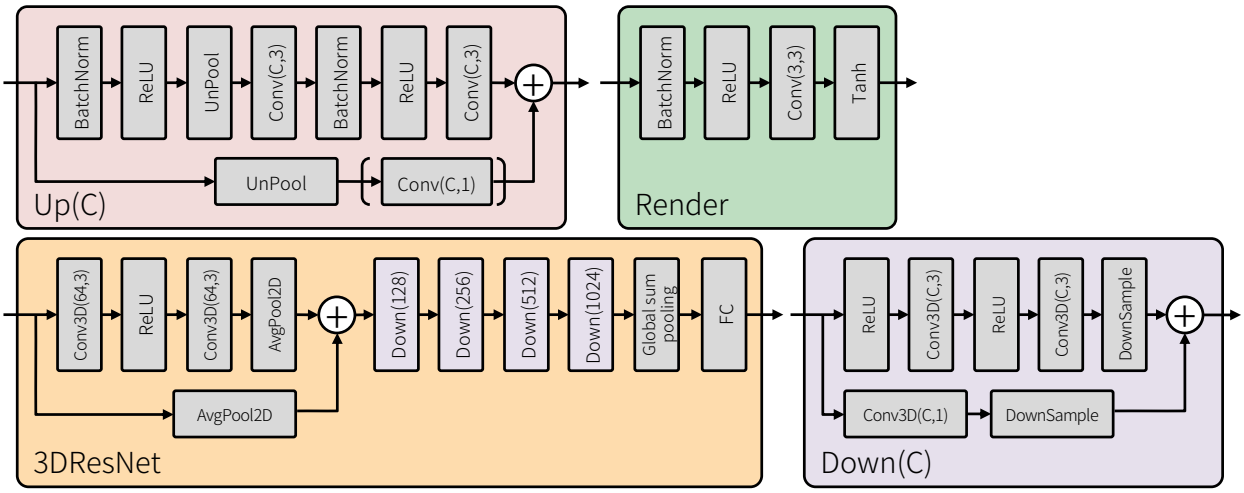


Fig. 4 Details of the blocks used in the main paper. "Conv(C, k)" denotes a 2D convolutional layer with C channels and $(k \times k)$ kernel. "Conv3D(C, k)" denotes a 3D convolutional layer with C channels and $(k \times k \times k)$ kernel. "UnPool" denotes a 2D unpooling layer with (2×2) kernel and stride 2. "AvgPool2D" denotes a 2D 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.

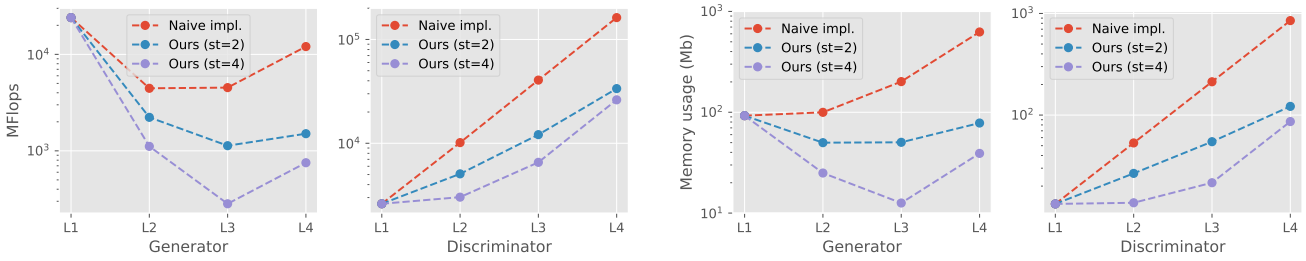
4.3 Computational cost and Memory

The computational cost and the memory consumption of the generator differ in nature from those of the discriminator (see Figure 5 and Table 1.)

While the computational cost of the generator is almost constant for each block, the cost of the discriminator increases exponentially with the level of the block. On the other hand, the memory consumption of both generator and discriminator grows exponentially with the level.

Method	GFlops	Ratio	Memory	Ratio
Gen (naive impl.)	45.07		1019	
Subsampling ($s_t = 2$)	28.91	1.56x	271	3.76x
Subsampling ($s_t = 4$)	26.20	1.72x	169	6.03x
Dis (naive impl.)	215.74		1130	
Subsampling ($s_t = 2$)	53.41	4.04x	217	5.22x
Subsampling ($s_t = 4$)	38.40	5.62x	135	8.37x
3D discriminator	162.38		851	
3D + 2D dis.	168.20		921	

Table 1 The total computational cost and intermediate memory consumption in megabytes when enabling frame and batch reductions. The number of frames used for generation is 16, and the batch size is 1.



(a) Computational cost of forward pass

(b) Total size of intermediate memory

Fig. 5 Graphs representing the cost savings by enabling multiple subsampling layers. The left figures show the theoretical computational cost of each block of the network, and the right figures denote the amount of intermediate memory in each block consumed by generated samples and hidden variables. The horizontal axis of all graphs represents the level of the block. Each block of the generator used to compute the cost contains layers inside the blue area in Figure 3 and the corresponding rendering block. Each block of the discriminator corresponds to each sub discriminator. The number of frames used for generation is 16, and the batch size is 1.

This is problematic because the computational cost and the memory consumption of 3D ResNet are particularly high for the discriminator.

As shown in Table 1, the computational cost for the discriminator tends to be significantly heavier than that of the generator even when using a single 3D discriminator in the setting of ordinary GAN.

This is also true when using a discriminator similar to the one used in MoCoGAN ("3D + 2D dis" in Table 1. See Section 5.3 for details.) The computational cost for the discriminator grows exponentially with the dimension of the video, and some countermeasure must be taken if one wishes to conduct unsupervised learning of high-resolution video with limited number of GPUs.

By successively reducing the frame-rate of the video with the aforementioned subsampling layer, however, we can prevent the computational cost from growing exponentially.

When the number of frames reduced by the subsampling layer is half of the original (i.e., $s_t = 2$), we can improve the total memory usage and computational cost of the model four∼five times over the vanilla model without subsampling layers.

Meanwhile, this mechanism does not improve the computational cost of the generator as much. However, most part of the overall computational cost comes from the discriminator, and this mechanism alone can reduce the overall computational cost to a reasonable level.

In general, the computational gain increases with the size of s_t. When the original number of frames is sufficiently large and $s_t \geq 4$, the computational cost for each sub-generator will be roughly kept constant because at each block, the number of frame decreases by the rate of 4 while the height and the width doubles.

The computational gain is not as high in our implementation because the number of original frames used in our experiments was just 16. For example, Figure 5 shows that the actual memory consumption at the level 4 in the generator is larger than that of the level 3 because the number of frames handled by the level 3 is one since the number of original frames is 16.

Even still, the memory consumption of our model with $s_t = 4$ was about 60% better than that of $s_t = 2$.

Thus, the computational gain from subsampling layer tends to increase with the size of s_t.

4.4 Dataset

We use three subsampling functions to make four subsampled videos from one example video in the dataset. For the sample to be used for the sub-discriminator with index= 1, we only apply one resize function to lower the resolution of the video by one eighth. Meanwhile, the last sub-discriminator receives a video that was produced by applying three subsampling layers defined in Equation (8) to the input example. This transformation reduces the number of frames to $1/s_t^3$ while maintaining the resolution.

4.5 Regularizer

To improve the performance of the generator, we augmented the loss of the discriminator at level l with a zero-centered gradient penalty evaluated over the target data distribution [33], given by

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

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

4.6 Conditional model

In the previous discussion, we only focus on the problem where the generator can only accept a noise vector as an input. However, as with the field of the image generation (c.f., BigGAN [6]), we can extend the proposed method to accept not only the noise vector but also a discrete integer label representing a category of the dataset. Giving such information as an argument of the generator and the discriminator, we can further improve the quality of the generated videos.

Specifically, we extend the generator and the discriminator according to the following. In the generator, we first transform the discrete label into one-hot vector $\mathbf{l}$, concatenate it with noise vector $\mathbf{z}$ to obtain another vector represented by $[\mathbf{l}, \mathbf{z}]$. Using this vector as the input of of the FC layer in Figure 3, the discrete label information can be propagated to the ConvLSTM. Regarding the extension of the image generator (i.e., the upsampling blocks after the ConvLSTM), we adopted the technique performed by Miyato et al. [34]; that is, we replace all the batch normalization layers with conditional batch normalization layers to ensure that all the blocks receive discrete label information. In the discriminator, we add a perturbation vector for the label to the final layer of each sub-discriminator according to a method of projection discriminator [35].

5 Experiments

5.1 Datasets

We used the following two datasets in the experiments.

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* [45]. 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 it 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. [52], we created the facial videos from FaceForensics [41] 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 these frames to the discriminator. As with UCF101, all the values are normalized to $[-1, 1]$.

5.2 Hyperparameters

We used the Adam [25] 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 local batch size of each GPU was selected so it fills the GPU memory of NVIDIA Tesla P100 (12Gb). The total batch size used for experiments depends on the experiments. 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 [47] and ChainerMN [2]. 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) [6].

We manually confirmed that the training time is approximately proportional to GFlops shown in Table 1 under the same number of GPUs. The whole training time including snapshot and computation of the inception score was about 61 hours under the environment of four GPUs, batch size 32, and $s_t = 2$. The training for each model used in the experiments ended within two to four days.

5.3 Baseline models

To confirm the effectiveness of our model, we introduced two baseline models that generate high-resolution videos. One is a high-resolution model consisting of a single generator and a single discriminator. Specifically, the network architecture of the generator in this model is almost identical to that of the proposed model, but it does not contain three rendering blocks that output low-resolution videos (that is, the generator only outputs a single high-resolution video during training). The network of the discriminator is the same as the sub-discriminator used in our method. Since the number of parameters of the generator is almost the same as that of our model,

we can see the difference of performance between the simple high-resolution model and our multi-scale model.

The other is a baseline model where the discriminator uses two different sub-discriminators similar to MoCoGAN. Specifically, although the network of the generator is the same as that of the model mentioned above, this discriminator consists of the 2D sub- discriminator in addition to the above 3D ones. The configuration of the 2D sub-discriminator is equivalent to the model of Miyato et al. [34] used for image generation.

The 3D sub-discriminator discriminates real videos from generated videos by directly classifying videos, while the 2D sub-discriminator only discriminates a randomly selected frame in videos. Using this baseline model, we can see the performance of simply inserting the subsampling layer only in the final layer instead of a middle layer.

To improve the quality of baseline models, we performed a grid search to find the optimal λ in Equation (10). We finally set $\lambda = 10.0$ for all experiments using these baseline models.

5.4 Qualitative comparison of generated videos

5.4.1 FaceForensics

To confirm the quality of the video generated by our model, we trained our models and two baselines using the two datasets described above. First, we trained these models using the FaceForensics [41] dataset. In order to check the effect of s_t in the subsampling layer, we trained the model with two values $s_t = 2, 4$. In the proposed model, the generator has four sub-generators and each of which generates videos at different resolution and different number of frames depending on the setting of subsampling layers (the value of s_t). Here, we denote the series of numbers of frames of those videos from different sub-generators as a tuple of the number of frames such as [16, 8, 4, 2]. Therefore, in the case of $s_t = 2$, the number of video frames produced by the generator is $[16, 8, 4, 2]$. On the other hand, in the case of $s_t = 4$, the number of video frames is $[16, 4, 1, 1]$ according to the definition of Equation (9). The batch size was set to 32 for both. Eight GPUs were used in this experiment.

The videos generated by our model are shown in Figure 6. We confirmed that at $s_t = 2$, our model generated high-resolution videos without collapsing the parts in a face. Specifically, we observed that our model was able to generate high-fidelity images as a single still image, and facial parts such as eyes and mouth move smoothly. To show clearly that our generated video is not just a sequence that continues a single still image, we also placed an enlarged view of the eyes and mouth in Figure 6.

In addition to this, we also confirmed that the proposed method could generate videos of various faces without causing mode collapse. To show this diversity, we show several samples generated by our model in Figure 7.

As for the facial video, MoCoGAN also shows similar qualitative results using a similar facial video dataset, but its resolution is relatively small (64×64 pixels). We confirmed that even though our model handles samples that consume 16 times more memory than the conventional small samples, it can be efficiently trained while saving memory and computational cost with the subsampling layer. As mentioned in Sections 1 and 4.3, if one tries to use MoCoGAN that does not exploit the subsampling layer to generate high resolution videos, there is a problem that the amount of consumed GPU memory is enormous (for example, MoCoGAN requires about 10Gb of memory to generate videos with 64×64 px even if batch size is 8). The result of Figure 6 shows that our model can be properly trained even at such a high resolution without extremely increasing the amount of memory consumed by the discriminator.

On the other hand, videos generated by the model of $s_t = 4$ tended to be inconsistent in the first few frames (note that we confirmed that both $s_t = 2$ and $s_t = 4$ generate stable videos for the remaining frames). This example is shown in Figure 8. We considered it is because that the increase in the number of subsampled frames leads to instability in the first few frames. We also confirmed that this behavior depends on the dataset. Specifically, the instability of $s_t = 4$ has been mitigated when trained with UCF101 dataset, and we observed that in the quantitative evaluation, the model with $s_t = 4$ outperformed the model with $s_t = 2$. We describe the detail in Sections 5.4.2 and 5.10.

5.4.2 UCF101

Next, we performed a similar experiment using UCF101, which is more challenging to generate videos that look more natural than FaceForensics. Unlike FaceForensics with 256×256 pixels, the resolution of the UCF101 used in experiments is 192×192 pixels; this is because the resolution of the original video is 320×240 pixels. Four GPUs were used to train models. Similar to the experiment in the FaceForensics, we trained two models of $s_t = 2$ and $s_t = 4$ with the UCF101 dataset. The total batch size was set to 32, which is identical to the experiment in the FaceForensics. To confirm the differences in qualitative results when using a conditional model, we also trained conditional models with discrete labels in UCF101. These hyperparameters are identical to those without conditions.

Although the metrics such as Inception Score and Fréchet Inception Distance used in this quantitative experiment have been applied in the field of video generation [42, 49, 50], it is difficult to know how much improvement in these values leads to qualitative improvement. To confirm this difference, we compared qualitatively with videos generated by five models. One is a TGAN [42] trained with UCF101. The second and

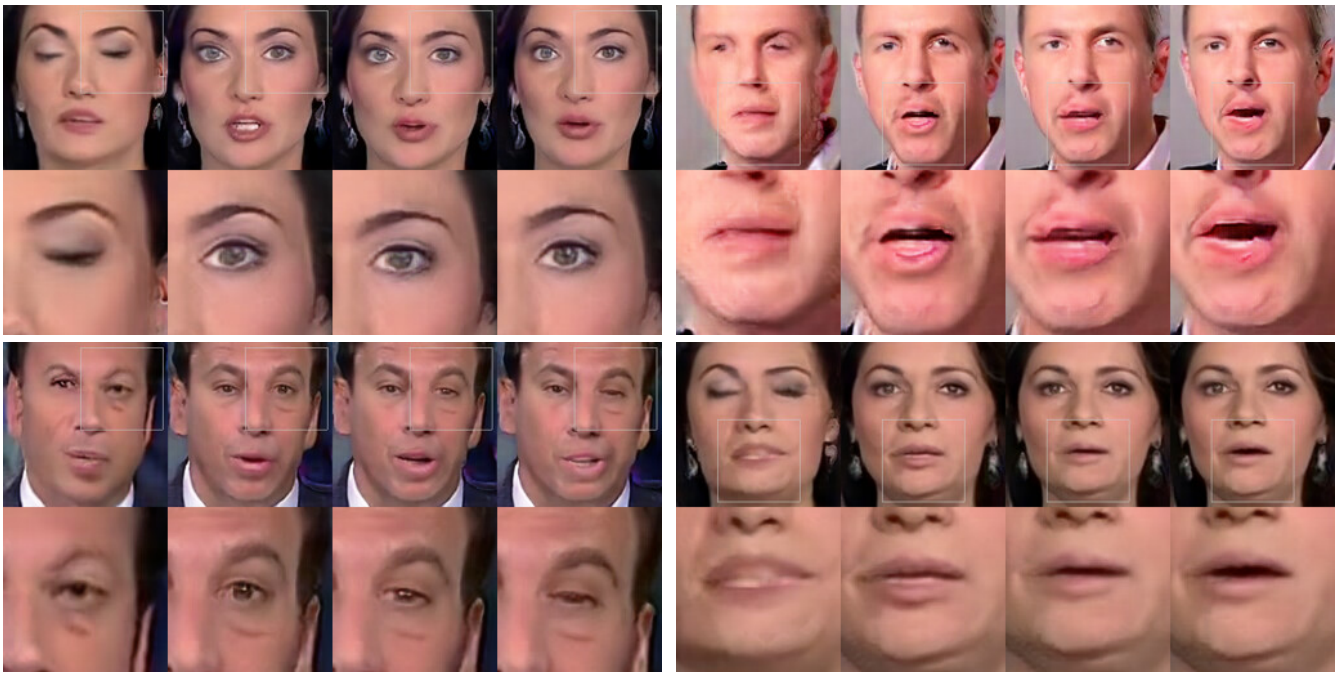


Fig. 6 An example of videos generated by our model ($s_t = 2$) trained with FaceForensics dataset. Every four frames out of 16 frames is shown in a row for the ease of identifing the motion in the video. The top row represents the frames of the whole video. The bottom row shows a magnified view of the area of the white box in the top row.

Fig. 7 A list of still images extracted from videos generated by our model ($s_t = 2$). The FaceForensics dataset was used for the training.

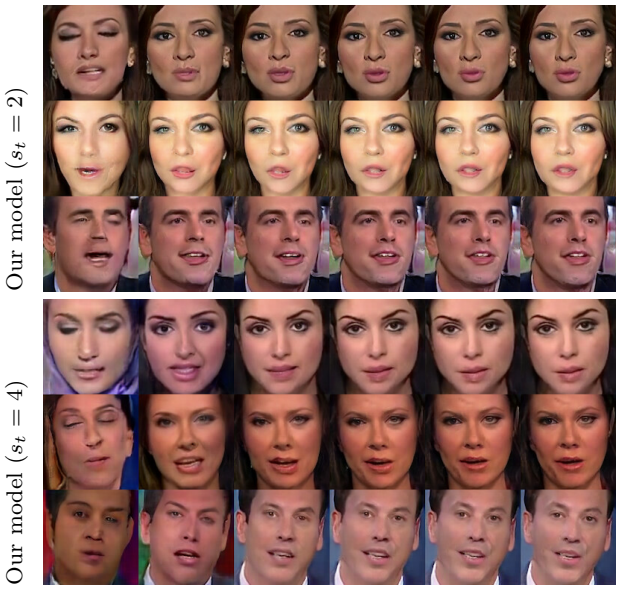


Fig. 8 Comparison of instability of first frames trained with FaceForensics. Only the initial six frames are shown.

third are our proposed models with $s_t = 2$ and $s_t = 4$. The rest are the two baseline models described in Section 5.3. For the MoCoGAN [49], we could not use it for the qualitative experiment because the pre-trained model was not available and there is no figure representing the generated result in the paper. However, we considered that TGAN can also be used to compare the qualitative quality of the current state-of-the-art methods because the TGAN achieves similar inception score to MoCoGAN.

Figure 9 shows the videos generated by the baseline and our two models. We confirmed that our proposed two models generated videos that are easier to interpret the content than the two baseline models. Specifically, the videos generated by our models tend to have some abstract motions such as moving shadows that look like a human, but its outline is relatively clear, and the background is more distinguishable (e.g., indoor, sea, grassland, and stadium) than baseline models. In particular, when the background of the video is static such as soccer or basketball in a distant view, our model generated more detailed videos than other conventional models.

The baseline models trained with UCF101 tended to produce the same meaningless videos since they often caused mode collapse. On the other hand, we did not find such a tendency in the proposed models. In order to show this behavior, we list the still images extracted from the generated videos in Figure 10. The inception scores and Fréchet inception distance measured by these models are much better than those observed by the baseline models. It suggests that both metrics have a certain degree of correlation with the quality of videos perceived by humans even in UCF101. For reference, we also put a list of frames extracted from the videos generated by TGAN [42]. We also confirmed that the TGAN tends to generate noisy videos and blurry backgrounds, whereas our model tends to produce a video with a relatively clear background and sharp shadows.

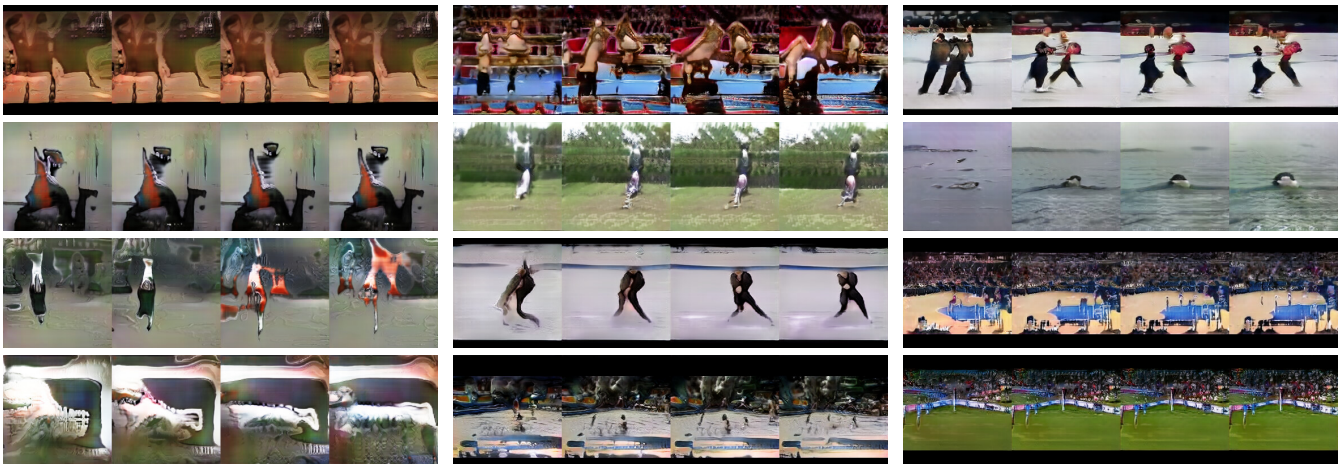
As for videos generated by the condtional model, we observed that it tends to yield a relatively stable video than the unconditional one. This was especially noticeable when the content of the video was a distant view. We have also confirmed through quantitative experiments described later that the quality of the videos by the conditional model tends to exceed that of the unconditional one. Because the quality of the video in which a person moves well is still comparable to the unconditional model, we inferred that such quantitative improvement of the quality was mainly because of the improvement of distant view videos.

Unlike the experiment of FaceForensics with $s_t = 4$, in the experiment of UCF101 with $s_t = 4$ we did not see such steep instability but observed gradual changes in the video, i.e., the category of the video does not change but topography changes gradually. The example is shown in Figure 11. We considered that it was due to differences in the domain of the dataset. If videos in the dataset are not very diverse as in FaceForensics, the transition of the domain of the first frames caused by the subsampling layer tends to occur. Contrarily, if the dataset is diverse as in UCF101, we considered that such steep transition would be less likely to occur since such transitions are easily identified by the discriminator.

5.5 Linear interpolation of the noise vector

We also generated a new sample with a intermediate vector between the two noise vectors to ensure that the trained network was not a model that memorized the dataset. The model trained with $s_t = 2$ and the FaceForensics dataset was used for the experiment.

The result is shown in Figure 12. The content of the video changed smoothly by moving $\mathbf{z}$ linearly. It means that our model is not just a "memorized" model of the original dataset. Changing $\mathbf{z}$ often tended to change the speaker itself, but on the other hand, there were several videos where the speaker did not change, but other attributes such as the orientation of the speaker's face changed smoothly. We inferred that it was mainly because that the input of the discriminator was a sequence clipped at 16 frames randomly from a video in the dataset.



Frame 1 Frame 16 Frame 1 Frame 16 Frame 1 Frame 16

(i) 3D + 2D discriminators (ii) Our model ($s_t = 2$, uncond.) (iii) Our model ($s_t = 4$, uncond.)

Fig. 9 Example of videos by the three models (our two unconditional models ($s_t = 2, 4$) and baseline ("3D + 2D discriminators")) trained with UCF101. Images excluding the intermediate three frames are shown to make it easy to identify the motion.

TGAN [42] Single 3D discriminator

Our model ($s_t = 4$, unconditional) Our model ($s_t = 4$, conditional)

Fig. 10 A list of still images extracted from videos generated by our models ($s_t = 4$), TGAN, and baseline model. The UCF101 dataset was used for the training. "unconditional" means a generator trained with videos only, whereas "conditional" is a conditional one trained with videos and corresponding labels.

5.6 Consistency of generated samples

To check whether each rendering block generates the same content under the same $\mathbf{z}$, we visualized the result of each rendering block. The example is shown in Figure 13. We observed that every rendering block in our method tends to generate the same content without a color-consistency regularization introduced in StackGAN++ [57].

Next, making use of the property that all blocks tend to output the same content, we observed which level of blocks the instability of the first frames at $s_t = 4$ resulted from. To check the results of the videos generated by all rendering blocks, we disabled every sub-

Fig. 11 Comparison of instability of first frames trained with UCF101. Only the initial six frames are shown.

Fig. 12 Example of linear interpolation of noise vectors. The first and last rows represent images sampled from the two noise vectors, and the middle row means an image generated from an intermediate vector between them. Only the result of the eight frame is shown for simplicity. The model trained with $s_t = 2$ was used.

sampling layer to make the rendering block of each level output a video with 16 frames. The results for each level at $s_t = 2, 4$ are shown in Figure 14. In contrast to the result of $s_t = 2$, where every rendering block generated a video with almost identical content, the first frames of the model of $s_t = 4$ tended to become unstable as the level increased. It indicates that the cause of instability is that the subsequent rendering blocks no longer output the same content.

To quantitatively confirm that this instability depends on the dataset, we calculated PSNR (Peak Signal-to-Noise Ratio) and SSIM (structural similarity) between the frame in a video generated by the rendering block of level 1 and the frame of level 4. We resized the videos in level 1 so that the resolutions of the two videos match. One thousand samples were used for evaluation.

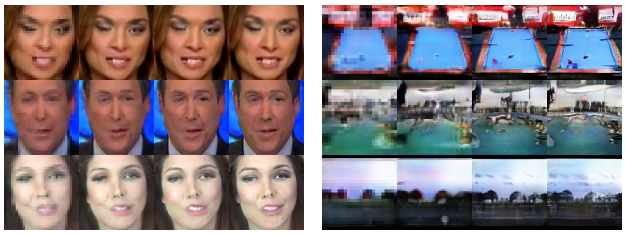


Fig. 13 Example of videos generated at different levels. The leftmost image represents the frame of the video generated by the initial rendering block, and the rightmost one denotes the final generated image.

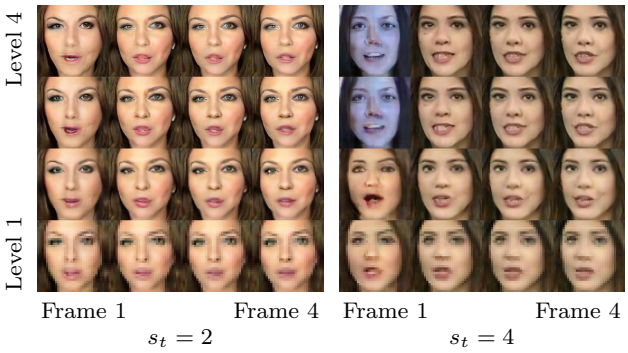


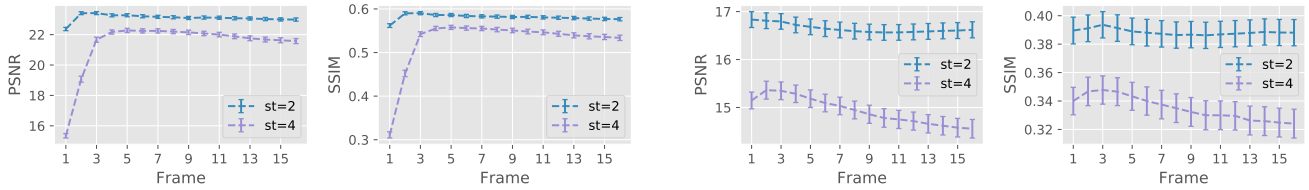
Fig. 14 The difference of the video which each rendering block outputs. The row represents levels of rendering blocks and time, and the column means time.

These quantitative results are shown in Figure 15. It indicates that the first frames at level 4 become unstable when trained with the FaceForensics dataset, while later it outputs almost identical content as level 1. However, the identity of $s_t = 4$ was worse than $s_t = 2$ even if enough time had passed. While such a rapid initial instability was not observed with UCF101, its identity after a sufficient period of time was slightly worse for both SSIM and PSNR than for FaceForensics. It means that the identity of the video generated by each block changes depending on the difficulty level of the dataset.

5.7 Comparison with other existing methods

We confirmed that the quality of samples generated by the proposed model is superior to other existing models with Inception Score (IS) [43] and Fréchet Inception Distance (FID) [19], which measure the quality of generated samples. We describe the details of the experimental environment used for measuring IS and FID in the following. See [5] for details of IS and FID.

As with other comparable methods [42, 49, 1], we employed the UCF101 dataset [45] for the quantitative experiments. Regarding the classifier used for comput-



(i) FaceForensics (ii) UCF101

Fig. 15 Quantitative differences of the videos generated by different rendering blocks. The row indicates the difference between the two frames generated by rendering block at level 1 and 4. The column means time. Error bars at the 95% confidence interval.

ing IS and FID, we used a pre-trained model of Convolutional 3D (C3D) network [48], which was first trained with Sports-1M dataset [23] and then fine-tuned on the UCF101 dataset[1]. Note that 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.

To calculate the IS and FID with the C3D model, we need a fourth-order tensor ($C \times T \times H \times W$) that represents the average for all the clips in the UCF101. We used the same average tensor used in TGAN.

In all experiments, we computed FID and IS according to the following procedure. First, we set the maximum number of iterations to 100,000 and took snapshots every 2,000 iterations. For each snapshot, we computed the IS and FID with 2,048 samples. The best snapshot with the largest IS was used for quantitative evaluation. We calculated the mean and standard deviation by performing the same procedure ten times for this snapshot. It took about four hours to compute IS and FID of one snapshot.

The resolutions of generated videos by all the existing methods are 64×64 pixels except for ProgressiveGAN (128×128 pixels). These scores shown in the quantitative experiments in these models can be compared with the IS calculated by our method because their models also compute the IS by resizing the generated videos to 128×128 pixels. However, even if the IS of our method exceeds the existing method, this increase may be due to simply increasing the resolution (in other words, the IS may be an indicator with a vulnerability that can be easily increased by increasing resolution). To confirm this, we measured IS and FID of two baseline models with a similar network structure as the existing method (c.f., Section 5.3), and observed the change of both scores in the case of high resolution.

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

Method	IS	FID
VGAN [51]	$8.31 \pm .09$	
TGAN [42]	$11.85 \pm .07$	
MoCoGAN [49]	$12.42 \pm .03$	
ProgressiveVGAN [1]	$13.59 \pm .07$	
ProgressiveVGAN w/ SWL [1]	$14.56 \pm .05$	
Single 3D discriminator only	$11.10 \pm .16$	8358 ± 81
3D + 2D discriminators	$10.47 \pm .12$	8304 ± 70
Naive implementation	$13.29 \pm .15$	5401 ± 31
Our model ($s_t = 2$)	$23.87 \pm .28$	3797 ± 20
Our model ($s_t = 4$)	$26.60 \pm .47$	3431 ± 19

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

The difference between our model and the existing ones is roughly divided into the two: multi-scale model and subsampling layer. To clarify the contribution of each factor to the IS and FID, we also defined a model without all subsampling layers as "naive implementation", and measured its IS and FID.

Four GPUs were used in all the experiments of this subsection. We set the maximum batch size within the memory of the GPU for each model. This means that the batch size of the two proposed methods was 32, whereas that of two baseline models was 8. The model without subsampling layers consumes much memory due to the resolution; therefore, we had to set the entire batch size to 4 even if we can use four GPUs.

The quantitative results are shown in Table 2. The inception scores of two baseline models are slightly lower than those of other existing methods. It indicates that it is difficult to improve scores by simply increasing the resolution of the samples from the generator. Although it may be possible that the baseline models could not allocate a sufficient mini-batch due to the high resolution, we will show in Section 5.9 that these scores do not increase dramatically even if the batch size is sufficiently large. The IS and FID of the baseline model with two sub-discriminators are almost the same as those of the baseline with a single 3D discriminator. It implies that it is difficult to improve the quality of the generated

Method	IS	FID
Single 3D discriminator only	$3.41 \pm .03$	13252 ± 50
3D + 2D discriminators	$1.01 \pm .01$	15380 ± 30
Naive implementation	$1.43 \pm .01$	13698 ± 12
Our model ($s_t = 2$)	$20.61 \pm .28$	3930 ± 25
Our model ($s_t = 4$)	$21.45 \pm .29$	3877 ± 15

Table 3 Inception Score and Fréchet Inception Distance when using a single GPU.

Method	IS	FID
Single 3D discriminator only	$11.72 \pm .20$	7754 ± 47
3D + 2D discriminators	$13.38 \pm .22$	6993 ± 25
Naive implementation	$14.44 \pm .20$	7613 ± 29
Our model ($s_t = 2$)	$23.87 \pm .28$	3797 ± 20
Our model ($s_t = 4$)	$26.60 \pm .47$	3431 ± 19

Table 4 Inception Score and Fréchet Inception Distance when setting 32 for batch size.

videos dramatically with the conventional approach of simply introducing multiple discriminators into the input layer. The IS and FID of the model without subsampling layers are much lower than those of our proposed model. It indicates that the subsampling layer contributes to the improvement of the quality in an environment of four GPUs.

The inception scores of our two models ($s_t = 2, 4$) are significantly higher than those of the other existing models. In other words, in the environment of four GPUs, it means that the quality of our proposed models exceeds the existing method. It is interesting to see that IS and FID computed by the model of $s_t = 4$ are better than those of $s_t = 2$. Thus, by increasing the number of frames to be reduced, our method can not only save the computational cost and the GPU memory more efficiency but also slightly improve the quality of the generated videos. As discussed in Sections 5.4.1 and 5.4.2, increasing s_t may contribute to instability of first few frames in some datasets; however, the increase of IS by setting $s_t = 2$ instead of not using subsampling layers is much higher than the increase of IS by setting $s_t = 4$ instead of $s_t = 2$. Based on these quantitative results, we concluded that an approach of setting $s_t = 2$ at the beginning and then gradually increasing would be appropriate when training with an unknown dataset.

5.8 Quantitative results under a single GPU

Although our proposed model outperforms the other existing methods in the environment of 4 GPUs, this improvement may be due to the advantageous setting where we used multiple GPUs. To confirm whether the IS and FID of our method exceed those of the existing method under the almost same environment, we measured IS and FID of the two baselines and the proposed models when training with a single GPU.

The results are shown in Table 3. IS and FID of the model without subsampling layers are much worse than those of the existing models. It is because that the video is a high resolution and the batch size is only one. It also illustrates the difficulty of training high resolution models in a limited resource. Even if we can use only one GPU, the IS of our model is significantly higher than those of the other existing models. In particular, the fact that inception scores of our models are higher than that of ProgressiveGAN, which also outputs high-resolution videos, demonstrates the effectiveness of our model under constrained computational resources.

5.9 Quantitative results under the same batch size

It can be seen that our models outperformed the other existing and baseline models under the environment of one or four GPUs. However, if we can increase the batch size without such limitations, i.e., if we can set the same batch size as our model by simply increasing the number of GPUs, baseline models may outperform the proposed method. Therefore, we measured IS and FID of the three baseline models under the same batch size. The batch size was set to 32. It means that the model without the subsampling layers consumed 32 GPUs, whereas the two baseline models used 16 GPUs and our method used only four GPUs.

Table 4 shows the results. Even if there is no limit on the number of GPUs, our proposed model outperformed all the baseline models. We can claim the following two from these results. First, the low IS and the high FID in the baseline models shown in the previous experiments are not mainly due to the small batch size. That is, it is hard to improve both scores dramatically with a simple model that introduces a single discriminator only to the input layer. Second, the reason why both scores of our models are much better than those of the existing methods is not a simple reason like the number of parameters is larger than others. Even in an environment where a sufficient number of GPUs can be used, the IS of the model without introducing the subsampling layer was similar to the other existing methods. It implies that such high scores are because of the introduction of multiple subsampling layers.

5.10 Effectiveness of frame sub-sampling layers

IS and FID highly depend on batch size. We confirmed that activating the subsampling layer improved IS and

Batchsize	Naive impl.	$s_t = 2$	$s_t = 4$
4	$13.29 \pm .15$	$14.64 \pm .25$	$17.12 \pm .26$
8	$11.26 \pm .19$	$20.61 \pm .28$	$21.45 \pm .29$
16	$12.79 \pm .10$	$22.81 \pm .52$	$22.69 \pm .47$
32	$14.44 \pm .20$	$23.87 \pm .28$	$26.60 \pm .47$
64	N/A	$24.39 \pm .50$	$26.89 \pm .40$
128	N/A	$25.65 \pm .34$	$28.87 \pm .67$

Table 5 Changes in the inception score when sub-sampling layers are enabled.

Batchsize	Naive impl.	$s_t = 2$	$s_t = 4$
4	5401 ± 31	5148 ± 33	4449 ± 14
8	7364 ± 51	3930 ± 25	3877 ± 15
16	7822 ± 27	3639 ± 15	3715 ± 19
32	7613 ± 29	3797 ± 20	3431 ± 19
64	N/A	4242 ± 32	3334 ± 26
128	N/A	3573 ± 28	3497 ± 26

Table 6 Changes in the Fréchet Inception Distance when sub-sampling layers are enabled.

FID of our model over the other existing methods, and the scores of the model of $s_t = 4$ are slightly better than those of $s_t = 2$. However, it is not clear whether these trends are the same even if the batch size is increased (for example, the score of the naive model may catch up with that of the model with subsampling layers). To see the effectiveness of the subsampling layer for batch size, we measured the change of the inception score when the subsampling layers were activated under the condition of the same batch size. In this subsection we performed three types of experiments; one is an experiment when all subsampling layers are disabled, and the remaining two are experiments when the subsampling layers are enabled with $s_t = 2$ and 4. Training these models with different batch sizes, we measured the change of IS and FID. For economic reasons, we did not measure scores of batch size 64 and 128 when disabling the subsampling layer.

The results are shown in Tables 5 and 6. In any batch size, IS and FID of the model with subsampling layers are significantly better than the model without them. It indicates that the subsampling layers are effective for improving the quality of the generated videos regardless of the batch size. As the discriminator at the later stage has to determine the authenticity of the video with a lower frame rate, it needs to judge the authenticity from the global motion and the quality of a still image instead of the difference of the fine movement. We considered that the subsampling layer played a role like *regularization*, resulting in improvement of IS and FID.

	$s_t = 2$		$s_t = 4$	
BS	pure	conditional	pure	conditional
16	$22.81 \pm .52$	$40.59 \pm .94$	$22.69 \pm .47$	$39.61 \pm .94$
32	$23.87 \pm .28$	48.00 ± 1.0	$26.60 \pm .47$	$49.30 \pm .56$
64	$24.39 \pm .50$	$54.93 \pm .68$	$26.89 \pm .40$	$51.21 \pm .49$

Table 7 Changes in the inception score when updating to a conditional model. "pure" denotes an unconditional model, whereas "conditional" is a conditional one described in Section 4.6. "BS" means a batch size.

	$s_t = 2$		$s_t = 4$	
BS	pure	conditional	pure	conditional
16	3639 ± 15	3390 ± 20	3715 ± 19	3487 ± 22
32	3797 ± 20	3253 ± 22	3431 ± 19	3186 ± 16
64	4242 ± 32	3934 ± 24	3334 ± 26	3352 ± 26

Table 8 Changes in the Fréchet Inception Distance when updating to a conditional model.

We also confirmed that IS and FID at $s_t = 4$ were significantly higher than those at $s_t = 2$ in almost all batch sizes. In particular, when the batch size is 128 and $s_t = 4$, the inception score of our model (28.87) dramatically exceeds the existing state-of-the-art method (14.56). It shows the superiority of our model over other existing methods. Note that our IS also exceeds all the scores of the existing methods when using one GPU only instead of such relatively large computational resources (c.f., Section 5.8).

5.11 Quantitative results of the conditional model

As we described in Section 4.6, the quality of our generated video can be improved with a conditional model. To see the changes of quantitative scores when using the conditional model, we measured the changes in IS and FID for each batch size. The experiment was performed under two conditions: $s_t = 2$ and $s_t = 4$. Three batch sizes of 16, 32, and 64 were used in the experiment.

The results are shown in Tables 7 and 8. They show that the scores improved significantly by extending to the conditional model regardless of the batch size and hyperparameters. In particular, the inception score (54.93) of the conditional model for $s_t = 2$ and the batch size of 64 is significantly higher than any previously reported scores. However, an increase in batch size did not lead to a significant increase in FID, but was sometimes worsened. When the batch size was 32, the FID of the conditional model was improved over all scores in Table 6, but its improvement is not as dramatic as that of IS. It is more intuitive when looking at the actual generated videos (see Figure 10).

Method	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	IS	FID
Naive impl.				$12.79 \pm .10$	7822 ± 27
	✓			$19.99 \pm .33$	4502 ± 37
	✓	✓		$21.38 \pm .38$	4192 ± 45
Full	✓	✓	✓	$22.81 \pm .52$	3639 ± 15

Table 9 Change in Inception Score and Fréchet Inception Distance when each subsampling layer is gradually enabled (batchsize=16, $s_t = 2$).

Method	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	IS	FID
Naive impl.				$12.79 \pm .10$	7822 ± 27
	✓			$20.95 \pm .12$	4433 ± 26
	✓		✓	$19.37 \pm .43$	4628 ± 20
Full	✓	✓		$22.69 \pm .47$	3715 ± 19

Table 10 Change in Inception Score and Fréchet Inception Distance when each subsampling layer is gradually enabled (batchsize=16, $s_t = 4$).

Method	Naive impl.	$s_t = 2$	$s_t = 4$
3D dis. only	$11.10 \pm .16$	$4.30 \pm .06$	$1.73 \pm .01$
3D + 2D dis.	$10.47 \pm .12$	$4.97 \pm .04$	$3.78 \pm .02$

Table 11 Changes in the Inception Score when applying a single frame subsampling layer to two baseline models. "Naive impl." means a model without any subsampling layers.

5.12 Effectiveness of each subsampling layer

In previous experiments, we enabled multiple subsampling layers simultaneously. To confirm whether all of the subsampling layers introduced by our method actually contribute to the improvement of the quality, we measured the change of IS and FID when multiple subsampling layers were gradually activated.

The concrete procedure is as follows. First, according to the notation in Figure 1 we defined the three subsampling layers introduced in our method as $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$. Next, we observed the change of IS and FID when these layers were activated sequentially from $\mathcal{S}_1$. Enabling $\mathcal{S}_i$ affects the number of video frames received by all subsequent discriminators in our method. Therefore, to investigate whether $\mathcal{S}_i$ contributes to the improvement of both scores in all cases, it is necessary to measure the change of the scores for every state of $\mathcal{S}_j$ except $\mathcal{S}_i$. Although strictly speaking, measuring the effect on the strategy to sequentially activate the subsampling layers does not fully illustrate the effectiveness of $\mathcal{S}_i$, we can see whether all subsampling layers are effective, at least in the general situation of activating them gradually.

We also confirmed the effectiveness of each subsampling layer for changes in hyperparameters by setting s_t to 2 and 4. The batch size for these experiments was set to 16. From the definition of b_t, the number of frames of four generated videos in the case of $s_t = 4$ is $[16, 4, 1, 1]$, that is, enabling both $\mathcal{S}_1$ and $\mathcal{S}_2$ is the same as enabling all subsampling layers. To see the effectiveness of $\mathcal{S}_3$ in $s_t = 4$, instead, we observed the change of IS and FID when $\mathcal{S}_3$ was activated in the situation where $\mathcal{S}_1$ was enabled.

The result of $s_t = 2$ is shown in Table 9. By gradually activating $\mathcal{S}_i$, both IS and FID significantly improved. We considered this is because that each subdiscriminator plays different roles in improving the quality by enabling the subsampling layers, and IS and FID gradually increased. Regarding the change in scores at each step, it can be seen that IS and FID improved most when enabling $\mathcal{S}_1$. It means that $\mathcal{S}_1$ contributed most to the improvement of the quality of videos in our method.

We also show the result of $s_t = 4$ in Table 10. As with the result of $s_t = 2$, IS and FID increased significantly by enabling $\mathcal{S}_i$ gradually. In particular, the improvement of IS and FID when enabling $\mathcal{S}_1$ represents the importance of $\mathcal{S}_1$ in our method. IS and FID did not improve when the number of frames of four generated videos was changed from $[16, 4, 4, 4]$ to $[16, 4, 1, 1]$, whereas both scores improved significantly when changing from $[16, 4, 4, 4]$ to $[16, 4, 1, 1]$. This implies that enabling $\mathcal{S}_2$ is more important than $\mathcal{S}_3$ for $s_t = 4$.

5.13 Frame subsampling in baseline models

Although we show that the subsampling layer in our model has the effect of improving the quality of videos, it is still unclear whether the subsampling layer is also effective in the conventional methods, that is, the essential contribution of our method may be only in the subsampling layer. To clarify that the contribution of this study is the combination of multi-scale model and subsampling layers, we measured the change of IS and FID when the subsampling layer was applied to the above two baseline models.

We describe the detail. Unlike our proposed model that outputs multiple videos for training, the generator in the baseline model generates only a single video. Thus, from the network architecture of the baseline model, it is clear that enabling $\mathcal{S}_2$ or $\mathcal{S}_3$ is equivalent to enabling only $\mathcal{S}_1$. In this experiment, we focused on $\mathcal{S}_1$ and measured only the change of both scores when $\mathcal{S}_1$ was activated. As in the previous experiment, we used the two hyperparameters ($s_t = 2, 4$) for the subsampling layer. The batch size was 16.

Method	Naive impl.	$s_t = 2$	$s_t = 4$
3D dis. only	8358 ± 81	11764 ± 54	13938 ± 26
3D + 2D dis.	8304 ± 70	11301 ± 26	11386 ± 12

Table 12 Changes in the Fréchet Inception Distance when applying a single frame subsampling layer to two baseline models.

The results are shown in Tables 11 and 12. Both IS and FID when activating the subsampling layer are significantly worse than those of the naive implementation. In particular, when the subsampling layer of $s_t = 4$ was applied, we observed that the training did not work at all for both models. While the introduction of the subsampling layer saves computational cost and memory consumption, the discriminator can identify the videos with reduced frame rates more easily than the original ones. We considered that it led to a decrease in the IS and an increase in FID. This result is in contrast to the result of our model, which improved both computational cost and quality by enabling the subsampling layer, and illustrates that our contribution is a combination of a multiscale model and subsampling layers.

6 Discussion

In this section, we summarize the empirical findings obtained through a series of experiments.

What contributed most to the improvement of the quality is the combination of the subsampling layer and the multiscale model (the quality cannot be dramatically improved by either the multiscale model or the subsampling layer). The score can be improved by simply increasing the number of GPUs without the subsampling layers, but its improvement is small. The score improved significantly by enabling the subsampling layers. However, this significant improvement does not occur in other models, but only in our multiscale model. That is, in order to improve the score in the task of video generation, it is important not to introduce either one but to combine both.

The introduction of the subsampling layer leads not only to the improvement of the quality but also to significantly saving the computational cost and memory consumption of the GPU (c.f., Table 1). Even in situations where only a single GPU can be used due to budget constraints, our method can efficiently solve high-resolution video generation problems that could not be solved with naive models, and its inception score is significantly higher than those of the existing methods. We have confirmed that this dramatic increase in scores cannot be achieved by only increasing the resolution with two baseline models.

Similar to the findings of BigGAN [6], the quality of the video can also be improved by increasing the batch size in our model. We confirmed that an increase in s_t leads to an improvement in the quality regardless of the batch size. However, on the other hand, the generation of the first frames may be unstable depending on the dataset. Since no such instability was observed in the case of $s_t = 2$, we consider that it would be appropriate to train the model with $s_t = 2$ at an early stage, gradually increase the value, and observe the change of IS and FID.

In our proposed method using multiple subsampling layers, enabling the first subsampling layer ($\mathcal{S}_1$) contributes the most to improving the score. However, in our experiments, we did not see any decrease in scores even when the subsequent subsampling layers were enabled. As there is no increase in computational cost and memory consumption by enabling the subsampling layer, we consider that it would be better to enable all subsampling layers first, and then to fine-tune by gradually activating the subsampling layers if there are enough computing resources and time.

6.1 Trials and their results

We share a process of trial and error that led up to the development of our method.

6.1.1 CLSTM layers

We tried stacked Convolutional LSTM layers but observed that a single CLSTM 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.

6.1.2 Generation of longer videos

We observed that our model sometimes succeeded to stably generate longer videos than the number of frames used for training (i.e., 16 frames) but sometimes generated broken videos. Although this behavior depends on the hyperparameters and the dataset, it is unclear under what conditions our model can generate stably[2].

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

6.1.3 Spectral normalization

We tried to insert the Spectral Normalization [34] into the 3D convolutional layers in the discriminator, but its performance decreased significantly.

6.1.4 Gradient penalty

Although we initially used a gradient penalty based on WGAN-GP [15] as a regularizer, but for simplicity we 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 could be performed normally without the gradient penalty. It may indicate that the method using multiple discriminators contributed to the stabilization of training.

6.1.5 Batch normalization

Mescheder et al. [33], the authors of the zero-centered gradient penalty, reported that they succeeded in generating high fidelity images without Batch Normalization layers in the generator, but in our experiments without Batch Normalization layers, the performance significantly decreased.

6.1.6 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.

7 Conclusion

In this paper we introduced the method of efficiently training the high-resolution model for video generation with GAN. The main idea is to generate videos in different ways during training and inference. During the training, the generator outputs multiple "sparse" samples useful for the training at low computational cost instead of directly generating high-resolution video. For inference, the generator outputs high-resolution "dense" videos over time. Using this method we can not only train our multi-scale model for video generation while saving computational cost and memory consumption, but the quality of our trained model is significantly superior to the baseline models and the existing ones. In both qualitative and quantitative experiments, we confirmed that our method could output high-resolution videos with higher quality the conventional ones, and the subsampling layers actually contribute to the improvement of the quality.

Our core idea may may not only to other fields that exploit videos such as video prediction but also to other domains with time series including audio generation. We are planning to find more practical applications by exploring these possibilities.

Acknowledgements We would like to acknowledge Takeru Miyato and Shoichiro Yamaguchi for helpful discussions. We would like to acknowledge Daichi Suzuo for providing a tool to calculate the cost of computation and the amount of memory consumed. We also would like to thank the developers of Chainer [47, 2].

References

1. Acharya, D., Huang, Z., Paudel, D.P., Gool, L.V.: Towards High Resolution Video Generation with Progressive Growing of Sliced Wasserstein GANs. In: Arxiv preprint arXiv:1810.02419 (2018)
2. Akiba, T., Fukuda, K., Suzuki, S.: ChainerMN: Scalable Distributed Deep Learning Framework. In: Proceedings of Workshop on ML Systems in NIPS (2017)
3. Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R.H., Levine, S.: Stochastic Variational Video Prediction. In: ICLR (2018)
4. Bansal, A., Ma, S., Ramanan, D., Sheikh, Y.: Recycle-GAN: Unsupervised Video Retargeting. In: ECCV (2018)
5. Borji, A.: Pros and Cons of GAN Evaluation Measures. In: Arxiv preprint arXiv:1802.03446 (2018)
6. Brock, A., Donahue, J., Simonyan, K.: Large Scale GAN Training for High Fidelity Natural Image Synthesis. In: Arxiv preprint arXiv:1809.11096 (2018)
7. Byeon, W., Wang, Q., Srivastava, R.K., Koumoutsakos, P.: ContextVP: Fully Context-Aware Video Prediction. In: ECCV (2018)
8. Cai, H., Bai, C., Tai, Y.W., Tang, C.K.: Deep Video Generation, Prediction and Completion of Human Action Sequences. In: ECCV (2018)
9. Denton, E., Chintala, S., Szlam, A., Fergus, R.: Deep Generative Image Models Using a Laplacian Pyramid of Adversarial Networks. In: NIPS (2015)
10. Denton, E., Fergus, R.: Stochastic Video Generation with a Learned Prior. In: Arxiv preprint arXiv:1802.07687 (2018)
11. Ebert, F., Finn, C., Lee, A.X., Levine, S.: Self-Supervised Visual Planning with Temporal Skip Connections. In: Conference on Robot Learning (CoRL) (2017)
12. Finn, C., Goodfellow, I., Levine, S.: Unsupervised Learning for Physical Interaction through Video Prediction. In: NIPS (2016)
13. Glorot, X., Bengio, Y.: Understanding the Difficulty of Training Deep Feedforward Neural Networks. In: AISTATS (2010)
14. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Nets. In: NIPS (2014)

15. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved Training of Wasserstein GANs. In: NIPS (2017)
16. Hao, Z., Huang, X., Belongie, S.: Controllable Video Generation with Sparse Trajectories. In: CVPR (2018)
17. Hara, K., Kataoka, H., Satoh, Y.: Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet? In: CVPR (2018)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: CVPR (2016)
19. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In: NIPS (2017)
20. Huang, X., Liu, M.Y., Belongie, S., Kautz, J.: Multimodal Unsupervised Image-to-Image Translation. In: ECCV (2018)
21. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-Image Translation with Conditional Adversarial Networks. In: CVPR (2017)
22. Kalchbrenner, N., van den Oord, A., Simonyan, K., Danihelka, I., Vinyals, O., Graves, A., Kavukcuoglu, K.: Video Pixel Networks. In: arxiv preprint arXiv:1610.00527 (2016)
23. Karpathy, A., Shetty, S., Toderici, G., Sukthankar, R., Leung, T., Li Fei-Fei: Large-scale Video Classification with Convolutional Neural Networks. In: CVPR (2014)
24. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive Growing of GANs for Improved Quality, Stability, and Variation. In: ICLR (2018)
25. Kingma, D., Ba, J.: Adam: A Method for Stochastic Optimization. In: ICLR (2015)
26. Lee, A.X., Zhang, R., Ebert, F., Abbeel, P., Finn, C., Levine, S.: Stochastic Adversarial Video Prediction. In: Arxiv preprint arXiv:1804.01523 (2018)
27. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Flow-Grounded Spatial-Temporal Video Prediction from Still Images. In: ECCV (2018)
28. Liang, X., Lee, L., Dai, W., Xing, E.P.: Dual Motion GAN for Future-Flow Embedded Video Prediction. In: ICCV (2017)
29. Liu, M.Y., Breuel, T., Kautz, J.: Unsupervised Image-to-Image Translation Networks. In: NIPS (2017)
30. Liu, Z., Yeh, R.A., Tang, X., Liu, Y., Agarwala, A.: Video Frame Synthesis using Deep Voxel Flow. In: ICCV (2017)
31. Lotter, W., Kreiman, G., Cox, D.: Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning. In: ICLR (2017)
32. Mathieu, M., Couprie, C., LeCun, Y.: Deep Multi-Scale Video Prediction beyond Mean Square Error. In: ICLR (2016)
33. Mescheder, L., Nowozin, S., Geiger, A.: Which Training Methods for GANs do actually Converge? In: ICML (2018)
34. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral Normalization for Generative Adversarial Networks. In: ICLR (2018)
35. Miyato, T., Koyama, M.: cGANs with Projection Discriminator. In: ICLR (2018)
36. Oh, J., Guo, X., Lee, H., Lewis, R., Singh, S.: Action-Conditional Video Prediction using Deep Networks in Atari Games. In: NIPS (2015)
37. Ohnishi, K., Yamamoto, S., Ushiku, Y., Harada, T.: Hierarchical Video Generation from Orthogonal Information: Optical Flow and Texture. In: AAAI (2018)
38. Oliphant, T.E.: Guide to NumPy, 2nd edn. CreateSpace Independent Publishing Platform, USA (2015)
39. Radford, A., Metz, L., Chintala, S.: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In: ICLR (2016)
40. Ranzato, M., Szlam, A., Bruna, J., Mathieu, M., Collobert, R., Chopra, S.: Video (Language) Modeling: A Baseline for Generative Models of Natural Videos. arXiv preprint arXiv:1412.6604 (2014)
41. Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., Nießner, M.: FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces. In: Arxiv preprint arXiv:1803.09179 (2018)
42. Saito, M., Matsumoto, E., Saito, S.: Temporal Generative Adversarial Nets with Singular Value Clipping. In: ICCV (2017)
43. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved Techniques for Training GANs. In: NIPS (2016)
44. Shi, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K., Woo, W.c.: Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In: NIPS (2015)
45. Soomro, K., Zamir, A.R., Shah, M.: UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. arXiv preprint arXiv:1212.0402 (2012)
46. Srivastava, N., Mansimov, E., Salakhutdinov, R.: Unsupervised Learning of Video Representations using LSTMs. In: ICML (2015)
47. Tokui, S., Oono, K., Hido, S., Clayton, J.: Chainer: a Next-Generation Open Source Framework for Deep Learning. In: Proceedings of Workshop on Machine Learning Systems in NIPS (2015)
48. Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning Spatiotemporal Features with 3D Convolutional Networks. In: ICCV (2015)
49. Tulyakov, S., Liu, M.Y., Yang, X., Kautz, J.: MoCoGAN: Decomposing Motion and Content for Video Generation. In: CVPR (2018)
50. Unterthiner, T., van Steenkiste, S., Kurach, K., Marinier, R., Michalski, M., Gelly, S.: Towards Accurate Generative Models of Video: A New Metric & Challenges. In: Arxiv preprint arXiv:1812.01717 (2018)
51. Vondrick, C., Pirsiavash, H., Torralba, A.: Generating Videos with Scene Dynamics. In: NIPS (2016)
52. Wang, T.C., Liu, M.Y., Zhu, J.Y., Liu, G., Tao, A., Kautz, J., Catanzaro, B.: Video-to-Video Synthesis. In: Arxiv preprint arXiv:1808.06601 (2018)
53. Wang, T.C., Liu, M.Y., Zhu, J.Y., Tao, A., Kautz, J., Catanzaro, B.: High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In: CVPR (2018)
54. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local Neural Networks. In: CVPR (2018)
55. Yang, C., Wang, Z., Zhu, X., Huang, C., Shi, J., Lin, D.: Pose Guided Human Video Generation. In: ECCV (2018)
56. Zhang, H., Goodfellow, I., Metaxas, D., Odena, A.: Self-Attention Generative Adversarial Networks. In: Arxiv preprint arXiv:1805.08318 (2018)
57. Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., Metaxas, D.: StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. In: arXiv preprint arXiv:1710.10916 (2017)
58. Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., Metaxas, D.: StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. In: ICCV (2017)
59. Zhang, Z., Xie, Y., Yang, L.: Photographic Text-to-Image Synthesis with a Hierarchically-nested Adversarial Network. In: CVPR (2018)

60. Zhao, L., Peng, X., Tian, Y., Kapadia, M., Metaxas, D.: Learning to Forecast and Refine Residual Motion for Image-to-Video Generation. In: ECCV (2018)
61. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In: ICCV (2017)

Semantic Image Synthesis with Spatially-Adaptive Normalization

Taesung Park[1,2*] Ming-Yu Liu[2] Ting-Chun Wang[2] Jun-Yan Zhu[2,3]

[1]UC Berkeley [2]NVIDIA [2,3]MIT CSAIL

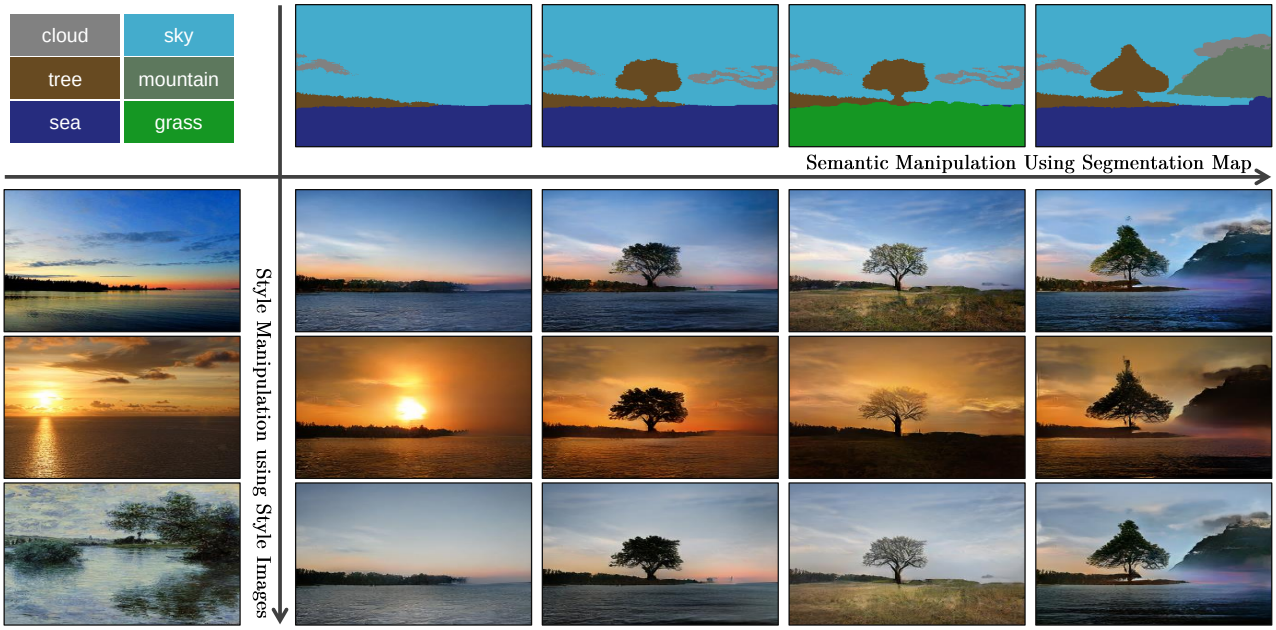


Figure 1: Our model allows user control over both semantic and style as synthesizing an image. The semantic (e.g., the existence of a tree) is controlled via a label map (the top row), while the style is controlled via the reference style image (the leftmost column). Please visit our website for interactive image synthesis demos.

Abstract

We propose spatially-adaptive normalization, a simple but effective layer for synthesizing photorealistic images given an input semantic layout. Previous methods directly feed the semantic layout as input to the deep network, which is then processed through stacks of convolution, normalization, and nonlinearity layers. We show that this is suboptimal as the normalization layers tend to "wash away" semantic information. To address the issue, we propose using the input layout for modulating the activations in normalization layers through a spatially-adaptive, learned transformation. Experiments on several challenging datasets demonstrate the advantage of the proposed method over existing approaches, regarding both visual fidelity and alignment with input layouts. Finally, our model allows user control over both semantic and style. Code is available at https://github.com/NVlabs/SPADE.

1. Introduction

Conditional image synthesis refers to the task of generating photorealistic images conditioning on certain input data. Seminal work computes the output image by stitching pieces from a single image (e.g., Image Analogies [16]) or using an image collection [7, 14, 23, 30, 35]. Recent methods directly learn the mapping using neural networks [3, 6, 22, 47, 48, 54, 55, 56]. The latter methods are faster and require no external database of images.

We are interested in a specific form of conditional image synthesis, which is converting a semantic segmentation mask to a photorealistic image. This form has a wide range of applications such as content generation and image editing [6, 22, 48]. We refer to this form as semantic image synthesis. In this paper, we show that the conventional network architecture [22, 48], which is built by stacking convolutional, normalization, and nonlinearity layers, is at best

*Taesung Park contributed to the work during his NVIDIA internship.

suboptimal because their normalization layers tend to "wash away" information contained in the input semantic masks. To address the issue, we propose *spatially-adaptive normalization*, a conditional normalization layer that modulates the activations using input semantic layouts through a spatially-adaptive, learned transformation and can effectively propagate the semantic information throughout the network.

We conduct experiments on several challenging datasets including the COCO-Stuff [4, 32], the ADE20K [58], and the Cityscapes [9]. We show that with the help of our spatially-adaptive normalization layer, a compact network can synthesize significantly better results compared to several state-of-the-art methods. Additionally, an extensive ablation study demonstrates the effectiveness of the proposed normalization layer against several variants for the semantic image synthesis task. Finally, our method supports multi-modal and style-guided image synthesis, enabling controllable, diverse outputs, as shown in Figure 1. Also, please see our SIGGRAPH 2019 Real-Time Live demo and try our online demo by yourself.

2. Related Work

Deep generative models can learn to synthesize images. Recent methods include generative adversarial networks (GANs) [13] and variational autoencoder (VAE) [28]. Our work is built on GANs but aims for the conditional image synthesis task. The GANs consist of a generator and a discriminator where the goal of the generator is to produce realistic images so that the discriminator cannot tell the synthesized images apart from the real ones.

Conditional image synthesis exists in many forms that differ in the type of input data. For example, class-conditional models [3, 36, 37, 39, 41] learn to synthesize images given category labels. Researchers have explored various models for generating images based on text [18,44,52,55]. Another widely-used form is image-to-image translation based on a type of conditional GANs [20, 22, 24, 25, 33, 57, 59, 60], where both input and output are images. Compared to earlier non-parametric methods [7, 16, 23], learning-based methods typically run faster during test time and produce more realistic results. In this work, we focus on converting segmentation masks to photorealistic images. We assume the training dataset contains registered segmentation masks and images. With the proposed spatially-adaptive normalization, our compact network achieves better results compared to leading methods.

Unconditional normalization layers have been an important component in modern deep networks and can be found in various classifiers, including the Local Response Normalization in the AlexNet [29] and the Batch Normalization (BatchNorm) in the Inception-v2 network [21]. Other popular normalization layers include the Instance Normalization (InstanceNorm) [46], the Layer Normalization [2], the Group Normalization [50], and the Weight Normalization [45]. We label these normalization layers as unconditional as they do not depend on external data in contrast to the conditional normalization layers discussed below.

Conditional normalization layers include the Conditional Batch Normalization (Conditional BatchNorm) [11] and Adaptive Instance Normalization (AdaIN) [19]. Both were first used in the style transfer task and later adopted in various vision tasks [3, 8, 10, 20, 26, 36, 39, 42, 49, 54]. Different from the earlier normalization techniques, conditional normalization layers require external data and generally operate as follows. First, layer activations are normalized to zero mean and unit deviation. Then the normalized activations are *denormalized* by modulating the activation using a learned affine transformation whose parameters are inferred from external data. For style transfer tasks [11, 19], the affine parameters are used to control the global style of the output, and hence are uniform across spatial coordinates. In contrast, our proposed normalization layer applies a spatially-varying affine transformation, making it suitable for image synthesis from semantic masks. Wang *et al*. proposed a closely related method for image super-resolution [49]. Both methods are built on spatially-adaptive modulation layers that condition on semantic inputs. While they aim to incorporate semantic information into super-resolution, our goal is to design a generator for style and semantics disentanglement. We focus on providing the semantic information in the context of modulating normalized activations. We use semantic maps in different scales, which enables coarse-to-fine generation. The reader is encouraged to review their work for more details.

3. Semantic Image Synthesis

Let $\mathbf{m} \in \mathbb{L}^{H \times W}$ be a semantic segmentation mask where $\mathbb{L}$ is a set of integers denoting the semantic labels, and H and W are the image height and width. Each entry in $\mathbf{m}$ denotes the semantic label of a pixel. We aim to learn a mapping function that can convert an input segmentation mask $\mathbf{m}$ to a photorealistic image.

Spatially-adaptive denormalization. Let $\mathbf{h}^i$ denote the activations of the i-th layer of a deep convolutional network for a batch of N samples. Let C^i be the number of channels in the layer. Let H^i and W^i be the height and width of the activation map in the layer. We propose a new conditional normalization method called the SPatially-Adaptive (DE)normalization[1] (SPADE). Similar to the Batch Normalization [21], the activation is normalized in the channel-wise manner and then modulated with learned scale and bias. Figure 2 illustrates the SPADE design. The activation

[1] Conditional normalization [11, 19] uses external data to denormalize the normalized activations; i.e., the denormalization part is conditional.

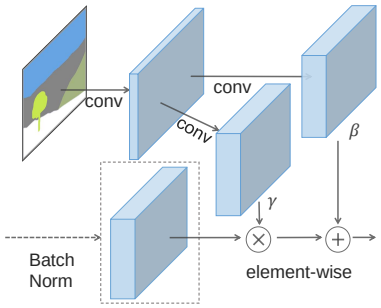


Figure 2: In the SPADE, the mask is first projected onto an embedding space and then convolved to produce the modulation parameters γ and β. Unlike prior conditional normalization methods, γ and β are not vectors, but tensors with spatial dimensions. The produced γ and β are multiplied and added to the normalized activation element-wise.

value at site $(n \in N, c \in C^i, y \in H^i, x \in W^i)$ is

$$\gamma^i_{c,y,x}(\mathbf{m})\frac{h^i_{n,c,y,x} - \mu^i_c}{\sigma^i_c} + \beta^i_{c,y,x}(\mathbf{m}) \tag{1}$$

where $h^i_{n,c,y,x}$ is the activation at the site before normalization and μ^i_c and σ^i_c are the mean and standard deviation of the activations in channel c:

$$\mu^i_c = \frac{1}{NH^iW^i}\sum_{n,y,x} h^i_{n,c,y,x} \tag{2}$$

$$\sigma^i_c = \sqrt{\frac{1}{NH^iW^i}\sum_{n,y,x}\left((h^i_{n,c,y,x})^2 - (\mu^i_c)^2\right)}. \tag{3}$$

The variables $\gamma^i_{c,y,x}(\mathbf{m})$ and $\beta^i_{c,y,x}(\mathbf{m})$ in (1) are the learned modulation parameters of the normalization layer. In contrast to the BatchNorm [21], they depend on the input segmentation mask and vary with respect to the location (y, x). We use the symbol $\gamma^i_{c,y,x}$ and $\beta^i_{c,y,x}$ to denote the functions that convert $\mathbf{m}$ to the scaling and bias values at the site (c, y, x) in the i-th activation map. We implement the functions $\gamma^i_{c,y,x}$ and $\beta^i_{c,y,x}$ using a simple two-layer convolutional network, whose design is in the appendix.

In fact, SPADE is related to, and is a generalization of several existing normalization layers. First, replacing the segmentation mask $\mathbf{m}$ with the image class label and making the modulation parameters spatially-invariant (i.e., $\gamma^i_{c,y_1,x_1} \equiv \gamma^i_{c,y_2,x_2}$ and $\beta^i_{c,y_1,x_1} \equiv \beta^i_{c,y_2,x_2}$ for any $y_1, y_2 \in \{1, 2, ..., H^i\}$ and $x_1, x_2 \in \{1, 2, ..., W^i\}$), we arrive at the form of the Conditional BatchNorm [11]. Indeed, for any spatially-invariant conditional data, our method reduces to the Conditional BatchNorm. Similarly, we can arrive at the AdaIN [19] by replacing $\mathbf{m}$ with a real image, making the modulation parameters spatially-invariant, and setting $N = 1$. As the modulation parameters are adaptive to the input segmentation mask, the proposed SPADE is better suited for semantic image synthesis.

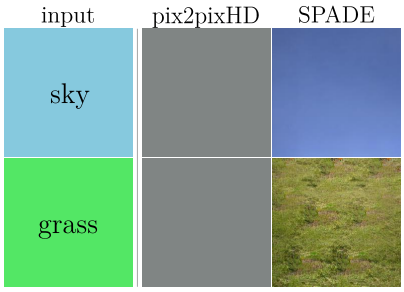


Figure 3: Comparing results given uniform segmentation maps: while the SPADE generator produces plausible textures, the pix2pixHD generator [48] produces two identical outputs due to the loss of the semantic information after the normalization layer.

SPADE generator. With the SPADE, there is no need to feed the segmentation map to the first layer of the generator, since the learned modulation parameters have encoded enough information about the label layout. Therefore, we discard encoder part of the generator, which is commonly used in recent architectures [22, 48]. This simplification results in a more lightweight network. Furthermore, similarly to existing class-conditional generators [36, 39, 54], the new generator can take a random vector as input, enabling a simple and natural way for multi-modal synthesis [20, 60].

Figure 4 illustrates our generator architecture, which employs several ResNet blocks [15] with upsampling layers. The modulation parameters of all the normalization layers are learned using the SPADE. Since each residual block operates at a different scale, we downsample the semantic mask to match the spatial resolution.

We train the generator with the same multi-scale discriminator and loss function used in pix2pixHD [48] except that we replace the least squared loss term [34] with the hinge loss term [31, 38, 54]. We test several ResNet-based discriminators used in recent unconditional GANs [1, 36, 39] but observe similar results at the cost of a higher GPU memory requirement. Adding the SPADE to the discriminator also yields a similar performance. For the loss function, we observe that removing any loss term in the pix2pixHD loss function lead to degraded generation results.

Why does the SPADE work better? A short answer is that it can better preserve semantic information against common normalization layers. Specifically, while normalization layers such as the InstanceNorm [46] are essential pieces in almost all the state-of-the-art conditional image synthesis models [48], they tend to wash away semantic information when applied to uniform or flat segmentation masks.

Let us consider a simple module that first applies convolution to a segmentation mask and then normalization. Furthermore, let us assume that a segmentation mask with a single label is given as input to the module (e.g., all the

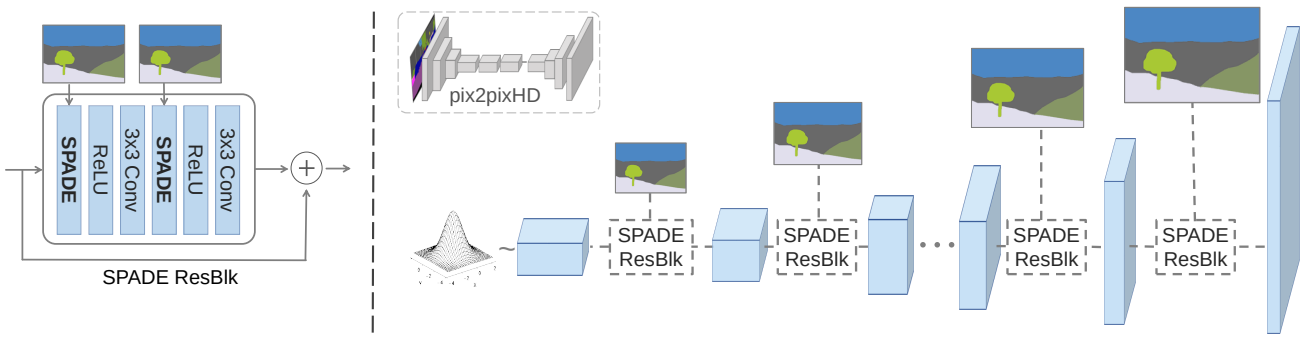


Figure 4: In the SPADE generator, each normalization layer uses the segmentation mask to modulate the layer activations. *(left)* Structure of one residual block with the SPADE. *(right)* The generator contains a series of the SPADE residual blocks with upsampling layers. Our architecture achieves better performance with a smaller number of parameters by removing the downsampling layers of leading image-to-image translation networks such as the pix2pixHD model [48].

pixels have the same label such as sky or grass). Under this setting, the convolution outputs are again uniform, with different labels having different uniform values. Now, after we apply InstanceNorm to the output, the normalized activation will become all zeros no matter what the input semantic label is given. Therefore, semantic information is totally lost. This limitation applies to a wide range of generator architectures, including pix2pixHD and its variant that concatenates the semantic mask at all intermediate layers, as long as a network applies convolution and then normalization to the semantic mask. In Figure 3, we empirically show this is precisely the case for pix2pixHD. Because a segmentation mask consists of a few uniform regions in general, the issue of information loss emerges when applying normalization.

In contrast, the segmentation mask in the SPADE Generator is fed through spatially adaptive modulation *without* normalization. Only activations from the previous layer are normalized. Hence, the SPADE generator can better preserve semantic information. It enjoys the benefit of normalization without losing the semantic input information.

Multi-modal synthesis. By using a random vector as the input of the generator, our architecture provides a simple way for multi-modal synthesis [20, 60]. Namely, one can attach an encoder that processes a real image into a random vector, which will be then fed to the generator. The encoder and generator form a VAE [28], in which the encoder tries to capture the style of the image, while the generator combines the encoded style and the segmentation mask information via the SPADEs to reconstruct the original image. The encoder also serves as a style guidance network at test time to capture the style of target images, as used in Figure 1. For training, we add a KL-Divergence loss term [28].

4. Experiments

Implementation details. We apply the Spectral Norm [38] to all the layers in both generator and discriminator. The learning rates for the generator and discriminator are 0.0001 and 0.0004, respectively [17]. We use the ADAM solver [27] with $\beta_1 = 0$ and $\beta_2 = 0.999$. All the experiments are conducted on an NVIDIA DGX1 with 8 32GB V100 GPUs. We use synchronized BatchNorm, i.e., these statistics are collected from all the GPUs.

Datasets. We conduct experiments on several datasets.

- *COCO-Stuff* [4] is derived from the COCO dataset [32]. It has 118,000 training images and 5,000 validation images captured from diverse scenes. It has 182 semantic classes. Due to its vast diversity, existing image synthesis models perform poorly on this dataset.
- *ADE20K* [58] consists of 20,210 training and 2,000 validation images. Similarly to the COCO, the dataset contains challenging scenes with 150 semantic classes.
- *ADE20K-outdoor* is a subset of the ADE20K dataset that only contains outdoor scenes, used in Qi *et al*. [43].
- *Cityscapes* dataset [9] contains street scene images in German cities. The training and validation set sizes are 3,000 and 500, respectively. Recent work has achieved photorealistic semantic image synthesis results [43, 47] on the Cityscapes dataset.
- *Flickr Landscapes.* We collect 41,000 photos from Flickr and use 1,000 samples for the validation set. To avoid expensive manual annotation, we use a well-trained DeepLabV2 [5] to compute input segmentation masks.

We train the competing semantic image synthesis methods on the same training set and report their results on the same validation set for each dataset.

Performance metrics. We adopt the evaluation protocol from previous work [6, 48]. Specifically, we run a semantic segmentation model on the synthesized images and compare how well the predicted segmentation mask matches the ground truth input. Intuitively, if the output images are realistic, a well-trained semantic segmentation model should be able to predict the ground truth label. For measuring the segmentation accuracy, we use both the mean Intersection-

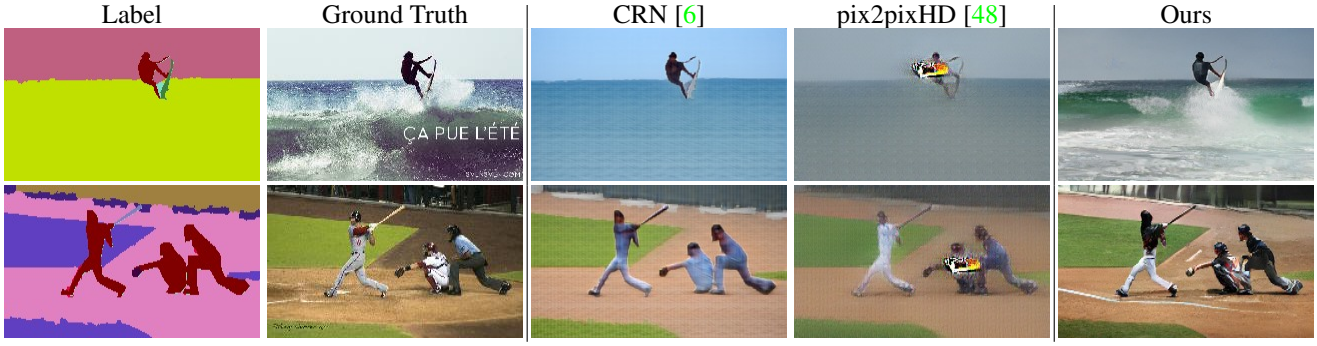


Figure 5: Visual comparison of semantic image synthesis results on the COCO-Stuff dataset. Our method successfully synthesizes realistic details from semantic labels.

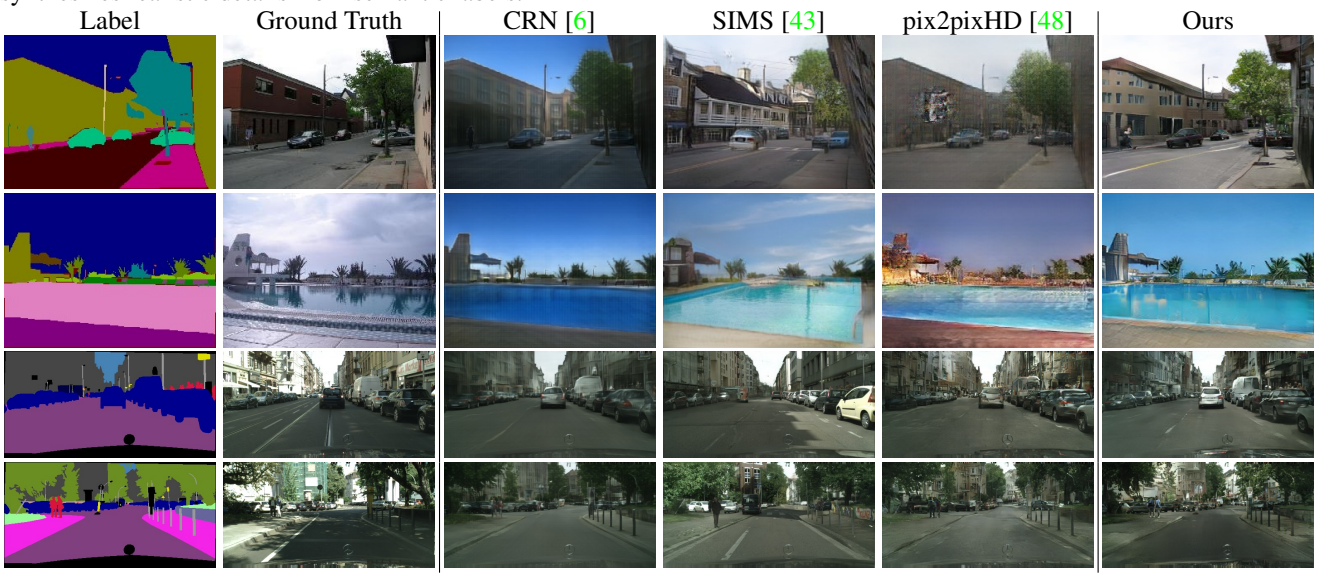


Figure 6: Visual comparison of semantic image synthesis results on the ADE20K outdoor and Cityscapes datasets. Our method produces realistic images while respecting the spatial semantic layout at the same time.

	COCO-Stuff			ADE20K			ADE20K-outdoor			Cityscapes		
Method	mIoU	accu	FID	mIoU	accu	FID	mIoU	accu	FID	mIoU	accu	FID
CRN [6]	23.7	40.4	70.4	22.4	68.8	73.3	16.5	68.6	99.0	52.4	77.1	104.7
SIMS [43]	N/A	N/A	N/A	N/A	N/A	N/A	13.1	74.7	67.7	47.2	75.5	**49.7**
pix2pixHD [48]	14.6	45.8	111.5	20.3	69.2	81.8	17.4	71.6	97.8	58.3	81.4	95.0
Ours	**37.4**	**67.9**	**22.6**	**38.5**	**79.9**	**33.9**	**30.8**	**82.9**	**63.3**	**62.3**	**81.9**	71.8

Table 1: Our method outperforms the current leading methods in semantic segmentation (mIoU and accu) and FID [17] scores on all the benchmark datasets. For the mIoU and accu, higher is better. For the FID, lower is better.

over-Union (mIoU) and the pixel accuracy (accu). We use the state-of-the-art segmentation networks for each dataset: DeepLabV2 [5, 40] for COCO-Stuff, UperNet101 [51] for ADE20K, and DRN-D-105 [53] for Cityscapes. In addition to the mIoU and the accu segmentation performance metrics, we use the Fréchet Inception Distance (FID) [17] to measure the distance between the distribution of synthesized results and the distribution of real images.

Baselines. We compare our method with 3 leading semantic image synthesis models: the pix2pixHD model [48], the cascaded refinement network (CRN) [6], and the semi-parametric image synthesis method (SIMS) [43]. The pix2pixHD is the current state-of-the-art GAN-based conditional image synthesis framework. The CRN uses a deep network that repeatedly refines the output from low to high resolution, while the SIMS takes a semi-parametric approach that composites real segments from a training set and refines the boundaries. Both the CRN and SIMS are mainly trained using image reconstruction loss. For a fair comparison, we train the CRN and pix2pixHD models using the implementations provided by the authors. As image synthesis using the SIMS requires many queries to the training

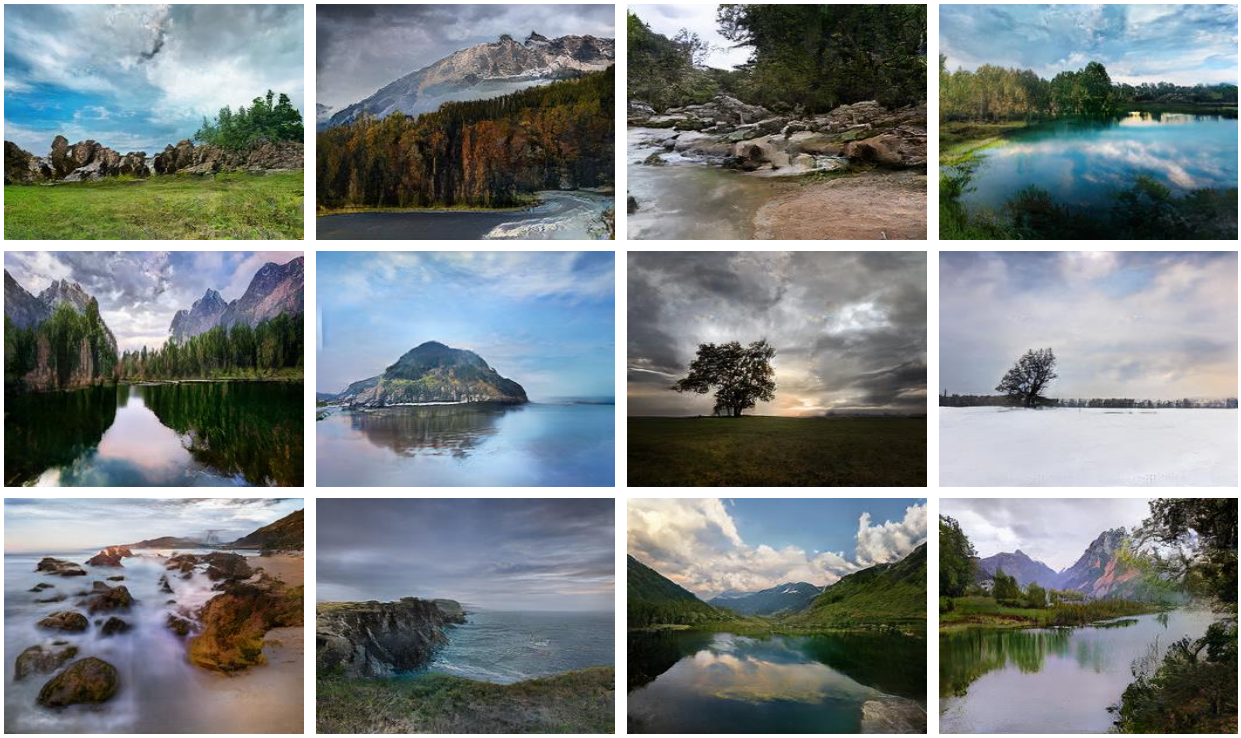


Figure 7: Semantic image synthesis results on the Flickr Landscapes dataset. The images were generated from semantic layout of photographs on the Flickr website.

dataset, it is computationally prohibitive for a large dataset such as the COCO-stuff and the full ADE20K. Therefore, we use the results provided by the authors when available.

Quantitative comparisons. As shown in Table 1, our method outperforms the current state-of-the-art methods by a large margin in all the datasets. For the COCO-Stuff, our method achieves an mIoU score of 35.2, which is about 1.5 times better than the previous leading method. Our FID is also 2.2 times better than the previous leading method. We note that the SIMS model produces a lower FID score but has poor segmentation performances on the Cityscapes dataset. This is because the SIMS synthesizes an image by first stitching image patches from the training dataset. As using the real image patches, the resulting image distribution can better match the distribution of real images. However, because there is no guarantee that a perfect query (e.g., a person in a particular pose) exists in the dataset, it tends to copy objects that do not match the input segments.

Qualitative results. In Figures 5 and 6, we provide qualitative comparisons of the competing methods. We find that our method produces results with much better visual quality and fewer visible artifacts, especially for diverse scenes in the COCO-Stuff and ADE20K dataset. When the training dataset size is small, the SIMS model also renders images with good visual quality. However, the depicted content often deviates from the input segmentation mask (e.g., the shape of the swimming pool in the second row of Figure 6).

Dataset	Ours vs. CRN	Ours vs. pix2pixHD	Ours vs. SIMS
COCO-Stuff	79.76	86.64	N/A
ADE20K	76.66	83.74	N/A
ADE20K-outdoor	66.04	79.34	85.70
Cityscapes	63.60	53.64	51.52

Table 2: User preference study. The numbers indicate the percentage of users who favor the results of the proposed method over those of the competing method.

In Figures 7 and 8, we show more example results from the Flickr Landscape and COCO-Stuff datasets. The proposed method can generate diverse scenes with high image fidelity. More results are included in the appendix.

Human evaluation. We use the Amazon Mechanical Turk (AMT) to compare the perceived visual fidelity of our method against existing approaches. Specifically, we give the AMT workers an input segmentation mask and two synthesis outputs from different methods and ask them to choose the output image that looks more like a corresponding image of the segmentation mask. The workers are given unlimited time to make the selection. For each comparison, we randomly generate 500 questions for each dataset, and each question is answered by 5 different workers. For quality control, only workers with a lifetime task approval rate greater than 98% can participate in our study.

Table 2 shows the evaluation results. We find that users

Figure 8: Semantic image synthesis results on COCO-Stuff. Our method successfully generates realistic images in diverse scenes ranging from animals to sports activities.

Method	#param	COCO.	ADE.	City.
decoder w/ SPADE (Ours)	96M	**35.2**	38.5	62.3
compact decoder w/ SPADE	61M	**35.2**	38.0	**62.5**
decoder w/ Concat	79M	31.9	33.6	61.1
pix2pixHD++ w/ SPADE	237M	34.4	**39.0**	62.2
pix2pixHD++ w/ Concat	195M	32.9	38.9	57.1
pix2pixHD++	183M	32.7	38.3	58.8
compact pix2pixHD++	103M	31.6	37.3	57.6
pix2pixHD [48]	183M	14.6	20.3	58.3

Table 3: The mIoU scores are boosted when the SPADE is used, for both the decoder architecture (Figure 4) and encoder-decoder architecture of pix2pixHD++ (our improved baseline over pix2pixHD [48]). On the other hand, simply concatenating semantic input at every layer fails to do so. Moreover, our compact model with smaller depth at all layers outperforms all the baselines.

Method	COCO	ADE20K	Cityscapes
segmap input	35.2	38.5	62.3
random input	35.3	38.3	61.6
kernelsize 5x5	35.0	39.3	61.8
kernelsize 3x3	35.2	38.5	62.3
kernelsize 1x1	32.7	35.9	59.9
#params 141M	35.3	38.3	62.5
#params 96M	35.2	38.5	62.3
#params 61M	35.2	38.0	62.5
Sync BatchNorm	35.0	39.3	61.8
BatchNorm	33.7	37.9	61.8
InstanceNorm	33.9	37.4	58.7

Table 4: The SPADE generator works with different configurations. We change the input of the generator, the convolutional kernel size acting on the segmentation map, the capacity of the network, and the parameter-free normalization method. The settings used in the paper are boldfaced.

strongly favor our results on all the datasets, especially on the challenging COCO-Stuff and ADE20K datasets. For the Cityscapes, even when all the competing methods achieve high image fidelity, users still prefer our results.

Effectiveness of the SPADE. For quantifying importance of the SPADE, we introduce a strong baseline called pix2pixHD++, which combines all the techniques we find useful for enhancing the performance of pix2pixHD except the SPADE. We also train models that receive the segmentation mask input at all the intermediate layers via feature concatenation in the channel direction, which is termed as pix2pixHD++ w/ Concat. Finally, the model that combines the strong baseline with the SPADE is denoted as pix2pixHD++ w/ SPADE.

As shown in Table 3, the architectures with the proposed SPADE consistently outperforms its counterparts, in both the decoder-style architecture described in Figure 4 and more traditional encoder-decoder architecture used in the pix2pixHD. We also find that concatenating segmentation masks at all intermediate layers, a reasonable alternative to the SPADE, does not achieve the same performance as SPADE. Furthermore, the decoder-style SPADE generator works better than the strong baselines even with a smaller number of parameters.

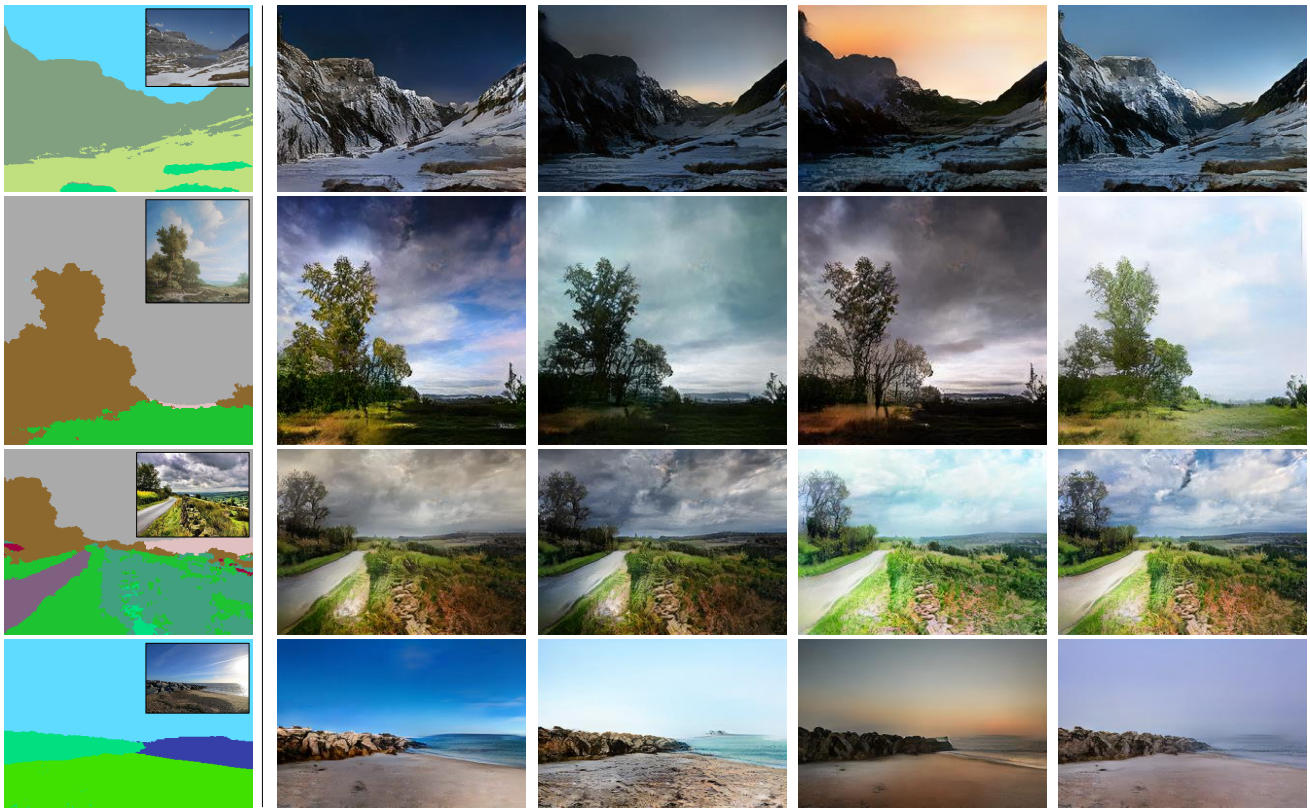


Figure 9: Our model attains multimodal synthesis capability when trained with the image encoder. During deployment, by using different random noise, our model synthesizes outputs with diverse appearances but all having the same semantic layouts depicted in the input mask. For reference, the ground truth image is shown inside the input segmentation mask.

Variations of SPADE generator. Table 4 reports the performance of several variations of our generator. First, we compare two types of input to the generator where one is the random noise while the other is the downsampled segmentation map. We find that both of the variants render similar performance and conclude that the modulation by SPADE alone provides sufficient signal about the input mask. Second, we vary the type of parameter-free normalization layers before applying the modulation parameters. We observe that the SPADE works reliably across different normalization methods. Next, we vary the convolutional kernel size acting on the label map, and find that kernel size of 1x1 hurts performance, likely because it prohibits utilizing the context of the label. Lastly, we modify the capacity of the generator by changing the number of convolutional filters. We present more variations and ablations in the appendix.

Multi-modal synthesis. In Figure 9, we show the multimodal image synthesis results on the Flickr Landscape dataset. For the same input segmentation mask, we sample different noise inputs to achieve different outputs. More results are included in the appendix.

Semantic manipulation and guided image synthesis. In Figure 1, we show an application where a user draws different segmentation masks, and our model renders the corresponding landscape images. Moreover, our model allows users to choose an external style image to control the global appearances of the output image. We achieve it by replacing the input noise with the embedding vector of the style image computed by the image encoder.

5. Conclusion

We have proposed the spatially-adaptive normalization, which utilizes the input semantic layout while performing the affine transformation in the normalization layers. The proposed normalization leads to the first semantic image synthesis model that can produce photorealistic outputs for diverse scenes including indoor, outdoor, landscape, and street scenes. We further demonstrate its application for multi-modal synthesis and guided image synthesis.

Acknowledgments. We thank Alexei A. Efros, Bryan Catanzaro, Andrew Tao, and Jan Kautz for insightful advice. We thank Chris Hebert, Gavriil Klimov, and Brad Nemire for their help in constructing the demo apps. Taesung Park contributed to the work during his internship at NVIDIA. His Ph.D. is supported by a Samsung Scholarship.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning (ICML)*, 2017. 3

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2

[3] A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations (ICLR)*, 2019. 1, 2

[4] H. Caesar, J. Uijlings, and V. Ferrari. Coco-stuff: Thing and stuff classes in context. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 4

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(4):834–848, 2018. 4, 5

[6] Q. Chen and V. Koltun. Photographic image synthesis with cascaded refinement networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 1, 4, 5, 13, 14, 15, 16, 17, 18

[7] T. Chen, M.-M. Cheng, P. Tan, A. Shamir, and S.-M. Hu. Sketch2photo: internet image montage. *ACM Transactions on Graphics (TOG)*, 28(5):124, 2009. 1, 2

[8] T. Chen, M. Lucic, N. Houlsby, and S. Gelly. On self modulation for generative adversarial networks. In *International Conference on Learning Representations*, 2019. 2

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2, 4

[10] H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville. Modulating early visual processing by language. In *Advances in Neural Information Processing Systems*, 2017. 2

[11] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. In *International Conference on Learning Representations (ICLR)*, 2016. 2, 3

[12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 12, 13

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014. 2

[14] J. Hays and A. A. Efros. Scene completion using millions of photographs. In *ACM SIGGRAPH*, 2007. 1

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[16] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. 2001. 1, 2

[17] 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 *Advances in Neural Information Processing Systems*, 2017. 4, 5, 13

[18] S. Hong, D. Yang, J. Choi, and H. Lee. Inferring semantic layout for hierarchical text-to-image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[19] X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 2, 3

[20] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz. Multimodal unsupervised image-to-image translation. *European Conference on Computer Vision (ECCV)*, 2018. 2, 3, 4

[21] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. 2, 3

[22] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 2, 3, 11, 12

[23] M. Johnson, G. J. Brostow, J. Shotton, O. Arandjelovic, V. Kwatra, and R. Cipolla. Semantic photo synthesis. In *Computer Graphics Forum*, volume 25, pages 407–413, 2006. 1, 2

[24] L. Karacan, Z. Akata, A. Erdem, and E. Erdem. Learning to generate images of outdoor scenes from attributes and semantic layouts. *arXiv preprint arXiv:1612.00215*, 2016. 2

[25] L. Karacan, Z. Akata, A. Erdem, and E. Erdem. Manipulating attributes of natural scenes via hallucination. *arXiv preprint arXiv:1808.07413*, 2018. 2

[26] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 4

[28] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014. 2, 4, 11, 12

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. 2

[30] J.-F. Lalonde, D. Hoiem, A. A. Efros, C. Rother, J. Winn, and A. Criminisi. Photo clip art. In *ACM transactions on graphics (TOG)*, volume 26, page 3. ACM, 2007. 1

[31] J. H. Lim and J. C. Ye. Geometric gan. *arXiv preprint arXiv:1705.02894*, 2017. 3, 11

[32] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014. 2, 4

[33] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. In *Advances in Neural Information Processing Systems*, 2017. 2

[34] X. Mao, Q. Li, H. Xie, Y. R. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 3, 11

[35] T. B. Mathias Eitz, Kristian Hildebrand and M. Alexa. Photosketch: A sketch based image query and compositing system. In *ACM SIGGRAPH 2009 Talk Program*, 2009. 1

[36] L. Mescheder, A. Geiger, and S. Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine Learning (ICML)*, 2018. 2, 3, 11

[37] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 2

[38] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018. 3, 4, 11

[39] T. Miyato and M. Koyama. cGANs with projection discriminator. In *International Conference on Learning Representations (ICLR)*, 2018. 2, 3, 11

[40] K. Nakashima. Deeplab-pytorch. https://github.com/kazuto1011/deeplab-pytorch, 2018. 5

[41] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier GANs. In *International Conference on Machine Learning (ICML)*, 2017. 2

[42] E. Perez, H. De Vries, F. Strub, V. Dumoulin, and A. Courville. Learning visual reasoning without strong priors. In *International Conference on Machine Learning (ICML)*, 2017. 2

[43] X. Qi, Q. Chen, J. Jia, and V. Koltun. Semi-parametric image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 4, 5, 13, 17, 18

[44] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *International Conference on Machine Learning (ICML)*, 2016. 2

[45] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, 2016. 2

[46] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. arxiv 2016. *arXiv preprint arXiv:1607.08022*, 2016. 2, 3

[47] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz, and B. Catanzaro. Video-to-video synthesis. In *Advances in Neural Information Processing Systems*, 2018. 1, 4

[48] 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1, 3, 4, 5, 7, 11, 12, 13, 14, 15, 16, 17, 18

[49] X. Wang, K. Yu, C. Dong, and C. Change Loy. Recovering realistic texture in image super-resolution by deep spatial feature transform. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 606–615, 2018. 2

[50] Y. Wu and K. He. Group normalization. In *European Conference on Computer Vision (ECCV)*, 2018. 2

[51] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun. Unified perceptual parsing for scene understanding. In *European Conference on Computer Vision (ECCV)*, 2018. 5

[52] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[53] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 5

[54] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. In *International Conference on Machine Learning (ICML)*, 2019. 1, 2, 3, 11

[55] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 1, 2

[56] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2018. 1

[57] B. Zhao, L. Meng, W. Yin, and L. Sigal. Image generation from layout. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[58] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 4

[59] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 2

[60] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman. Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems*, 2017. 2, 3, 4

A. Additional Implementation Details

Generator. The architecture of the generator consists of a series of the proposed SPADE ResBlks with nearest neighbor upsampling. We train our network using 8 GPUs simultaneously and use the synchronized version of the BatchNorm. We apply the Spectral Norm [38] to all the convolutional layers in the generator. The architectures of the proposed SPADE and SPADE ResBlk are given in Figure 10 and Figure 11, respectively. The architecture of the generator is shown in Figure 12.

Discriminator. The architecture of the discriminator follows the one used in the pix2pixHD method [48], which uses a multi-scale design with the InstanceNorm (IN). The only difference is that we apply the Spectral Norm to all the convolutional layers of the discriminator. The details of the discriminator architecture is shown in Figure 13.

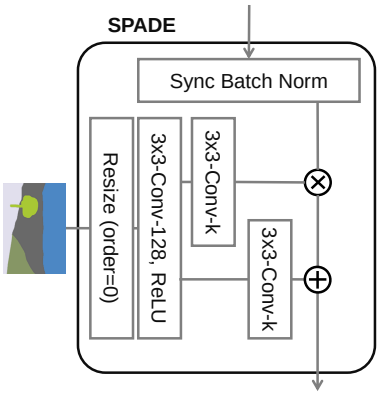


Figure 10: SPADE Design. The term 3x3-Conv-k denotes a 3-by-3 convolutional layer with k convolutional filters. The segmentation map is resized to match the resolution of the corresponding feature map using nearest-neighbor downsampling.

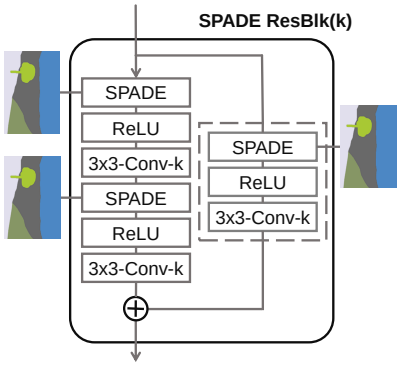


Figure 11: SPADE ResBlk. The residual block design largely follows that in Mescheder *et al*. [36] and Miyato *et al*. [39]. We note that for the case that the number of channels before and after the residual block is different, the skip connection is also learned (dashed box in the figure).

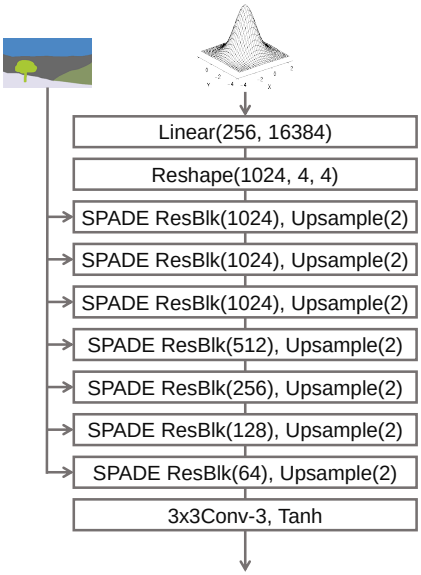


Figure 12: SPADE Generator. Different from prior image generators [22, 48], the semantic segmentation mask is passed to the generator through the proposed SPADE ResBlks in Figure 11.

Image Encoder. The image encoder consists of 6 stride-2 convolutional layers followed by two linear layers to produce the mean and variance of the output distribution as shown in Figure 14.

Learning objective. We use the learning objective function in the pix2pixHD work [48] except that we replace its LSGAN loss [34] term with the Hinge loss term [31, 38, 54]. We use the same weighting among the loss terms in the objective function as that in the pix2pixHD work.

When training the proposed framework with the image encoder for multi-modal synthesis and style-guided image synthesis, we include a KL Divergence loss:

$$\mathcal{L}_{\text{KLD}} = \mathcal{D}_{\text{KL}}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

where the prior distribution $p(\mathbf{z})$ is a standard Gaussian distribution and the variational distribution q is fully determined by a mean vector and a variance vector [28]. We use the reparamterization trick [28] for back-propagating the gradient from the generator to the image encoder. The weight for the KL Divergence loss is 0.05.

In Figure 15, we overview the training data flow. The image encoder encodes a real image to a mean vector and a variance vector. They are used to compute the noise input to the generator via the reparameterization trick [28]. The generator also takes the segmentation mask of the input image as input with the proposed SPADE ResBlks. The

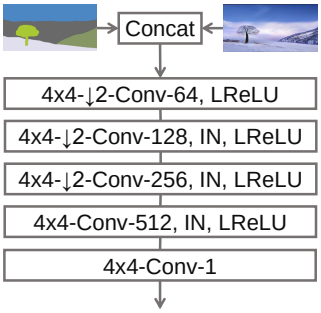


Figure 13: Our discriminator design largely follows that in the pix2pixHD [48]. It takes the concatenation the segmentation map and the image as input. It is based on the PatchGAN [22]. Hence, the last layer of the discriminator is a convolutional layer.

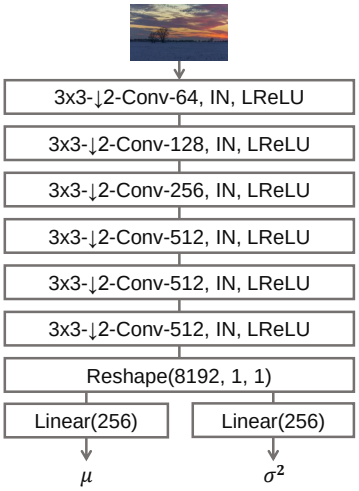


Figure 14: The image encoder consists a series of convolutional layers with stride 2 followed by two linear layers that output a mean vector μ and a variance vector σ.

discriminator takes concatenation of the segmentation mask and the output image from the generator as input and aims to classify that as fake.

Training details. We perform 200 epochs of training on the Cityscapes and ADE20K datasets, 100 epochs of training on the COCO-Stuff dataset, and 50 epochs of training on the Flickr Landscapes dataset. The image sizes are 256×256, except the Cityscapes at 512×256. We linearly decay the learning rate to 0 from epoch 100 to 200 for the Cityscapes and ADE20K datasets. The batch size is 32. We initialize the network weights using thes Glorot initialization [12].

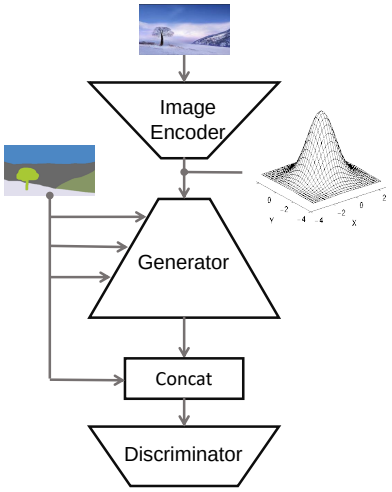


Figure 15: The image encoder encodes a real image to a latent representation for generating a mean vector and a variance vector. They are used to compute the noise input to the generator via the reparameterization trick [28]. The generator also takes the segmentation mask of the input image as input via the proposed SPADE ResBlks. The discriminator takes concatenation of the segmentation mask and the output image from the generator as input and aims to classify that as fake.

B. Additional Ablation Study

Method	COCO.	ADE.	City.
Ours	35.2	38.5	62.3
Ours w/o Perceptual loss	24.7	30.1	57.4
Ours w/o GAN feature matching loss	33.2	38.0	62.2
Ours w/ a deeper discriminator	34.9	38.3	60.9
pix2pixHD++ w/ SPADE	34.4	39.0	62.2
pix2pixHD++	32.7	38.3	58.8
pix2pixHD++ w/o Sync BatchNorm	27.4	31.8	51.1
pix2pixHD++ w/o Sync BatchNorm, and w/o Spectral Norm	26.0	31.9	52.3
pix2pixHD [48]	14.6	20.3	58.3

Table 5: Additional ablation study results using the mIoU metric: the table shows that both the perceptual loss and GAN feature matching loss terms are important. Making the discriminator deeper does not lead to a performance boost. The table also shows that all the components (Synchronized BatchNorm, Spectral Norm, TTUR, the Hinge loss objective, and the SPADE) used in the proposed method helps our strong baseline, pix2pixHD++.

Table 5 provides additional ablation study results analyzing the contribution of individual components in the proposed method. We first find that both of the perceptual loss and GAN feature matching loss inherited from the learning objective function of the pix2pixHD [48] are important. Removing any of them leads to a performance drop. We also find that increasing the depth of the discriminator by inserting one more convolutional layer to the top of the pix2pixHD discriminator does not improve the results.

In Table 5, we also analyze the effectiveness of each component used in our strong baseline, the pix2pixHD++ method, derived from the pix2pixHD method. We found that the Spectral Norm, synchronized BatchNorm, TTUR [17], and the hinge loss objective all contribute to the performance boost. Adding the SPADE to the strong baseline further improves the performance. Note that the pix2pixHD++ w/o Sync BatchNorm and w/o Spectral Norm still differs from the pix2pixHD in that it uses the hinge loss objective, TTUR, a large batch size, and the Glorot initialization [12].

C. Additional Results

In Figure 16, 17, and 18, we show additional synthesis results from the proposed method on the COCO-Stuff and ADE20K datasets with comparisons to those from the CRN [6] and pix2pixHD [48] methods.

In Figure 19 and 20, we show additional synthesis results from the proposed method on the ADE20K-outdoor and Cityscapes datasets with comparison to those from the CRN [6], SIMS [43], and pix2pixHD [48] methods.

In Figure 21, we show additional multi-modal synthesis results from the proposed method. As sampling different $\mathbf{z}$ from a standard multivariate Gaussian distribution, we synthesize images of diverse appearances.

In the accompanying video, we demonstrate our semantic image synthesis interface. We show how a user can create photorealistic landscape images by painting semantic labels on a canvas. We also show how a user can synthesize images of diverse appearances for the same semantic segmentation mask as well as transfer the appearance of a provided style image to the synthesized one.

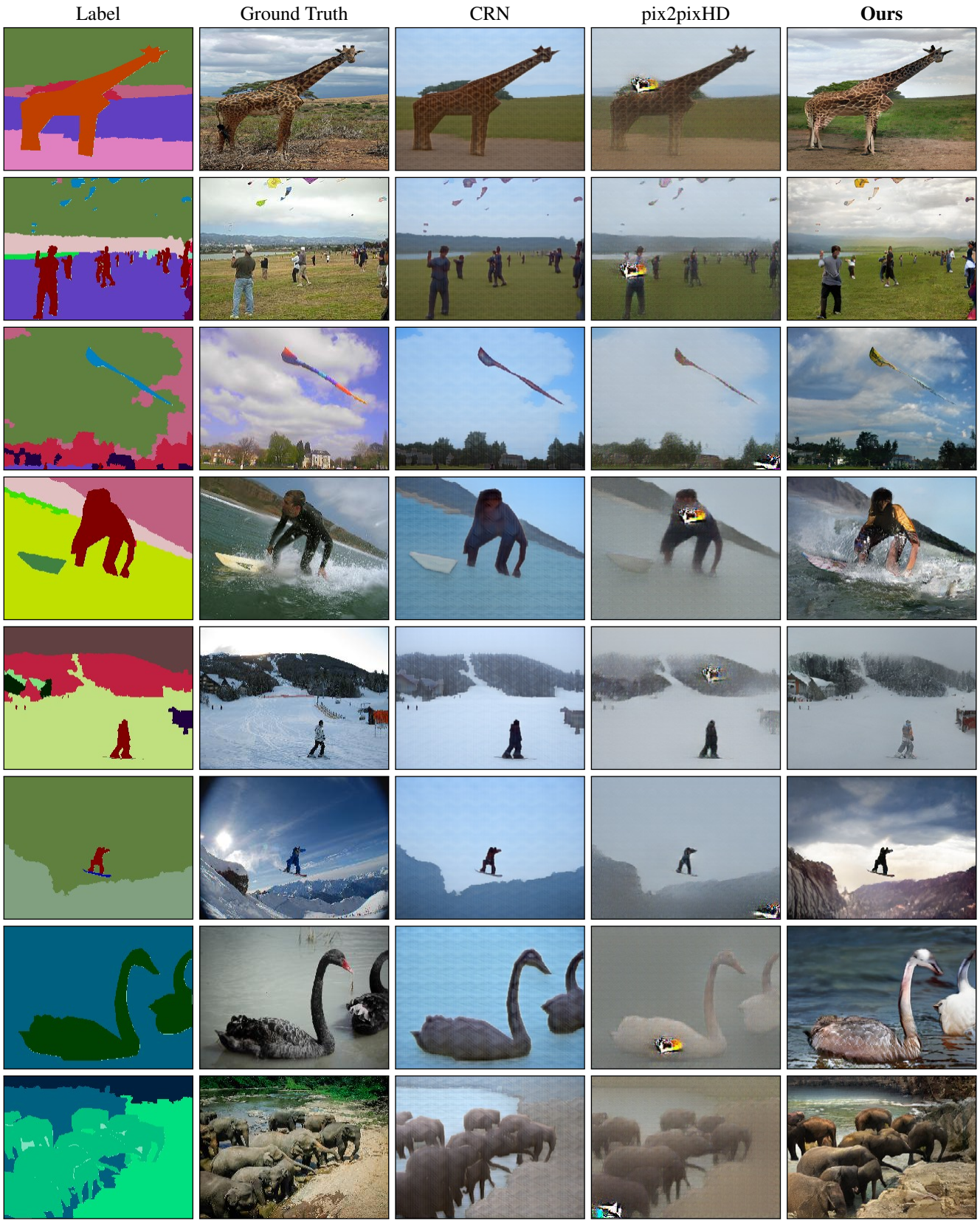


Figure 16: Additional results with comparison to those from the CRN [6] and pix2pixHD [48] methods on the COCO-Stuff dataset.

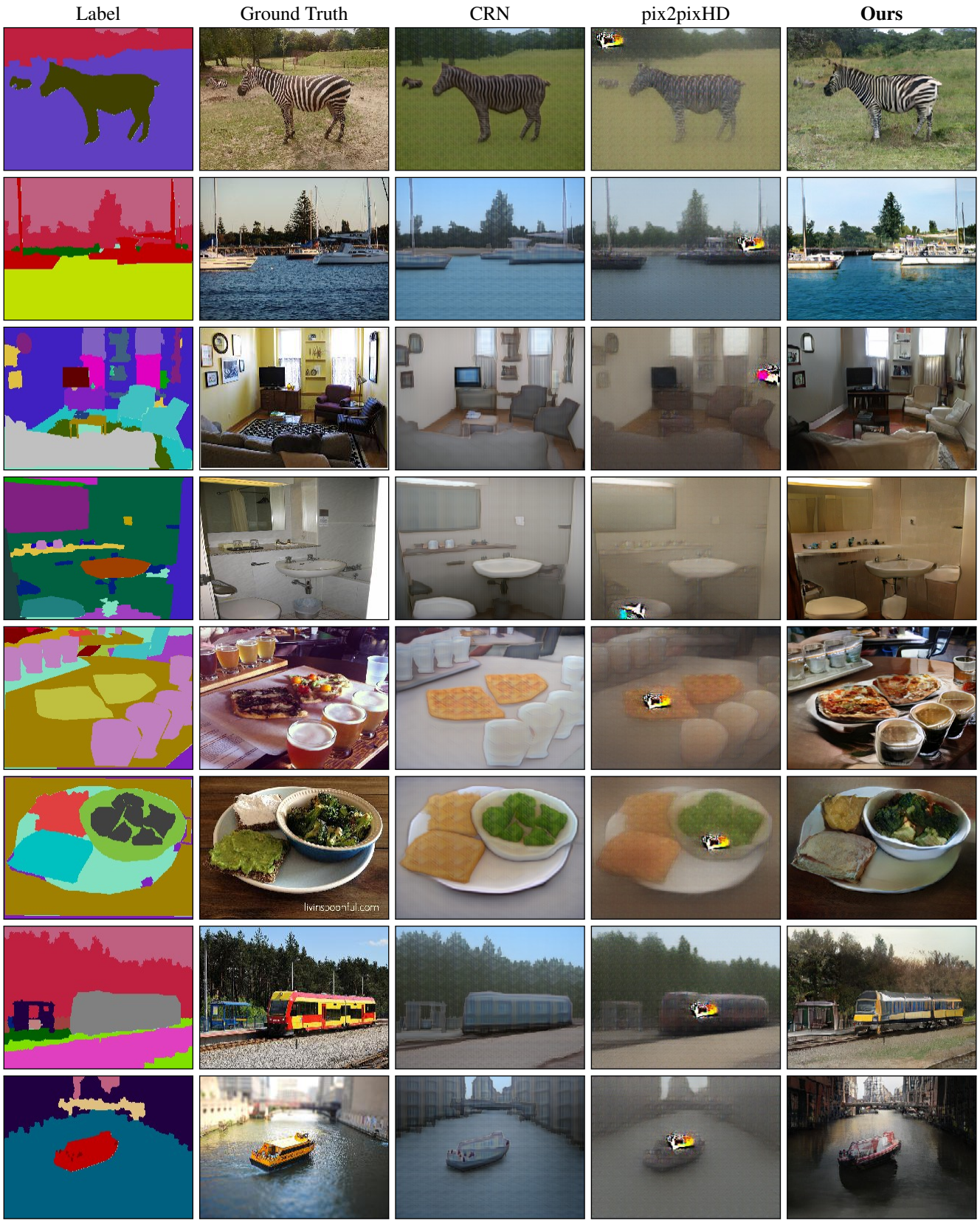


Figure 17: Additional results with comparison to those from the CRN [6] and pix2pixHD [48] methods on the COCO-Stuff dataset.

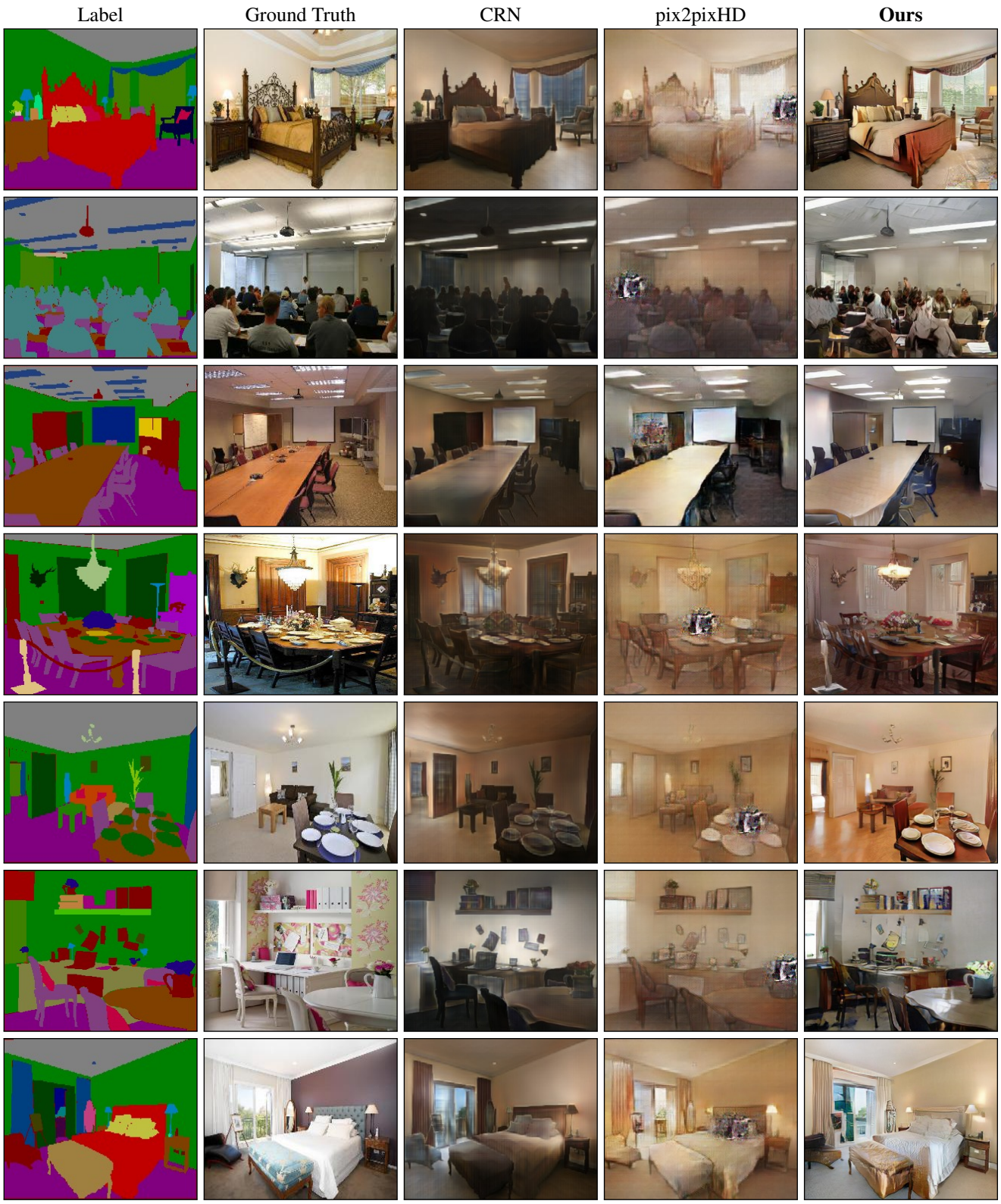


Figure 18: Additional results with comparison to those from the CRN [6] and pix2pixHD [48] methods on the ADE20K dataset.

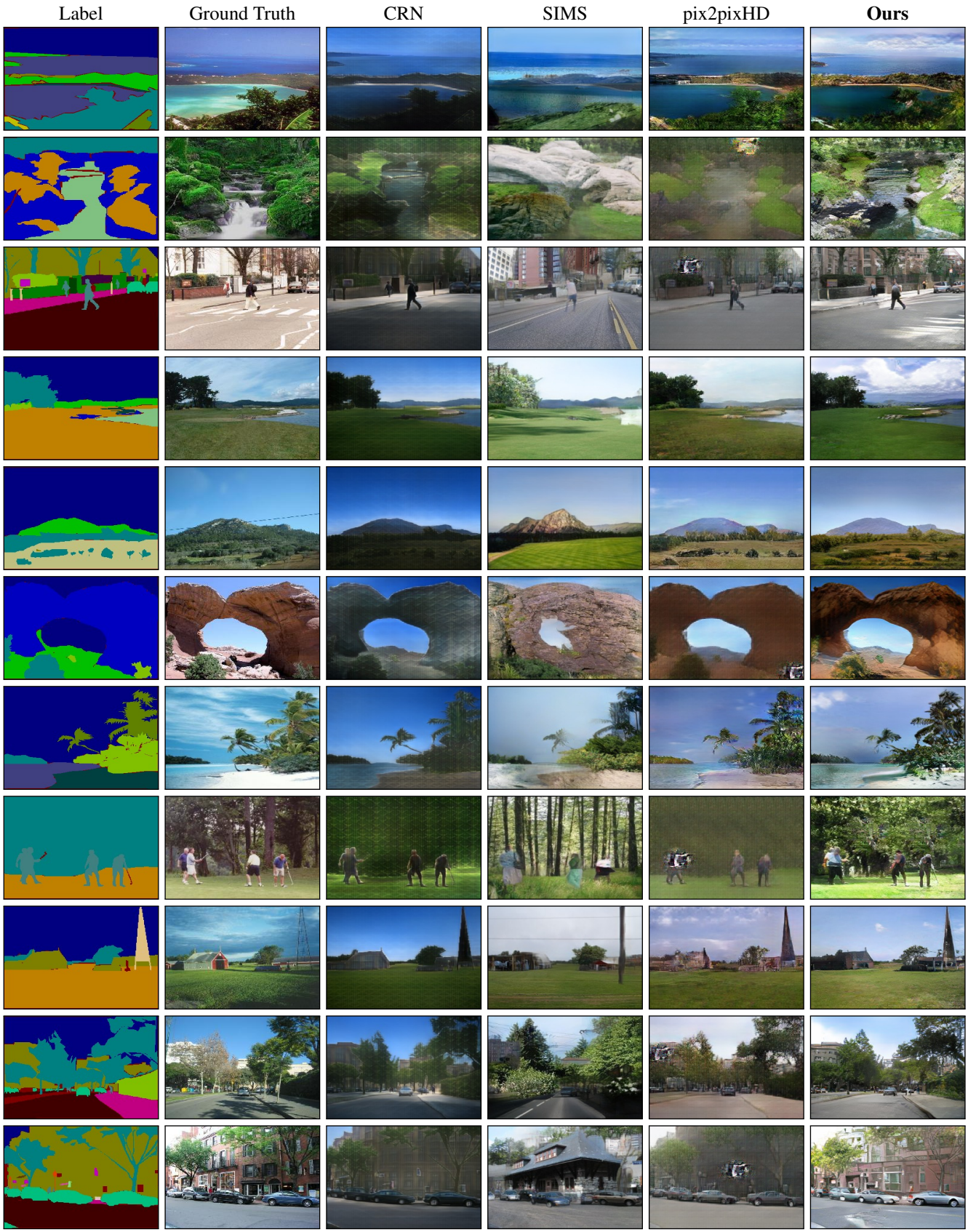


Figure 19: Additional results with comparison to those from the CRN [6], SIMS [43], and pix2pixHD [48] methods on the ADE20K-outdoor dataset.

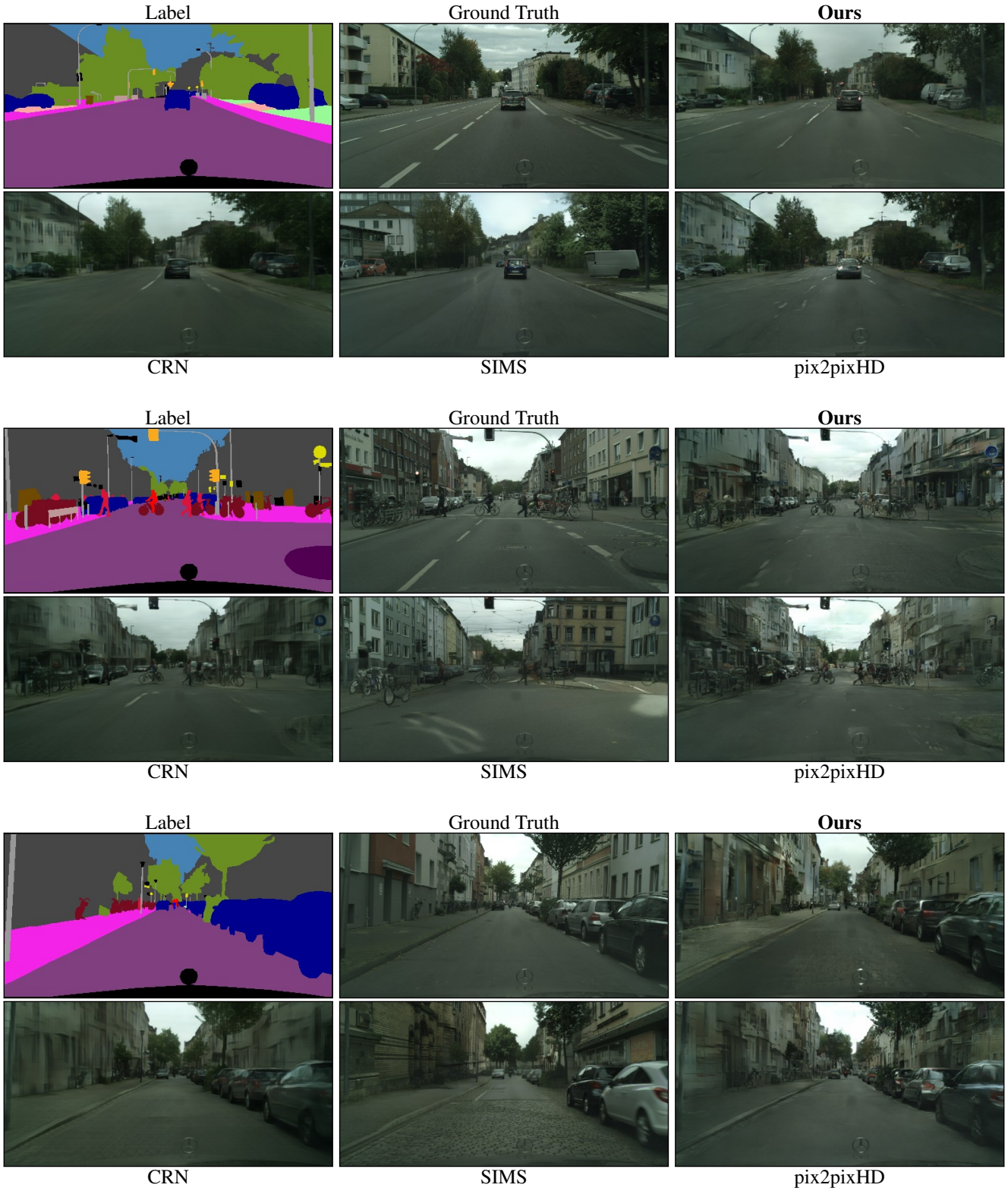


Figure 20: Additional results with comparison to those from the CRN [6], SIMS [43], and pix2pixHD [48] methods on the Cityscapes dataset.

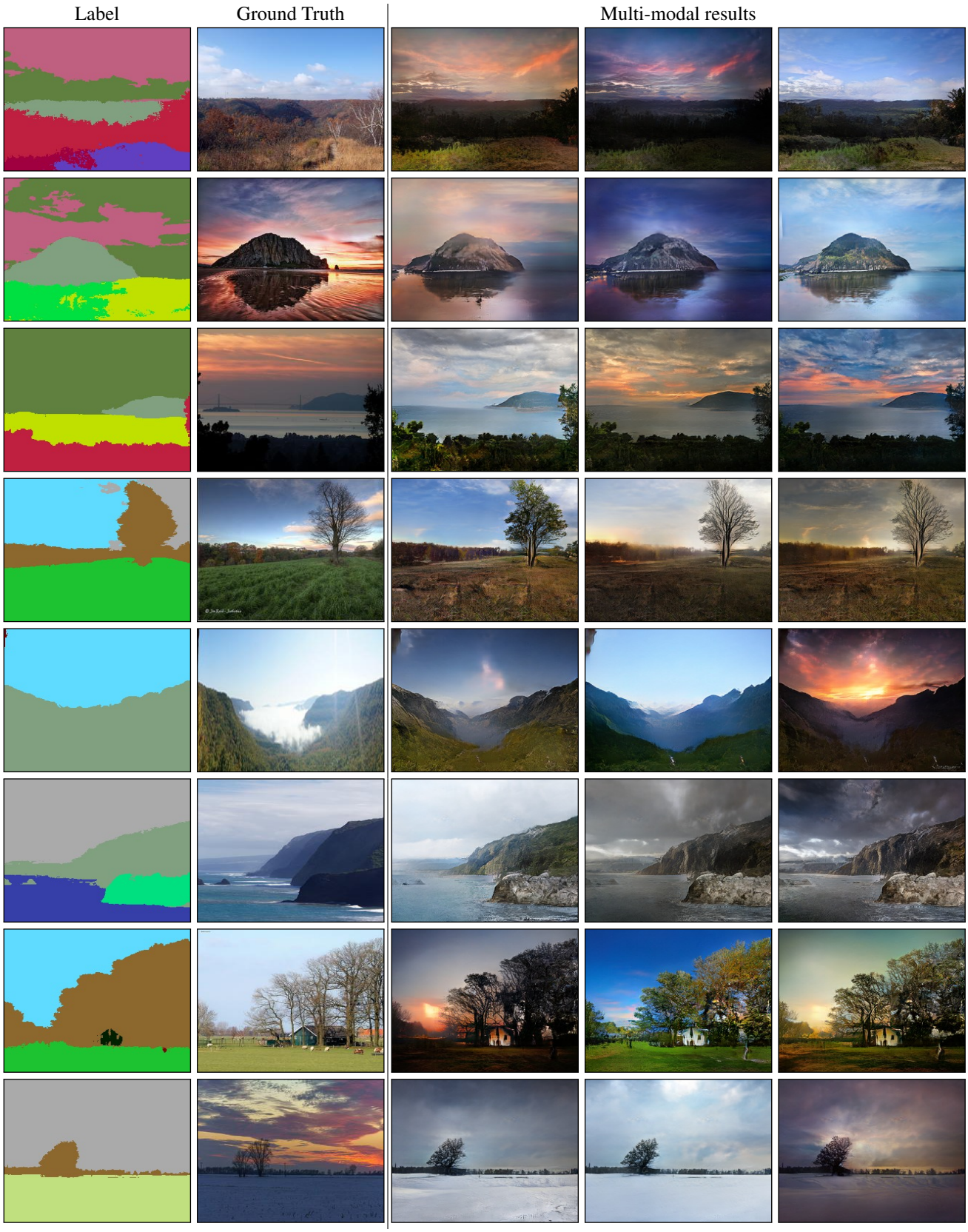


Figure 21: Additional multi-modal synthesis results on the Flickr Landscapes Dataset. By sampling latent vectors from a standard Gaussian distribution, we synthesize images of diverse appearances.

Few-Shot Adversarial Learning of Realistic Neural Talking Head Models

Egor Zakharov[1,2] Aliaksandra Shysheya[1,2] Egor Burkov[1,2] Victor Lempitsky[1,2]
[1]Samsung AI Center, Moscow [2]Skolkovo Institute of Science and Technology

Figure 1: The results of talking head image synthesis using face landmark tracks extracted from a different video sequence of the same person (on the left), and using face landmarks of a different person (on the right). The **results** are conditioned on the **landmarks** taken from the **target** frame, while the **source** frame is an example from the training set. The talking head models on the left were trained using eight frames, while the models on the right were trained in a one-shot manner.

Abstract

Several recent works have shown how highly realistic human head images can be obtained by training convolutional neural networks to generate them. In order to create a personalized talking head model, these works require training on a large dataset of images of a single person. However, in many practical scenarios, such personalized talking head models need to be learned from a few image views of a person, potentially even a single image. Here, we present a system with such few-shot capability. It performs lengthy meta-learning on a large dataset of videos, and after that is able to frame few- and one-shot learning of neural talking head models of previously unseen people as adversarial training problems with high capacity generators and discriminators. Crucially, the system is able to initialize the parameters of both the generator and the discriminator in a person-specific way, so that training can be based on just a few images and done quickly, despite the need to tune tens of millions of parameters. We show that such an approach is able to learn highly realistic and personalized talking head models of new people and even portrait paintings.

1. Introduction

In this work, we consider the task of creating personalized photorealistic talking head models, i.e. systems that can synthesize plausible video-sequences of speech expressions and mimics of a particular individual. More specifically, we consider the problem of synthesizing photorealistic personalized head images given a set of face landmarks, which drive the animation of the model. Such ability has practical applications for telepresence, including videoconferencing and multi-player games, as well as special effects industry. Synthesizing realistic talking head sequences is known to be hard for two reasons. First, human heads have high photometric, geometric and kinematic complexity. This complexity stems not only from modeling faces (for which a large number of modeling approaches exist) but also from modeling mouth cavity, hair, and garments. The second complicating factor is the acuteness of the human visual system towards even minor mistakes in the appearance modeling of human heads (the so-called *uncanny valley* effect [25]). Such low tolerance to modeling mistakes explains the current prevalence of non-photorealistic cartoon-like avatars in many practically-deployed teleconferencing systems.

To overcome the challenges, several works have proposed to synthesize articulated head sequences by warping a single or multiple static frames. Both classical warping algorithms [4, 30] and warping fields synthesized using machine learning (including deep learning) [11, 31, 42] can be used for such purposes. While warping-based systems can create talking head sequences from as little as a single image, the amount of motion, head rotation, and disocclusion

that they can handle without noticeable artifacts is limited.

Direct (warping-free) synthesis of video frames using adversarially-trained deep convolutional networks (ConvNets) presents the new hope for photorealistic talking heads. Very recently, some remarkably realistic results have been demonstrated by such systems [17, 21, 39]. However, to succeed, such methods have to train large networks, where both generator and discriminator have tens of millions of parameters for each talking head. These systems, therefore, require a several-minutes-long video [21, 39] or a large dataset of photographs [17] as well as hours of GPU training in order to create a new personalized talking head model. While this effort is lower than the one required by systems that construct photo-realistic head models using sophisticated physical and optical modeling [1], it is still excessive for most practical telepresence scenarios, where we want to enable users to create their personalized head models with as little effort as possible.

In this work, we present a system for creating talking head models from a handful of photographs (so-called *few-shot learning*) and with limited training time. In fact, our system can generate a reasonable result based on a single photograph (*one-shot learning*), while adding a few more photographs increases the fidelity of personalization. Similarly to [17, 21, 39], the talking heads created by our model are deep ConvNets that synthesize video frames in a direct manner by a sequence of convolutional operations rather than by warping. The talking heads created by our system can, therefore, handle a large variety of poses that goes beyond the abilities of warping-based systems.

The few-shot learning ability is obtained through extensive pre-training (*meta-learning*) on a large corpus of talking head videos corresponding to different speakers with diverse appearance. In the course of meta-learning, our system simulates few-shot learning tasks and learns to transform landmark positions into realistically-looking personalized photographs, given a small training set of images with this person. After that, a handful of photographs of a new person sets up a new adversarial learning problem with high-capacity generator and discriminator pre-trained via meta-learning. The new adversarial problem converges to the state that generates realistic and personalized images after a few training steps.

In the experiments, we provide comparisons of talking heads created by our system with alternative neural talking head models [17, 42] via quantitative measurements and a user study, where our approach generates images of sufficient realism and personalization fidelity to deceive the study participants. We demonstrate several uses of our talking head models, including video synthesis using landmark tracks extracted from video sequences of the same person, as well as *puppeteering* (video synthesis of a certain person based on the face landmark tracks of a different person).

2. Related work

A huge body of works is devoted to statistical modeling of the appearance of human faces [5], with remarkably good results obtained both with classical techniques [37] and, more recently, with deep learning [23, 26] (to name just a few). While modeling faces is a highly related task to talking head modeling, the two tasks are not identical, as the latter also involves modeling non-face parts such as hair, neck, mouth cavity and often shoulders/upper garment. These non-face parts cannot be handled by some trivial extension of the face modeling methods since they are much less amenable for registration and often have higher variability and higher complexity than the face part. In principle, the results of face modeling [37] or lips modeling [33] can be stitched into an existing head video. Such design, however, does not allow full control over the head rotation in the resulting video and therefore does not result in a fully-fledged talking head system.

The design of our system borrows a lot from the recent progress in generative modeling of images. Thus, our architecture uses adversarial training [12] and, more specifically, the ideas behind conditional discriminators [24], including projection discriminators [34]. Our meta-learning stage uses the adaptive instance normalization mechanism [14], which was shown to be useful in large-scale conditional generation tasks [6, 36]. We also find an idea of content-style decomposition [15] to be extremely useful for separating the texture from the body pose.

The model-agnostic meta-learner (MAML) [10] uses meta-learning to obtain the initial state of an image classifier, from which it can quickly converge to image classifiers of unseen classes, given few training samples. This high-level idea is also utilized by our method, though our implementation of it is rather different. Several works have further proposed to combine adversarial training with meta-learning. Thus, data-augmentation GAN [2], Meta-GAN [45], adversarial meta-learning [43] use adversarially-trained networks to generate additional examples for classes unseen at the meta-learning stage. While these methods are focused on boosting the few-shot classification performance, our method deals with the training of image generation models using similar adversarial objectives. To summarize, we bring the adversarial fine-tuning into the meta-learning framework. The former is applied after we obtain initial state of the generator and the discriminator networks via the meta-learning stage.

Finally, very related to ours are the two recent works on text-to-speech generation [3, 19]. Their setting (few-shot learning of generative models) and some of the components (standalone embedder network, generator fine-tuning) are are also used in our case. Our work differs in the application domain, the use of adversarial learning, its adaptation to the meta-learning process and implementation details.

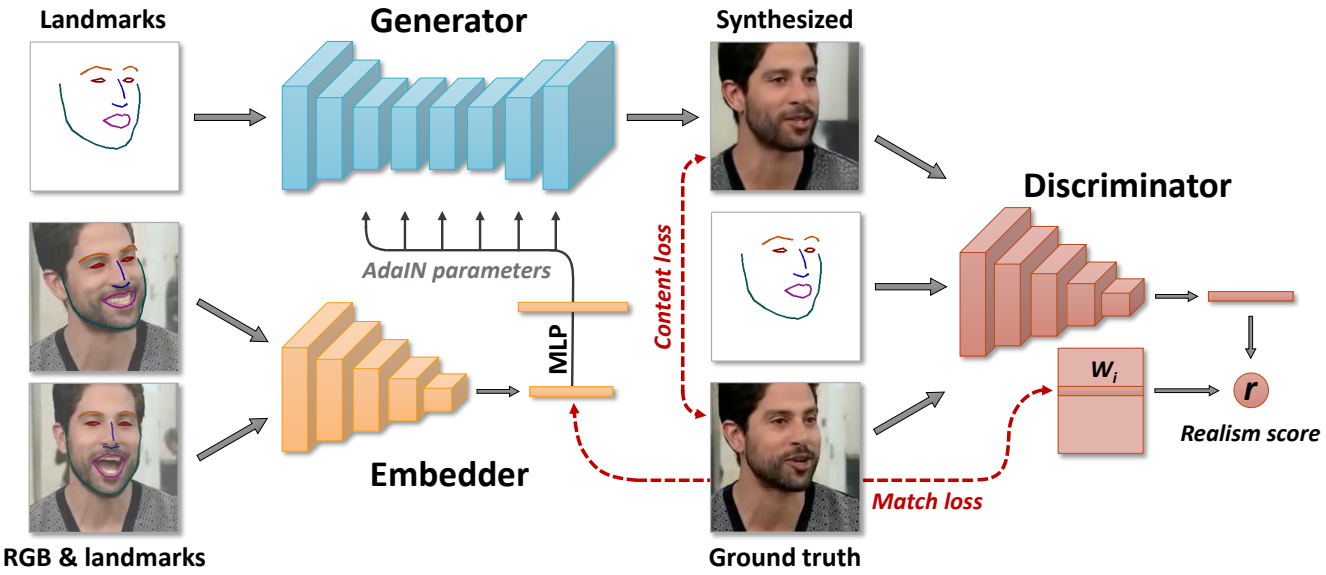


Figure 2: Our meta-learning architecture involves the embedder network that maps head images (with estimated face landmarks) to the embedding vectors, which contain pose-independent information. The generator network maps input face landmarks into output frames through the set of convolutional layers, which are modulated by the embedding vectors via adaptive instance normalization. During meta-learning, we pass sets of frames from the same video through the embedder, average the resulting embeddings and use them to predict adaptive parameters of the generator. Then, we pass the landmarks of a different frame through the generator, comparing the resulting image with the ground truth. Our objective function includes perceptual and adversarial losses, with the latter being implemented via a conditional projection discriminator.

3. Methods

3.1. Architecture and notation

The meta-learning stage of our approach assumes the availability of M video sequences, containing talking heads of different people. We denote with $\mathbf{x}_i$ the i-th video sequence and with $\mathbf{x}_i(t)$ its t-th frame. During the learning process, as well as during test time, we assume the availability of the face landmarks' locations for all frames (we use an off-the-shelf face alignment code [7] to obtain them). The landmarks are rasterized into three-channel images using a predefined set of colors to connect certain landmarks with line segments. We denote with $\mathbf{y}_i(t)$ the resulting *landmark image* computed for $\mathbf{x}_i(t)$.

In the meta-learning stage of our approach, the following three networks are trained (Figure 2):

- The *embedder* $E(\mathbf{x}_i(s), \mathbf{y}_i(s);\, \phi)$ takes a video frame $\mathbf{x}_i(s)$, an associated landmark image $\mathbf{y}_i(s)$ and maps these inputs into an N-dimensional vector $\hat{\mathbf{e}}_i(s)$. Here, ϕ denotes network parameters that are learned in the meta-learning stage. In general, during meta-learning, we aim to learn ϕ such that the vector $\hat{\mathbf{e}}_i(s)$ contains video-specific information (such as the person's identity) that is invariant to the pose and mimics in a particular frame s. We denote embedding vectors computed by the embedder as $\hat{\mathbf{e}}_i$.
- The *generator* $G(\mathbf{y}_i(t), \hat{\mathbf{e}}_i;\, \psi, \mathbf{P})$ takes the landmark image $\mathbf{y}_i(t)$ for the video frame not seen by the embedder, the predicted video embedding $\hat{\mathbf{e}}_i$ and outputs a synthesized video frame $\hat{\mathbf{x}}_i(t)$. The generator is trained to maximize the similarity between its outputs and the ground truth frames. All parameters of the generator are split into two sets: the person-generic parameters ψ, and the person-specific parameters $\hat{\psi}_i$. During meta-learning, only ψ are trained directly, while $\hat{\psi}_i$ are predicted from the embedding vector $\hat{\mathbf{e}}_i$ using a trainable projection matrix $\mathbf{P}$: $\hat{\psi}_i = \mathbf{P}\hat{\mathbf{e}}_i$.
- The *discriminator* $D(\mathbf{x}_i(t), \mathbf{y}_i(t), i;\, \theta, \mathbf{W}, \mathbf{w}_0, b)$ takes a video frame $\mathbf{x}_i(t)$, an associated landmark image $\mathbf{y}_i(t)$ and the index of the training sequence i. Here, $\theta, \mathbf{W}, \mathbf{w}_0$ and b denote the learnable parameters associated with the discriminator. The discriminator contains a ConvNet part $V(\mathbf{x}_i(t), \mathbf{y}_i(t);\, \theta)$ that maps the input frame and the landmark image into an N-dimensional vector. The discriminator predicts a single scalar (realism score) r, that indicates whether the input frame $\mathbf{x}_i(t)$ is a real frame of the i-th video sequence and whether it matches the input pose $\mathbf{y}_i(t)$, based on the output of its ConvNet part and the parameters $\mathbf{W}, \mathbf{w}_0, b$.

3.2. Meta-learning stage

During the meta-learning stage of our approach, the parameters of all three networks are trained in an adversarial

fashion. It is done by simulating episodes of K-shot learning ($K = 8$ in our experiments). In each episode, we randomly draw a training video sequence i and a single frame t from that sequence. In addition to t, we randomly draw additional K frames $s_1, s_2, \ldots, s_K$ from the same sequence. We then compute the estimate $\hat{\mathbf{e}}_i$ of the i-th video embedding by simply averaging the embeddings $\hat{\mathbf{e}}_i(s_k)$ predicted for these additional frames:

$$\hat{\mathbf{e}}_i = \frac{1}{K}\sum_{k=1}^{K} E\left(\mathbf{x}_i(s_k), \mathbf{y}_i(s_k);\, \phi\right) . \tag{1}$$

A reconstruction $\hat{\mathbf{x}}_i(t)$ of the t-th frame, based on the estimated embedding $\hat{\mathbf{e}}_i$, is then computed:

$$\hat{\mathbf{x}}_i(t) = G\left(\mathbf{y}_i(t), \hat{\mathbf{e}}_i;\, \psi, \mathbf{P}\right) . \tag{2}$$

The parameters of the embedder and the generator are then optimized to minimize the following objective that comprises the content term, the adversarial term, and the embedding match term:

$$\begin{aligned}\mathcal{L}(\phi, \psi, \mathbf{P}, \theta, \mathbf{W}, \mathbf{w}_0, b) = \mathcal{L}_{\text{CNT}}(\phi, \psi, \mathbf{P}) + \\ \mathcal{L}_{\text{ADV}}(\phi, \psi, \mathbf{P}, \theta, \mathbf{W}, \mathbf{w}_0, b) + \mathcal{L}_{\text{MCH}}(\phi, \mathbf{W}) .\end{aligned} \tag{3}$$

In (3), the content loss term $\mathcal{L}_{\text{CNT}}$ measures the distance between the ground truth image $\mathbf{x}_i(t)$ and the reconstruction $\hat{\mathbf{x}}_i(t)$ using the perceptual similarity measure [20], corresponding to VGG19 [32] network trained for ILSVRC classification and VGGFace [28] network trained for face verification. The loss is calculated as the weighted sum of L_1 losses between the features of these networks.

The adversarial term in (3) corresponds to the realism score computed by the discriminator, which needs to be maximized, and a feature matching term [40], which essentially is a perceptual similarity measure, computed using discriminator (it helps with the stability of the training):

$$\begin{aligned}\mathcal{L}_{\text{ADV}}(\phi, \psi, \mathbf{P}, \theta, \mathbf{W}, \mathbf{w}_0, b) = \\ -D(\hat{\mathbf{x}}_i(t), \mathbf{y}_i(t), i; \theta, \mathbf{W}, \mathbf{w}_0, b) + \mathcal{L}_{\text{FM}} .\end{aligned} \tag{4}$$

Following the projection discriminator idea [34], the columns of the matrix $\mathbf{W}$ contain the embeddings that correspond to individual videos. The discriminator first maps its inputs to an N-dimensional vector $V(\mathbf{x}_i(t), \mathbf{y}_i(t); \theta)$ and then computes the realism score as:

$$\begin{aligned}D(\hat{\mathbf{x}}_i(t), \mathbf{y}_i(t), i;\, \theta, \mathbf{W}, \mathbf{w}_0, b) = \\ V\left(\hat{\mathbf{x}}_i(t), \mathbf{y}_i(t);\, \theta\right)^T (\mathbf{W}_i + \mathbf{w}_0) + b ,\end{aligned} \tag{5}$$

where $\mathbf{W}_i$ denotes the i-th column of the matrix $\mathbf{W}$. At the same time, $\mathbf{w}_0$ and b do not depend on the video index, so these terms correspond to the general realism of $\hat{\mathbf{x}}_i(t)$ and its compatibility with the landmark image $\mathbf{y}_i(t)$.

Thus, there are two kinds of video embeddings in our system: the ones computed by the embedder, and the ones that correspond to the columns of the matrix $\mathbf{W}$ in the discriminator. The match term $\mathcal{L}_{\text{MCH}}(\phi, \mathbf{W})$ in (3) encourages the similarity of the two types of embeddings by penalizing the L_1-difference between $E\left(\mathbf{x}_i(s_k), \mathbf{y}_i(s_k);\, \phi\right)$ and $\mathbf{W}_i$.

As we update the parameters ϕ of the embedder and the parameters ψ of the generator, we also update the parameters $\theta, \mathbf{W}, \mathbf{w}_0, b$ of the discriminator. The update is driven by the minimization of the following hinge loss, which encourages the increase of the realism score on real images $\mathbf{x}_i(t)$ and its decrease on synthesized images $\hat{\mathbf{x}}_i(t)$:

$$\begin{aligned}\mathcal{L}_{\text{DSC}}(\phi, \psi, \mathbf{P}, \theta, \mathbf{W}, \mathbf{w}_0, b) = \\ \max(0, 1 + D(\hat{\mathbf{x}}_i(t), \mathbf{y}_i(t), i;\, \phi, \psi, \theta, \mathbf{W}, \mathbf{w}_0, b)) + \\ \max(0, 1 - D(\mathbf{x}_i(t), \mathbf{y}_i(t), i;\, \theta, \mathbf{W}, \mathbf{w}_0, b)) .\end{aligned} \tag{6}$$

The objective (6) thus compares the realism of the fake example $\hat{\mathbf{x}}_i(t)$ and the real example $\mathbf{x}_i(t)$ and then updates the discriminator parameters to push these scores below -1 and above $+1$ respectively. The training proceeds by alternating updates of the embedder and the generator that minimize the losses $\mathcal{L}_{\text{CNT}}, \mathcal{L}_{\text{ADV}}$ and $\mathcal{L}_{\text{MCH}}$ with the updates of the discriminator that minimize the loss $\mathcal{L}_{\text{DSC}}$.

3.3. Few-shot learning by fine-tuning

Once the meta-learning has converged, our system can learn to synthesize talking head sequences for a new person, unseen during meta-learning stage. As before, the synthesis is conditioned on the landmark images. The system is learned in a few-shot way, assuming that T training images $\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(T)$ (e.g. T frames of the same video) are given and that $\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(T)$ are the corresponding landmark images. Note that the number of frames T needs not to be equal to K used in the meta-learning stage.

Naturally, we can use the meta-learned embedder to estimate the embedding for the new talking head sequence:

$$\hat{\mathbf{e}}_{\text{NEW}} = \frac{1}{T}\sum_{t=1}^{T} E(\mathbf{x}(t), \mathbf{y}(t);\, \phi) , \tag{7}$$

reusing the parameters ϕ estimated in the meta-learning stage. A straightforward way to generate new frames, corresponding to new landmark images, is then to apply the generator using the estimated embedding $\hat{\mathbf{e}}_{\text{NEW}}$ and the meta-learned parameters ψ, as well as projection matrix $\mathbf{P}$. By doing so, we have found out that the generated images are plausible and realistic, however, there often is a considerable identity gap that is not acceptable for most applications aiming for high personalization degree.

This identity gap can often be bridged via the *fine-tuning stage*. The fine-tuning process can be seen as a simplified version of meta-learning with a single video sequence and a

smaller number of frames. The fine-tuning process involves the following components:

- The generator $G(\mathbf{y}(t), \hat{\mathbf{e}}_{\text{NEW}}; \psi, \mathbf{P})$ is now replaced with $G'(\mathbf{y}(t); \psi, \psi')$. As before, it takes the landmark image $\mathbf{y}(t)$ and outputs the synthesized frame $\hat{\mathbf{x}}(t)$. Importantly, the person-specific generator parameters, which we now denote with ψ', are now directly optimized alongside the person-generic parameters ψ. We still use the computed embeddings $\hat{\mathbf{e}}_{\text{NEW}}$ and the projection matrix $\mathbf{P}$ estimated at the meta-learning stage to initialize ψ', i.e. we start with $\psi' = \mathbf{P}\hat{\mathbf{e}}_{\text{NEW}}$.
- The discriminator $D'(\mathbf{x}(t), \mathbf{y}(t); \theta, \mathbf{w}', b)$, as before, computes the realism score. Parameters θ of its ConvNet part $V(\mathbf{x}(t), \mathbf{y}(t); \theta)$ and bias b are initialized to the result of the meta-learning stage. The initialization of $\mathbf{w}'$ is discussed below.

During fine-tuning, the realism score of the discriminator is obtained in a similar way to the meta-learning stage:

$$D'(\hat{\mathbf{x}}(t), \mathbf{y}(t); \theta, \mathbf{w}', b) = V\left(\hat{\mathbf{x}}(t), \mathbf{y}(t); \theta\right)^T \mathbf{w}' + b\,. \tag{8}$$

As can be seen from the comparison of expressions (5) and (8), the role of the vector $\mathbf{w}'$ in the fine-tuning stage is the same as the role of the vector $\mathbf{W}_i + \mathbf{w}_0$ in the meta-learning stage. For the intiailization, we do not have access to the analog of $\mathbf{W}_i$ for the new personality (since this person is not in the meta-learning dataset). However, the match term $\mathcal{L}_{\text{MCH}}$ in the meta-learning process ensures the similarity between the discriminator video-embeddings and the vectors computed by the embedder. Hence, we can initialize $\mathbf{w}'$ to the sum of $\mathbf{w}_0$ and $\hat{\mathbf{e}}_{\text{NEW}}$.

Once the new learning problem is set up, the loss functions of the fine-tuning stage follow directly from the meta-learning variants. Thus, the generator parameters ψ and ψ' are optimized to minimize the simplified objective:

$$\mathcal{L}'(\psi, \psi', \theta, \mathbf{w}', b) = \mathcal{L}'_{\text{CNT}}(\psi, \psi') + \mathcal{L}'_{\text{ADV}}(\psi, \psi', \theta, \mathbf{w}', b)\,, \tag{9}$$

where $t \in \{1 \dots T\}$ is the number of the training example. The discriminator parameters $\theta, \mathbf{w}_{\text{NEW}}, b$ are optimized by minimizing the same hinge loss as in (6):

$$\mathcal{L}'_{\text{DSC}}(\psi, \psi', \theta, \mathbf{w}', b) = \max(0, 1 + D(\hat{\mathbf{x}}(t), \mathbf{y}(t);\, \psi, \psi', \theta, \mathbf{w}', b)) + \max(0, 1 - D(\mathbf{x}(t), \mathbf{y}(t);\, \theta, \mathbf{w}', b))\,. \tag{10}$$

In most situations, the fine-tuned generator provides a much better fit of the training sequence. The initialization of all parameters via the meta-learning stage is also crucial. As we show in the experiments, such initialization injects a strong realistic talking head prior, which allows our model to extrapolate and predict realistic images for poses with varying head poses and facial expressions.

3.4. Implementation details

We base our generator network $G(\mathbf{y}_i(t), \hat{\mathbf{e}}_i; \psi, \mathbf{P})$ on the image-to-image translation architecture proposed by Johnson et. al. [20], but replace downsampling and upsampling layers with residual blocks similarly to [6] (with batch normalization [16] replaced by instance normalization [38]). The person-specific parameters $\hat{\psi}_i$ serve as the affine coefficients of instance normalization layers, following the adaptive instance normalization technique proposed in [14], though we still use regular (non-adaptive) instance normalization layers in the downsampling blocks that encode landmark images $\mathbf{y}_i(t)$.

For the embedder $E(\mathbf{x}_i(s), \mathbf{y}_i(s); \phi)$ and the convolutional part of the discriminator $V(\mathbf{x}_i(t), \mathbf{y}_i(t); \theta)$, we use similar networks, which consist of residual downsampling blocks (same as the ones used in the generator, but without normalization layers). The discriminator network, compared to the embedder, has an additional residual block at the end, which operates at 4×4 spatial resolution. To obtain the vectorized outputs in both networks, we perform global sum pooling over spatial dimensions followed by ReLU.

We use spectral normalization [35] for all convolutional and fully connected layers in all the networks. We also use self-attention blocks, following [6] and [44]. They are inserted at 32×32 spatial resolution in all downsampling parts of the networks and at 64×64 resolution in the upsampling part of the generator.

For the calculation of $\mathcal{L}_{\text{CNT}}$, we evaluate L_1 loss between activations of `Conv1,6,11,20,29` VGG19 layers and `Conv1,6,11,18,25` VGGFace layers for real and fake images. We sum these losses with the weights equal to $1.5 \cdot 10^{-1}$ for VGG19 and $2.5 \cdot 10^{-2}$ for VGGFace terms. We use Caffe [18] trained versions for both of these networks. For $\mathcal{L}_{\text{FM}}$, we use activations after each residual block of the discriminator network and the weights equal to 10. Finally, for $\mathcal{L}_{\text{MCH}}$ we also set the weight to 10.

We set the minimum number of channels in convolutional layers to 64 and the maximum number of channels as well as the size N of the embedding vectors to 512. In total, the embedder has 15 million parameters, the generator has 38 million parameters. The convolutional part of the discriminator has 20 million parameters. The networks are optimized using Adam [22]. We set the learning rate of the embedder and the generator networks to 5×10^{-5} and to 2×10^{-4} for the discriminator, doing two update steps for the latter per one of the former, following [44].

4. Experiments

Two datasets with talking head videos are used for quantitative and qualitative evaluation: VoxCeleb1 [27] (256p videos at 1 fps) and VoxCeleb2 [8] (224p videos at 25 fps), with the latter having approximately 10 times more videos

Method (T)	FID↓	SSIM↑	CSIM↑	USER↓
VoxCeleb1				
X2Face (1)	45.8	**0.68**	**0.16**	0.82
Pix2pixHD (1)	**42.7**	0.56	0.09	0.82
Ours (1)	43.0	0.67	0.15	**0.62**
X2Face (8)	51.5	**0.73**	**0.17**	0.83
Pix2pixHD (8)	**35.1**	0.64	0.12	0.79
Ours (8)	38.0	0.71	**0.17**	**0.62**
X2Face (32)	56.5	**0.75**	0.18	0.85
Pix2pixHD (32)	**24.0**	0.70	0.16	0.71
Ours (32)	29.5	0.74	**0.19**	**0.61**
VoxCeleb2				
Ours-FF (1)	**46.1**	0.61	**0.42**	**0.43**
Ours-FT (1)	48.5	**0.64**	0.35	0.46
Ours-FF (8)	42.2	0.64	**0.47**	0.40
Ours-FT (8)	**42.2**	**0.68**	0.42	**0.39**
Ours-FF (32)	40.4	0.65	**0.48**	0.38
Ours-FT (32)	**30.6**	**0.72**	0.45	**0.33**

Table 1: Quantitative comparison of methods on different datasets with multiple few-shot learning settings. Please refer to the text for more details and discussion.

than the former. VoxCeleb1 is used for comparison with baselines and ablation studies, while by using VoxCeleb2 we show the full potential of our approach.

Metrics. For the quantitative comparisons, we fine-tune all models on few-shot learning sets of size T for a person not seen during meta-learning (or pretraining) stage. After the few-shot learning, the evaluation is performed on the hold-out part of the same sequence (so-called *self-reenactment* scenario). For the evaluation, we uniformly sampled 50 videos from VoxCeleb test sets and 32 hold-out frames for each of these videos (the fine-tuning and the hold-out parts do not overlap).

We use multiple comparison metrics to evaluate photo-realism and identity preservation of generated images. Namely, we use Frechet-inception distance (FID) [13], mostly measuring perceptual realism, structured similarity (SSIM) [41], measuring low-level similarity to the ground truth images, and cosine similarity (CSIM) between embedding vectors of the state-of-the-art face recognition network [9] for measuring identity mismatch (note that this network has quite different architecture from VGGFace used within content loss calculation during training).

We also perform a user study in order to evaluate perceptual similarity and realism of the results as seen by the human respondents. We show people the triplets of images of the same person taken from three different video sequences. Two of these images are real and one is fake, produced by one of the methods, which are being compared. We ask the user to find the fake image given that all of these images are of the same person. This evaluates both photo-realism and identity preservation because the user can infer the identity from the two real images (and spot an identity mismatch even if the generated image is perfectly realistic). We use the user accuracy (success rate) as our metric. The lower bound here is the accuracy of one third (when users cannot spot fakes based on non-realism or identity mismatch and have to guess randomly). Generally, we believe that this user-driven metric (USER) provides a much better idea of the quality of the methods compared to FID, SSIM, or CSIM.

Methods. On the VoxCeleb1 dataset we compare our model against two other systems: X2Face [42] and Pix2pixHD [40]. For X2Face, we have used the model, as well as pretrained weights, provided by the authors (in the original paper it was also trained and evaluated on the VoxCeleb1 dataset). For Pix2pixHD, we pretrained the model from scratch on the whole dataset for the same amount of iterations as our system without any changes to thetraining pipeline proposed by the authors. We picked X2Face as a strong baseline for warping-based methods and Pix2pixHD for direct synthesis methods.

In our comparison, we evaluate the models in several scenarios by varying the number of frames T used in few-shot learning. X2Face, as a feed-forward method, is simply initialized via the training frames, while Pix2pixHD and our model are being additionally fine-tuned for 40 epochs on the few-shot set. Notably, in the comparison, X2Face uses dense correspondence field, computed on the ground truth image, to synthesize the generated one, while our method and Pix2pixHD use very sparse landmark information, which arguably gives X2Face an unfair advantage.

Comparison results. We perform comparison with baselines in three different setups, with 1, 8 and 32 frames in the fine-tuning set. Test set, as mentioned before, consists of 32 hold-out frames for each of the 50 test video sequences. Moreover, for each test frame we sample two frames at random from the other video sequences with the same person. These frames are used in triplets alongside with fake frames for user-study.

As we can see in Table 1-Top, baselines consistently outperform our method on the two of our similarity metrics. We argue that this is intrinsic to the methods themselves: X2Face uses L_2 loss during optimization [42], which leads to a good SSIM score. On the other hand, Pix2pixHD maximizes only perceptual metric, without identity preservation loss, leading to minimization of FID, but has bigger identity mismatch, as seen from the CSIM column. Moreover, these metrics do not correlate well with human perception, since both of these methods produce uncanny valley artifacts, as can be seen from qualitative comparison Figure 3 and the

Figure 3: Comparison on the VoxCeleb1 dataset. For each of the compared methods, we perform one- and few-shot learning on a video of a person not seen during meta-learning or pretraining. We set the number of training frames equal to **T** (the leftmost column). One of the training frames is shown in the **source** column. Next columns show **ground truth** image, taken from the test part of the video sequence, and the generated results of the compared methods.

user study results. Cosine similarity, on the other hand, better correlates with visual quality, but still favours blurry, less realistic images, and that can also be seen by comparing Table 1-Top with the results presented in Figure 3.

While the comparison in terms of the objective metrics is inconclusive, the user study (that included 4800 triplets, each shown to 5 users) clearly reveals the much higher realism and personalization degree achieved by our method.

We have also carried out the ablation study of our system and the comparison of the few-shot learning timings. Both are provided in the Supplementary material.

Large-scale results. We then scale up the available data and train our method on a larger VoxCeleb2 dataset. Here, we train two variants of our method. FF (feed-forward) variant is trained for 150 epochs without the embedding matching loss $\mathcal{L}_{\text{MCH}}$ and, therefore, we only use it without fine-tuning (by simply predicting adaptive parameters ψ' via the projection of the embedding $\hat{\mathbf{e}}_{\text{NEW}}$). The FT variant is trained for half as much (75 epochs) but with $\mathcal{L}_{\text{MCH}}$, which allows fine-tuning. We run the evaluation for both of these models since they allow to trade off few-shot learning speed versus the results quality. Both of them achieve considerably higher scores, compared to smaller-scale models trained on VoxCeleb1. Notably, the FT model reaches the lower bound of 0.33 for the user study accuracy in $T = 32$ setting, which is a perfect score. We present results for both of these models in Figure 4 and more results (including results, where animation is driven by landmarks from a different video of the same person) are given in the supplementary material and in Figure 1.

Generally, judging by the results of comparisons (Table 1-Bottom) and the visual assessment, the FF model performs better for low-shot learning (e.g. one-shot), while the FT model achieves higher quality for bigger T via adversarial fine-tuning.

Puppeteering results. Finally, we show the results for the puppeteering of photographs and paintings. For that, we evaluate the model, trained in one-shot setting, on poses from test videos of the VoxCeleb2 dataset. We rank these videos using CSIM metric, calculated between the original image and the generated one. This allows us to find persons with similar geometry of the landmarks and use them for the puppeteering. The results can be seen in Figure 5 as well as in Figure 1.

Figure 4: Results for our best models on the VoxCeleb2 dataset. The number of training frames is, again, equal to **T** (the leftmost column) and the example training frame in shown in the **source** column. Next columns show **ground truth** image and the results for **Ours-FF** feed-forward model, **Ours-FT** model **before** and **after fine-tuning**. While the feed-forward variant allows fast (real-time) few-shot learning of new avatars, fine-tuning ultimately provides better realism and fidelity.

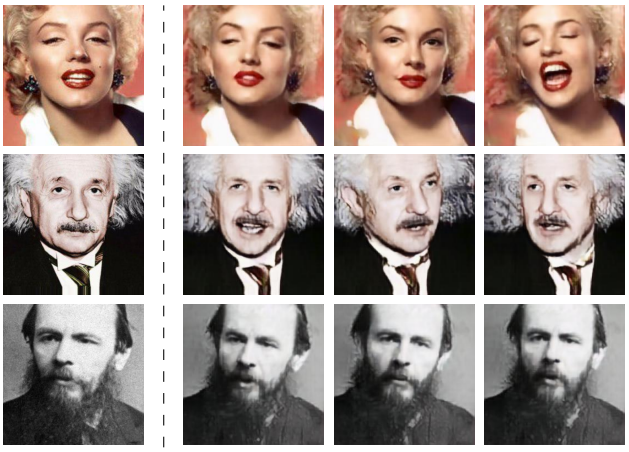


Figure 5: Bringing still photographs to life. We show the puppeteering results for one-shot models learned from photographs in the **source** column. Driving poses were taken from the VoxCeleb2 dataset. Digital zoom recommended.

5. Conclusion

We have presented a framework for meta-learning of adversarial generative models, which is able to train highly-realistic virtual talking heads in the form of deep generator networks. Crucially, only a handful of photographs (as little as one) is needed to create a new model, whereas the model trained on 32 images achieves perfect realism and personalization score in our user study (for 224p static images).

Currently, the key limitations of our method are the mimics representation (in particular, the current set of landmarks does not represent the gaze in any way) and the lack of landmark adaptation. Using landmarks from a different person leads to a noticeable personality mismatch. So, if one wants to create "fake" puppeteering videos without such mismatch, some landmark adaptation is needed. We note, however, that many applications do not require puppeteering a different person and instead only need the ability to drive one's own talking head. For such scenario, our approach already provides a high-realism solution.

References

[1] Oleg Alexander, Mike Rogers, William Lambeth, Jen-Yuan Chiang, Wan-Chun Ma, Chuan-Chang Wang, and Paul Debevec. The Digital Emily project: Achieving a photorealistic digital actor. *IEEE Computer Graphics and Applications*, 30(4):20–31, 2010. 2

[2] Antreas Antoniou, Amos J. Storkey, and Harrison Edwards. Augmenting image classifiers using data augmentation generative adversarial networks. In *Artificial Neural Networks and Machine Learning - ICANN*, pages 594–603, 2018. 2

[3] Sercan Arik, Jitong Chen, Kainan Peng, Wei Ping, and Yanqi Zhou. Neural voice cloning with a few samples. In *Proc. NIPS*, pages 10040–10050, 2018. 2

[4] Hadar Averbuch-Elor, Daniel Cohen-Or, Johannes Kopf, and Michael F Cohen. Bringing portraits to life. *ACM Transactions on Graphics (TOG)*, 36(6):196, 2017. 1, 14

[5] Volker Blanz, Thomas Vetter, et al. A morphable model for the synthesis of 3d faces. In *Proc. SIGGRAPH*, volume 99, pages 187–194, 1999. 2

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019. 2, 5, 12

[7] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230, 000 3d facial landmarks). In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1021–1030, 2017. 3

[8] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. Voxceleb2: Deep speaker recognition. In *INTERSPEECH*, 2018. 6

[9] Jiankang Deng, Jia Guo, Xue Niannan, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *CVPR*, 2019. 6

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. ICML*, pages 1126–1135, 2017. 2

[11] Yaroslav Ganin, Daniil Kononenko, Diana Sungatullina, and Victor Lempitsky. Deepwarp: Photorealistic image resynthesis for gaze manipulation. In *European Conference on Computer Vision*, pages 311–326. Springer, 2016. 1

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2

[13] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6626–6637. Curran Associates, Inc., 2017. 6

[14] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proc. ICCV*, 2017. 2, 5

[15] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *ECCV*, 2018. 2

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015. 5

[17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proc. CVPR*, pages 5967–5976, 2017. 2

[18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 5

[19] Ye Jia, Yu Zhang, Ron Weiss, Quan Wang, Jonathan Shen, Fei Ren, Patrick Nguyen, Ruoming Pang, Ignacio Lopez Moreno, Yonghui Wu, et al. Transfer learning from speaker verification to multispeaker text-to-speech synthesis. In *Proc. NIPS*, pages 4485–4495, 2018. 2

[20] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV*, pages 694–711, 2016. 4, 5

[21] Hyeongwoo Kim, Pablo Garrido, Ayush Tewari, Weipeng Xu, Justus Thies, Matthias Nießner, Patrick Pérez, Christian Richardt, Michael Zollhöfer, and Christian Theobalt. Deep video portraits. *arXiv preprint arXiv:1805.11714*, 2018. 2

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 5

[23] Stephen Lombardi, Jason Saragih, Tomas Simon, and Yaser Sheikh. Deep appearance models for face rendering. *ACM Transactions on Graphics (TOG)*, 37(4):68, 2018. 2

[24] Simon Osindero Mehdi Mirza. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014. 2

[25] Masahiro Mori. The uncanny valley. *Energy*, 7(4):33–35, 1970. 1

[26] Koki Nagano, Jaewoo Seo, Jun Xing, Lingyu Wei, Zimo Li, Shunsuke Saito, Aviral Agarwal, Jens Fursund, Hao Li, Richard Roberts, et al. paGAN: real-time avatars using dynamic textures. In *SIGGRAPH Asia 2018 Technical Papers*, page 258. ACM, 2018. 2

[27] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. Voxceleb: a large-scale speaker identification dataset. In *INTERSPEECH*, 2017. 5

[28] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep face recognition. In *Proc. BMVC*, 2015. 4

[29] Albert Pumarola, Antonio Agudo, Aleix M Martinez, Alberto Sanfeliu, and Francesc Moreno-Noguer. Ganimation: Anatomically-aware facial animation from a single image. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 818–833, 2018. 14

[30] Steven M Seitz and Charles R Dyer. View morphing. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 21–30. ACM, 1996. 1

[31] Zhixin Shu, Mihir Sahasrabudhe, Riza Alp Guler, Dimitris Samaras, Nikos Paragios, and Iasonas Kokkinos. Deforming autoencoders: Unsupervised disentangling of shape and appearance. In *The European Conference on Computer Vision (ECCV)*, September 2018. 1

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015. 4

[33] Supasorn Suwajanakorn, Steven M Seitz, and Ira Kemelmacher-Shlizerman. Synthesizing Obama: learning lip sync from audio. *ACM Transactions on Graphics (TOG)*, 36(4):95, 2017. 2

[34] Masanori Koyama Takeru Miyato. cgans with projection discriminator. *arXiv:1802.05637*, 2018. 2, 4

[35] Masanori Koyama Yuichi Yoshida Takeru Miyato, Toshiki Kataoka. Spectral normalization for generative adversarial networks. *arXiv:1802.05957*, 2018. 5

[36] Timo Aila Tero Karras, Samuli Laine. A style-based generator architecture for generative adversarial networks. *arXiv:1812.04948*, 2018. 2

[37] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of RGB videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2387–2395, 2016. 2, 12

[38] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. 5, 12

[39] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018. 2

[40] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 4, 6

[41] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *Trans. Img. Proc.*, 13(4):600–612, Apr. 2004. 6

[42] Olivia Wiles, A. Sophia Koepke, and Andrew Zisserman. X2face: A network for controlling face generation using images, audio, and pose codes. In *The European Conference on Computer Vision (ECCV)*, September 2018. 1, 2, 6

[43] Chengxiang Yin, Jian Tang, Zhiyuan Xu, and Yanzhi Wang. Adversarial meta-learning. *CoRR*, abs/1806.03316, 2018. 2

[44] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. 5, 12

[45] Ruixiang Zhang, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. Metagan: An adversarial approach to few-shot learning. In *NeurIPS*, pages 2371–2380, 2018. 2

A. Supplementary material

In the supplementary material, we provide additional qualitative results as well as an ablation study and a time comparison between our method and the baselines for both inference and training.

A.1. Time comparison results.

In Table 2, we provide a comparison of timings for the three methods. Additionally, we included the feed-forward variant of our method in the comparison, which was trained only for the VoxCeleb2 dataset. The comparison was carried out on a single NVIDIA P40 GPU. For Pix2pixHD and our method, few-shot learning was done via fine-tuning for 40 epochs on the training set of size T. For T larger than 1, we trained the models on batches of 8 images. Each measurement was averaged over 100 iterations.

We see that, given enough training data, our method in feed-forward variant can outpace all other methods by a large margin in terms of few-shot training time, while keeping personalization fidelity and realism of the outputs on quite a high level (as can be seen in Figure 4). But in order to achieve the best results in terms of quality, fine-tuning has to be performed, which takes approximately four and a half minutes on the P40 GPU for 32 training images. The number of epochs and, hence, the fine-tuning speed can be optimized further on a case by case basis or via the introduction of a training scheduler, which we did not perform.

On the other hand, inference speed for our method is comparable or slower than other methods, which is caused by a large number of parameters we need to encode the prior knowledge about talking heads. Though, this figure can be drastically improved via the usage of more modern GPUs (on an NVIDIA 2080 Ti, the inference time can be decreased down to 13ms per frame, which is enough for most real-time applications).

Method (T)	Time, s
Few-shot learning	
X2Face (1)	0.236
Pix2pixHD (1)	33.92
Ours (1)	43.84
Ours-FF (1)	**0.061**
X2Face (8)	1.176
Pix2pixHD (8)	52.40
Ours (8)	85.48
Ours-FF (8)	**0.138**
X2Face (32)	7.542
Pix2pixHD (32)	122.6
Ours (32)	258.0
Ours-FF (32)	**0.221**
Inference	
X2Face	0.110
Pix2pixHD	**0.034**
Ours	0.139

Table 2: Quantitative comparison of few-shot learning and inference timings for the three models.

A.2. Ablation study

In this section, we evaluate the contributions related to the losses we use in the training of our model, as well as motivate the training procedure. We have already shown in Figure 4 the effect that the fine-tuning has on the quality of the results, so we do not evaluate it here. Instead, we focus on the details of fine-tuning.

The first question we asked was about the importance of person-specific parameters initialization via the embedder. We tried different types of random initialization for both the embedding vector $\hat{\mathbf{e}}_{\text{NEW}}$ and the adaptive parameters $\hat{\psi}$ of the generator, but these experiments did not yield any plausible images after the fine-tuning. Hence we realized that the person-specific initialization of the generator provided by the embedder is important for convergence of the fine-tuning problem.

Then, we evaluated the contribution of the person-specific initialization of the discriminator. We remove $\mathcal{L}_{\text{MCH}}$ term from the objective and perform meta-learning. The use of multiple training frames in few-shot learning problems, like in our final method, leads to optimization instabilities, so we used a one-shot meta-learning configuration, which turned out to be stable. After meta-learning, we randomly initialize the person-specific vector $\mathbf{W}_i$ of the discriminator. The results can be seen in Figure 7. We notice that the results for random initialization are plausible but introduce a noticeable gap in terms of realism and personalization fidelity. We, therefore, came to the conclusion that person-specific initialization of the discriminator also contributes to the quality of the results, albeit in a lesser way than the initialization of the generator does.

Finally, we evaluate the contribution of adversarial term $\mathcal{L}'_{\text{ADV}}$ during the fine-tuning. We, therefore, remove it from the fine-tuning objective and compare the results to our best model (see Figure 7). While the difference between these variants is quite subtle, we note that adversarial fine-tuning leads to crisper images that better match ground truth both in terms of pose and image details. The close-up images in Figure 8 were chosen in order to highlight these differences.

A.3. Additional qualitative results

More comparisons with other methods are available in Figure 9, Figure 10, Figure 6. More puppeteering results for one-shot learned portraits and photographs are presented in Figure 11. We also show the results for talking heads learned from selfies in Figure 13. Additional comparisons between the methods are provided in the rest of the figures.

Figure 6: Comparison with Thies et al.[37]. We used 32 frames for the fine-tuning, while 1100 frames were used to train the Face2Face model. Note that the output resolution of our model is constrained by the training dataset. Also, our model is able to synthesize a naturally looking frame from different viewpoints for a fixed pose (given 3D face landmarks), which is a limitation of the Face2Face system.

A.4. Training and architecture details

As stated in the paper, we used the architecture similar to the one in [6]. The convolutional parts of the embedder and the discriminator are the same networks with 6 residual downsampling blocks, each performing downsampling by a factor of 2. The inputs of these convolutional networks are RGB images concatenated with the landmark images, in total there are 6 input channels. The initial number of channels is 64, increased by a factor of two in each block, up to a maximum of 512. The blocks are pre-activated residual blocks with no normalization, as described in the paper [6]. The first block is a regular residual block with activation function not being applied in the end. Each skip connection has a linear layer inside if the spatial resolution is being changed. Self-attention [44] blocks are inserted after three downsampling blocks. Downsampling is performed via average pooling. Then, after applying ReLU activation function to the output tensor, we perform sum-pooling over spatial dimensions.

For the embedder, the resulting vectorized embeddings for each training image are stored (in order to apply $\mathcal{L}_{\text{MCH}}$ element-wise), and the averaged embeddings are fed into the generator. For the discriminator, the resulting vector is used to calculate the realism score.

The generator consists of three parts: 4 residual downsampling blocks (with self-attention inserted before the last block), 4 blocks operating at bottleneck resolution and 4 upsampling blocks (self-attention is inserted after 2 upsampling blocks). Upsampling is performed in the end of the block, following [6]. The number of channels in bottleneck layers is 512. Downsampling blocks are normalized via instance normalization [38], while bottleneck and upsampling blocks are normalized via adaptive instance normalization. A single linear layer is used to map an embedding vector to all adaptive parameters. After the last upsampling block, we insert a final adaptive normalization layer, followed by a ReLU and a convolution. The output is then mapped into $[-1, 1]$ via Tanh.

The training was carried out on 8 NVIDIA P40 GPUs, with batch size 48 via simultaneous gradient descend, with 2 updates of the discriminator per 1 of the generator. In our experiments, we used PyTorch distributed module and have performed reduction of the gradients across the GPUs only for the generator and the embedder.

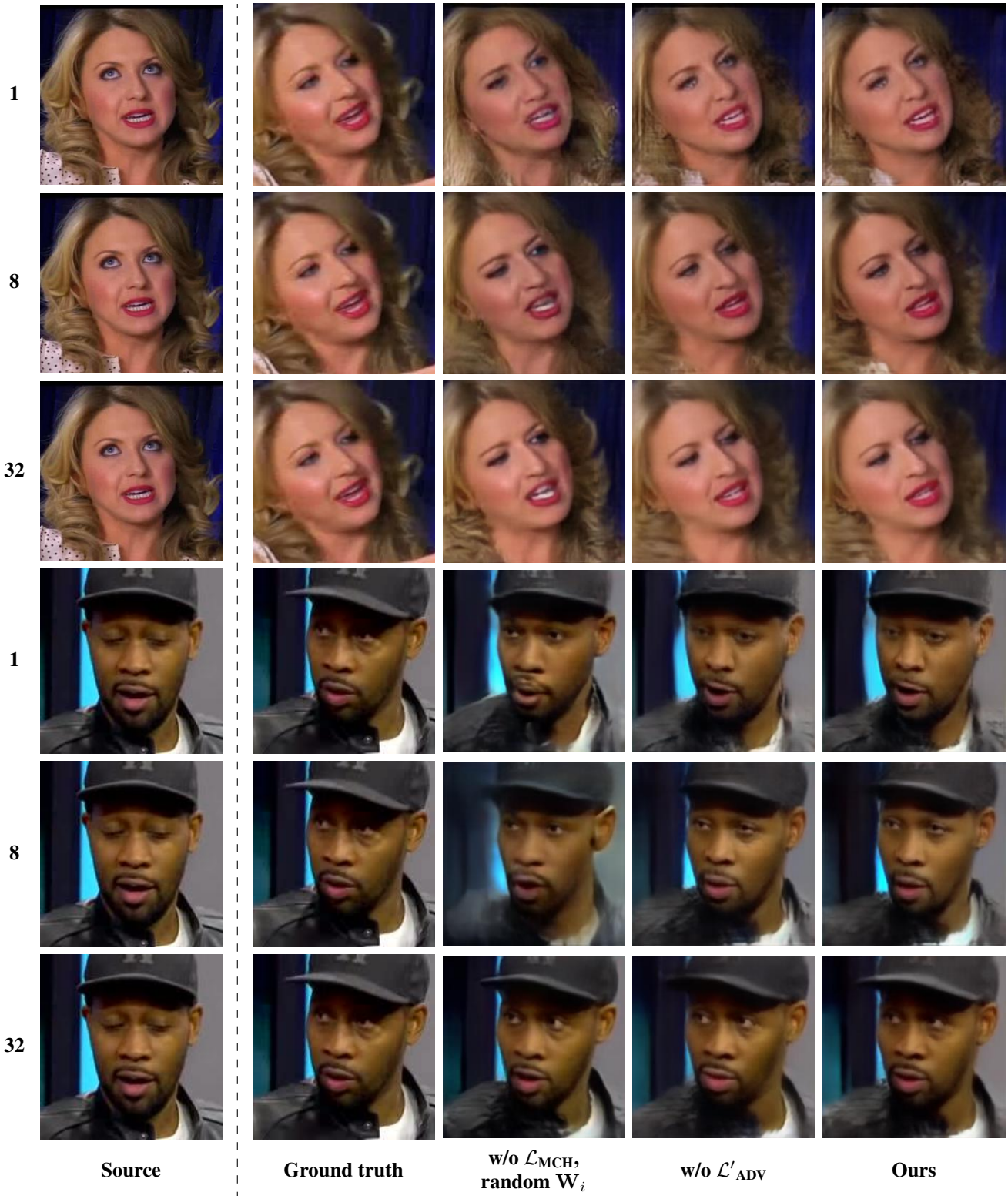


Figure 7: Ablation study of our contributions. The number of training frames is, again, equal to **T** (the leftmost column), the example training frame in shown in the **source** column and the next column shows **ground truth** image. Then, we remove $\mathcal{L}_{\text{MCH}}$ from the meta-learning objective and initialize the embedding vector of the discriminator **randomly** (third column) and evaluate the contribution of adversarial fine-tuning compared to the regular fine-tuning **with no** $\mathcal{L}'_{\text{ADV}}$ in the objective (fifth column). The last column represents results from our final model.

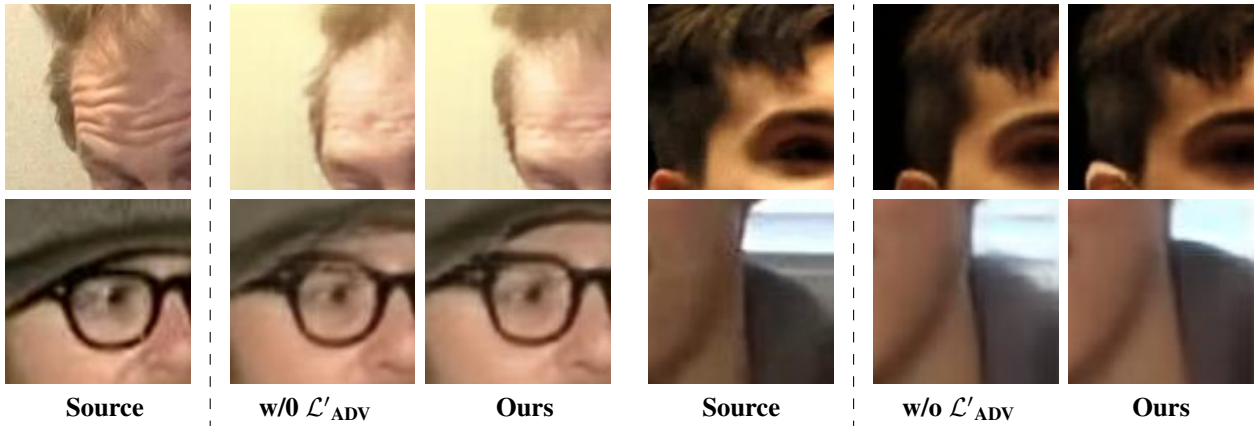


Figure 8: More close-up examples of the ablation study examples for the comparison against the model **w/o** $\mathcal{L}'_{\text{ADV}}$. We used 8 training frames. Notice the geometry gap (top row) and additional artifacts (bottom row) introduced by the removal of $\mathcal{L}'_{\text{ADV}}$ during fine-tuning.

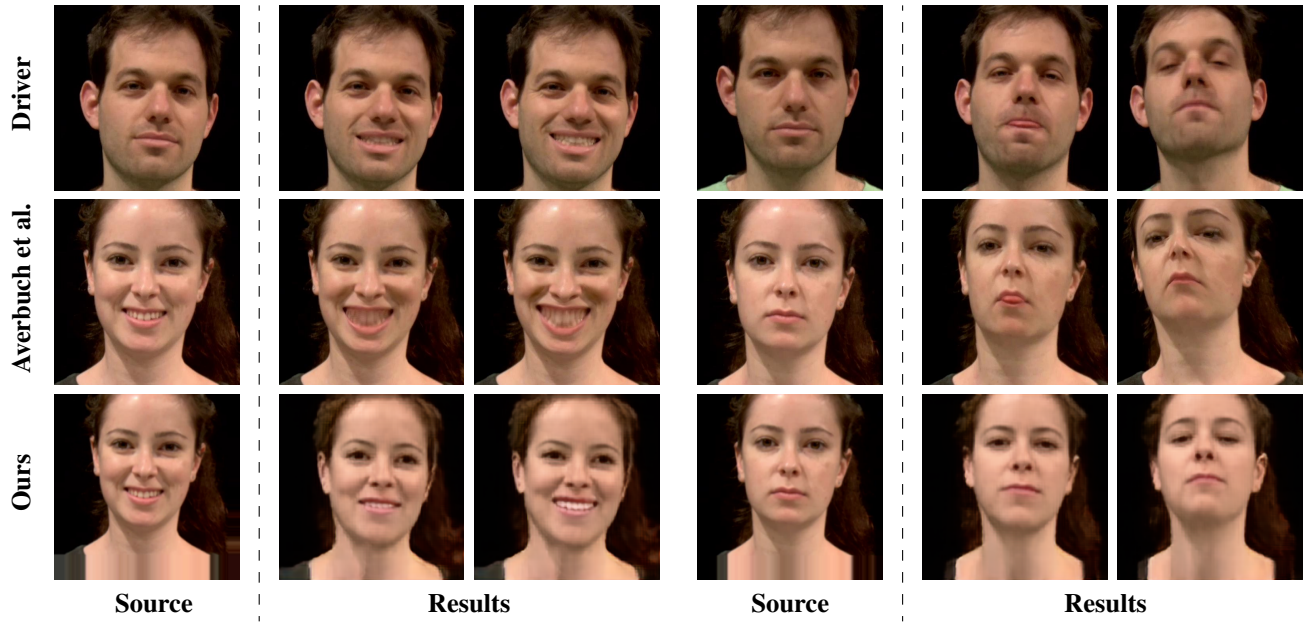


Figure 9: Comparison with Averbuch-Elor et al. [4] on the failure cases mentioned in the paper. Notice that our model better transfers the input pose and also is unaffected by the pose of the original frame, which lifts the "neutral face" constraint on the source image assumed in [4].

Figure 10: Comparison with Pumarola et al. [29] (second column) and our method (right four columns). We perform the driving in the same way as we animate still images in the paper. Note that in the VoxCeleb datasets face cropping have been performed differently, so we had to manually crop our results, effectively decreasing the resolution.

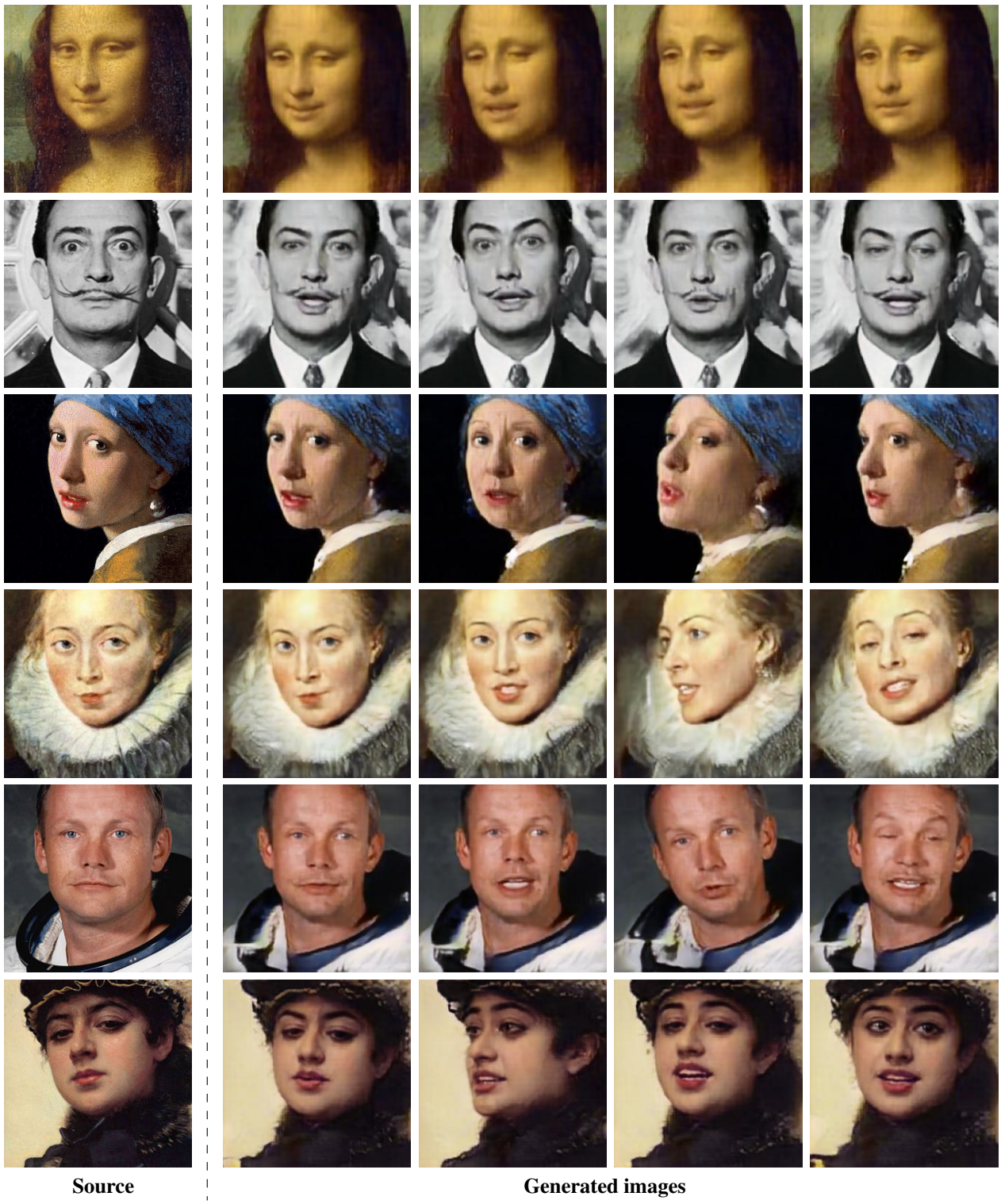


Figure 11: More puppeteering results for talking head models trained in one-shot setting. The image used for one-shot training problem is in the **source** column. The next columns show **generated images**, which were conditioned on the video sequence of a different person.

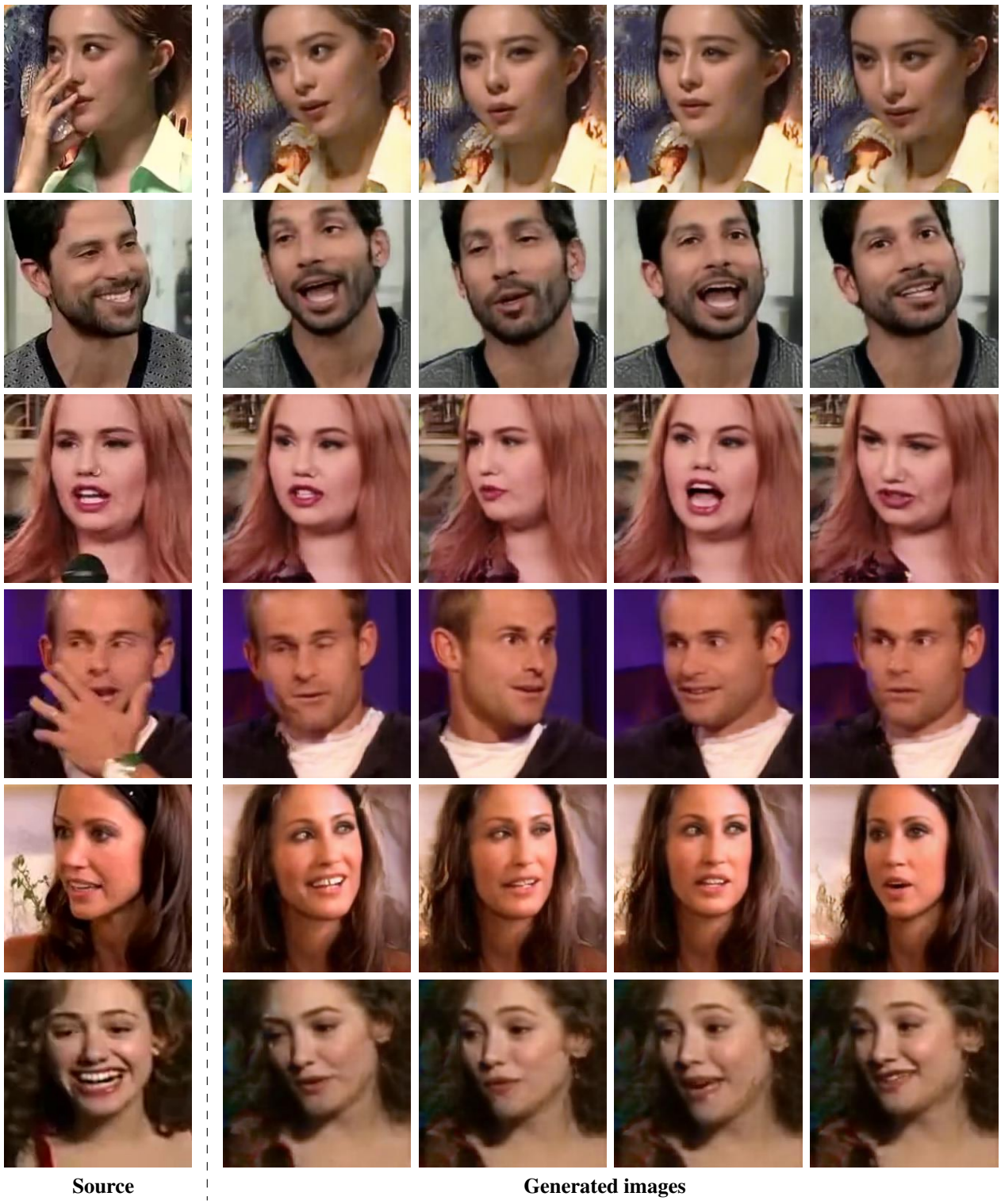


Figure 12: Results for talking head models trained in eight-shot setting. Example training frame is in the **source** column. The next columns show **generated images**, which were conditioned on the pose tracks taken from a different video sequence with the same person.

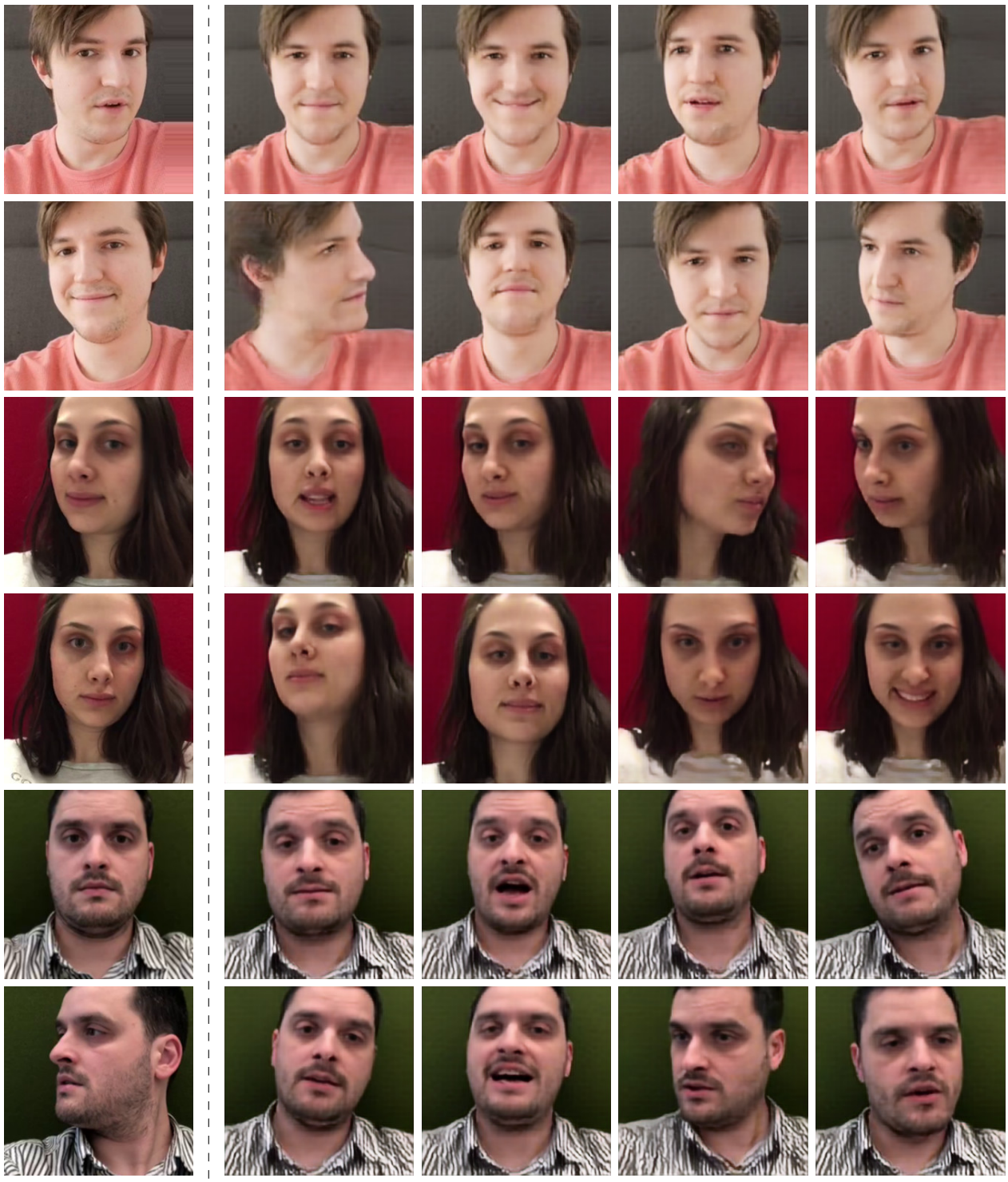


Figure 13: Results for talking head models trained in 16-shot setting on selfie photographs with driving landmarks taken from the different video of the same person. Example training frames are shown in the **source** column. The next columns show **generated images**, which were conditioned on the different video sequence of the same person.

Figure 14: First of the extended qualitative comparisons on the VoxCeleb1 dataset. Here, the comparison is carried out with respect to both the qualitative performance of each method and the way the amount of the training data affects the results. The notation for the columns follows Figure 3 in the main paper.

Figure 15: Second extended qualitative comparison on the VoxCeleb1 dataset. Here, we compare qualitative performance of the three methods on different people not seen during meta-learning or pretraining. We used eight shot learning problem formulation. The notation for the columns follows Figure 3 in the main paper.

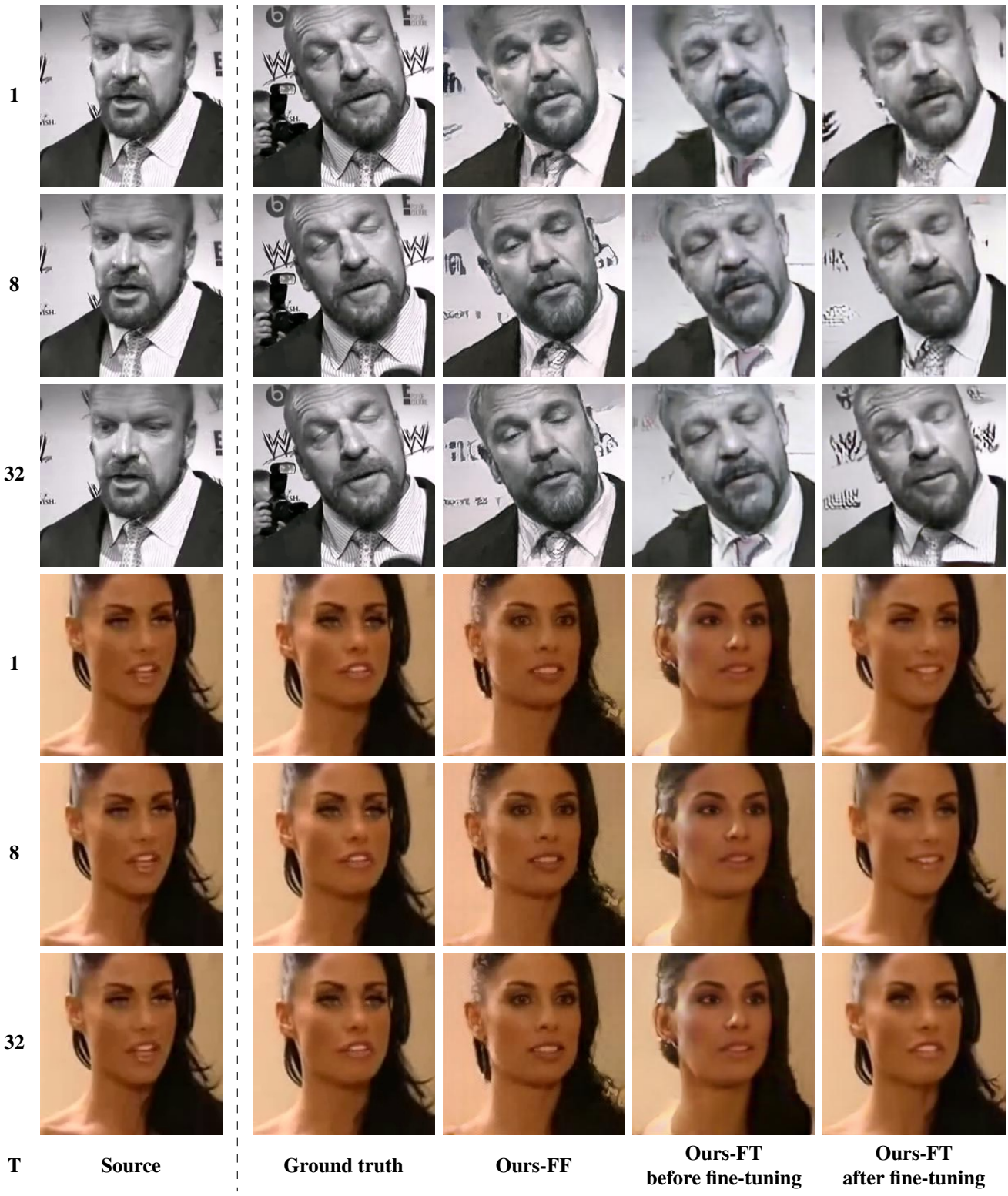


Figure 16: First of the extended qualitative comparisons on the VoxCeleb2 dataset. Here, the comparison is carried out with respect to both the qualitative performance of each variant of our method and the way the amount of the training data affects the results. The notation for the columns follows Figure 4 in the main paper.

Figure 17: Second extended qualitative comparison on the VoxCeleb2 dataset. Here, we compare qualitative performance of the three variants of our method on different people not seen during meta-learning or pretraining. We used eight shot learning problem formulation. The notation for the columns follows Figure 4 in the main paper.

Interpreting the Latent Space of GANs for Semantic Face Editing

Yujun Shen[1], Jinjin Gu[2], Xiaoou Tang[1], Bolei Zhou[1]
[1]The Chinese University of Hong Kong [2]The Chinese University of Hong Kong, Shenzhen
{sy116, xtang, bzhou}@ie.cuhk.edu.hk, jinjingu@link.cuhk.edu.cn

Figure 1: Manipulating various facial attributes through varying the latent codes of a well-trained GAN model. The first column shows the original synthesis from PGGAN [21], while each of the other columns shows the results of manipulating a specific attribute.

Abstract

Despite the recent advance of Generative Adversarial Networks (GANs) in high-fidelity image synthesis, there lacks enough understanding of how GANs are able to map a latent code sampled from a random distribution to a photo-realistic image. Previous work assumes the latent space learned by GANs follows a distributed representation but observes the vector arithmetic phenomenon. In this work, we propose a novel framework, called InterFaceGAN, for semantic face editing by interpreting the latent semantics learned by GANs. In this framework, we conduct a detailed study on how different semantics are encoded in the latent space of GANs for face synthesis. We find that the latent code of well-trained generative models actually learns a disentangled representation after linear transformations. We explore the disentanglement between various semantics and manage to decouple some entangled semantics with subspace projection, leading to more precise control of facial attributes. Besides manipulating gender, age, expression, and the presence of eyeglasses, we can even vary the face pose as well as fix the artifacts accidentally generated by GAN models. The proposed method is further applied to achieve real image manipulation when combined with GAN inversion methods or some encoder-involved models. Extensive results suggest that learning to synthesize faces spontaneously brings a disentangled and controllable facial attribute representation.[1]

1. Introduction

Generative Adversarial Networks (GANs) [15] have significantly advanced image synthesis in recent years. The rationale behind GANs is to learn the mapping from a latent distribution to the real data through adversarial training. After learning such a non-linear mapping, GAN is capable of producing photo-realistic images from randomly sampled latent codes. However, it is uncertain how semantics originate and are organized in the latent space. Taking face synthesis as an example, when sampling a latent code to produce an image, how the code is able to determine various semantic attributes (*e.g.*, gender and age) of the output face, and how these attributes are entangled with each other?

[1]Code and models are available at this link.

Existing work typically focuses on improving the synthesis quality of GANs [40, 28, 21, 8, 22], however, few efforts have been made on studying what a GAN actually learns with respect to the latent space. Radford *et al.* [31] first observes the vector arithmetic property in the latent space. A recent work [4] further shows that some units from intermediate layers of the GAN generator are specialized to synthesize certain visual concepts, such as sofa and TV for living room generation. Even so, there lacks enough understanding of how GAN connects the latent space and the image semantic space, as well as how the latent code can be used for image editing.

In this paper, we propose a framework *InterFaceGAN*, short for *Inter*preting *Face GAN*s, to identify the semantics encoded in the latent space of well-trained face synthesis models and then utilize them for semantic face editing. Beyond the vector arithmetic property, this framework provides both theoretical analysis and experimental results to verify that *linear* subspaces align with different *true-or-false* semantics emerging in the latent space. We further study the disentanglement between different semantics and show that we can decouple some entangled attributes (*e.g.*, old people are more likely to wear eyeglasses then young people) through the linear subspace projection. These disentangled semantics enable precise control of facial attributes with any given GAN model *without retraining*.

Our contributions are summarized as follows:

- We propose InterFaceGAN to explore how a single or multiple semantics are encoded in the latent space of GANs, such as PGGAN [21] and StyleGAN [22], and observe that GANs spontaneously learn various latent subspaces corresponding to specific attributes. These attribute representations become disentangled after some linear transformations.
- We show that InterFaceGAN enables semantic face editing with any *fixed* pre-trained GAN model. Some results are shown in Fig.1. Besides gender, age, expression, and the presence of eyeglasses, we can noticeably also vary the face pose or correct some artifacts produced by GANs.
- We extend InterFaceGAN to real image editing with GAN inversion methods and encoder-involved models. We successfully manipulate the attributes of real faces by simply varying the latent code, even with GANs that are not specifically designed for the editing task.

1.1. Related Work

Generative Adversarial Networks. GAN [15] has brought wide attention in recent years due to its great potential in producing photo-realistic images [1, 17, 6, 40, 28, 21, 8, 22]. It typically takes a sampled latent code as the input and outputs an image synthesis. To make GANs applicable for real image processing, existing methods proposed to reverse the mapping from the latent space to the image space [30, 42, 27, 5, 16] or learn an additional encoder associated with the GAN training [13, 12, 41]. Despite this tremendous success, little work has been done on understanding how GANs learn to connect the input latent space with the semantics in the real visual world.

Study on Latent Space of GANs. Latent space of GANs is generally treated as Riemannian manifold [9, 2, 23]. Prior work focused on exploring how to make the output image vary smoothly from one synthesis to another through interpolation in the latent space, regardless of whether the image is semantically controllable [24, 32]. GLO [7] optimized the generator and latent code simultaneously to learn a better latent space. However, the study on how a well-trained GAN is able to encode different semantics inside the latent space is still missing. Some work has observed the vector arithmetic property [31, 36]. Beyond that, this work provides a detailed analysis of the semantics encoded in the latent space from both the property of a single semantic and the disentanglement of multiple semantics. Some concurrent work also explores the latent semantics learned by GANs. Jahanian *et al.* [20] studies the steerability of GANs concerning camera motion and image color tone. Goetschalckx *et al.* [14] improves the memorability of the output image. Yang *et al.* [38] explores the hierarchical semantics in the deep generative representations for scene synthesis. Unlike them, we focus on facial attributes emerging in GANs for face synthesis and extend our method to real image manipulation.

Semantic Face Editing with GANs. Semantic face editing aims at manipulating facial attributes of a given image. Compared to unconditional GANs which can generate image arbitrarily, semantic editing expects the model to only change the target attribute but maintain other information of the input face. To achieve this goal, current methods required carefully designed loss functions [29, 10, 35], introduction of additional attribute labels or features [25, 39, 3, 37, 34], or special architectures [11, 33] to train new models. However, the synthesis resolution and quality of these models are far behind those of native GANs, like PGGAN [21] and StyleGAN [22]. Different from previous learning-based methods, this work explores the interpretable semantics inside the latent space of *fixed* GAN models, and *turns unconstrained GANs to controllable GANs* by varying the latent code.

2. Framework of InterFaceGAN

In this section, we introduce the framework of InterFaceGAN, which first provides a rigorous analysis of the semantic attributes emerging in the latent space of well-trained GAN models, and then constructs a manipulation pipeline of leveraging the semantics in the latent code for facial attribute editing.

2.1. Semantics in the Latent Space

Given a well-trained GAN model, the generator can be formulated as a deterministic function $g : \mathcal{Z} \to \mathcal{X}$. Here, $\mathcal{Z} \subseteq \mathbb{R}^d$ denotes the d-dimensional latent space, for which Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ is commonly used [28, 21, 8, 22]. $\mathcal{X}$ stands for the image space, where each sample $\mathbf{x}$ possesses certain semantic information, like gender and age for face model. Suppose we have a semantic scoring function $f_S : \mathcal{X} \to \mathcal{S}$, where $\mathcal{S} \subseteq \mathbb{R}^m$ represents the semantic space with m semantics. We can bridge the latent space $\mathcal{Z}$ and the semantic space $\mathcal{S}$ with $\mathbf{s} = f_S(g(\mathbf{z}))$, where $\mathbf{s}$ and $\mathbf{z}$ denote the semantic scores and the sampled latent code respectively.

Single Semantic. It has been widely observed that when linearly interpolating two latent codes $\mathbf{z}_1$ and $\mathbf{z}_2$, the appearance of the corresponding synthesis changes continuously [31, 8, 22]. It implicitly means that the semantics contained in the image also change gradually. According to *Property 1*, the linear interpolation between $\mathbf{z}_1$ and $\mathbf{z}_2$ forms a direction in $\mathcal{Z}$, which further defines a hyperplane. We therefore make an assumption[2] that for any binary semantic (*e.g.*, male *v.s.* female), there exists a hyperplane in the latent space serving as the separation boundary. Semantic remains the same when the latent code walks within the same side of the hyperplane yet turns into the opposite when across the boundary.

Given a hyperplane with a unit normal vector $\mathbf{n} \in \mathbb{R}^d$, we define the "distance" from a sample $\mathbf{z}$ to this hyperplane as

$$\mathrm{d}(\mathbf{n}, \mathbf{z}) = \mathbf{n}^T \mathbf{z}. \tag{1}$$

Here, $\mathrm{d}(\cdot, \cdot)$ is not a strictly defined distance since it can be negative. When $\mathbf{z}$ lies near the boundary and is moved toward and across the hyperplane, both the "distance" and the semantic score vary accordingly. And it is just at the time when the "distance" changes its numerical sign that the semantic attribute reverses. We therefore expect these two to be linearly dependent with

$$f(g(\mathbf{z})) = \lambda \mathrm{d}(\mathbf{n}, \mathbf{z}), \tag{2}$$

where $f(\cdot)$ is the scoring function for a particular semantic, and $\lambda > 0$ is a scalar to measure how fast the semantic varies along with the change of distance. According to *Property 2*, random samples drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ are very likely to locate close enough to a given hyperplane. Therefore, the corresponding semantic can be modeled by the linear subspace that is defined by $\mathbf{n}$.

Property 1 *Given $\mathbf{n} \in \mathbb{R}^d$ with $\mathbf{n} \neq \mathbf{0}$, the set $\{\mathbf{z} \in \mathbb{R}^d : \mathbf{n}^T \mathbf{z} = 0\}$ defines a hyperplane in $\mathbb{R}^d$, and $\mathbf{n}$ is called the normal vector. All vectors $\mathbf{z} \in \mathbb{R}^d$ satisfying $\mathbf{n}^T \mathbf{z} > 0$ locate from the same side of the hyperplane.*

[2]This assumption is empirically verified in Sec.3.1.

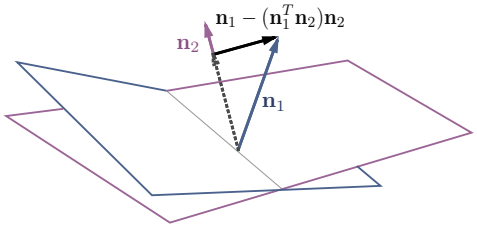


Figure 2: Illustration of the conditional manipulation in subspace. The projection of $\mathbf{n}_1$ onto $\mathbf{n}_2$ is subtracted from $\mathbf{n}_1$, resulting in a new direction $\mathbf{n}_1 - (\mathbf{n}_1^T \mathbf{n}_2)\mathbf{n}_2$.

Property 2 *Given $\mathbf{n} \in \mathbb{R}^d$ with $\mathbf{n}^T \mathbf{n} = 1$, which defines a hyperplane, and a multivariate random variable $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, we have $\mathrm{P}(|\mathbf{n}^T \mathbf{z}| \leq 2\alpha \sqrt{\frac{d}{d-2}}) \geq (1 - 3e^{-cd})(1 - \frac{2}{\alpha} e^{-\alpha^2/2})$ for any $\alpha \geq 1$ and $d \geq 4$. Here, $\mathrm{P}(\cdot)$ stands for probability and c is a fixed positive constant.*[3]

Multiple Semantics. When the case comes to m different semantics, we have

$$\mathbf{s} \equiv f_S(g(\mathbf{z})) = \Lambda \mathbf{N}^T \mathbf{z}, \tag{3}$$

where $\mathbf{s} = [s_1, \ldots, s_m]^T$ denotes the semantic scores, $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ is a diagonal matrix containing the linear coefficients, and $\mathbf{N} = [\mathbf{n}_1, \ldots, \mathbf{n}_m]$ indicates the separation boundaries. Aware of the distribution of random sample $\mathbf{z}$, which is $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, we can easily compute the mean and covariance matrix of the semantic scores $\mathbf{s}$ as

$$\boldsymbol{\mu}_\mathbf{s} = \mathbb{E}(\Lambda \mathbf{N}^T \mathbf{z}) = \Lambda \mathbf{N}^T \mathbb{E}(\mathbf{z}) = \mathbf{0}, \tag{4}$$

$$\begin{aligned} \boldsymbol{\Sigma}_\mathbf{s} &= \mathbb{E}(\Lambda \mathbf{N}^T \mathbf{z}\mathbf{z}^T \mathbf{N} \Lambda^T) = \Lambda \mathbf{N}^T \mathbb{E}(\mathbf{z}\mathbf{z}^T) \mathbf{N} \Lambda^T \\ &= \Lambda \mathbf{N}^T \mathbf{N} \Lambda. \end{aligned} \tag{5}$$

We therefore have $\mathbf{s} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\mathbf{s})$, which is a multivariate normal distribution. Different entries of $\mathbf{s}$ are disentangled if and only if $\boldsymbol{\Sigma}_\mathbf{s}$ is a diagonal matrix, which requires $\{\mathbf{n}_1, \ldots, \mathbf{n}_m\}$ to be orthogonal with each other. If this condition does not hold, some semantics will correlate with each other and $\mathbf{n}_i^T \mathbf{n}_j$ can be used to measure the entanglement between the i-th and j-th semantics.

2.2. Manipulation in the Latent Space

In this part, we introduce how to use the semantics found in latent space for image editing.

Single Attribute Manipulation. According to Eq.(2), to manipulate the attribute of a synthesized image, we can easily edit the original latent code $\mathbf{z}$ with $\mathbf{z}_{edit} = \mathbf{z} + \alpha \mathbf{n}$. It will make the synthesis look more positive on such semantic with $\alpha > 0$, since the score becomes $f(g(\mathbf{z}_{edit})) = f(g(\mathbf{z})) + \lambda\alpha$ after editing. Similarly, $\alpha < 0$ will make the synthesis look more negative.

[3]When $d = 512$, we have $P(|\mathbf{n}^T \mathbf{z}| > 5.0) < 1e^{-6}$. It suggests that almost all sampled latent codes are expected to locate within 5 unit-length to the boundary. Proof can be found in **Appendix**.

Conditional Manipulation. When there is more than one attribute, editing one may affect another since some semantics can be coupled with each other. To achieve more precise control, we propose *conditional manipulation* by manually forcing $\mathbf{N}^T\mathbf{N}$ in Eq.(5) to be diagonal. In particular, we use projection to orthogonalize different vectors. As shown in Fig.2, given two hyperplanes with normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$, we find a projected direction $\mathbf{n}_1 - (\mathbf{n}_1^T\mathbf{n}_2)\mathbf{n}_2$, such that moving samples along this new direction can change "attribute 1" without affecting "attribute 2". We call this operation as conditional manipulation. If there is more than one attribute to be conditioned on, we just subtract the projection from the primal direction onto the plane that is constructed by all conditioned directions.

Real Image Manipulation. Since our approach enables semantic editing from the latent space of a *fixed* GAN model, we need to first map a real image to a latent code before performing manipulation. For this purpose, existing methods have proposed to directly optimize the latent code to minimize the reconstruction loss [27], or to learn an extra encoder to invert the target image back to latent space [42, 5]. There are also some models that have already involved an encoder along with the training process of GANs [13, 12, 41], which we can directly use for inference.

3. Experiments

In this section, we evaluate InterFaceGAN with state-of-the-art GAN models, PGGAN [21] and StyleGAN [22]. Specifically, the experiments in Sec.3.1, Sec.3.2, and Sec.3.3 are conducted on PGGAN to interpret the latent space of the traditional generator. Experiments in Sec.3.4 are carried out on StyleGAN to investigate the style-based generator and also compare the differences between the two sets of latent representations in StyleGAN. We also apply our approach to real images in Sec.3.5 to see how the semantics implicitly learned by GANs can be applied to real face editing. Implementation details can be found in **Appendix**.

3.1. Latent Space Separation

As mentioned in Sec.2.1, our framework is based on an assumption that for any binary attribute, there exists a hyperplane in latent space such that all samples from the same side are with the same attribute. Accordingly, we would like to first evaluate the correctness of this assumption to make the remaining analysis considerable.

We train five independent linear SVMs on pose, smile, age, gender, eyeglasses, and then evaluate them on the validation set ($6K$ samples with high confidence level on attribute scores) as well as the entire set ($480K$ random samples). Tab.1 shows the results. We find that all linear boundaries achieve over 95% accuracy on the validation set and over 75% on the entire set, suggesting that for a binary attribute, there exists a linear hyperplane in the latent space that can well separate the data into two groups.

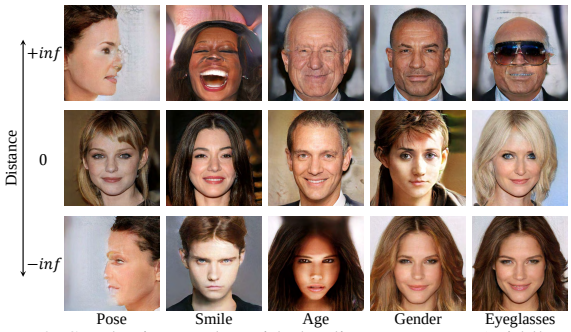


Figure 3: Synthesis samples with the distance near to (middle row) and extremely far away from (top and bottom rows) the separation boundary. Each column corresponds to a particular attribute.

Table 1: Classification accuracy (%) on separation boundaries in latent space with respect to different attributes.

Dataset	Pose	Smile	Age	Gender	Eyeglasses
Validation	100.0	96.9	97.9	98.7	95.6
All	90.3	78.5	75.3	84.2	80.1

We also visualize some samples in Fig.3 by ranking them with the distance to the decision boundary. Note that those extreme cases (first and last row in Fig.3) are very unlikely to be directly sampled, instead constructed by moving a latent code towards the normal direction "infinitely". From Fig.3, we can tell that the positive samples and negative samples are distinguishable to each other with respect to the corresponding attribute.

3.2. Latent Space Manipulation

In this part, we verify whether the semantics found by InterFaceGAN are manipulable.

Manipulating Single Attribute. Fig.4 plots the manipulation results on five different attributes. It suggests that our manipulation approach performs well on all attributes in both positive and negative directions. Particularly on *pose* attribute, we observe that even the boundary is searched by solving a bi-classification problem, moving the latent code can produce continuous changing. Furthermore, although there lacks enough data with extreme poses in the training set, GAN is capable of imagining how profile faces should look like. The same situation also happens on eyeglasses attribute. We can manually create a lot of faces wearing eyeglasses despite the inadequate data in the training set. These two observations provide strong evidence that GAN does not produce images randomly, but learns some interpretable semantics from the latent space.

Distance Effect of Semantic Subspace. When manipulating the latent code, we observe an interesting distance effect that the samples will suffer from severe changes in appearance if being moved too far from the boundary, and

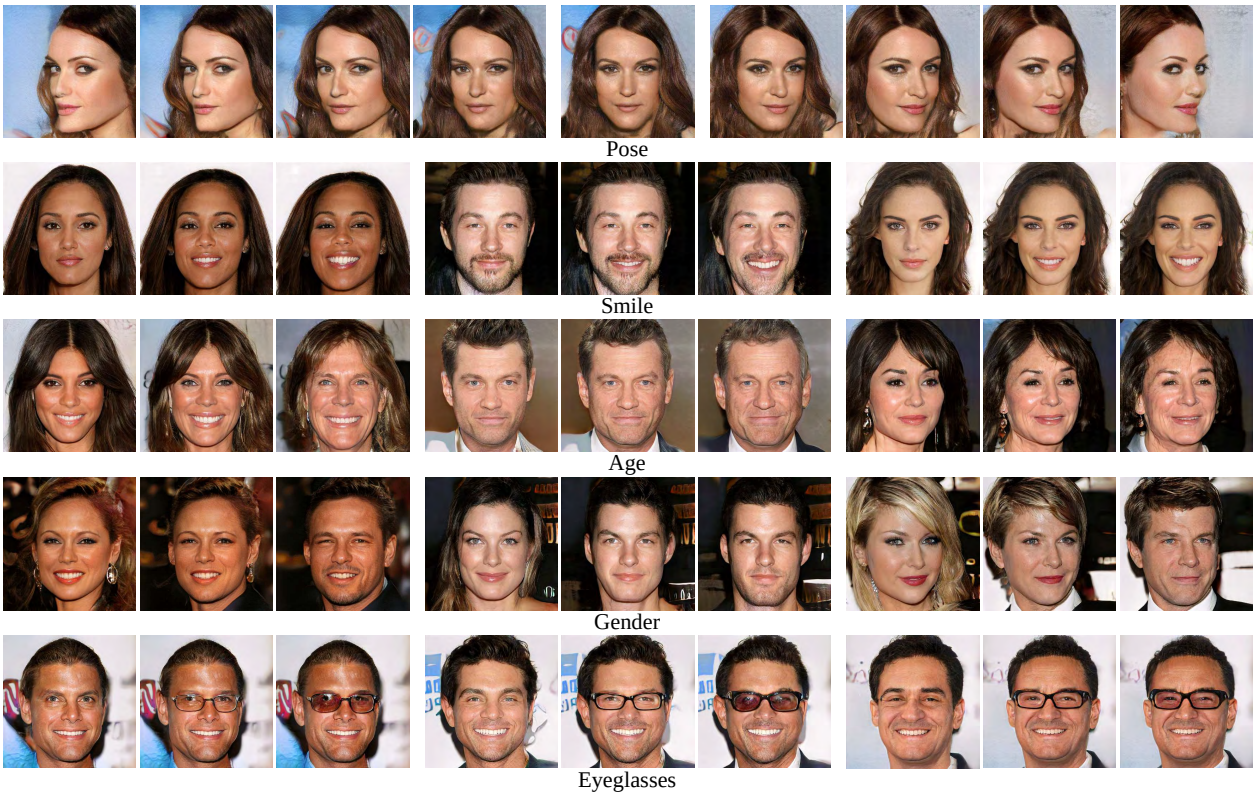


Figure 4: Single attribute manipulation results. The first row shows the same person under gradually changed poses. The following rows correspond to the results of manipulating four different attributes. For each set of three samples in a row, the central one is the original synthesis, while the left and right stand for the results by moving the latent code along negative and positive direction respectively.

Figure 5: Illustration of the distance effect by taking gender manipulation as an example. The image in the red dashed box stands for the original synthesis. Our approach performs well when the latent code locates close to the boundary. However, when the distance keeps increasing, the synthesized images are no longer like the same person.

finally tend to become the extreme cases shown in Fig.3. Fig.5 illustrates this phenomenon by taking gender editing as an instance. Near-boundary manipulation works well. When samples go beyond a certain region[4], however, the editing results are no longer like the original face anymore. But this effect does not affect our understanding of the disentangled semantics in latent space. That is because such extreme samples are very unlikely to be directly drawn from a standard normal distribution, which is pointed out in **Property 2** in Sec.2.1. Instead, they are constructed manually by keeping moving a normally sampled latent code along a certain direction. In this way, we can get a better interpretation on the latent semantics of GANs.

Artifacts Correction. We further apply our approach to fix the artifacts that sometimes occurred in the synthesized

Figure 6: Examples on fixing the artifacts that GAN has generated. First row shows some bad generation results, while the following two rows present the gradually corrected synthesis by moving the latent codes along the positive "quality" direction.

[4]We choose 5.0 as the threshold.

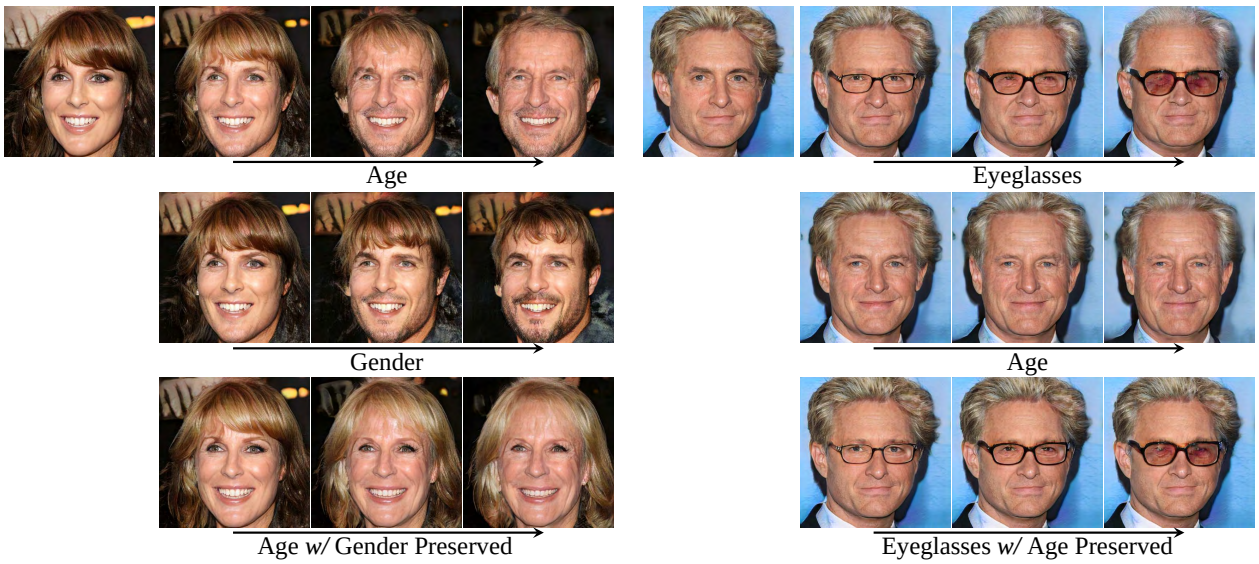


Figure 7: Examples for conditional manipulation. The first two rows show the manipulation results along with the original directions learned by SVMs for two attributes independently. The last row edits the faces by varying one attribute with the other one unchanged.

outputs. We manually labeled $4K$ bad synthesis and then trained a linear SVM to find the separation hyperplane, same as other attributes. We surprisingly find that GAN also encodes such information in latent space. Based on this discovery, we are capable of correcting some mistakes GAN has made in the generation process, as shown in Fig.6.

3.3. Conditional Manipulation

In this section, we study the disentanglement between different attributes and evaluate the conditional manipulation approach.

Correlation between Attributes. Different from [22] which introduced perceptual path length and linear separability to measure the disentanglement property of latent space, we focus more on the relationships between different hidden semantics and study how they are coupled with each other. Here, two different metrics are used to measure the correlation between two attributes. (i) We compute the cosine similarity between two directions as $\cos(\mathbf{n}_1, \mathbf{n}_2) = \mathbf{n}_1^T \mathbf{n}_2$, where $\mathbf{n}_1$ and $\mathbf{n}_2$ stand for unit vectors. (ii) We treat each attribute score as a random variable, and use the attribute distribution observed from all $500K$ synthesized data to compute the correlation coefficient ρ. Here, we have $\rho_{A_1 A_2} = \frac{Cov(A_1, A_2)}{\sigma_{A_1} \sigma_{A_2}}$, where A_1 and A_2 represent two random variables with respect to two attributes. $Cov(\cdot, \cdot)$ stands for covariance, and σ denotes standard deviation.

Tab.2 and Tab.3 report the results. We can tell that attributes behave similarly under these two metrics, showing that our InterFaceGAN is able to accurately identify the semantics hidden in latent space. We also find that pose and smile are almost orthogonal to other attributes. Nevertheless, gender, age, and eyeglasses are highly correlated with each other. This observation reflects the attribute correlation in the training dataset (*i.e.*, CelebA-HQ [21]) to some extent, where male old people are more likely to wear eyeglasses. This characteristic is also captured by GAN when learning to produce real observation.

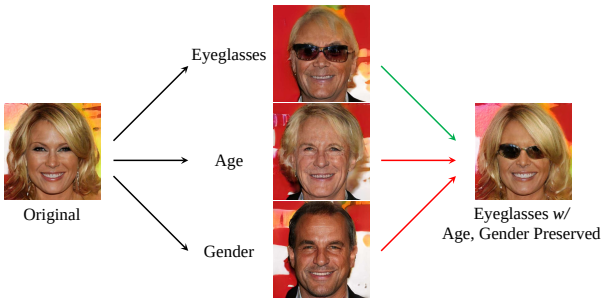


Figure 8: Examples for conditional manipulation with more than one conditions. Left: Original synthesis. Middle: Manipulations along single boundary. Right: Conditional manipulation. **Green** arrow: Primal direction. **Red** arrows: Projection subtraction.

Table 2: Correlation matrix of attribute boundaries.

	Pose	Smile	Age	Gender	Eyeglasses
Pose	1.00	-0.04	-0.06	-0.05	-0.04
Smile	-	1.00	0.04	-0.10	-0.05
Age	-	-	1.00	0.49	0.38
Gender	-	-	-	1.00	0.52
Eyeglasses	-	-	-	-	1.00

Table 3: Correlation matrix of synthesized attribute distributions.

	Pose	Smile	Age	Gender	Eyeglasses
Pose	1.00	-0.01	-0.01	-0.02	0.00
Smile	-	1.00	0.02	-0.08	-0.01
Age	-	-	1.00	0.42	0.35
Gender	-	-	-	1.00	0.47
Eyeglasses	-	-	-	-	1.00

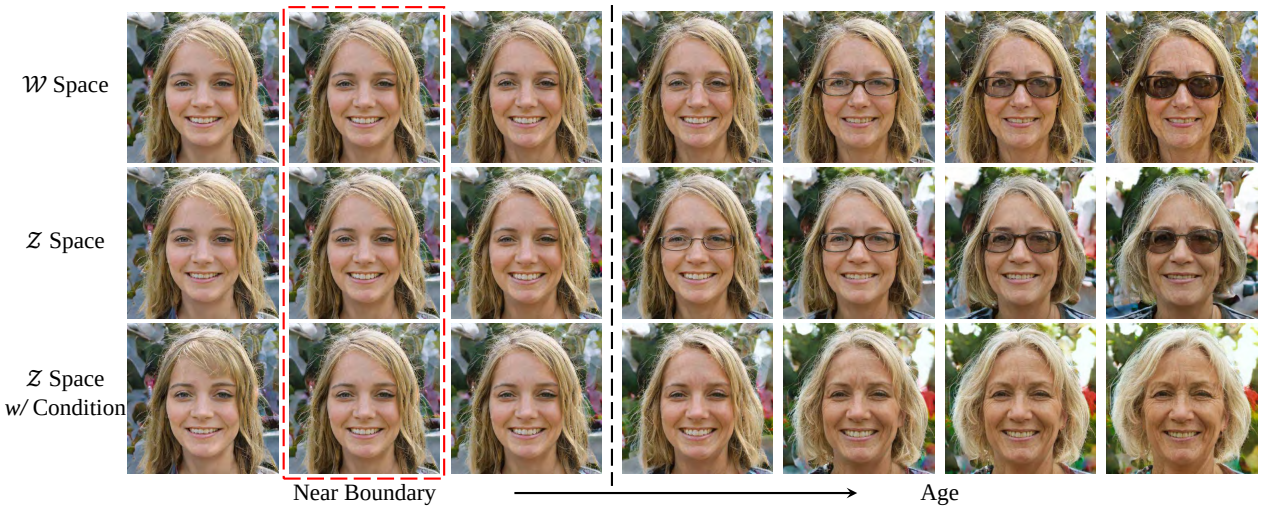


Figure 9: Analysis on the latent space $\mathcal{Z}$ and disentangled latent space $\mathcal{W}$ of StyleGAN [22] by taking age manipulation as an example. $\mathcal{W}$ space behaves better for long term manipulation, but the flaw in $\mathcal{Z}$ space can be fixed by projection (*i.e.*, conditional manipulation) to achieve better performance.

Conditional Manipulation. To decorrelate different semantics for independent facial attribute editing, we propose conditional manipulation in Sec.2.2. Fig.7 shows some results by manipulating one attribute with another one as a condition. Taking the left sample in Fig.7 as an example, the results tend to become male when being edited to get old (first row). We fix this problem by subtracting its projection onto the gender direction (second row) from age direction, resulting in a new direction. In this way, we can make sure the gender component is barely affected when the sample is moved along the projected direction (third row). Fig.8 shows conditional manipulation with more than one constraint, where we add glasses by conditionally preserving age and gender. In the beginning, adding eyeglasses is entangled with changing both age and gender. But we manage to add glasses without affecting age and gender with projection operation. These two experiments show that our proposed conditional approach helps to achieve independent and precise attribute control.

3.4. Results on StyleGAN

Different from conventional GANs, StyleGAN [22] proposed style-based generator. Basically, StyleGAN learns to map the latent code from space $\mathcal{Z}$ to another high dimensional space $\mathcal{W}$ before feeding it into the generator. As pointed out in [22], $\mathcal{W}$ shows much stronger disentanglement property than $\mathcal{Z}$, since $\mathcal{W}$ is not restricted to any certain distribution and can better model the underlying character of real data.

We did a similar analysis on both $\mathcal{Z}$ and $\mathcal{W}$ spaces of StyleGAN as did to PGGAN and found that $\mathcal{W}$ space indeed learns a more disentangled representation, as pointed out by [22]. Such disentanglement helps $\mathcal{W}$ space achieve strong superiority over $\mathcal{Z}$ space for attribute editing. As shown in Fig.9, age and eyeglasses are also entangled in StyleGAN model. Compared to $\mathcal{Z}$ space (second row), $\mathcal{W}$ space (first row) performs better, especially in long-distance manipulation. Nevertheless, we can use the conditional manipulation trick described in Sec.2.2 to decorrelate these two attributes in $\mathcal{Z}$ space (third row), resulting in more appealing results. This trick, however, cannot be applied to $\mathcal{W}$ space. We found that $\mathcal{W}$ space sometimes captures the attributes correlation that happens in training data and encodes them together as a coupled "style". Taking Fig.9 as an example, "age" and "eyeglasses" are supported to be two independent semantics, but StyleGAN actually learns an eyeglasses-included age direction such that this new direction is somehow orthogonal to the eyeglasses direction itself. In this way, subtracting the projection, which is almost zero, will hardly affect the final results.

3.5. Real Image Manipulation

In this part, we manipulate real faces with the proposed InterFaceGAN to verify whether the semantic attributes learned by GAN can be applied to real faces. Recall that InterFaceGAN achieves semantic face editing by moving the latent code along a certain direction. Accordingly, we need to first invert the given real image back to the latent code. It turns out to be a non-trivial task because GANs do not fully capture all the modes as well as the diversity of the true distribution. To invert a pre-trained GAN model, there are two typical approaches. One is the optimization-based approach, which directly optimizes the latent code with the fixed generator to minimize the pixel-wise reconstruction error [27]. The other is the encoder-based, where an extra encoder network is trained to learn the inverse mapping [42]. We tested the two baseline approaches on PGGAN and StyleGAN.

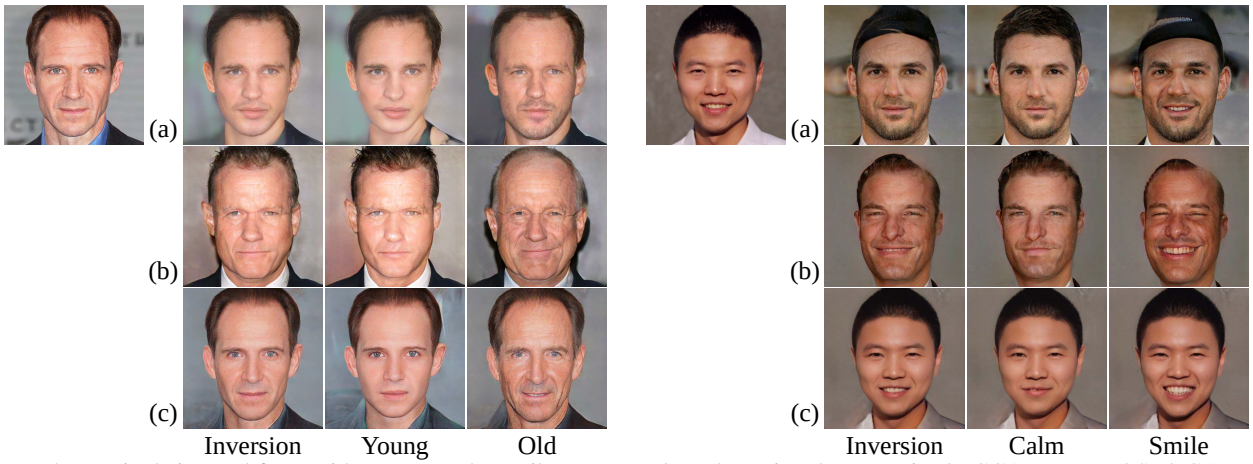


Figure 10: Manipulating real faces with respect to the attributes age and gender, using the pre-trained PGGAN [21] and StyleGAN [22]. Given an image to edit, we first invert it back to the latent code and then manipulate the latent code with InterFaceGAN. On the top left corner is the input real face. From top to bottom: (a) PGGAN with optimization-based inversion method, (b) PGGAN with encoder-based inversion method, (c) StyleGAN with optimization-based inversion method.

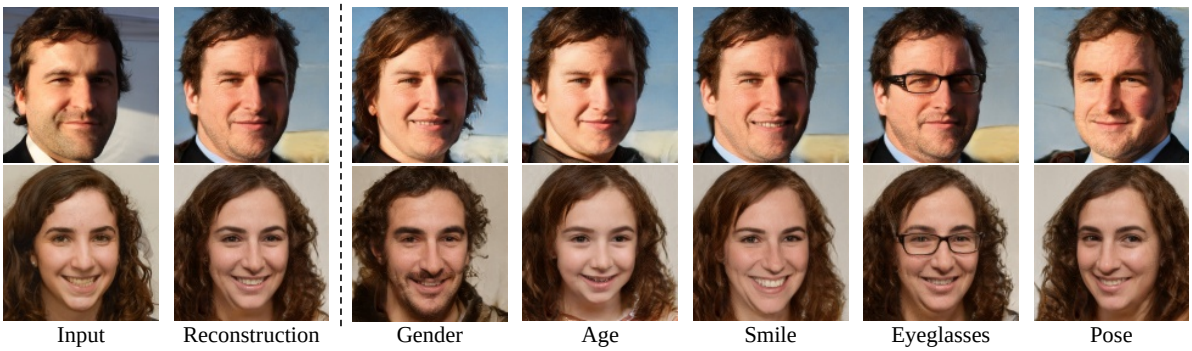


Figure 11: Manipulating real faces with LIA [41], which is a encoder-decoder generative model for high-resolution face synthesis.

Results are shown in Fig.10. We can tell that both optimization-based (first row) and encoder-based (second row) methods show poor performance when inverting PGGAN. This can be imputed to the strong discrepancy between training and testing data distributions. For example, the model tends to generate Western people even the input is an Easterner (see the right example in Fig.10). Even unlike the inputs, however, the inverted images can still be semantically edited with InterFaceGAN. Compared to PGGAN, the results on StyleGAN (third row) are much better. Here, we treat the layer-wise styles (*i.e.*, $\mathbf{w}$ for all layers) as the optimization target. When editing an instance, we push all style codes towards the same direction. As shown in Fig.10, we successfully change the attributes of real face images *without* retraining StyleGAN but leveraging the interpreted semantics from latent space.

We also test InterFaceGAN on encoder-decoder generative models, which train an encoder together with the generator and discriminator. After the model converges, the encoder can be directly used for inference to map a given image to latent space. We apply our method to interpret the latent space of the recent encoder-decoder model LIA [41]. The manipulation result is shown in Fig.11 where we successfully edit the input faces with various attributes, like age and face pose. It suggests that the latent code in the encoder-decoder based generative models also supports semantic manipulation. In addition, compared to Fig.10 (b) where the encoder is separately learned after the GAN model is well-prepared, the encoder trained together with the generator gives better reconstruction as well as manipulation results.

4. Conclusion

We propose InterFaceGAN to interpret the semantics encoded in the latent space of GANs. By leveraging the interpreted semantics as well as the proposed conditional manipulation technique, we are able to precisely control the facial attributes with any fixed GAN model, even turning unconditional GANs to controllable GANs. Extensive experiments suggest that InterFaceGAN can also be applied to real image editing.

Acknowledgement: This work is supported in part by the Early Career Scheme (ECS) through the Research Grants Council of Hong Kong under Grant No.24206219 and in part by SenseTime Collaborative Grant.

Appendix

A. Overview

This appendix contains the following information:

- We introduce the implementation details of the proposed InterFaceGAN in Sec.B.
- We provide the detailed proof of *Property 2* in the main paper in Sec.C.
- Please also refer to this video to see continuous attribute editing results.

B. Implementation Details

We choose five key facial attributes for analysis, including pose, smile (expression), age, gender, and eyeglasses. The corresponding positive directions are defined as turning right, laughing, getting old, changing to male, and wearing eyeglasses. Note that we can always plug in more attributes easily as long as the attribute detector is available.

To better predict these attributes from synthesized images, we train an auxiliary attribute prediction model using the annotations from the CelebA dataset [26] with ResNet-50 network [18]. This model is trained with multi-task losses to simultaneously predict smile, age, gender, eyeglasses, as well as the 5-point facial landmarks. Here, the facial landmarks will be used to compute yaw pose, which is also treated as a binary attribute (left or right) in further analysis. Besides the landmarks, all other attributes are learned as bi-classification problem with softmax cross-entropy loss, while landmarks are optimized with l_2 regression loss. As images produced by PGGAN and StyleGAN are with 1024×1024 resolution, we resize them to 224×224 before feeding them to the attribute model.

Given the pre-trained GAN model, we synthesize $500K$ images by randomly sampling the latent space. There are mainly two reasons in preparing such large-scale data: (i) to eliminate the randomness caused by sampling and make sure the distribution of the latent codes is as expected, and (ii) to get enough wearing-glasses samples, which are really rare in PGGAN model.

To find the semantic boundaries in the latent space, we use the pre-trained attribute prediction model to assign attribute scores for all $500K$ synthesized images. For each attribute, we sort the corresponding scores, and choose $10K$ samples with highest scores and $10K$ with lowest ones as candidates. The reason in doing so is that the prediction model is not absolutely accurate and may produce wrong prediction for ambiguous samples, *e.g.*, middle-aged person for age attribute. We then randomly choose 70% samples from the candidates as the training set to learn a linear SVM, resulting in a decision boundary. Recall that, normal directions of all boundaries are normalized to unit vectors.

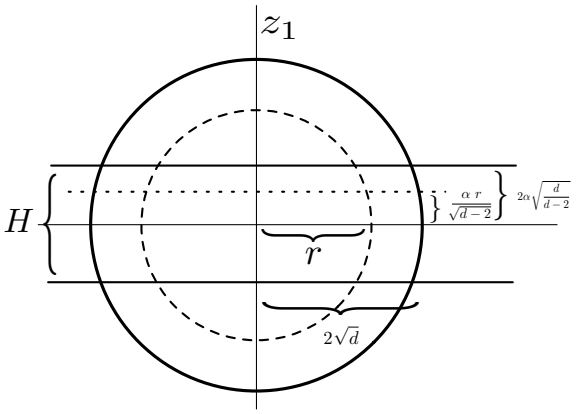


Figure 12: Illustration of *Property 2*, which shows that most of the probability mass of high-dimensional Gaussian distribution lies in the thin slab near the "equator".

Remaining 30% are used for verifying how the linear classifier behaves. Here, for SVM training, the inputs are the $512d$ latent codes, while the binary labels are assigned by the auxiliary attribute prediction model.

C. Proof

In this part, we provide detailed proof of *Property 2* in the main paper. Recall this property as follow.

Property 2 *Given $\mathbf{n} \in \mathbb{R}^d$ with $\mathbf{n}^T\mathbf{n} = 1$, which defines a hyperplane, and a multivariate random variable $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, we have $\mathrm{P}(|\mathbf{n}^T\mathbf{z}| \leq 2\alpha\sqrt{\frac{d}{d-2}}) \geq (1 - 3e^{-cd})(1 - \frac{2}{\alpha}e^{-\alpha^2/2})$ for any $\alpha \geq 1$ and $d \geq 4$. Here $\mathrm{P}(\cdot)$ stands for probability and c is a fixed positive constant.*

Proof.

Without loss of generality, we fix $\mathbf{n}$ to be the first coordinate vector. Accordingly, it suffices to prove that $\mathrm{P}(|z_1| \leq 2\alpha\sqrt{\frac{d}{d-2}}) \geq (1 - 3e^{-cd})(1 - \frac{2}{\alpha}e^{-\alpha^2/2})$, where z_1 denotes the first entry of $\mathbf{z}$.

As shown in Fig.12, let H denote the set

$$\{\mathbf{z} \sim \mathbf{N}(\mathbf{0}, \mathbf{I}_d) : ||\mathbf{z}||_2 \leq 2\sqrt{d}, |z_1| \leq 2\alpha\sqrt{\frac{d}{d-2}}\},$$

where $||\cdot||_2$ stands for the l_2 norm. Obviously, we have $\mathrm{P}(H) \leq \mathrm{P}(|z_1| \leq 2\alpha\sqrt{\frac{d}{d-2}})$. Now, we will show $\mathrm{P}(H) \geq (1 - 3e^{-cd})(1 - \frac{2}{\alpha}e^{-\alpha^2/2})$

Considering the random variable $R = ||\mathbf{z}||_2$, with cumulative distribution function $F(R \leq r)$ and density function $f(r)$, we have

$$\begin{aligned}\mathrm{P}(H) &= \mathrm{P}(|z_1| \leq 2\alpha\sqrt{\frac{d}{d-2}}|R \leq 2\sqrt{d})\mathrm{P}(R \leq 2\sqrt{d}) \\ &= \int_0^{2\sqrt{d}} \mathrm{P}(|z_1| \leq 2\alpha\sqrt{\frac{d}{d-2}}|R = r)f(r)dr.\end{aligned}$$

According to *Theorem 1* below, when $r \leq 2\sqrt{d}$, we have

$$\begin{aligned}
\mathrm{P}(H) &= \int_0^{2\sqrt{d}} \mathrm{P}(|z_1| \leq 2\alpha\sqrt{\frac{d}{d-2}} | R = r) f(r) dr \\
&= \int_0^{2\sqrt{d}} \mathrm{P}(|z_1| \leq \frac{2\sqrt{d}}{r} \frac{\alpha}{\sqrt{d-2}} | R = 1) f(r) dr \\
&\geq \int_0^{2\sqrt{d}} \mathrm{P}(|z_1| \leq \frac{\alpha}{\sqrt{d-2}} | R = 1) f(r) dr \\
&\geq \int_0^{2\sqrt{d}} (1 - \frac{2}{\alpha} e^{-\alpha^2/2}) f(r) dr \\
&= (1 - \frac{2}{\alpha} e^{-\alpha^2/2}) \int_0^{2\sqrt{d}} f(r) dr \\
&= (1 - \frac{2}{\alpha} e^{-\alpha^2/2}) \mathrm{P}(0 \leq R \leq 2\sqrt{d}).
\end{aligned}$$

Then, according to *Theorem 2* below, by setting $\beta = \sqrt{d}$, we have

$$\begin{aligned}
\mathrm{P}(H) &= (1 - \frac{2}{\alpha} e^{-\alpha^2/2}) \mathrm{P}(0 \leq R \leq 2\sqrt{d}) \\
&\geq (1 - \frac{2}{\alpha} e^{-\alpha^2/2})(1 - 3e^{-cd}).
\end{aligned}$$

Q.E.D.

Theorem 1 *Given a unit spherical* $\{\mathbf{z} \in \mathbb{R}^d : ||\mathbf{z}||_2 = 1\}$, *we have* $\mathrm{P}(|z_1| \leq \frac{\alpha}{\sqrt{d-2}}) \geq 1 - \frac{2}{\alpha} e^{-\alpha^2/2}$ *for any* $\alpha \geq 1$ *and* $d \geq 4$.

Proof.

By symmetry, we just prove the case where $z_1 \geq 0$. Also, we only consider about the case where $\frac{\alpha}{\sqrt{d-2}} \leq 1$.

Let U denote the set $\{\mathbf{z} \in \mathbb{R}^d : ||\mathbf{z}||_2 = 1, z_1 \geq \frac{\alpha}{\sqrt{d-2}}\}$, and K denote the set $\{\mathbf{z} \in \mathbb{R}^d : ||\mathbf{z}||_2 = 1, z_1 \geq 0\}$. It suffices to prove that the surface of U area and the surface of K area in Fig.13 satisfy

$$\frac{surf(U)}{surf(K)} \leq \frac{2}{\alpha} e^{-\alpha^2/2},$$

where $surf(\cdot)$ stands for the surface area of a high dimensional geometry. Let $A(d)$ denote the surface area of a d-dimensional unit-radius ball. Then, we have

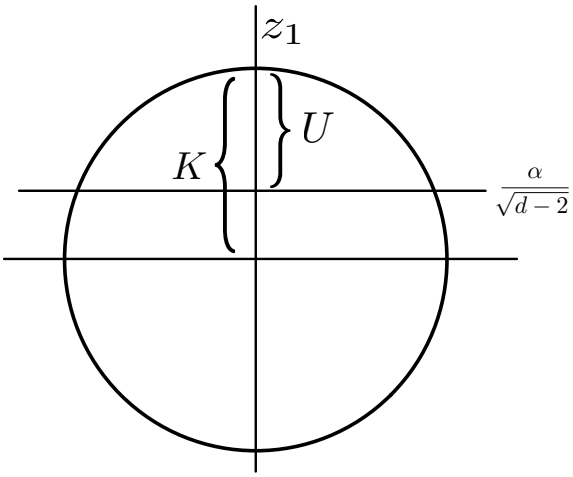


Figure 13: Diagram for *Theorem 1*.

$$\begin{aligned}
surf(U) &= \int_{\frac{\alpha}{\sqrt{d-2}}}^1 (1 - z_1^2)^{\frac{d-2}{2}} A(d-1) dz_1 \\
&\leq \int_{\frac{\alpha}{\sqrt{d-2}}}^1 e^{-\frac{d-2}{2} z_1^2} A(d-1) dz_1 \\
&\leq \int_{\frac{\alpha}{\sqrt{d-2}}}^1 \frac{z_1\sqrt{d-2}}{\alpha} e^{-\frac{d-2}{2} z_1^2} A(d-1) dz_1 \\
&\leq \int_{\frac{\alpha}{\sqrt{d-2}}}^\infty \frac{z_1\sqrt{d-2}}{\alpha} e^{-\frac{d-2}{2} z_1^2} A(d-1) dz_1 \\
&= \frac{A(d-1)}{\alpha\sqrt{d-2}} e^{-\alpha^2/2}.
\end{aligned}$$

Similarly, we have

$$\begin{aligned}
surf(K) &= \int_0^1 (1 - z_1^2)^{\frac{d-2}{2}} A(d-1) dz_1 \\
&\geq \int_0^{\frac{1}{\sqrt{d-2}}} (1 - z_1^2)^{\frac{d-2}{2}} A(d-1) dz_1 \\
&\geq \frac{1}{\sqrt{d-2}} (1 - \frac{1}{d-2})^{\frac{d-2}{2}} A(d-1).
\end{aligned}$$

Considering the fact that $(1-x)^a \geq 1 - ax$ for any $a \geq 1$ and $0 \leq x \leq 1$, we have

$$\begin{aligned}
surf(K) &\geq \frac{1}{\sqrt{d-2}} (1 - \frac{1}{d-2})^{\frac{d-2}{2}} A(d-1) \\
&\geq \frac{1}{\sqrt{d-2}} (1 - \frac{1}{d-2} \frac{d-2}{2}) A(d-1) \\
&= \frac{A(d-1)}{2\sqrt{d-2}}.
\end{aligned}$$

Accordingly,

$$\frac{surf(U)}{surf(K)} \leq \frac{\frac{A(d-1)}{\alpha\sqrt{d-2}}e^{-\alpha^2/2}}{\frac{A(d-1)}{2\sqrt{d-2}}} = \frac{2}{\alpha}e^{-\alpha^2/2}.$$

Q.E.D.

Theorem 2 (Gaussian Annulus Theorem [19]) *For a d-dimensional spherical Gaussian with unit variance in each direction, for any $\beta \leq \sqrt{d}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within the annulus $\sqrt{d} - \beta \leq ||\mathbf{z}||_2 \leq \sqrt{d} + \beta$, where c is a fixed positive constant.*

That is to say, given $\mathbf{z} \sim \mathbf{N}(\mathbf{0}, \mathbf{I}_d)$, $\beta \leq \sqrt{d}$, and a constant $c > 0$, we have

$$\mathrm{P}(\sqrt{d} - \beta \leq ||\mathbf{z}||_2 \leq \sqrt{d} + \beta) \geq (1 - 3e^{-c\beta^2}).$$

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017. 2

[2] Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. Latent space oddity: on the curvature of deep generative models. In *ICLR*, 2018. 2

[3] Jianmin Bao, Dong Chen, Fang Wen, Houqiang Li, and Gang Hua. Towards open-set identity preserving face synthesis. In *CVPR*, 2018. 2

[4] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. Visualizing and understanding generative adversarial networks. In *ICLR*, 2019. 2

[5] David Bau, Jun-Yan Zhu, Jonas Wulff, William Peebles, Hendrik Strobelt, Bolei Zhou, and Antonio Torralba. Seeing what a gan cannot generate. In *ICCV*, 2019. 2, 4

[6] David Berthelot, Thomas Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017. 2

[7] Piotr Bojanowski, Armand Joulin, David Lopez-Pas, and Arthur Szlam. Optimizing the latent space of generative networks. In *ICML*, 2018. 2

[8] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019. 2, 3

[9] Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick van der Smagt. Metrics for deep generative models. In *AISTAT*, 2018. 2

[10] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NeurIPS*, 2016. 2

[11] Chris Donahue, Akshay Balsubramani, Julian McAuley, and Zachary C. Lipton. Semantically decomposing the latent spaces of generative adversarial networks. In *ICLR*, 2018. 2

[12] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *ICLR*, 2017. 2, 4

[13] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. In *ICLR*, 2017. 2, 4

[14] Lore Goetschalckx, Alex Andonian, Aude Oliva, and Phillip Isola. Ganalyze: Toward visual definitions of cognitive image properties. In *ICCV*, 2019. 2

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014. 1, 2

[16] Jinjin Gu, Yujun Shen, and Bolei Zhou. Image processing using multi-code gan prior. In *CVPR*, 2020. 2

[17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017. 2

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 9

[19] John Hopcroft and Ravi Kannan. *Foundations of Data Science*. 2014. 11

[20] Ali Jahanian, Lucy Chai, and Phillip Isola. On the "steerability" of generative adversarial networks. In *ICLR*, 2020. 2

[21] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *ICLR*, 2018. 1, 2, 3, 4, 6, 8

[22] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019. 2, 3, 4, 6, 7, 8

[23] Line Kuhnel, Tom Fletcher, Sarang Joshi, and Stefan Sommer. Latent space non-linear statistics. *arXiv preprint arXiv:1805.07632*, 2018. 2

[24] Samuli Laine. Feature-based metrics for exploring the latent space of generative models. In *ICLR Workshop*, 2018. 2

[25] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'Aurelio Ranzato. Fader networks: Manipulating images by sliding attributes. In *NeurIPS*, 2017. 2

[26] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, 2015. 9

[27] Fangchang Ma, Ulas Ayaz, and Sertac Karaman. Invertibility of convolutional generative networks from partial measurements. In *NeurIPS*, 2018. 2, 4, 7

[28] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018. 2, 3

[29] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *ICML*, 2017. 2

[30] Guim Perarnau, Joost Van De Weijer, Bogdan Raducanu, and Jose M Álvarez. Invertible conditional gans for image editing. In *NeurIPS Workshop*, 2016. 2

[31] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 2, 3

[32] Hang Shao, Abhishek Kumar, and P Thomas Fletcher. The riemannian geometry of deep generative models. In *CVPR Workshop*, 2018. 2

[33] Yujun Shen, Ping Luo, Junjie Yan, Xiaogang Wang, and Xiaoou Tang. Faceid-gan: Learning a symmetry three-player gan for identity-preserving face synthesis. In *CVPR*, 2018. 2

[34] Yujun Shen, Bolei Zhou, Ping Luo, and Xiaoou Tang. Facefeat-gan: a two-stage approach for identity-preserving face synthesis. *arXiv preprint arXiv:1812.01288*, 2018. 2

[35] Luan Tran, Xi Yin, and Xiaoming Liu. Disentangled representation learning gan for pose-invariant face recognition. In *CVPR*, 2017. 2

[36] Paul Upchurch, Jacob Gardner, Geoff Pleiss, Robert Pless, Noah Snavely, Kavita Bala, and Kilian Weinberger. Deep feature interpolation for image content changes. In *CVPR*, 2017. 2

[37] Taihong Xiao, Jiapeng Hong, and Jinwen Ma. Elegant: Exchanging latent encodings with gan for transferring multiple face attributes. In *ECCV*, 2018. 2

[38] Ceyuan Yang, Yujun Shen, and Bolei Zhou. Semantic hierarchy emerges in deep generative representations for scene synthesis. *arXiv preprint arXiv:1911.09267*, 2019. 2

[39] Xi Yin, Xiang Yu, Kihyuk Sohn, Xiaoming Liu, and Manmohan Chandraker. Towards large-pose face frontalization in the wild. In *ICCV*, 2017. 2

[40] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *ICML*, 2019. 2

[41] Jiapeng Zhu, Deli Zhao, and Bo Zhang. Lia: Latently invertible autoencoder with adversarial learning. *arXiv preprint arXiv:1906.08090*, 2019. 2, 4, 8

[42] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, 2016. 2, 4, 7

Analyzing and Improving the Image Quality of StyleGAN

Tero Karras NVIDIA
Samuli Laine NVIDIA
Miika Aittala NVIDIA
Janne Hellsten NVIDIA
Jaakko Lehtinen NVIDIA and Aalto University
Timo Aila NVIDIA

Abstract

The style-based GAN architecture (StyleGAN) yields state-of-the-art results in data-driven unconditional generative image modeling. We expose and analyze several of its characteristic artifacts, and propose changes in both model architecture and training methods to address them. In particular, we redesign the generator normalization, revisit progressive growing, and regularize the generator to encourage good conditioning in the mapping from latent codes to images. In addition to improving image quality, this path length regularizer yields the additional benefit that the generator becomes significantly easier to invert. This makes it possible to reliably attribute a generated image to a particular network. We furthermore visualize how well the generator utilizes its output resolution, and identify a capacity problem, motivating us to train larger models for additional quality improvements. Overall, our improved model redefines the state of the art in unconditional image modeling, both in terms of existing distribution quality metrics as well as perceived image quality.

1. Introduction

The resolution and quality of images produced by generative methods, especially generative adversarial networks (GAN) [16], are improving rapidly [23, 31, 5]. The current state-of-the-art method for high-resolution image synthesis is StyleGAN [24], which has been shown to work reliably on a variety of datasets. Our work focuses on fixing its characteristic artifacts and improving the result quality further.

The distinguishing feature of StyleGAN [24] is its unconventional generator architecture. Instead of feeding the input latent code $\mathbf{z} \in \mathcal{Z}$ only to the beginning of a the network, the *mapping network* f first transforms it to an intermediate latent code $\mathbf{w} \in \mathcal{W}$. Affine transforms then produce *styles* that control the layers of the *synthesis network* g via adaptive instance normalization (AdaIN) [21, 9, 13, 8]. Additionally, stochastic variation is facilitated by providing additional random noise maps to the synthesis network. It has been demonstrated [24, 38] that this design allows the intermediate latent space $\mathcal{W}$ to be much less entangled than the input latent space $\mathcal{Z}$. In this paper, we focus all analysis solely on $\mathcal{W}$, as it is the relevant latent space from the synthesis network's point of view.

Many observers have noticed characteristic artifacts in images generated by StyleGAN [3]. We identify two causes for these artifacts, and describe changes in architecture and training methods that eliminate them. First, we investigate the origin of common blob-like artifacts, and find that the generator creates them to circumvent a design flaw in its architecture. In Section 2, we redesign the normalization used in the generator, which removes the artifacts. Second, we analyze artifacts related to progressive growing [23] that has been highly successful in stabilizing high-resolution GAN training. We propose an alternative design that achieves the same goal — training starts by focusing on low-resolution images and then progressively shifts focus to higher and higher resolutions — without changing the network topology during training. This new design also allows us to reason about the effective resolution of the generated images, which turns out to be lower than expected, motivating a capacity increase (Section 4).

Quantitative analysis of the quality of images produced using generative methods continues to be a challenging topic. Fréchet inception distance (FID) [20] measures differences in the density of two distributions in the high-dimensional feature space of an InceptionV3 classifier [39]. Precision and Recall (P&R) [36, 27] provide additional visibility by explicitly quantifying the percentage of generated images that are similar to training data and the percentage of training data that can be generated, respectively. We use these metrics to quantify the improvements.

Both FID and P&R are based on classifier networks that have recently been shown to focus on textures rather than shapes [12], and consequently, the metrics do not accurately capture all aspects of image quality. We observe that the perceptual path length (PPL) metric [24], originally introduced as a method for estimating the quality of latent space

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

interpolations, correlates with consistency and stability of shapes. Based on this, we regularize the synthesis network to favor smooth mappings (Section 3) and achieve a clear improvement in quality. To counter its computational expense, we also propose executing all regularizations less frequently, observing that this can be done without compromising effectiveness.

Finally, we find that projection of images to the latent space $\mathcal{W}$ works significantly better with the new, path-length regularized StyleGAN2 generator than with the original StyleGAN. This makes it easier to attribute a generated image to its source (Section 5).

Our implementation and trained models are available at `https://github.com/NVlabs/stylegan2`

2. Removing normalization artifacts

We begin by observing that most images generated by StyleGAN exhibit characteristic blob-shaped artifacts that resemble water droplets. As shown in Figure 1, even when the droplet may not be obvious in the final image, it is present in the intermediate feature maps of the generator.[1] The anomaly starts to appear around 64×64 resolution, is present in all feature maps, and becomes progressively stronger at higher resolutions. The existence of such a consistent artifact is puzzling, as the discriminator should be able to detect it.

We pinpoint the problem to the AdaIN operation that normalizes the mean and variance of each feature map separately, thereby potentially destroying any information found in the magnitudes of the features relative to each other. We hypothesize that the droplet artifact is a result of the generator intentionally sneaking signal strength information past instance normalization: by creating a strong, localized spike that dominates the statistics, the generator can effectively scale the signal as it likes elsewhere. Our hypothesis is supported by the finding that when the normalization step is removed from the generator, as detailed below, the droplet artifacts disappear completely.

[1] In rare cases (perhaps 0.1% of images) the droplet is missing, leading to severely corrupted images. See Appendix A for details.

2.1. Generator architecture revisited

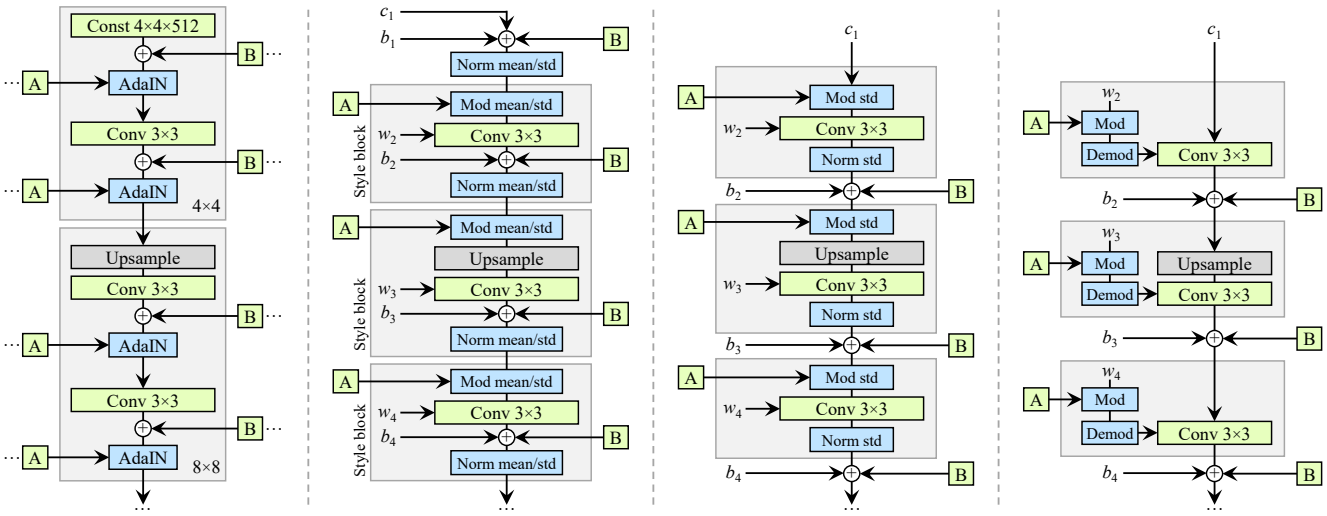
We will first revise several details of the StyleGAN generator to better facilitate our redesigned normalization. These changes have either a neutral or small positive effect on their own in terms of quality metrics.

Figure 2a shows the original StyleGAN synthesis network g [24], and in Figure 2b we expand the diagram to full detail by showing the weights and biases and breaking the AdaIN operation to its two constituent parts: normalization and modulation. This allows us to re-draw the conceptual gray boxes so that each box indicates the part of the network where one style is active (i.e., "style block"). Interestingly, the original StyleGAN applies bias and noise within the style block, causing their relative impact to be inversely proportional to the current style's magnitudes. We observe that more predictable results are obtained by moving these operations outside the style block, where they operate on normalized data. Furthermore, we notice that after this change it is sufficient for the normalization and modulation to operate on the standard deviation alone (i.e., the mean is not needed). The application of bias, noise, and normalization to the constant input can also be safely removed without observable drawbacks. This variant is shown in Figure 2c, and serves as a starting point for our redesigned normalization.

2.2. Instance normalization revisited

One of the main strengths of StyleGAN is the ability to control the generated images via *style mixing*, i.e., by feeding a different latent $\mathbf{w}$ to different layers at inference time. In practice, style modulation may amplify certain feature maps by an order of magnitude or more. For style mixing to work, we must explicitly counteract this amplification on a per-sample basis — otherwise the subsequent layers would not be able to operate on the data in a meaningful way.

If we were willing to sacrifice scale-specific controls (see video), we could simply remove the normalization, thus removing the artifacts and also improving FID slightly [27]. We will now propose a better alternative that removes the artifacts while retaining full controllability. The main idea is to base normalization on the *expected* statistics of the incoming feature maps, but without explicit forcing.



(a) StyleGAN (b) StyleGAN (detailed) (c) Revised architecture (d) Weight demodulation

Figure 2. We redesign the architecture of the StyleGAN synthesis network. (a) The original StyleGAN, where A denotes a learned affine transform from $\mathcal{W}$ that produces a style and B is a noise broadcast operation. (b) The same diagram with full detail. Here we have broken the AdaIN to explicit normalization followed by modulation, both operating on the mean and standard deviation per feature map. We have also annotated the learned weights (w), biases (b), and constant input (c), and redrawn the gray boxes so that one style is active per box. The activation function (leaky ReLU) is always applied right after adding the bias. (c) We make several changes to the original architecture that are justified in the main text. We remove some redundant operations at the beginning, move the addition of b and B to be outside active area of a style, and adjust only the standard deviation per feature map. (d) The revised architecture enables us to replace instance normalization with a "demodulation" operation, which we apply to the weights associated with each convolution layer.

Recall that a style block in Figure 2c consists of modulation, convolution, and normalization. Let us start by considering the effect of a modulation followed by a convolution. The modulation scales each input feature map of the convolution based on the incoming style, which can alternatively be implemented by scaling the convolution weights:

$$w'_{ijk} = s_i \cdot w_{ijk}, \tag{1}$$

where w and w' are the original and modulated weights, respectively, s_i is the scale corresponding to the ith input feature map, and j and k enumerate the output feature maps and spatial footprint of the convolution, respectively.

Now, the purpose of instance normalization is to essentially remove the effect of s from the statistics of the convolution's output feature maps. We observe that this goal can be achieved more directly. Let us assume that the input activations are i.i.d. random variables with unit standard deviation. After modulation and convolution, the output activations have standard deviation of

$$\sigma_j = \sqrt{\sum_{i,k} {w'_{ijk}}^2}, \tag{2}$$

i.e., the outputs are scaled by the L_2 norm of the corresponding weights. The subsequent normalization aims to restore the outputs back to unit standard deviation. Based on Equation 2, this is achieved if we scale ("demodulate") each output feature map j by $1/\sigma_j$. Alternatively, we can again bake this into the convolution weights:

$$w''_{ijk} = w'_{ijk} \Big/ \sqrt{\sum_{i,k} {w'_{ijk}}^2 + \epsilon}, \tag{3}$$

where ϵ is a small constant to avoid numerical issues.

We have now baked the entire style block to a single convolution layer whose weights are adjusted based on s using Equations 1 and 3 (Figure 2d). Compared to instance normalization, our demodulation technique is weaker because it is based on statistical assumptions about the signal instead of actual contents of the feature maps. Similar statistical analysis has been extensively used in modern network initializers [14, 19], but we are not aware of it being previously used as a replacement for data-dependent normalization. Our demodulation is also related to weight normalization [37] that performs the same calculation as a part of reparameterizing the weight tensor. Prior work has identified weight normalization as beneficial in the context of GAN training [43].

Our new design removes the characteristic artifacts (Figure 3) while retaining full controllability, as demonstrated in the accompanying video. FID remains largely unaffected (Table 1, rows A, B), but there is a notable shift from precision to recall. We argue that this is generally desirable, since recall can be traded into precision via truncation, whereas

Configuration	FFHQ, 1024×1024				LSUN Car, 512×384			
	FID ↓	Path length ↓	Precision ↑	Recall ↑	FID ↓	Path length ↓	Precision ↑	Recall ↑
A Baseline StyleGAN [24]	4.40	212.1	**0.721**	0.399	3.27	1484.5	**0.701**	0.435
B + Weight demodulation	4.39	175.4	0.702	0.425	3.04	862.4	0.685	0.488
C + Lazy regularization	4.38	158.0	0.719	0.427	2.83	981.6	0.688	0.493
D + Path length regularization	4.34	**122.5**	0.715	0.418	3.43	651.2	0.697	0.452
E + No growing, new G & D arch.	3.31	124.5	0.705	0.449	3.19	471.2	0.690	0.454
F + Large networks (StyleGAN2)	**2.84**	145.0	0.689	**0.492**	**2.32**	**415.5**	0.678	**0.514**
Config A with large networks	3.98	199.2	0.716	0.422	–	–	–	–

Table 1. Main results. For each training run, we selected the training snapshot with the lowest FID. We computed each metric 10 times with different random seeds and report their average. *Path length* corresponds to the PPL metric, computed based on path endpoints in $\mathcal{W}$ [24], without the central crop used by Karras et al. [24]. The FFHQ dataset contains 70k images, and the discriminator saw 25M images during training. For LSUN CAR the numbers were 893k and 57M. ↑ indicates that higher is better, and ↓ that lower is better.

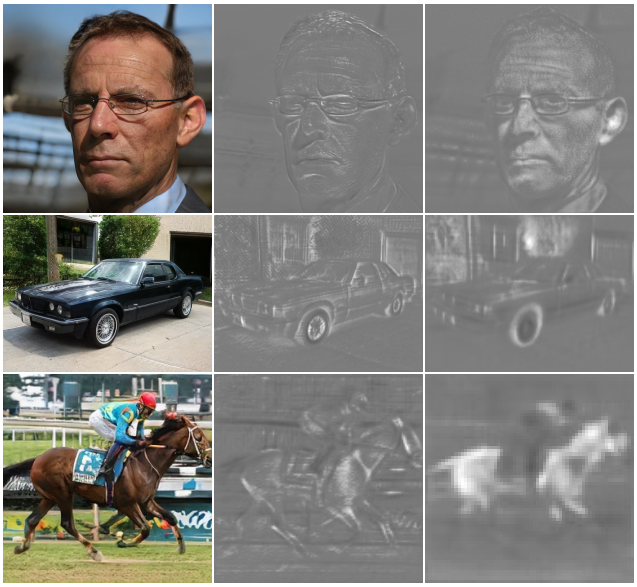
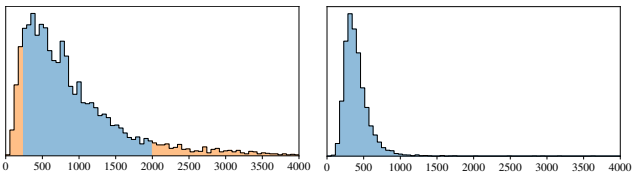


Figure 3. Replacing normalization with demodulation removes the characteristic artifacts from images and activations.

(a) Low PPL scores (b) High PPL scores

Figure 4. Connection between perceptual path length and image quality using baseline StyleGAN (config A) with LSUN CAT. (a) Random examples with low PPL ($\leq 10^{\text{th}}$ percentile). (b) Examples with high PPL ($\geq 90^{\text{th}}$ percentile). There is a clear correlation between PPL scores and semantic consistency of the images.



(a) StyleGAN (config A) (b) StyleGAN2 (config F)

Figure 5. (a) Distribution of PPL scores of individual images generated using baseline StyleGAN (config A) with LSUN CAT (FID = 8.53, PPL = 924). The percentile ranges corresponding to Figure 4 are highlighted in orange. (b) StyleGAN2 (config F) improves the PPL distribution considerably (showing a snapshot with the same FID = 8.53, PPL = 387).

the opposite is not true [27]. In practice our design can be implemented efficiently using grouped convolutions, as detailed in Appendix B. To avoid having to account for the activation function in Equation 3, we scale our activation functions so that they retain the expected signal variance.

3. Image quality and generator smoothness

While GAN metrics such as FID or Precision and Recall (P&R) successfully capture many aspects of the generator, they continue to have somewhat of a blind spot for image quality. For an example, refer to Figures 13 and 14 that contrast generators with identical FID and P&R scores but markedly different overall quality.[2]

We observe a correlation between perceived image quality and perceptual path length (PPL) [24], a metric that was originally introduced for quantifying the smoothness of the mapping from a latent space to the output image by measuring average LPIPS distances [50] between generated images under small perturbations in latent space. Again consulting Figures 13 and 14, a smaller PPL (smoother generator mapping) appears to correlate with higher overall image qual-

[2]We believe that the key to the apparent inconsistency lies in the particular choice of feature space rather than the foundations of FID or P&R. It was recently discovered that classifiers trained using ImageNet [35] tend to base their decisions much more on texture than shape [12], while humans strongly focus on shape [28]. This is relevant in our context because FID and P&R use high-level features from InceptionV3 [39] and VGG-16 [39], respectively, which were trained in this way and are thus expected to be biased towards texture detection. As such, images with, e.g., strong cat textures may appear more similar to each other than a human observer would agree, thus partially compromising density-based metrics (FID) and manifold coverage metrics (P&R).

ity, whereas other metrics are blind to the change. Figure 4 examines this correlation more closely through per-image PPL scores on LSUN CAT, computed by sampling the latent space around $\mathbf{w} \sim f(\mathbf{z})$. Low scores are indeed indicative of high-quality images, and vice versa. Figure 5a shows the corresponding histogram and reveals the long tail of the distribution. The overall PPL for the model is simply the expected value of these per-image PPL scores. We always compute PPL for the entire image, as opposed to Karras et al. [24] who use a smaller central crop.

It is not immediately obvious why a low PPL should correlate with image quality. We hypothesize that during training, as the discriminator penalizes broken images, the most direct way for the generator to improve is to effectively stretch the region of latent space that yields good images. This would lead to the low-quality images being squeezed into small latent space regions of rapid change. While this improves the average output quality in the short term, the accumulating distortions impair the training dynamics and consequently the final image quality.

Clearly, we cannot simply encourage minimal PPL since that would guide the generator toward a degenerate solution with zero recall. Instead, we will describe a new regularizer that aims for a smoother generator mapping without this drawback. As the resulting regularization term is somewhat expensive to compute, we first describe a general optimization that applies to any regularization technique.

3.1. Lazy regularization

Typically the main loss function (e.g., logistic loss [16]) and regularization terms (e.g., R_1 [30]) are written as a single expression and are thus optimized simultaneously. We observe that the regularization terms can be computed less frequently than the main loss function, thus greatly diminishing their computational cost and the overall memory usage. Table 1, row C shows that no harm is caused when R_1 regularization is performed only once every 16 minibatches, and we adopt the same strategy for our new regularizer as well. Appendix B gives implementation details.

3.2. Path length regularization

We would like to encourage that a fixed-size step in $\mathcal{W}$ results in a non-zero, fixed-magnitude change in the image. We can measure the deviation from this ideal empirically by stepping into random directions in the image space and observing the corresponding $\mathbf{w}$ gradients. These gradients should have close to an equal length regardless of $\mathbf{w}$ or the image-space direction, indicating that the mapping from the latent space to image space is well-conditioned [33].

At a single $\mathbf{w} \in \mathcal{W}$, the local metric scaling properties of the generator mapping $g(\mathbf{w}) : \mathcal{W} \mapsto \mathcal{Y}$ are captured by the Jacobian matrix $\mathbf{J}_\mathbf{w} = \partial g(\mathbf{w})/\partial \mathbf{w}$. Motivated by the desire to preserve the expected lengths of vectors regardless of the direction, we formulate our regularizer as

$$\mathbb{E}_{\mathbf{w},\mathbf{y}\sim\mathcal{N}(0,\mathbf{I})} \left(\left\| \mathbf{J}_\mathbf{w}^T \mathbf{y} \right\|_2 - a \right)^2, \tag{4}$$

where $\mathbf{y}$ are random images with normally distributed pixel intensities, and $\mathbf{w} \sim f(\mathbf{z})$, where $\mathbf{z}$ are normally distributed. We show in Appendix C that, in high dimensions, this prior is minimized when $\mathbf{J}_\mathbf{w}$ is orthogonal (up to a global scale) at any $\mathbf{w}$. An orthogonal matrix preserves lengths and introduces no squeezing along any dimension.

To avoid explicit computation of the Jacobian matrix, we use the identity $\mathbf{J}_\mathbf{w}^T \mathbf{y} = \nabla_\mathbf{w}(g(\mathbf{w}) \cdot \mathbf{y})$, which is efficiently computable using standard backpropagation [6]. The constant a is set dynamically during optimization as the long-running exponential moving average of the lengths $\|\mathbf{J}_\mathbf{w}^T \mathbf{y}\|_2$, allowing the optimization to find a suitable global scale by itself.

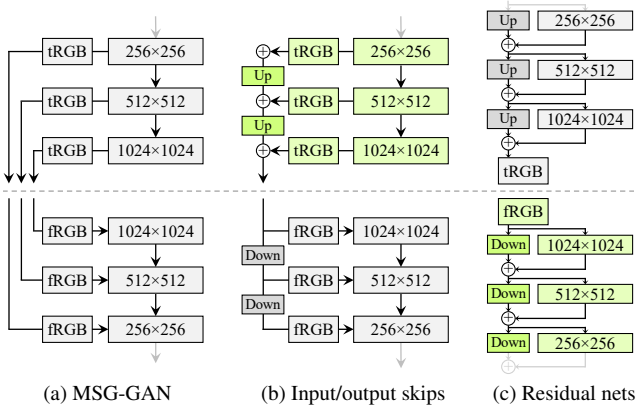
Our regularizer is closely related to the Jacobian clamping regularizer presented by Odena et al. [33]. Practical differences include that we compute the products $\mathbf{J}_\mathbf{w}^T \mathbf{y}$ analytically whereas they use finite differences for estimating $\mathbf{J}_\mathbf{w}\boldsymbol{\delta}$ with $\mathcal{Z} \ni \boldsymbol{\delta} \sim \mathcal{N}(0, \mathbf{I})$. It should be noted that spectral normalization [31] of the generator [46] only constrains the largest singular value, posing no constraints on the others and hence not necessarily leading to better conditioning. We find that enabling spectral normalization in addition to our contributions — or instead of them — invariably compromises FID, as detailed in Appendix E.

In practice, we notice that path length regularization leads to more reliable and consistently behaving models, making architecture exploration easier. We also observe that the smoother generator is significantly easier to invert (Section 5). Figure 5b shows that path length regularization clearly tightens the distribution of per-image PPL scores, without pushing the mode to zero. However, Table 1, row D points toward a tradeoff between FID and PPL in datasets that are less structured than FFHQ.

4. Progressive growing revisited

Progressive growing [23] has been very successful in stabilizing high-resolution image synthesis, but it causes its own characteristic artifacts. The key issue is that the progressively grown generator appears to have a strong location preference for details; the accompanying video shows that when features like teeth or eyes should move smoothly over the image, they may instead remain stuck in place before jumping to the next preferred location. Figure 6 shows a related artifact. We believe the problem is that in progressive growing each resolution serves momentarily as the output resolution, forcing it to generate maximal frequency details, which then leads to the trained network to have excessively high frequencies in the intermediate layers, compromising shift invariance [49]. Appendix A shows an example. These

Figure 6. Progressive growing leads to "phase" artifacts. In this example the teeth do not follow the pose but stay aligned to the camera, as indicated by the blue line.



(a) MSG-GAN (b) Input/output skips (c) Residual nets

Figure 7. Three generator (above the dashed line) and discriminator architectures. Up and Down denote bilinear up and downsampling, respectively. In residual networks these also include 1×1 convolutions to adjust the number of feature maps. tRGB and fRGB convert between RGB and high-dimensional per-pixel data. Architectures used in configs E and F are shown in green.

issues prompt us to search for an alternative formulation that would retain the benefits of progressive growing without the drawbacks.

4.1. Alternative network architectures

While StyleGAN uses simple feedforward designs in the generator (synthesis network) and discriminator, there is a vast body of work dedicated to the study of better network architectures. Skip connections [34, 22], residual networks [18, 17, 31], and hierarchical methods [7, 47, 48] have proven highly successful also in the context of generative methods. As such, we decided to re-evaluate the network design of StyleGAN and search for an architecture that produces high-quality images without progressive growing.

Figure 7a shows MSG-GAN [22], which connects the matching resolutions of the generator and discriminator using multiple skip connections. The MSG-GAN generator is modified to output a mipmap [42] instead of an image, and a similar representation is computed for each real image as well. In Figure 7b we simplify this design by upsampling and summing the contributions of RGB outputs corresponding to different resolutions. In the discriminator, we similarly provide the downsampled image to each resolution block of the discriminator. We use bilinear filtering in all up and downsampling operations. In Figure 7c we further modify the design to use residual connections.[3] This design is similar to LAPGAN [7] without the per-resolution discriminators employed by Denton et al.

FFHQ	D original		D input skips		D residual	
	FID	PPL	FID	PPL	FID	PPL
G original	4.32	265	4.18	235	3.58	269
G output skips	4.33	169	3.77	127	**3.31**	**125**
G residual	4.35	203	3.96	229	3.79	243

LSUN Car	D original		D input skips		D residual	
	FID	PPL	FID	PPL	FID	PPL
G original	3.75	905	3.23	758	3.25	802
G output skips	3.77	544	3.86	**316**	3.19	471
G residual	3.93	981	3.40	667	**2.66**	645

Table 2. Comparison of generator and discriminator architectures without progressive growing. The combination of generator with output skips and residual discriminator corresponds to configuration E in the main result table.

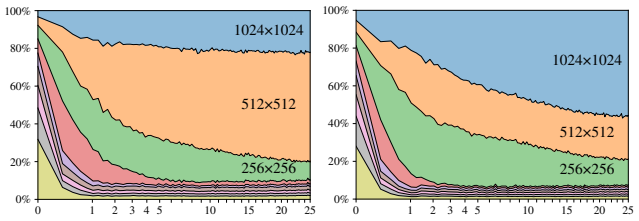
Table 2 compares three generator and three discriminator architectures: original feedforward networks as used in StyleGAN, skip connections, and residual networks, all trained without progressive growing. FID and PPL are provided for each of the 9 combinations. We can see two broad trends: skip connections in the generator drastically improve PPL in all configurations, and a residual discriminator network is clearly beneficial for FID. The latter is perhaps not surprising since the structure of discriminator resembles classifiers where residual architectures are known to be helpful. However, a residual architecture was harmful in the generator — the lone exception was FID in LSUN CAR when both networks were residual.

For the rest of the paper we use a skip generator and a residual discriminator, without progressive growing. This corresponds to configuration E in Table 1, and it significantly improves FID and PPL.

4.2. Resolution usage

The key aspect of progressive growing, which we would like to preserve, is that the generator will initially focus on low-resolution features and then slowly shift its attention to finer details. The architectures in Figure 7 make it possible for the generator to first output low resolution images that are not affected by the higher-resolution layers in a significant way, and later shift the focus to the higher-resolution

[3] In residual network architectures, the addition of two paths leads to a doubling of signal variance, which we cancel by multiplying with $1/\sqrt{2}$. This is crucial for our networks, whereas in classification resnets [18] the issue is typically hidden by batch normalization.



(a) StyleGAN-sized (config E) (b) Large networks (config F)

Figure 8. Contribution of each resolution to the output of the generator as a function of training time. The vertical axis shows a breakdown of the relative standard deviations of different resolutions, and the horizontal axis corresponds to training progress, measured in millions of training images shown to the discriminator. We can see that in the beginning the network focuses on low-resolution images and progressively shifts its focus on larger resolutions as training progresses. In (a) the generator basically outputs a 512^2 image with some minor sharpening for 1024^2, while in (b) the larger network focuses more on the high-resolution details.

layers as the training proceeds. Since this is not enforced in any way, the generator will do it only if it is beneficial. To analyze the behavior in practice, we need to quantify how strongly the generator relies on particular resolutions over the course of training.

Since the skip generator (Figure 7b) forms the image by explicitly summing RGB values from multiple resolutions, we can estimate the relative importance of the corresponding layers by measuring how much they contribute to the final image. In Figure 8a, we plot the standard deviation of the pixel values produced by each tRGB layer as a function of training time. We calculate the standard deviations over 1024 random samples of $\mathbf{w}$ and normalize the values so that they sum to 100%.

At the start of training, we can see that the new skip generator behaves similar to progressive growing — now achieved without changing the network topology. It would thus be reasonable to expect the highest resolution to dominate towards the end of the training. The plot, however, shows that this fails to happen in practice, which indicates that the generator may not be able to "fully utilize" the target resolution. To verify this, we inspected the generated images manually and noticed that they generally lack some of the pixel-level detail that is present in the training data — the images could be described as being sharpened versions of 512^2 images instead of true 1024^2 images.

This leads us to hypothesize that there is a capacity problem in our networks, which we test by doubling the number of feature maps in the highest-resolution layers of both networks.[4] This brings the behavior more in line with expectations: Figure 8b shows a significant increase in the contribution of the highest-resolution layers, and Table 1, row F shows that FID and Recall improve markedly. The last row shows that baseline StyleGAN also benefits from additional capacity, but its quality remains far below StyleGAN2.

[4] We double the number of feature maps in resolutions 64^2–1024^2 while keeping other parts of the networks unchanged. This increases the total number of trainable parameters in the generator by 22% (25M → 30M) and in the discriminator by 21% (24M → 29M).

Dataset	Resolution	StyleGAN (A)		StyleGAN2 (F)	
		FID	PPL	FID	PPL
LSUN CAR	512×384	3.27	1485	**2.32**	**416**
LSUN CAT	256×256	8.53	924	**6.93**	**439**
LSUN CHURCH	256×256	4.21	742	**3.86**	**342**
LSUN HORSE	256×256	3.83	1405	**3.43**	**338**

Table 3. Improvement in LSUN datasets measured using FID and PPL. We trained CAR for 57M images, CAT for 88M, CHURCH for 48M, and HORSE for 100M images.

Table 3 compares StyleGAN and StyleGAN2 in four LSUN categories, again showing clear improvements in FID and significant advances in PPL. It is possible that further increases in the size could provide additional benefits.

5. Projection of images to latent space

Inverting the synthesis network g is an interesting problem that has many applications. Manipulating a given image in the latent feature space requires finding a matching latent code $\mathbf{w}$ for it first. Previous research [1, 10] suggests that instead of finding a common latent code $\mathbf{w}$, the results improve if a separate $\mathbf{w}$ is chosen for each layer of the generator. The same approach was used in an early encoder implementation [32]. While extending the latent space in this fashion finds a closer match to a given image, it also enables projecting arbitrary images that should have no latent representation. Instead, we concentrate on finding latent codes in the original, unextended latent space, as these correspond to images that the generator could have produced.

Our projection method differs from previous methods in two ways. First, we add ramped-down noise to the latent code during optimization in order to explore the latent space more comprehensively. Second, we also optimize the stochastic noise inputs of the StyleGAN generator, regularizing them to ensure they do not end up carrying coherent signal. The regularization is based on enforcing the autocorrelation coefficients of the noise maps to match those of unit Gaussian noise over multiple scales. Details of our projection method can be found in Appendix D.

5.1. Attribution of generated images

Detection of manipulated or generated images is a very important task. At present, classifier-based methods can quite reliably detect generated images, regardless of their exact origin [29, 45, 40, 51, 41]. However, given the rapid pace of progress in generative methods, this may not be a lasting situation. Besides general detection of fake images, we may also consider a more limited form of the problem:

StyleGAN — generated images | StyleGAN2 — generated images | StyleGAN2 — real images

Figure 9. Example images and their projected and re-synthesized counterparts. For each configuration, top row shows the target images and bottom row shows the synthesis of the corresponding projected latent vector and noise inputs. With the baseline StyleGAN, projection often finds a reasonably close match for generated images, but especially the backgrounds differ from the originals. The images generated using StyleGAN2 can be projected almost perfectly back into generator inputs, while projected real images (from the training set) show clear differences to the originals, as expected. All tests were done using the same projection method and hyperparameters.

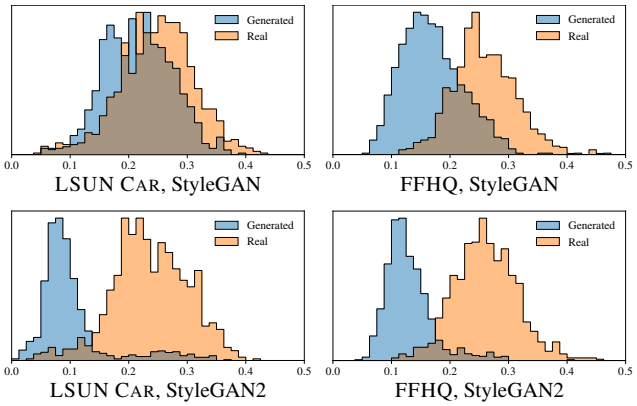


Figure 10. LPIPS distance histograms between original and projected images for generated (blue) and real images (orange). Despite the higher image quality of our improved generator, it is much easier to project the generated images into its latent space $\mathcal{W}$. The same projection method was used in all cases.

being able to attribute a fake image to its specific source [2]. With StyleGAN, this amounts to checking if there exists a $\mathbf{w} \in \mathcal{W}$ that re-synthesis the image in question.

We measure how well the projection succeeds by computing the LPIPS [50] distance between original and re-synthesized image as $D_{\text{LPIPS}}[\boldsymbol{x}, g(\tilde{g}^{-1}(\boldsymbol{x}))]$, where $\boldsymbol{x}$ is the image being analyzed and $\tilde{g}^{-1}$ denotes the approximate projection operation. Figure 10 shows histograms of these distances for LSUN CAR and FFHQ datasets using the original StyleGAN and StyleGAN2, and Figure 9 shows example projections. The images generated using StyleGAN2 can be projected into $\mathcal{W}$ so well that they can be almost unambiguously attributed to the generating network. However, with the original StyleGAN, even though it should technically be possible to find a matching latent code, it appears that the mapping from $\mathcal{W}$ to images is too complex for this to succeed reliably in practice. We find it encouraging that StyleGAN2 makes source attribution easier even though the image quality has improved significantly.

6. Conclusions and future work

We have identified and fixed several image quality issues in StyleGAN, improving the quality further and considerably advancing the state of the art in several datasets. In some cases the improvements are more clearly seen in motion, as demonstrated in the accompanying video. Appendix A includes further examples of results obtainable using our method. Despite the improved quality, StyleGAN2 makes it easier to attribute a generated image to its source.

Training performance has also improved. At 1024^2 resolution, the original StyleGAN (config A in Table 1) trains at 37 images per second on NVIDIA DGX-1 with 8 Tesla V100 GPUs, while our config E trains 40% faster at 61 img/s. Most of the speedup comes from simplified dataflow due to weight demodulation, lazy regularization, and code optimizations. StyleGAN2 (config F, larger networks) trains at 31 img/s, and is thus only slightly more expensive to train than original StyleGAN. Its total training time was 9 days for FFHQ and 13 days for LSUN CAR.

The entire project, including all exploration, consumed 132 MWh of electricity, of which 0.68 MWh went into training the final FFHQ model. In total, we used about 51 single-GPU years of computation (Volta class GPU). A more detailed discussion is available in Appendix F.

In the future, it could be fruitful to study further improvements to the path length regularization, e.g., by replacing the pixel-space L_2 distance with a data-driven feature-space metric. Considering the practical deployment of GANs, we feel that it will be important to find new ways to reduce the training data requirements. This is especially crucial in applications where it is infeasible to acquire tens of thousands of training samples, and with datasets that include a lot of intrinsic variation.

Acknowledgements We thank Ming-Yu Liu for an early review, Timo Viitanen for help with the public release, David Luebke for in-depth discussions and helpful comments, and Tero Kuosmanen for technical support with the compute infrastructure.

References

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2StyleGAN: How to embed images into the StyleGAN latent space? In *ICCV*, 2019. 7

[2] Michael Albright and Scott McCloskey. Source generator attribution via inversion. In *CVPR Workshops*, 2019. 8

[3] Carl Bergstrom and Jevin West. Which face is real? http://www.whichfaceisreal.com/learn.html, Accessed November 15, 2019. 1

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. 17

[5] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018. 1

[6] Yann N. Dauphin, Harm de Vries, and Yoshua Bengio. Equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390, 2015. 5

[7] Emily L. Denton, Soumith Chintala, Arthur Szlam, and Robert Fergus. Deep generative image models using a Laplacian pyramid of adversarial networks. *CoRR*, abs/1506.05751, 2015. 6

[8] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations. 1

[9] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *CoRR*, abs/1610.07629, 2016. 1

[10] Aviv Gabbay and Yedid Hoshen. Style generator inversion for image enhancement and animation. *CoRR*, abs/1906.11880, 2019. 7

[11] R. Ge, X. Feng, H. Pyla, K. Cameron, and W. Feng. Power measurement tutorial for the Green500 list. https://www.top500.org/green500/resources/tutorials/, Accessed March 1, 2020. 21

[12] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *CoRR*, abs/1811.12231, 2018. 1, 4

[13] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *CoRR*, abs/1705.06830, 2017. 1

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010. 3

[15] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013. 17

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *NIPS*, 2014. 1, 5, 11

[17] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of Wasserstein GANs. *CoRR*, abs/1704.00028, 2017. 6

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 6

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *CoRR*, abs/1502.01852, 2015. 3

[20] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Proc. NIPS*, pages 6626–6637, 2017. 1

[21] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR*, abs/1703.06868, 2017. 1

[22] Animesh Karnewar and Oliver Wang. MSG-GAN: multi-scale gradients for generative adversarial networks. In *Proc. CVPR*, 2020. 6

[23] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. 1, 5, 11

[24] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proc. CVPR*, 2018. 1, 2, 4, 5, 11, 13, 16, 20

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 11, 19

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105. 2012. 11

[27] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *Proc. NeurIPS*, 2019. 1, 2, 4

[28] Barbara Landau, Linda B. Smith, and Susan S. Jones. The importance of shape in early lexical learning. *Cognitive Development*, 3(3), 1988. 4

[29] Haodong Li, Han Chen, Bin Li, and Shunquan Tan. Can forensic detectors identify GAN generated images? In *Proc. Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2018. 7

[30] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? *CoRR*, abs/1801.04406, 2018. 5, 11

[31] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018. 1, 5, 6, 20

[32] Dmitry Nikitko. StyleGAN – Encoder for official TensorFlow implementation. https://github.com/Puzer/stylegan-encoder/, 2019. 7

[33] Augustus Odena, Jacob Buckman, Catherine Olsson, Tom B. Brown, Christopher Olah, Colin Raffel, and Ian Goodfellow. Is generator conditioning causally related to GAN performance? *CoRR*, abs/1802.08768, 2018. 5, 18

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proc. Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015. 6

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet large scale visual recognition challenge. In *Proc. CVPR*, 2015. 4

[36] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. *CoRR*, abs/1806.00035, 2018. 1

[37] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016. 3

[38] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of GANs for semantic face editing. *CoRR*, abs/1907.10786, 2019. 1

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 4

[40] Run Wang, Lei Ma, Felix Juefei-Xu, Xiaofei Xie, Jian Wang, and Yang Liu. FakeSpotter: A simple baseline for spotting AI-synthesized fake faces. *CoRR*, abs/1909.06122, 2019. 7

[41] Sheng-Yu Wang, Oliver Wang, Richard Zhang, Andrew Owens, and Alexei A. Efros. CNN-generated images are surprisingly easy to spot... for now. *CoRR*, abs/1912.11035, 2019. 7

[42] Lance Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, 1983. 6

[43] Sitao Xiang and Hao Li. On the effects of batch and weight normalization in generative adversarial networks. *CoRR*, abs/1704.03971, 2017. 3

[44] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2015. 11

[45] Ning Yu, Larry Davis, and Mario Fritz. Attributing fake images to GANs: Analyzing fingerprints in generated images. *CoRR*, abs/1811.08180, 2018. 7

[46] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *CoRR*, abs/1805.08318, 2018. 5

[47] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris N. Metaxas. Stack-GAN: text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017. 6

[48] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N. Metaxas. Stack-GAN++: realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1710.10916, 2017. 6

[49] Richard Zhang. Making convolutional networks shift-invariant again. In *Proc. ICML*, 2019. 5, 11

[50] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, 2018. 4, 8, 19

[51] Xu Zhang, Svebor Karaman, and Shih-Fu Chang. Detecting and simulating artifacts in GAN fake images. *CoRR*, abs/1907.06515, 2019. 7

A. Image quality

We include several large images that illustrate various aspects related to image quality. Figure 11 shows hand-picked examples illustrating the quality and diversity achievable using our method in FFHQ, while Figure 12 shows uncurated results for all datasets mentioned in the paper.

Figures 13 and 14 demonstrate cases where FID and P&R give non-intuitive results, but PPL seems to be more in line with human judgement.

We also include images relating to StyleGAN artifacts. Figure 15 shows a rare case where the blob artifact fails to appear in StyleGAN activations, leading to a seriously broken image. Figure 16 visualizes the activations inside Table 1 configurations A and F. It is evident that progressive growing leads to higher-frequency content in the intermediate layers, compromising shift invariance of the network. We hypothesize that this causes the observed uneven location preference for details when progressive growing is used.

B. Implementation details

We implemented our techniques on top of the official TensorFlow implementation of StyleGAN[5] corresponding to configuration A in Table 1. We kept most of the details unchanged, including the dimensionality of $\mathcal{Z}$ and $\mathcal{W}$ (512), mapping network architecture (8 fully connected layers, $100\times$ lower learning rate), equalized learning rate for all trainable parameters [23], leaky ReLU activation with $\alpha = 0.2$, bilinear filtering [49] in all up/downsampling layers [24], minibatch standard deviation layer at the end of the discriminator [23], exponential moving average of generator weights [23], style mixing regularization [24], nonsaturating logistic loss [16] with R_1 regularization [30], Adam optimizer [25] with the same hyperparameters ($\beta_1 = 0, \beta_2 = 0.99, \epsilon = 10^{-8}, \text{minibatch} = 32$), and training datasets [24, 44]. We performed all training runs on NVIDIA DGX-1 with 8 Tesla V100 GPUs using TensorFlow 1.14.0 and cuDNN 7.4.2.

Generator redesign In configurations B–F we replace the original StyleGAN generator with our revised architecture. In addition to the changes highlighted in Section 2, we initialize components of the constant input c_1 using $\mathcal{N}(0, 1)$ and simplify the noise broadcast operations to use a single shared scaling factor for all feature maps. Similar to Karras et al. [24], we initialize all weights using $\mathcal{N}(0, 1)$ and all biases and noise scaling factors to zero, except for the biases of the affine transformation layers, which we initialize to one. We employ weight modulation and demodulation in all convolution layers, except for the output layers (tRGB in Figure 7) where we leave out the demodulation. With 1024^2 output resolution, the generator contains a total of 18 affine transformation layers where the first one corresponds to 4^2 resolution, the next two correspond to 8^2, and so forth.

[5] https://github.com/NVlabs/stylegan

Weight demodulation Considering the practical implementation of Equations 1 and 3, it is important to note that the resulting set of weights will be different for each sample in a minibatch, which rules out direct implementation using standard convolution primitives. Instead, we choose to employ *grouped convolutions* [26] that were originally proposed as a way to reduce computational costs by dividing the input feature maps into multiple independent groups, each with their own dedicated set of weights. We implement Equations 1 and 3 by temporarily reshaping the weights and activations so that each convolution sees one sample with N groups — instead of N samples with one group. This approach is highly efficient because the reshaping operations do not actually modify the contents of the weight and activation tensors.

Lazy regularization In configurations C–F we employ lazy regularization (Section 3.1) by evaluating the regularization terms (R_1 and path length) in a separate regularization pass that we execute once every k training iterations. We share the internal state of the Adam optimizer between the main loss and the regularization terms, so that the optimizer first sees gradients from the main loss for k iterations, followed by gradients from the regularization terms for one iteration. To compensate for the fact that we now perform $k+1$ training iterations instead of k, we adjust the optimizer hyperparameters $\lambda' = c \cdot \lambda$, $\beta_1' = (\beta_1)^c$, and $\beta_2' = (\beta_2)^c$, where $c = k/(k+1)$. We also multiply the regularization term by k to balance the overall magnitude of its gradients. We use $k = 16$ for the discriminator and $k = 8$ for the generator.

Path length regularization Configurations D–F include our new path length regularizer (Section 3.2). We initialize the target scale a to zero and track it on a per-GPU basis as the exponential moving average of $\left\|\mathbf{J}_{\mathbf{w}}^T \mathbf{y}\right\|_2$ using decay coefficient $\beta_{\text{pl}} = 0.99$. We weight our regularization term by

$$\gamma_{\text{pl}} = \frac{\ln 2}{r^2(\ln r - \ln 2)}, \tag{5}$$

where r specifies the output resolution (e.g. $r = 1024$). We have found these parameter choices to work reliably across all configurations and datasets. To ensure that our regularizer interacts correctly with style mixing regularization, we compute it as an average of all individual layers of the synthesis network. Appendix C provides detailed analysis of the effects of our regularizer on the mapping between $\mathcal{W}$ and image space.

Figure 11. Four hand-picked examples illustrating the image quality and diversity achievable using StylegGAN2 (config F).

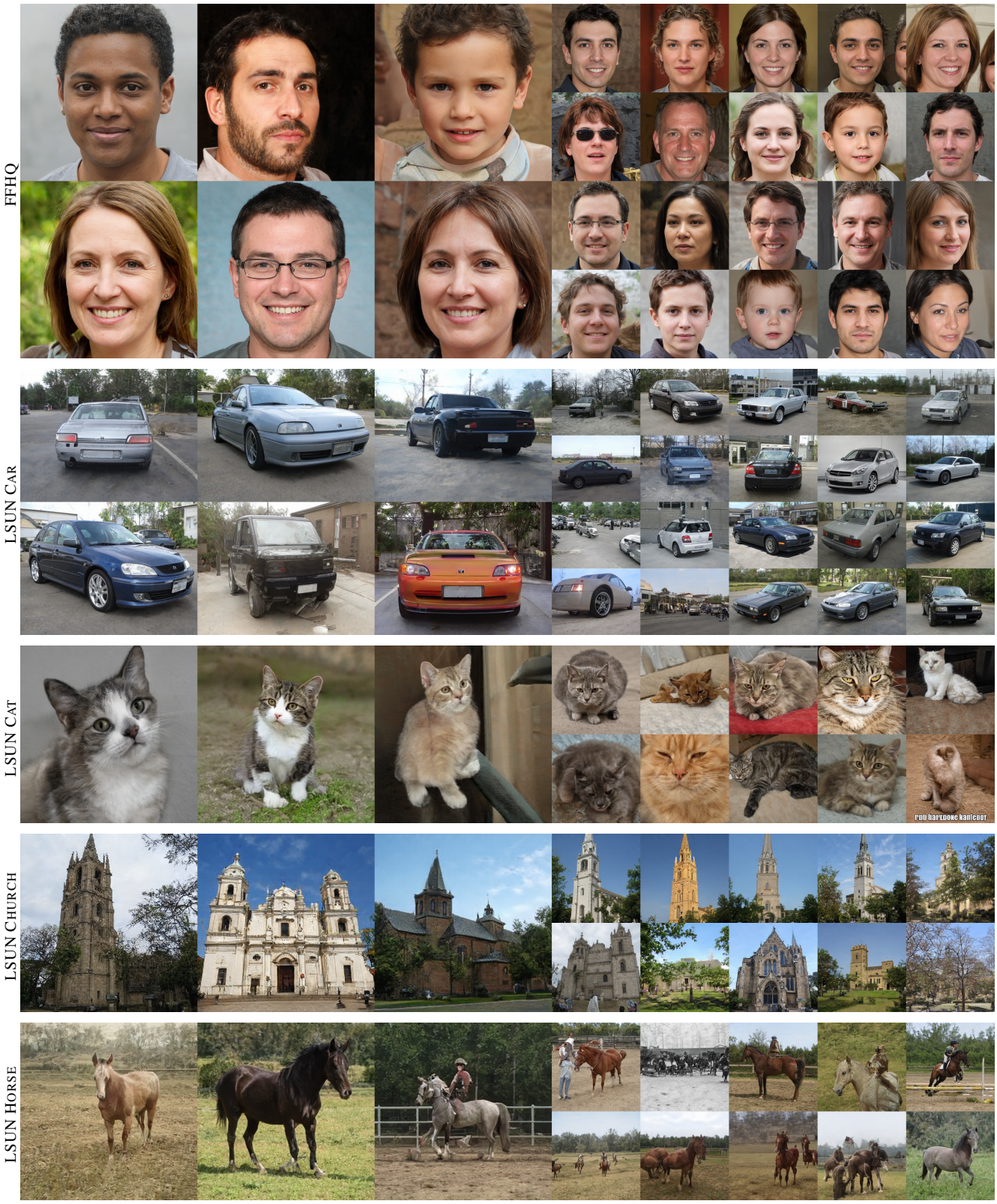
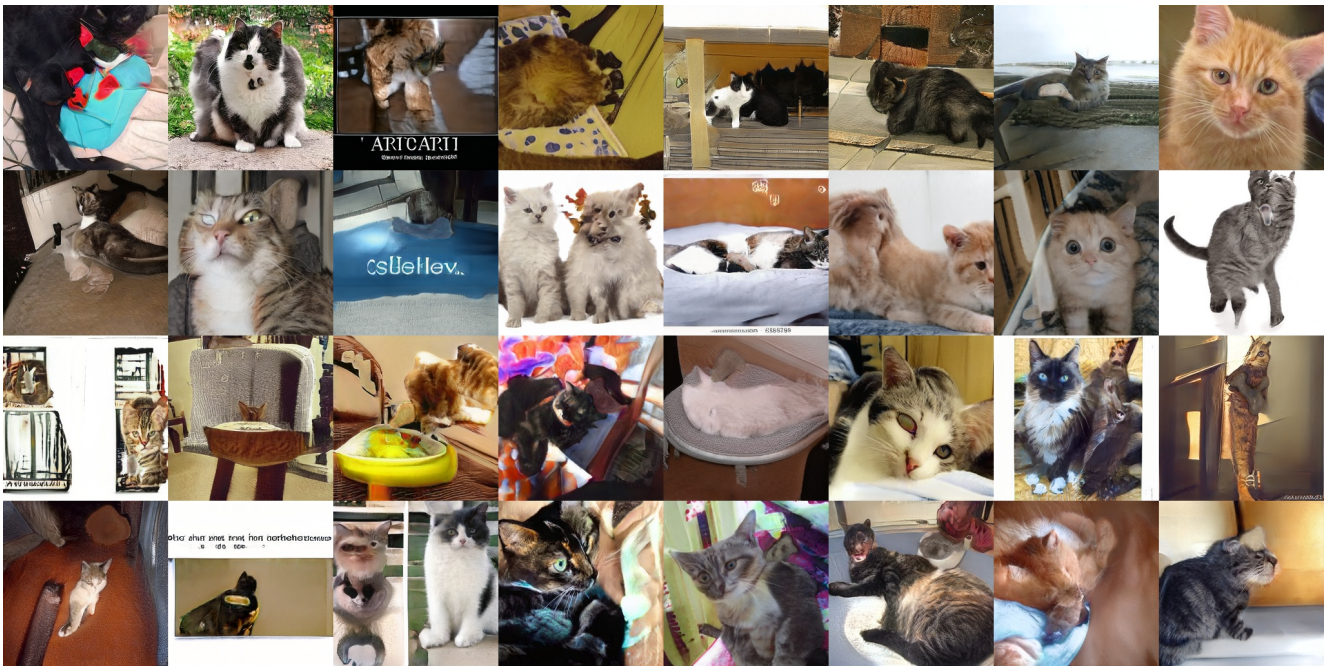


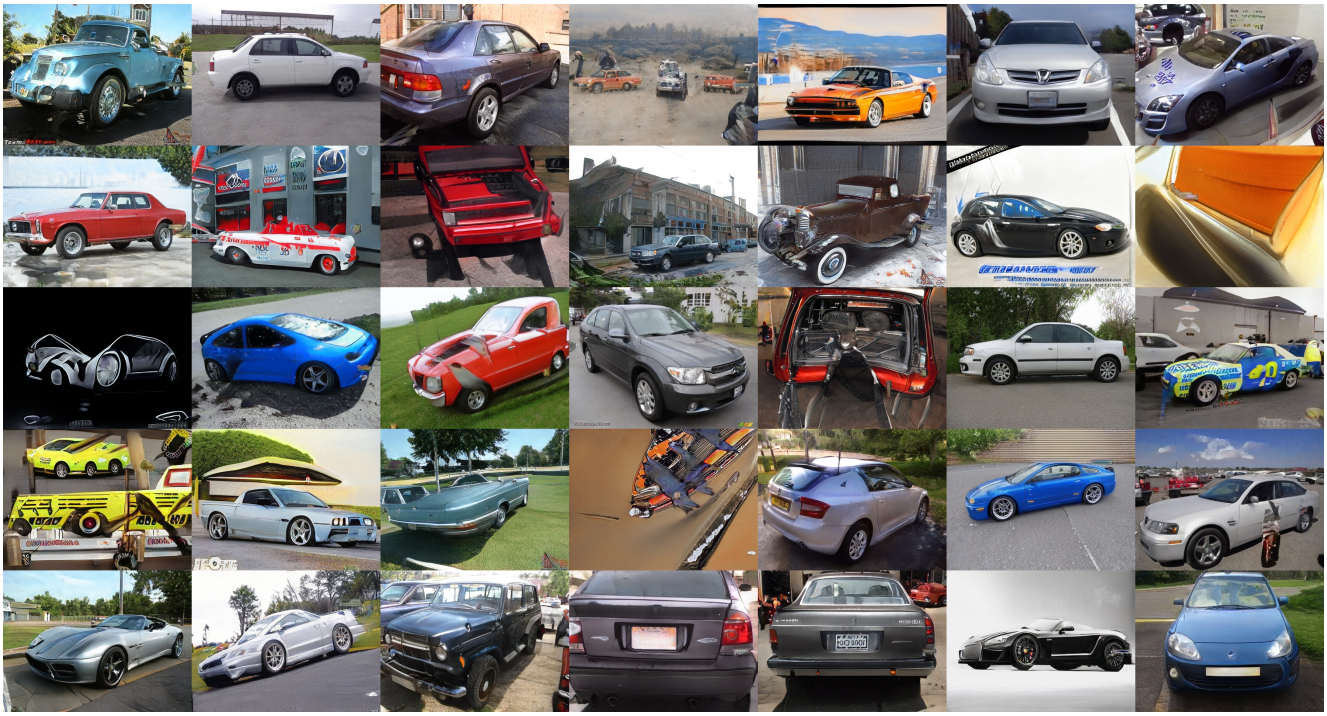
Figure 12. Uncurated results for each dataset used in Tables 1 and 3. The images correspond to random outputs produced by our generator (config F), with truncation applied at all resolutions using $\psi = 0.5$ [24].



Model 1: FID = 8.53, P = 0.64, R = 0.28, PPL = 924

Model 2: FID = 8.53, P = 0.62, R = 0.29, PPL = 387

Figure 13. Uncurated examples from two generative models trained on LSUN CAT without truncation. FID, precision, and recall are similar for models 1 and 2, even though the latter produces cat-shaped objects more often. Perceptual path length (PPL) indicates a clear preference for model 2. Model 1 corresponds to configuration A in Table 3, and model 2 is an early training snapshot of configuration F.



Model 1: FID = 3.27, P = 0.70, R = 0.44, PPL = 1485

Model 2: FID = 3.27, P = 0.67, R = 0.48, PPL = 437

Figure 14. Uncurated examples from two generative models trained on LSUN CAR without truncation. FID, precision, and recall are similar for models 1 and 2, even though the latter produces car-shaped objects more often. Perceptual path length (PPL) indicates a clear preference for model 2. Model 1 corresponds to configuration A in Table 3, and model 2 is an early training snapshot of configuration F.

Figure 15. An example of the importance of the droplet artifact in StyleGAN generator. We compare two generated images, one successful and one severely corrupted. The corresponding feature maps were normalized to the viewable dynamic range using instance normalization. For the top image, the droplet artifact starts forming in 64^2 resolution, is clearly visible in 128^2, and increasingly dominates the feature maps in higher resolutions. For the bottom image, 64^2 is qualitatively similar to the top row, but the droplet does not materialize in 128^2. Consequently, the facial features are stronger in the normalized feature map. This leads to an overshoot in 256^2, followed by multiple spurious droplets forming in subsequent resolutions. Based on our experience, it is rare that the droplet is missing from StyleGAN images, and indeed the generator fully relies on its existence.

(a) Progressive growing (config A)

(b) Without progressive growing (config F)

Figure 16. Progressive growing leads to significantly higher frequency content in the intermediate layers. This compromises shift-invariance of the network and makes it harder to localize features precisely in the higher-resolution layers.

Progressive growing In configurations A–D we use progressive growing with the same parameters as Karras et al. [24] (start at 8^2 resolution and learning rate $\lambda = 10^{-3}$, train for 600k images per resolution, fade in next resolution for 600k images, increase learning rate gradually by $3\times$). In configurations E–F we disable progressive growing and set the learning rate to a fixed value $\lambda = 2 \cdot 10^{-3}$, which we found to provide the best results. In addition, we use output skips in the generator and residual connections in the discriminator as detailed in Section 4.1.

Dataset-specific tuning Similar to Karras et al. [24], we augment the FFHQ dataset with horizontal flips to effectively increase the number of training images from 70k to 140k, and we do not perform any augmentation for the LSUN datasets. We have found that the optimal choices for the training length and R_1 regularization weight γ tend to vary considerably between datasets and configurations. We use $\gamma = 10$ for all training runs except for configuration E in Table 1, as well as LSUN CHURCH and LSUN HORSE in Table 3, where we use $\gamma = 100$. It is possible that further tuning of γ could provide additional benefits.

Performance optimizations We profiled our training runs extensively and found that — in our case — the default primitives for image filtering, up/downsampling, bias addition, and leaky ReLU had surprisingly high overheads in terms of training time and GPU memory footprint. This motivated us to optimize these operations using hand-written CUDA kernels. We implemented filtered up/downsampling as a single fused operation, and bias and activation as another one. In configuration E at 1024^2 resolution, our optimizations improved the overall training time by about 30% and memory footprint by about 20%.

C. Effects of path length regularization

The path length regularizer described in Section 3.2 is of the form:

$$\mathcal{L}_{\text{pl}} = \mathbb{E}_{\mathbf{w}}\mathbb{E}_{\mathbf{y}}\left(\left\|\mathbf{J}_{\mathbf{w}}^T\mathbf{y}\right\|_2 - a\right)^2, \tag{6}$$

where $\mathbf{y} \in \mathbb{R}^M$ is a unit normal distributed random variable in the space of generated images (of dimension $M = 3wh$, namely the RGB image dimensions), $\mathbf{J}_{\mathbf{w}} \in \mathbb{R}^{M\times L}$ is the Jacobian matrix of the generator function $g : \mathbb{R}^L \mapsto \mathbb{R}^M$ at a latent space point $\mathbf{w} \in \mathbb{R}^L$, and $a \in \mathbb{R}$ is a global value that expresses the desired scale of the gradients.

C.1. Effect on pointwise Jacobians

The value of this prior is minimized when the inner expectation over $\mathbf{y}$ is minimized at every latent space point $\mathbf{w}$ separately. In this subsection, we show that the inner expectation is (approximately) minimized when the Jacobian matrix $\mathbf{J}_{\mathbf{w}}$ is orthogonal, up to a global scaling factor. The general strategy is to use the well-known fact that, in high dimensions L, the density of a unit normal distribution is concentrated on a spherical shell of radius $\sqrt{L}$. The inner expectation is then minimized when the matrix $\mathbf{J}_{\mathbf{w}}^T$ scales the function under expectation to have its minima at this radius. This is achieved by any orthogonal matrix (with suitable global scale that is the same at every $\mathbf{w}$).

We begin by considering the inner expectation

$$\mathcal{L}_{\mathbf{w}} := \mathbb{E}_{\mathbf{y}}\left(\left\|\mathbf{J}_{\mathbf{w}}^T\mathbf{y}\right\|_2 - a\right)^2.$$

We first note that the radial symmetry of the distribution of $\mathbf{y}$, as well as of the l_2 norm, allows us to focus on diagonal matrices only. This is seen using the Singular Value Decomposition $\mathbf{J}_{\mathbf{w}}^T = \mathbf{U}\tilde{\mathbf{\Sigma}}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{L\times L}$ and $\mathbf{V} \in \mathbb{R}^{M\times M}$ are orthogonal matrices, and $\tilde{\mathbf{\Sigma}} = [\mathbf{\Sigma}\ \mathbf{0}]$ is a horizontal concatenation of a diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{L\times L}$ and a zero matrix $\mathbf{0} \in \mathbb{R}^{L\times(M-L)}$ [15]. Because rotating a unit normal random variable by an orthogonal matrix leaves the distribution unchanged, and rotating a vector leaves its norm unchanged, the expression simplifies to

$$\begin{aligned}\mathcal{L}_{\mathbf{w}} &= \mathbb{E}_{\mathbf{y}}\left(\left\|\mathbf{U}\tilde{\mathbf{\Sigma}}\mathbf{V}^T\mathbf{y}\right\|_2 - a\right)^2 \\ &= \mathbb{E}_{\mathbf{y}}\left(\left\|\tilde{\mathbf{\Sigma}}\mathbf{y}\right\|_2 - a\right)^2.\end{aligned}$$

Furthermore, the zero matrix in $\tilde{\mathbf{\Sigma}}$ drops the dimensions of $\mathbf{y}$ beyond L, effectively marginalizing its distribution over those dimensions. The marginalized distribution is again a unit normal distribution over the remaining L dimensions. We are then left to consider the minimization of the expression

$$\mathcal{L}_{\mathbf{w}} = \mathbb{E}_{\tilde{\mathbf{y}}}\left(\left\|\mathbf{\Sigma}\tilde{\mathbf{y}}\right\|_2 - a\right)^2,$$

over diagonal square matrices $\mathbf{\Sigma} \in \mathbb{R}^{L\times L}$, where $\tilde{\mathbf{y}}$ is unit normal distributed in dimension L. To summarize, all matrices $\mathbf{J}_{\mathbf{w}}^T$ that share the same singular values with $\mathbf{\Sigma}$ produce the same value for the original loss.

Next, we show that this expression is minimized when the diagonal matrix $\mathbf{\Sigma}$ has a specific identical value at every diagonal entry, i.e., it is a constant multiple of an identity matrix. We first write the expectation as an integral over the probability density of $\tilde{\mathbf{y}}$:

$$\begin{aligned}\mathcal{L}_{\mathbf{w}} &= \int \left(\left\|\mathbf{\Sigma}\tilde{\mathbf{y}}\right\|_2 - a\right)^2 p_{\tilde{\mathbf{y}}}(\tilde{\mathbf{y}})\, \mathrm{d}\tilde{\mathbf{y}} \\ &= (2\pi)^{-\frac{L}{2}} \int \left(\left\|\mathbf{\Sigma}\tilde{\mathbf{y}}\right\|_2 - a\right)^2 \exp\left(-\frac{\tilde{\mathbf{y}}^T\tilde{\mathbf{y}}}{2}\right) \mathrm{d}\tilde{\mathbf{y}}\end{aligned}$$

Observing the radially symmetric form of the density, we change into a polar coordinates $\tilde{\mathbf{y}} = r\phi$, where $r \in \mathbb{R}_+$ is the distance from origin, and $\phi \in \mathbb{S}^{L-1}$ is a unit vector, i.e., a point on the $L-1$-dimensional unit sphere. This change of variables introduces a Jacobian factor r^{L-1}:

$$\tilde{\mathcal{L}}_{\mathbf{w}} = (2\pi)^{-\frac{L}{2}} \int_{\mathbb{S}}\int_0^\infty \left(r\left\|\mathbf{\Sigma}\phi\right\|_2 - a\right)^2 r^{L-1} \exp\left(-\frac{r^2}{2}\right) \mathrm{d}r\, \mathrm{d}\phi$$

The probability density $(2\pi)^{-L/2}r^{L-1}\exp\left(-\frac{r^2}{2}\right)$ is then an L-dimensional unit normal density expressed in polar coordinates, dependent only on the radius and not on the angle. A standard argument by Taylor approximation shows that when L is high, for any ϕ the density is well approximated by density $(2\pi e/L)^{-L/2}\exp\left(-\frac{1}{2}(r-\mu)^2/\sigma^2\right)$, which is a (unnormalized) one-dimensional normal density in r, centered at $\mu = \sqrt{L}$ of standard deviation $\sigma = 1/\sqrt{2}$ [4]. In other words, the density of the L-dimensional unit normal distribution is concentrated on a shell of radius $\sqrt{L}$. Substituting this density into the integral, the loss becomes

approximately

$$\mathcal{L}_{\mathbf{w}} \approx (2\pi e/L)^{-L/2} \int_{\mathbb{S}} \int_{0}^{\infty} \left(r \left\|\mathbf{\Sigma}\phi\right\|_2 - a\right)^2 \exp\left(-\frac{\left(r-\sqrt{L}\right)^2}{2\sigma^2}\right) \mathrm{d}r \, \mathrm{d}\phi, \quad (7)$$

where the approximation becomes exact in the limit of infinite dimension L.

To minimize this loss, we set $\mathbf{\Sigma}$ such that the function $\left(r \left\|\mathbf{\Sigma}\phi\right\|_2 - a\right)^2$ obtains minimal values on the spherical shell of radius $\sqrt{L}$. This is achieved by $\mathbf{\Sigma} = \frac{a}{\sqrt{L}}\mathbf{I}$, whereby the function becomes constant in ϕ and the expression reduces to

$$\mathcal{L}_{\mathbf{w}} \approx (2\pi e/L)^{-L/2} \mathcal{A}(\mathbb{S}) a^2 L^{-1} \int_{0}^{\infty} \left(r-\sqrt{L}\right)^2 \exp\left(-\frac{\left(r-\sqrt{L}\right)^2}{2\sigma^2}\right) \mathrm{d}r,$$

where $\mathcal{A}(\mathbb{S})$ is the surface area of the unit sphere (and like the other constant factors, irrelevant for minimization). Note that the zero of the parabola $(r-\sqrt{L})^2$ coincides with the maximum of the probability density, and therefore this choice of Σ minimizes the inner integral in Eq. 7 separately for every ϕ.

In summary, we have shown that — assuming a high dimensionality L of the latent space — the value of the path length prior (Eq. 6) is minimized when all singular values of the Jacobian matrix of the generator are equal to a global constant, at every latent space point $\mathbf{w}$, i.e., they are orthogonal up to a globally constant scale.

While in theory a merely scales the values of the mapping without changing its properties and could be set to a fixed value (e.g., 1), in practice it does affect the dynamics of the training. If the imposed scale does not match the scale induced by the random initialization of the network, the training spends its critical early steps in pushing the weights towards the required overall magnitudes, rather than enforcing the actual objective of interest. This may degrade the internal state of the network weights and lead to sub-optimal performance in later training. Empirically we find that setting a fixed scale reduces the consistency of the training results across training runs and datasets. Instead, we set a dynamically based on a running average of the existing scale of the Jacobians, namely $a \approx \mathbb{E}_{\mathbf{w},\mathbf{y}}\left(\left\|\mathbf{J}_w^T\mathbf{y}\right\|_2\right)$. With this choice the prior targets the scale of the local Jacobians towards whatever global average already exists, rather than forcing a specific global average. This also eliminates the need to measure the appropriate scale of the Jacobians explicitly, as is done by Odena et al. [33] who consider a related conditioning prior.

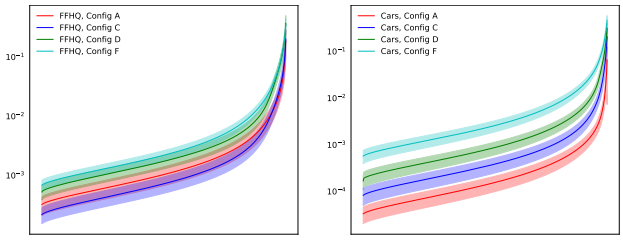


Figure 17. The mean and standard deviation of the magnitudes of sorted singular values of the Jacobian matrix evaluated at random latent space points $\mathbf{w}$, with largest eigenvalue normalized to 1. In both datasets, path length regularization (Config D) and novel architecture (Config F) exhibit better conditioning; notably, the effect is more pronounced in the Cars dataset that contains much more variability, and where path length regularization has a relatively stronger effect on the PPL metric (Table 1).

Figure 17 shows empirically measured magnitudes of singular values of the Jacobian matrix for networks trained with and without path length regularization. While orthogonality is not reached, the eigenvalues of the regularized network are closer to one another, implying better conditioning, with the strength of the effect correlated with the PPL metric (Table 1).

C.2. Effect on global properties of generator mapping

In the previous subsection, we found that the prior encourages the Jacobians of the generator mapping to be everywhere orthogonal. While Figure 17 shows that the mapping does not satisfy this constraint exactly in practice, it is instructive to consider what global properties the constraint implies for mappings that do. Without loss of generality, we assume unit global scale for the matrices to simplify the presentation.

The key property is that that a mapping $g : \mathbb{R}^L \mapsto \mathbb{R}^M$ with everywhere orthogonal Jacobians preserves the lengths of curves. To see this, let $u : [t_0, t_1] \mapsto \mathbb{R}^L$ parametrize a curve in the latent space. Mapping the curve through the generator g, we obtain a curve $\tilde{u} = g \circ u$ in the space of images. Its arc length is

$$L = \int_{t_0}^{t_1} |\tilde{u}'(t)| \, \mathrm{d}t, \quad (8)$$

where prime denotes derivative with respect to t. By chain rule, this equals

$$L = \int_{t_0}^{t_1} |J_g(u(t))u'(t)| \, \mathrm{d}t, \quad (9)$$

where $J_g \in \mathbb{R}^{L \times M}$ is the Jacobian matrix of g evaluated at $u(t)$. By our assumption, the Jacobian is orthogonal, and

consequently it leaves the 2-norm of the vector $u'(t)$ unaffected:

$$L = \int_{t_0}^{t_1} |u'(t)| \, \mathrm{d}t. \tag{10}$$

This is the length of the curve u in the latent space, prior to mapping with g. Hence, the lengths of u and $\tilde{u}$ are equal, and so g preserves the length of any curve.

In the language of differential geometry, g isometrically embeds the Euclidean latent space $\mathbb{R}^L$ into a submanifold $\mathcal{M}$ in $\mathbb{R}^M$ — e.g., the manifold of images representing faces, embedded within the space of all possible RGB images. A consequence of isometry is that straight line segments in the latent space are mapped to geodesics, or shortest paths, on the image manifold: a straight line v that connects two latent space points cannot be made any shorter, so neither can there be a shorter on-manifold image-space path between the corresponding images than $g \circ v$. For example, a geodesic on the manifold of face images is a continuous morph between two faces that incurs the minimum total amount of change (as measured by l_2 difference in RGB space) when one sums up the image difference in each step of the morph.

Isometry is not achieved in practice, as demonstrated in empirical experiments in the previous subsection. The full loss function of the training is a combination of potentially conflicting criteria, and it is not clear if a genuinely isometric mapping would be capable of expressing the image manifold of interest. Nevertheless, a pressure to make the mapping as isometric as possible has desirable consequences. In particular, it discourages unnecessary "detours": in a non-constrained generator mapping, a latent space interpolation between two similar images may pass through any number of distant images in RGB space. With regularization, the mapping is encouraged to place distant images in different regions of the latent space, so as to obtain short image paths between any two endpoints.

D. Projection method details

Given a target image $\boldsymbol{x}$, we seek to find the corresponding $\mathbf{w} \in \mathcal{W}$ and per-layer noise maps denoted $\boldsymbol{n}_i \in \mathbb{R}^{r_i \times r_i}$ where i is the layer index and r_i denotes the resolution of the ith noise map. The baseline StyleGAN generator in 1024×1024 resolution has 18 noise inputs, i.e., two for each resolution from 4×4 to 1024×1024 pixels. Our improved architecture has one fewer noise input because we do not add noise to the learned 4×4 constant (Figure 2).

Before optimization, we compute $\boldsymbol{\mu}_{\mathbf{w}} = \mathbb{E}_{\mathbf{z}} \, f(\mathbf{z})$ by running 10 000 random latent codes $\mathbf{z}$ through the mapping network f. We also approximate the scale of $\mathcal{W}$ by computing $\sigma_{\mathbf{w}}^2 = \mathbb{E}_{\mathbf{z}} \, \|f(\mathbf{z}) - \boldsymbol{\mu}_{\mathbf{w}}\|_2^2$, i.e., the average square Euclidean distance to the center.

At the beginning of optimization, we initialize $\mathbf{w} = \boldsymbol{\mu}_{\mathbf{w}}$ and $\boldsymbol{n}_i = \mathcal{N}(\mathbf{0}, \mathbf{I})$ for all i. The trainable parameters are the components of $\mathbf{w}$ as well as all components in all noise maps $\boldsymbol{n}_i$. The optimization is run for 1000 iterations using Adam optimizer [25] with default parameters. Maximum learning rate is $\lambda_{max} = 0.1$, and it is ramped up from zero linearly during the first 50 iterations and ramped down to zero using a cosine schedule during the last 250 iterations. In the first three quarters of the optimization we add Gaussian noise to $\mathbf{w}$ when evaluating the loss function as $\tilde{\mathbf{w}} = \mathbf{w} + \mathcal{N}(0, 0.05 \, \sigma_{\mathbf{w}} t^2)$, where t goes from one to zero during the first 750 iterations. This adds stochasticity to the optimization and stabilizes finding of the global optimum.

Given that we are explicitly optimizing the noise maps, we must be careful to avoid the optimization from sneaking actual signal into them. Thus we include several noise map regularization terms in our loss function, in addition to an image quality term. The image quality term is the LPIPS [50] distance between target image $\boldsymbol{x}$ and the synthesized image: $L_{image} = D_{\mathrm{LPIPS}}[\boldsymbol{x}, g(\tilde{\mathbf{w}}, \boldsymbol{n}_0, \boldsymbol{n}_1, \ldots)]$. For increased performance and stability, we downsample both images to 256×256 resolution before computing the LPIPS distance. Regularization of the noise maps is performed on

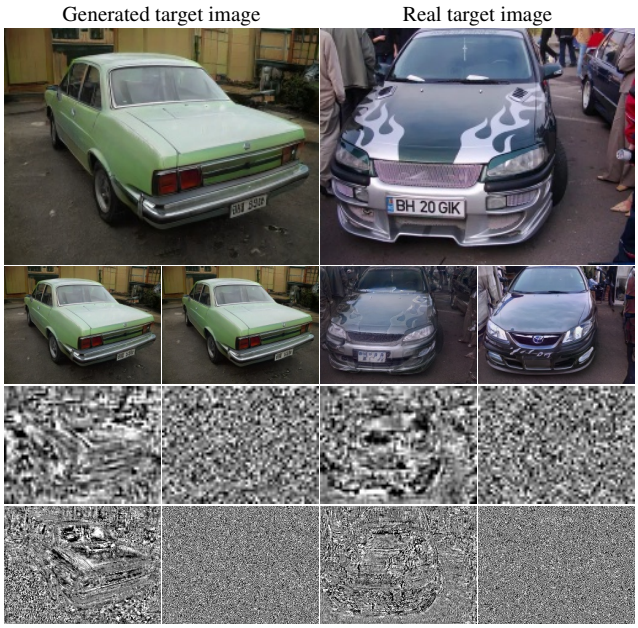


Figure 18. Effect of noise regularization in latent-space projection where we also optimize the contents of the noise inputs of the synthesis network. Top to bottom: target image, re-synthesized image, contents of two noise maps at different resolutions. When regularization is turned off in this test, we only normalize the noise maps to zero mean and unit variance, which leads the optimization to sneak signal into the noise maps. Enabling the noise regularization prevents this. The model used here corresponds to configuration F in Table 1.

	SN-G	SN-D	Demod	P.reg	FID ↓	PPL ↓	Pre. ↑	Rec. ↑
1	–	–	✓	✓	**2.83**	145.0	0.689	**0.492**
2	–	✓	✓	✓	2.98	131.4	0.700	0.469
3	✓	✓	✓	✓	3.40	**130.9**	**0.720**	0.435
4	✓	✓	–	✓	3.38	162.6	0.705	0.468
5	✓	✓	–	–	3.33	394.9	0.705	0.463
6	✓	–	–	✓	3.36	217.1	0.695	0.464
7	✓	–	–	–	3.22	394.4	0.692	0.489

Table 4. Effect of spectral normalization with FFHQ at 1024^2. The first row corresponds to StyleGAN2, i.e., config F in Table 1. In the subsequent rows, we enable spectral normalization in the generator (*SN-G*) and in the discriminator (*SN-D*). We also test the training without weight demodulation (*Demod*) and path length regularization (*P.reg*). All of these configurations are highly detrimental to FID, as well as to Recall. ↑ indicates that higher is better, and ↓ that lower is better.

multiple resolution scales. For this purpose, we form for each noise map greater than 8×8 in size a pyramid down to 8×8 resolution by averaging 2×2 pixel neighborhoods and multiplying by 2 at each step to retain the expected unit variance. These downsampled noise maps are used for regularization only and have no part in synthesis.

Let us denote the original noise maps by $\boldsymbol{n}_{i,0} = \boldsymbol{n}_i$ and the downsampled versions by $\boldsymbol{n}_{i,j>0}$. Similarly, let $r_{i,j}$ be the resolution of an original ($j = 0$) or downsampled ($j > 0$) noise map so that $r_{i,j+1} = r_{i,j}/2$. The regularization term for noise map $\boldsymbol{n}_{i,j}$ is then

$$\begin{aligned} L_{i,j} &= \left(\frac{1}{r_{i,j}^2} \cdot \sum_{x,y} \boldsymbol{n}_{i,j}(x,y) \cdot \boldsymbol{n}_{i,j}(x-1,y) \right)^2 \\ &+ \left(\frac{1}{r_{i,j}^2} \cdot \sum_{x,y} \boldsymbol{n}_{i,j}(x,y) \cdot \boldsymbol{n}_{i,j}(x,y-1) \right)^2, \end{aligned}$$

where the noise map is considered to wrap at the edges. The regularization term is thus sum of squares of the resolution-normalized autocorrelation coefficients at one pixel shifts horizontally and vertically, which should be zero for a normally distributed signal. The overall loss term is then $L_{total} = L_{image} + \alpha \sum_{i,j} L_{i,j}$. In all our tests, we have used noise regularization weight $\alpha = 10^5$. In addition, we renormalize all noise maps to zero mean and unit variance after each optimization step. Figure 18 illustrates the effect of noise regularization on the resulting noise maps.

E. Results with spectral normalization

Since spectral normalization (SN) is widely used in GANs [31], we investigated its effect on StyleGAN2. Table 4 gives the results for a variety of configurations where spectral normalization is enabled in addition to our techniques (weight demodulation, path length regularization) or instead of them.

Item	GPU years (Volta)	Electricity (MWh)
Initial exploration	20.25	58.94
Paper exploration	13.71	31.49
FFHQ config F	0.23	0.68
Other runs in paper	7.20	16.77
Backup runs left out	4.73	12.08
Video, figures, etc.	0.31	0.82
Public release	4.62	10.82
Total	51.05	131.61

Table 5. Computational effort expenditure and electricity consumption data for this project. The unit for computation is GPU-years on a single NVIDIA V100 GPU — it would have taken approximately 51 years to execute this project using a single GPU. See the text for additional details about the computation and energy consumption estimates. *Initial exploration* includes all training runs after the release of StyleGAN [24] that affected our decision to start this project. *Paper exploration* includes all training runs that were done specifically for this project, but were not intended to be used in the paper as-is. *FFHQ config F* refers to the training of the final network. This is approximately the cost of training the network for another dataset without hyperparameter tuning. *Other runs in paper* covers the training of all other networks shown in the paper. *Backup runs left out* includes the training of various networks that could potentially have been shown in the paper, but were ultimately left out to keep the exposition more focused. *Video, figures, etc.* includes computation that was spent on producing the images and graphs in the paper, as well as on the result video. *Public release* covers testing, benchmarking, and large-scale image dumps related to the public release.

Interestingly, adding spectral normalization to our generator is almost a no-op. On an implementation level, SN scales the weight tensor of each layer with a scalar value $1/\sigma(w)$. The effect of such scaling, however, is overridden by Equation 3 for the main convolutional layers as well as the affine transformation layers. Thus, the only thing that SN adds on top of weight demodulation is through its effect on the `tRGB` layers.

When we enable spectral normalization in the discriminator, FID is slightly compromised. Enabling it in the generator as well leads to significantly worse results, even though its effect is isolated to the `tRGB` layers. Leaving SN enabled, but disabling a subset of our contributions does not improve the situation. Thus we conclude that StyleGAN2 gives better results without spectral normalization.

F. Energy consumption

Computation is a core resource in any machine learning project: its availability and cost, as well as the associated energy consumption, are key factors in both choosing research directions and practical adoption. We provide a detailed breakdown for our entire project in Table 5 in terms of both GPU time and electricity consumption.

We report expended computational effort as single-GPU years (Volta class GPU). We used a varying number of

NVIDIA DGX-1s for different stages of the project, and converted each run to single-GPU equivalents by simply scaling by the number of GPUs used.

The entire project consumed approximately 131.61 megawatt hours (MWh) of electricity. We followed the Green500 power measurements guidelines [11] as follows. For each job, we logged the exact duration, number of GPUs used, and which of our two separate compute clusters the job was executed on. We then measured the actual power draw of an 8-GPU DGX-1 when it was training FFHQ config F. A separate estimate was obtained for the two clusters because they use different DGX-1 SKUs. The vast majority of our training runs used 8 GPUs, and for the rest we approximated the power draw by scaling linearly with $n/8$, where n is the number of GPUs.

Approximately half of the total energy was spent on early exploration and forming ideas. Then subsequently a quarter was spent on refining those ideas in more targeted experiments, and finally a quarter on producing this paper and preparing the public release of code, trained models, and large sets of images. Training a single FFHQ network (config F) took approximately 0.68 MWh (0.5% of the total project expenditure). This is the cost that one would pay when training the network from scratch, possibly using a different dataset. In short, vast majority of the electricity used went into shaping the ideas, testing hypotheses, and hyperparameter tuning. We did not use automated tools for finding hyperparameters or optimizing network architectures.

Latent Neural Differential Equations for Video Generation

Cade Gordon CADEGORDONML@GMAIL.COM and **Natalie Parde** PARDE@UIC.EDU
Department of Computer Science
University of Illinois at Chicago

Abstract

Generative Adversarial Networks have recently shown promise for video generation, building off of the success of image generation while also addressing a new challenge: time. Although time was analyzed in some early work, the literature has not adequately grown with temporal modeling developments. We study the effects of Neural Differential Equations to model the temporal dynamics of video generation. The paradigm of Neural Differential Equations presents many theoretical strengths including the first continuous representation of time within video generation. In order to address the effects of Neural Differential Equations, we investigate how changes in temporal models affect generated video quality. Our results give support to the usage of Neural Differential Equations as a simple replacement for older temporal generators. While keeping run times similar and decreasing parameter count, we produce a new state-of-the-art model in 64×64 pixel unconditional video generation, with an Inception Score of 15.20.

1. Introduction

Generative modeling remains an important problem within computer vision, with new developments providing a better understanding of high-dimensional data modeling and even aiding the supervised learning sphere. Good representations of distributions improve feature space visualisation, clustering, and classification. Many approaches have tackled the problem of representing a distribution, including Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), which have recently shown immense potential for image generation. As time progresses GANs become more robust, allowing for greater image size (Radford et al., 2015; Karras et al., 2018; Brock et al., 2018) and quality (Karras et al., 2019, 2020).

The success of GANs in image generation propelled them towards being the prominent methodology for video generation. However, the application of GANs to video generation has come with new challenges. Adding time to the preexisting color, width, and height dimensions has increased computational costs and complexity by an order of magnitude. Early models generated videos of a meagerly 64 by 64 pixels (Vondrick et al., 2016; Saito et al., 2017; Tulyakov et al., 2018). The addition of the new temporal component not only restricted video size, it also opened many questions regarding the best way to navigate an entirely new dimension. The first model to use GANs for video generation was VGAN (Vondrick et al., 2016), which used 3D convolutional kernels to account for time, framing it as no more than an extra feature channel blended in with color, width and height.

Treating temporal features as a separate dimensional scope allowed for the subsequent TGAN (Saito et al., 2017) to outperform VGAN in terms of Inception Score (IS) (Salimans

et al., 2016). The authors proposed two separate generative architectures: a 1D convolutional temporal generator and an image generator. Further investigation of the temporal latent space was done by MoCoGAN (Tulyakov et al., 2018), in which the authors proposed decomposing the image generator's input into a single content vector and an evolving motion vector. Experimentation has also gone into increasing frame size and network depth, with some reflections on computational mitigation (Saito et al., 2020; Clark et al., 2019; Li et al., 2020a), but little work has gone into rigorously examining time.

Our work reopens the discussion of the temporal latent space. After the revelation of separate temporal generation, researchers have stopped asking questions about the temporal generator. Works after TGAN employed Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) or Convolutional LSTM (CLSTM) (Xingjian et al., 2015) blocks. To this day, the LSTM remains and has never been fully ablated or examined with control. A similar previously unproven but accepted notion was content motion decomposition. First published in 2018 as part of MoCoGAN (Tulyakov et al., 2018), content and motion decomposition was not ablated until the publication of MoFlowGAN (Li et al., 2020a) in 2020. With very limited analysis, much of the temporal space remains an open question within these models.

While able to model temporal dynamics, recurrent models such as the LSTM and its variants only represent discrete samples. We propose to re-explore the temporal space under a continuous paradigm. Neural Ordinary Differential Equations (NODEs) (Chen et al., 2018) offer the potential for a continuous representation of the temporal dimension. Extending the paradigm of Neural Differential Equations, we propose the first continuous video generation model. Our work makes the following contributions:

- We establish the first continuous GAN for video generation.
- We experiment with multiple novel architectures for video generation.
- We analyze how changes in the temporal latent space modality affect visual fidelity through an ablation study.

2. Related Work

2.1. Generative Adversarial Networks

Two neural networks compose GANs: a Discriminator D and Generator G. The generator transforms a sampled noise vector z from a distribution p_z and maps it to an image (or in our case a video). The Discriminator functions by taking an input image or video x and mapping it to a value representing whether it believes x is sampled from the real distribution p_x or the distribution produced by the generator p_g. The two compete to minimize or maximize a loss function that may be represented generically as shown below, where ϕ is a function of the Discriminator's prediction and the truth label represented as 1 (real) or 0 (fake):

$$\max_G \min_D \mathbb{E}_{x\sim p_x}[\phi(D(x), 1)] + \mathbb{E}_{z\sim p_z}[\phi(D(G(z)), 0)]$$

ϕ is typically the identity function, cross entropy function, or hinge loss function. Loss choice has been shown to be less consequential so long as a Lipschitz constraint is met (Qin

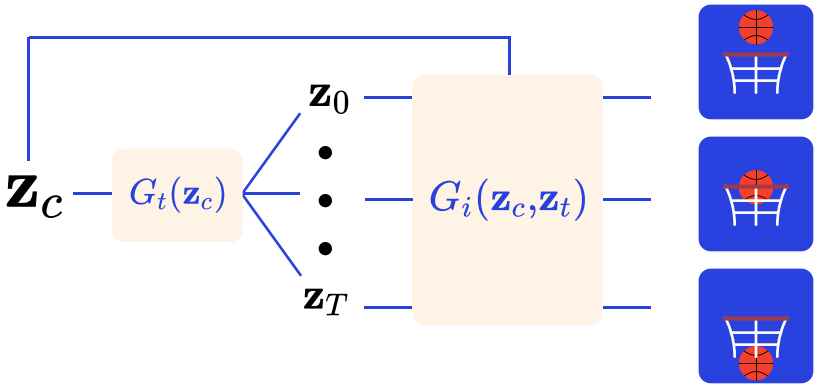


Figure 1: A latent variable $\mathbf{z}_c$ is transformed by G_t into a series of temporal vectors $\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_T$. Each temporal vector $\mathbf{z}_t$ is concatenated with $\mathbf{z}_c$ and transformed into an image. Said images are joined to compose a video.

et al., 2018). GANs are often difficult to train, and two approaches to increasing their stability during training time are through applying a form of Lipschitz constraint or multi-scale generation. WGAN (Arjovsky et al., 2017) and WGAN-GP (Gulrajani et al., 2017) showed the effectiveness of the Lipschitz regularization. SNGAN (Miyato et al., 2018) subsequently showed a refined way to enforce the constraint through spectral normalization. Progressive GAN (Karras et al., 2018) stabilized training by increasing the generated resolution over time.

2.2. Video Prediction

Video prediction conditions a model on a sample of frames, and models the subsequent frames. A common approach is the use of a recurrent architecture such as an LSTM (Ranzato et al., 2014; Srivastava et al., 2015; Finn et al., 2016; Hsieh et al., 2018; Byeon et al., 2018; Luc et al., 2020). Another common methodology is using optical flow (Liang et al., 2017; Liu et al., 2017; Hao et al., 2018; Li et al., 2018). Prior work has also explored the stochastic nature of videos (Denton and Fergus, 2018; Babaeizadeh et al., 2018; Lee et al., 2018; Franceschi et al., 2020; Villegas et al., 2019).

2.3. Video Generation

To the best of our knowledge, the first work to use a GAN to generate videos was VGAN (Vondrick et al., 2016). VGAN generated videos using spatio-temporal convolutions with 3D kernels and fractional strides, separately generating the motion and background. In order to combine the two it used a learned mask to produce the final output.

Its successor, TGAN (Saito et al., 2017), separately generated temporal and frame features. TGAN transformed a single noise vector into multiple vectors accounting for time with a temporal generator G_t, a series of 1D convolutions. The generated vectors concatenated with the starting single noise vector were then fed into an image generator

G_i. By separating temporal generation into its own process, TGAN outperformed VGAN (Salimans et al., 2016). The general form of G may be seen in Figure 1.

MoCoGAN (Tulyakov et al., 2018) continued in the line of temporal manipulation by using an LSTM to generate temporal features. The authors assumed that the temporal space was composed of a motion and content subspace. Though their work outperformed TGAN, it was not until MoFlowGAN (Li et al., 2020a), two years later, that the content and motion decomposition was fairly ablated, with results showing that it led to a positive increase in IS. It is hard to know for many of these models which features actually allowed for their success, since the discriminator and image generation architectures change and increase in complexity from one paper to the next. In light of this, a properly controlled analysis of our proposed model will fill in many of the gaps in the current literature.

Other papers focus on increasing the dimension of the video output. DVD-GAN and MoFlowGAN (Clark et al., 2019; Li et al., 2020a) produce 128x128 pixel videos. The current state-of-the-art TGANv2 (Saito et al., 2020) even boasts 192x192 pixel videos. Recently TGAN-F (Kahembwe and Ramamoorthy, 2019) further improved performance by simplifying the discriminator of TGAN.

2.4. Neural Differential Equations

NODEs (Chen et al., 2018) transformed the vision of ResNets (He et al., 2016) by giving them a continuous definition. Instead of the singular discrete additions of a neural network function $f(x)$, they proposed integrating using ordinary differential equation (ODE) solvers. The new interpretation allows for an approximate continuous temporal representation, where $\mathbf{h}$ represents the hidden state of a layer, and t represents the ordering of layers:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t) = \mathbf{h}_t + \int_t^{t+1} g(\mathbf{h}_t, t)t$$

$$\frac{d\mathbf{h}_t}{dt} = g(\mathbf{h}_t, t)$$

We use t to represent time. Works like ODE^{2}VAE (Yildiz et al., 2019) extended this to second order ODEs. Much of the work surrounding NODEs revolves around Variational Autoencoders (Chen et al., 2018; Grathwohl et al., 2018; Yildiz et al., 2019). Recently, even more differential equation families have been explored; Neural Stochastic Differential Equations (NSDEs) are a successful example (Li et al., 2020b; Tzen and Raginsky, 2019).

3. Neural Differential Equation Video GANs

There is a significant gap in the literature explaining the choice for temporal generators. To remedy this, we propose to explore it under a paradigm common to general physics: using differential equations to represent temporal dynamics. While using historical image generator functions, we will observe changes in performance metrics with different temporal generator functions. Comparisons between the families of generative functions may be visualized by Figure 2.

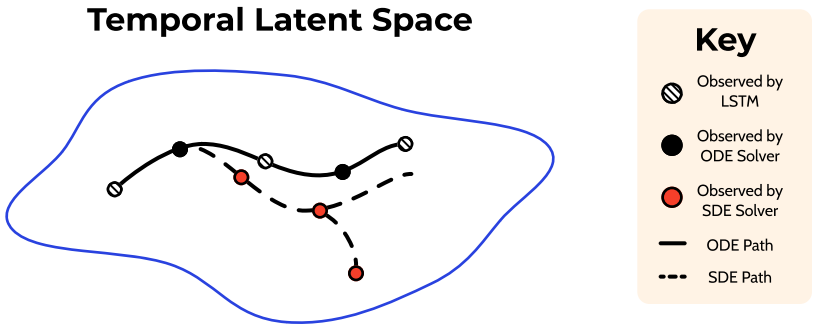


Figure 2: When compared with typical LSTMs, neural differential equations have more frequent observations, and SDEs have greater potentiality for solutions.

3.1. Ordinary Differential Equations (ODEs)

Instead of an auto-regressive LSTM or 1D Kernel, a differential equation may be used to model the evolution of a latent variable $\mathbf{z}_t$. By a learned function $f(\mathbf{z}_t)$, future $\mathbf{z}_T$ may be found by integrating:

$$\mathbf{z}_T = \mathbf{z}_0 + \int_0^T f(\mathbf{z}_t, t)dt$$

$$\dot{\mathbf{z}}_t = \frac{d\mathbf{z}_t}{dt} = f(\mathbf{z}_t, t)$$

The image generator $G_i(\mathbf{z})$ may then produce an image from $\mathbf{z}_t$. Using a differential equation the model may account for the finer nuances of traversing the latent space accounting for motion in $\mathbf{z}_{t<t+\epsilon<t+1}$. LSTMs only view sparse time steps; the model moves from $\mathbf{z}_t$ to $\mathbf{z}_{t+1}$ never accounting for a $\mathbf{z}_{t+0.5}$. NODEs allow for the intermediate $\mathbf{z}_t$ values to be traversed, which may potentially lead to better performance as this can more closely approximate a latent trajectory.

The family of NODEs also allows for higher order interpretations of the model. Our $f(\mathbf{z}_t)$ may represent higher orders than simple $\dot{\mathbf{z}}_t$, such as $\ddot{\mathbf{z}}_t$ or higher. A first-order ODE parameterizes more immediate changes during integration, whereas higher orders represent much more long-term shifts, such as concavity, in the latent variable.

3.2. Stochastic Differential Equations (SDEs)

NODEs allow for path approximations in determinate systems. Every $\mathbf{z}_t$ will produce a single $\mathbf{z}_{t+1}$, but this isn't reflective of how videos truly function. There is an inherent stochastic nature to how videos progress—actors have a branching tree of decisions and so do particles for their motion. NSDEs may be a good way to represent the random nature present in videos, offering all of the benefits of NODEs while allowing for randomness with

their added noise. Under this form, and letting $\mu(\mathbf{z}_t)$ and $\sigma(\mathbf{z}_t)$ represent drift and diffusion respectively, we find $\mathbf{z}_T$ with:

$$\mathbf{z}_T = \mathbf{z}_0 + \int_0^T \mu(\mathbf{z}_t, t)dt + \int_0^T \sigma(\mathbf{z}_t, t)dW_t$$

Each of $\mu(\mathbf{z}_t)$ and $\sigma(\mathbf{z}_t)$ are parameterized by a neural network. W_t is a Wiener process, a continuous series of values with Gaussian increments. The validity of this formulation may be exemplified by thinking about a video of a face changing expressions. If the actor starts out with a neutral face they may then produce a sad one after that. However, a smile would be equally likely. By injecting randomness either path may be explored by the model.

3.3. Benefits of Differential Equations

Differential equations allow for increased control over how paths are traversed because of their continuous properties. Because $\mathbf{z}_t$ is found by integration, there are two unique characteristics that other modalities do not possess. First, $\mathbf{z}_t$ can be integrated backwards in time allowing the discovery of $\mathbf{z}_{t-n}$. This can be thought of as what happens before the first frame. Second, if increased frame rates are desired, they can easily be accounted for. The differential equation solver will necessitate evaluations of $\mathbf{z}_{t<t+\epsilon<t+1}$. To achieve a higher frame rate, the image generator simply needs to sample some of the intermediate $\mathbf{z}_t$ evaluations. Control like this is impossible in recurrent models.

4. Experimental Protocol

The most widely used and comparable metrics for video generation are IS and Fréchet Inception Distance (FID) (Heusel et al., 2017). These are calculated by a C3D model (Tran et al., 2015) pretrained on the UCF101 dataset (Soomro et al., 2012). Their values quantify visual fidelity of the generated videos. We observe changes to these metrics as we alter G_t. We train the following models on UCF101:

- TGAN
- MoCoGAN
- TGANv2 (used for Effects of Family and Order only)

Each model will run for 100,000 epochs using the model's originally proposed hyperparameters. IS will be calculated on samples of 2,048 videos every 2,000 epochs. The epoch with the highest IS will be used to calculate the model's final statistics. Using the best performing epoch, five batches of 2,048 videos will be created. FID and IS will be calculated on each batch, and we will report the mean value and standard deviation for each metric.

4.1. Finding $f(x)$

In order to generate videos under this paradigm an appropriate neural network architecture for $f(x)$ needs to be studied. To find $f(x)$, TGAN and MoCoGAN will be trained under different G_ts. For each, $\mathbf{z}_t$ will be found by integrating $f(x)$ as a first order NODE. Ablation

will occur with the following $f(x)$s: $f : \mathbb{R}^d \to \mathbb{R}^d$ using a single learned layer with a nonlinearity; $f, g, h : \mathbb{R}^d \to \mathbb{R}^d$ where $f(x) = (g \circ h)(x)$, with g and h being also single learned layers with a nonlinearity; and the same functions as the previous setup but with g and h equalizing parameters of each model's original G_t. Testing these choices of $f(x)$ across both TGAN and MoCoGAN allows for greater evidence for or against how well each $f(x)$ generalizes to the task and architecture.

4.2. Effects of Family and Order

With an effective $f(x)$, we can ablate the multiple families and orders. TGAN, MoCoGAN, and now TGANv2 will be tested under the following motion generators: the model's original G_t, the first order ODE, the second order ODE, the third order ODE, and the SDE. For each configuration we will report IS and FID using the process specified earlier.

5. Implementation

In this section we further detail model architectures and explain minor alterations to the planned experimental design. All experiments are performed on an NVIDIA RTX 2080 TI GPU and are written in PyTorch. To promote future research and replication we make our code available to the community.[1]

5.1. Further Specification

As outlined above, said $f(x)$ designs necessitated careful measures to make them functional and fairly comparable to their original model's counterparts. In all models except TGANv2, tanh composes the final or intermediate activation function. It satisfies the continuous and Lipschitz constraint for uniqueness specified by Chen et al. (2018). Furthermore, as the final layer it permits both positive and negative resulting values stopping $\mathbf{z}_t$ from being monotonically increasing with time.

Additionally, we feed the starting noise vector through a fully connected network (FCN) aiming to equalize the number of nonlinearities. This design choice originated from comparing the number of activation functions in G_t in our experimental groups to those of the original models. For example, in TGAN the temporal generator is composed of four ReLUs and a final tanh. As it stands, $f(x)$ has only one nonlinearity. To amend the nonlinearity gap between our proposed G_t and the original models, we prepended an FCN to $f(x)$ with equal nonlinearity count to the original model's G_t. This means when integrating, individual z_ts will be in a comparatively complex space. We follow this protocol of prepending the FCN to the integration for all experiments except those with TGANv2 and MoCoGAN with equal parameterization.

5.2. Deviations from Original Plans

Instead of calculating IS every 2,000 training iterations, we calculated IS every 1,000 iterations. We also increased the number of samples used to compute an IS and FID mean

1. https://github.com/Zasder3/Latent-Neural-Differential-Equations-for-Video-Generation

and standard deviations. The original 5 measures became 10 to increase precision and to become more inline with that of TGANv2's protocol.

We also found it infeasible to compute every $f(x)$ family variation of TGANv2 under our setup due to limited computational resources. Opting to train only a first order ODE, with a batch size of 32, the model took three days to train on an A100 GPU costing over $350 using Google Cloud Platform.

6. Results

6.1. Finding $f(x)$

$f(x)$ Type	Original	Single Layer	Two Layers	Equal Parameters
TGAN	15.06±0.25	**15.20±0.26**	14.39±0.27	14.08±0.18
MoCoGAN	10.86±0.16	10.24±0.16	9.70±0.14	**12.61±0.21**

Table 1: Inception Score by type of $f(x)$ (higher is better)

$f(x)$ Type	Original	Single Layer	Two Layers	Equal Parameters
TGAN	**26512**±27	26678±21	26750±21	26751±27
MoCoGAN	27951±28	28767±61	28967±41	**26998±33**

Table 2: Fréchet Inception Distance by type of $f(x)$ (lower is better)

Looking to the IS and FID across different variations of $f(x)$ in Tables 1 and 2, we find a loose trend relating performance of the model to parameter count. Within the TGAN runs, parameters increase from left to right, but the IS decreases from left to right. In the case of MoCoGAN, equal parameters actually significantly increase performance in comparison to the single layer and two layer models as this variation forced the removal of the embedding FCN. More research needs to be done to conclude this hypothesis, but as it stands parameter count has predictive power on model performance.

The previous state-of-the-art IS for unconditional 64×64 pixel on UCF101 was held by TGAN-F, with an average IS of 13.62. Our variant which we will term TGAN-ODE outperforms this mark to become the new state-of-the-art, with an average IS of 15.20.

6.2. Effects of Family and Order

Family	Original	1st Order	2nd Order	3rd Order	SDE
TGAN	15.06±0.25	**15.20±0.26**	13.96±0.23	13.39±0.20	14.62±0.28
MoCoGAN	10.86±0.16	12.61±0.21	11.84±0.22	11.16±0.18	**14.33±0.21**
TGANv2	**26.60±0.47**[2]	21.02±0.28	-	-	-

Table 3: Inception Score by Family and Order (higher is better)

Family	Original	1st Order	2nd Order	3rd Order	SDE
TGAN	**26512±27**	26678±21	26963±26	27223±23	27252±11
MoCoGAN	27951±28	**26998±33**	27889±47	28164±25	28064±33
TGANv2	**3431±19**[2]	26017±29	-	-	-

Table 4: Fréchet Inception Distance by Family and Order (lower is better)

By nature, these experiments were more exploratory than those in §6.1; however, they produced some noteworthy anomalies and trends. Within our setup we found that performance degrades with increasing order of the ODE across both TGAN and MoCoGAN. The most surprising result is that of MoCoGAN-SDE, which outperformed the baseline and first order implementation by a large margin.

Our entries for TGANv2's original scores are sourced from the paper. Differences in data pipelines, framework, and implementation makes the direct comparison imperfect, but a good proxy for current results. Discrepancies are most noticeable in FID because we did not have access to the original dataset statistics, hence we had to calculate our own. Further work must be done to provide a thorough outcome, but as it stands a first order ODE performs adequately on a large scale, albeit not yet competitively.

7. Discussion

From our analysis on small scale models, we find promising results in the usage of ODEs and SDEs as drop-in replacements for the typical temporal generator. Within our experiments we achieve success at parameter counts equal to or lesser than baseline models. Run times also remained nearly identical to the original models. Differential equations seem to provide theoretical and quantitative boosts without harming speed. We find promising evidence of differential equation success at smaller scales, but not yet at larger ones. This opens room for future researchers to more thoroughly investigate scaling the presented technique.

In order to achieve success with these models in higher dimensions, several considerations are necessary. First, with respect to the actual $f(x)$ to be integrated, although we found a suitable function in our quite small search space, it's evident that the choice in function can have drastic effects on our results. Second, larger models come with increased VRAM necessities. A single consumer GPU will no longer be able to handle the current models at scale.

Although not strictly related to our questions of interest, during our training we additionally noted a troubling phenomenon with regard to IS score. From one calculation to the next there was extreme variation in the observed value—at one point in time the model weights may produce that of state-of-the-art, and the next nowhere close. Under older measurement frameworks (for example, only calculating IS on the training end) true model improvements may have been missed. On the other hand, this may have confounded success in models with no true advantage, but rather more luck on the final IS evaluation.

2. Value sourced from original paper instead of reproduced.

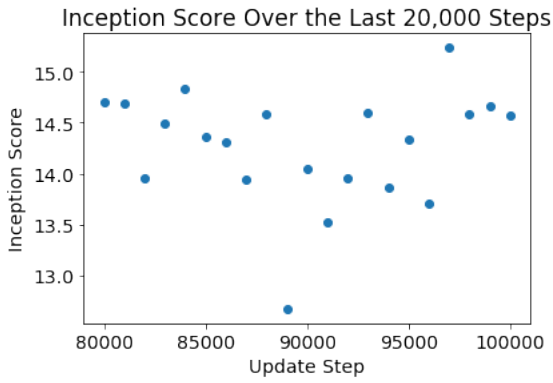


Figure 3: The last 21 calculated IS values from TGAN trained with a first order ODE. Observations range from above 15 (state-of-the-art) to below 13 showing how without careful observation great models might be missed.

8. Conclusion

Our work presents the first continuous time GAN for video generation and seeks to reopen the question of temporal generation. We find evidence supporting the use of differential equations as potential drop-in replacements for common temporal generators. We ablate under different integrated functions, differential equation orders, and families to investigate the robustness of differential equations in video generation. On the UCF101 dataset, our variant, termed TGAN-ODE, presents a new state-of-the-art on unconditional 64×64 pixel image generation.

The results of this work reopen the case for investigating the temporal generator and provide a novel direction for others to build upon. We are eager to see the outcomes of researchers' efforts as they scale the video size, use the models under different problem formulations, and increase the frame-rate to further explore this paradigm.

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 214–223, 2017.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. In *International Conference on Learning Representations*, 2018.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

Wonmin Byeon, Qin Wang, Rupesh Kumar Srivastava, and Petros Koumoutsakos. Contextvp: Fully context-aware video prediction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 753–769, 2018.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

Aidan Clark, Jeff Donahue, and Karen Simonyan. Adversarial video generation on complex datasets. *arXiv*, pages arXiv–1907, 2019.

Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. In *International Conference on Machine Learning*, pages 1174–1183, 2018.

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

Jean-Yves Franceschi, Edouard Delasalles, Mickaël Chen, Sylvain Lamprier, and Patrick Gallinari. Stochastic latent residual video prediction, 2020.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*, 2018.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

Z. Hao, X. Huang, and S. Belongie. Controllable video generation with sparse trajectories. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7854–7863, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li F Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 517–526. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7333-learning-to-decompose-and-disentangle-representations-for-video-prediction.pdf.

Emmanuel Kahembwe and Subramanian Ramamoorthy. Lower dimensional kernels for video discriminators. *arXiv preprint arXiv:1912.08860*, 2019.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

Alex X. Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction, 2018.

Wei Li, Zehuan Yuan, Xiangzhong Fang, and Changhu Wang. Moflowgan: Video generation with flow guidance. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2020a.

Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *arXiv preprint arXiv:2001.01328*, 2020b.

Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Flow-grounded spatial-temporal video prediction from still images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 9 2018.

Xiaodan Liang, Lisa Lee, Wei Dai, and Eric P. Xing. Dual motion gan for future-flow embedded video prediction. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 10 2017.

Ziwei Liu, Raymond A Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4463–4471, 2017.

Pauline Luc, Aidan Clark, Sander Dieleman, Diego de Las Casas, Yotam Doron, Albin Cassirer, and Karen Simonyan. Transformation-based adversarial video prediction on large-scale data. *arXiv preprint arXiv:2003.04035*, 2020.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

Yipeng Qin, Niloy Mitra, and Peter Wonka. How does lipschitz regularization influence gan training?, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

Masaki Saito, Eiichi Matsumoto, and Shunta Saito. Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE international conference on computer vision*, pages 2830–2839, 2017.

Masaki Saito, Shunta Saito, Masanori Koyama, and Sosuke Kobayashi. Train sparsely, generate densely: Memory-efficient unsupervised training of high-resolution temporal gan. *International Journal of Computer Vision*, May 2020. doi: 10.1007/s11263-020-01333-y. URL https://doi.org/10.1007/s11263-020-01333-y.

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

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

Ruben Villegas, Arkanath Pathak, Harini Kannan, Dumitru Erhan, Quoc V Le, and Honglak Lee. High fidelity video prediction with large stochastic recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 81–91, 2019.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Advances in neural information processing systems*, pages 613–621, 2016.

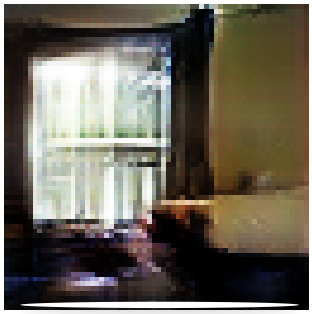
SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

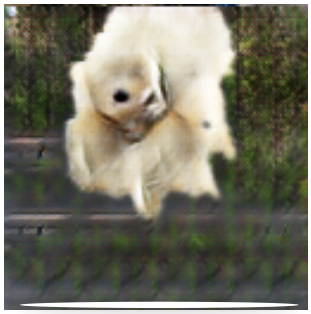
Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. Ode2vae: Deep generative second order odes with bayesian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 13412–13421. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9497-ode2vae-deep-generative-second-order-odes-with-bayesian-neural-networks.pdf.

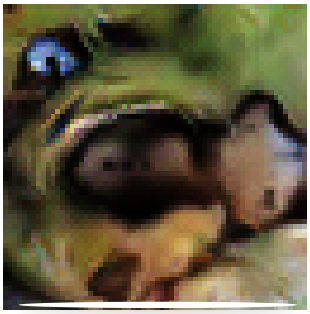
Deconvolution and Checkerboard Artifacts

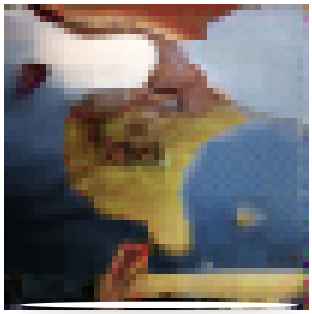
AUGUSTUS ODENA Google Brain
VINCENT DUMOULIN Université de Montréal
CHRIS OLAH Google Brain
Oct. 17 2016
Citation: Odena, et al., 2016

When we look very closely at images generated by neural networks, we often see a strange checkerboard pattern of artifacts. It's more obvious in some cases than others, but a large fraction of recent models exhibit this behavior.


Radford, et al., 2015 [1]


Salimans et al., 2016 [2]


Donahue, et al., 2016 [3]


Dumoulin, et al., 2016 [4]

Mysteriously, the checkerboard pattern tends to be most prominent in images with strong colors. What's going on? Do neural networks hate bright colors? The actual cause of these artifacts is actually remarkably simple, as is a method for avoiding them.

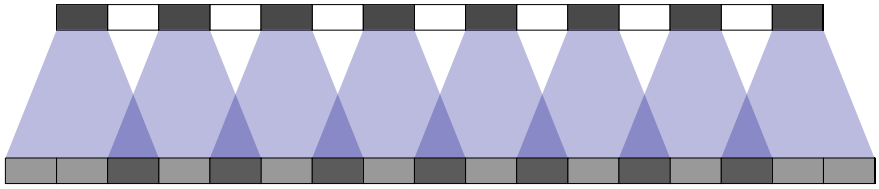
Deconvolution & Overlap

When we have neural networks generate images, we often have them build them up from low resolution, high-level descriptions. This allows the network to describe the rough image and then fill in the details.

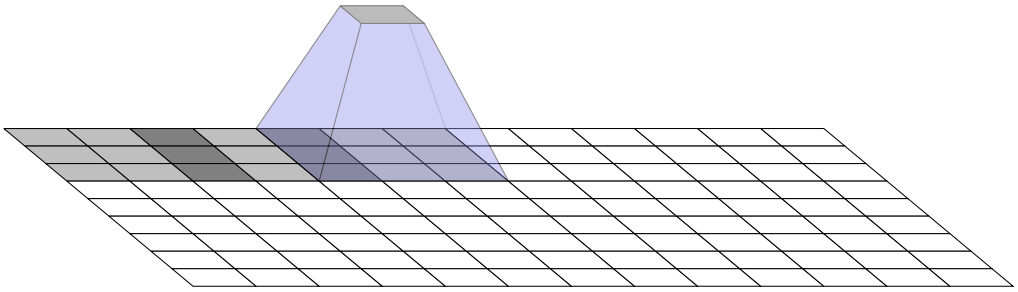
In order to do this, we need some way to go from a lower resolution image to a higher one. We generally do this with the *deconvolution* operation. Roughly, deconvolution layers allow the model to use every point in the small image to "paint" a square in the larger one.

(Deconvolution has a number of interpretations and different names, including "transposed convolution." We use the name "deconvolution" in this article for brevity. For excellent discussion of deconvolution, see [5, 6].)

Unfortunately, deconvolution can easily have "uneven overlap," putting more of the metaphorical paint in some places than others [7]. In particular, deconvolution has uneven overlap when the kernel size (the output window size) is not divisible by the stride (the spacing between points on the top). While the network could, in principle, carefully learn weights to avoid this — as we'll discuss in more detail later — in practice neural networks struggle to avoid it completely.

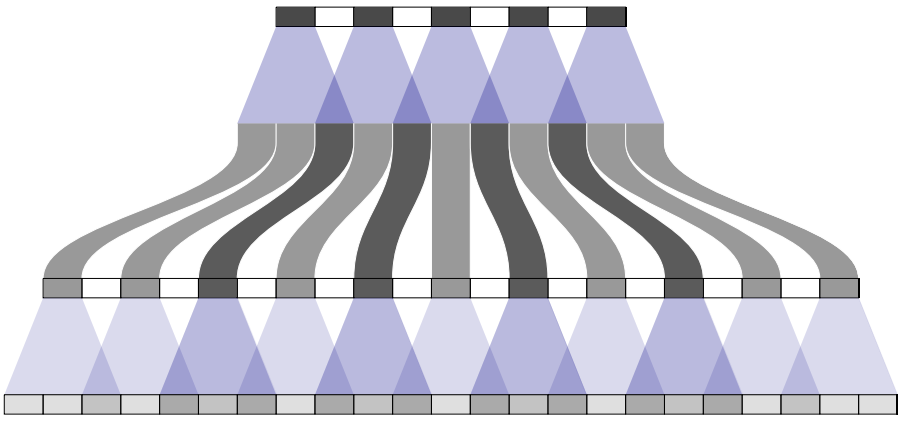


The overlap pattern also forms in two dimensions. The uneven overlaps on the two axes multiply together, creating a characteristic checkerboard-like pattern of varying magnitudes.



In fact, the uneven overlap tends to be more extreme in two dimensions! Because the two patterns are multiplied together, the unevenness gets squared. For example, in one dimension, a stride 2, size 3 deconvolution has some outputs with twice the number of inputs as others, but in two dimensions this becomes a factor of four.

Now, neural nets typically use multiple layers of deconvolution when creating images, iteratively building a larger image out of a series of lower resolution descriptions. While it's possible for these stacked deconvolutions to cancel out artifacts, they often compound, creating artifacts on a variety of scales.



Stride 1 deconvolutions — which we often see as the last layer in successful models (eg. [2]) — are quite effective at dampening artifacts. They can remove artifacts of frequencies that divide their size, and reduce others artifacts of frequency less than their size. However, artifacts can still leak through, as seen in many recent models.

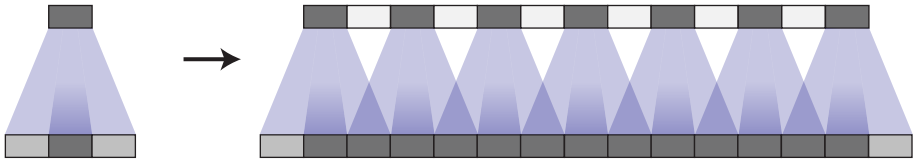
In addition to the high frequency checkerboard-like artifacts we observed above, early deconvolutions can create lower-frequency artifacts, which we'll explore in more detail later.

These artifacts tend to be most prominent when outputting unusual colors. Since neural network layers typically have a bias (a learned value added to the output) it's easy to output the average color. The further a color — like bright red — is away from the average color, the more deconvolution needs to contribute.

Overlap & Learning

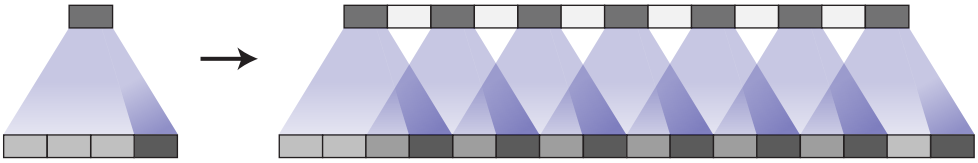
Thinking about things in terms of uneven overlap is — while a useful framing — kind of simplistic. For better or worse, our models learn weights for their deconvolutions.

In theory, our models could learn to carefully write to unevenly overlapping positions so that the output is evenly balanced.



This is a tricky balancing act to achieve, especially when one has multiple channels interacting. Avoiding artifacts significantly restricts the possible filters, sacrificing model capacity. In practice, neural networks struggle to learn to completely avoid these patterns.

In fact, not only do models with uneven overlap not learn to avoid this, but models with even overlap often learn kernels that cause similar artifacts! While it isn't their default behavior the way it is for uneven overlap, it's still very easy for even overlap deconvolution to cause artifacts.



Completely avoiding artifacts is still a significant restriction on filters, and in practice the artifacts are still present in these models, although they seem milder. (See [4], which uses stride 2 size 4 deconvolutions, as an example.)

There are probably a lot of factors at play here. For example, in the case of Generative Adversarial Networks (GANs), one issue may be the discriminator and its gradients (we'll discuss this more later). But a big part of the problem seems to be deconvolution. At best, deconvolution is fragile because it very easily represents artifact creating functions, even when the size is carefully chosen. At worst, creating artifacts is the default behavior of deconvolution.

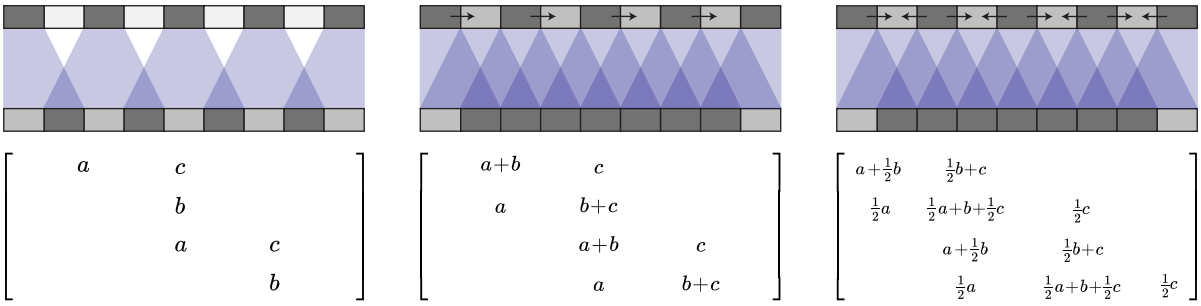
Is there a different way to upsample that is more resistant to artifacts?

Better Upsampling

To avoid these artifacts, we'd like an alternative to regular deconvolution ("transposed convolution"). Unlike deconvolution, this approach to upsampling shouldn't have artifacts as its default behavior. Ideally, it would go further, and be biased against such artifacts.

One approach is to make sure you use a kernel size that is divided by your stride, avoiding the overlap issue. This is equivalent to "sub-pixel convolution," a technique which has recently had success in image super-resolution [8]. However, while this approach helps, it is still easy for deconvolution to fall into creating artifacts.

Another approach is to separate out upsampling to a higher resolution from convolution to compute features. For example, you might resize the image (using nearest-neighbor interpolation or bilinear interpolation) and then do a convolutional layer. This seems like a natural approach, and roughly similar methods have worked well in image super-resolution (eg. [9]).



Deconvolution **NN-Resize Convolution** **Bilinear-Resize Convolution**

Both deconvolution and the different resize-convolution approaches are linear operations, and can be interpreted as matrices. This a helpful way to see the differences between them. Where deconvolution has a unique entry for each output window, resize-convolution is implicitly weight-tying in a way that discourages high frequency artifacts.

We've had our best results with nearest-neighbor interpolation, and had difficulty making bilinear resize work. This may simply mean that, for our models, the nearest-neighbor happened to work well with hyper-parameters optimized for deconvolution. It might also point at trickier issues with naively using bilinear interpolation, where it resists high-frequency image features too strongly. We don't necessarily think that either approach is the final solution to upsampling, but they do fix the checkerboard artifacts.

Code

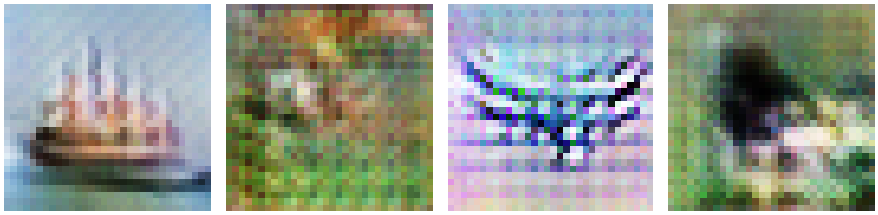
Resize-convolution layers can be easily implemented in TensorFlow using `tf.image.resize_images()`. For best results, use `tf.pad()` before doing convolution with `tf.nn.conv2d()` to avoid boundary artifacts.

Image Generation Results

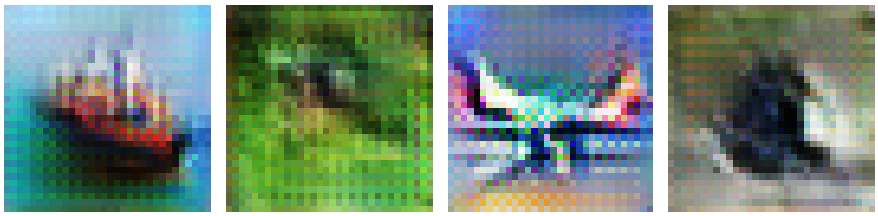
Our experience has been that nearest-neighbor resize followed by a convolution works very well, in a wide variety of contexts.

One case where we've found this approach to help is Generative Adversarial Networks. Simply switching out the standard deconvolutional layers for nearest-neighbor resize followed by convolution causes artifacts of different frequencies to disappear.

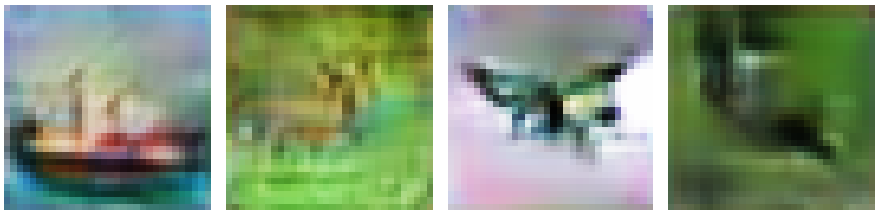
Deconv in last two layers.



Other layers use resize-convolution.
Artifacts of frequency 2 and 4.

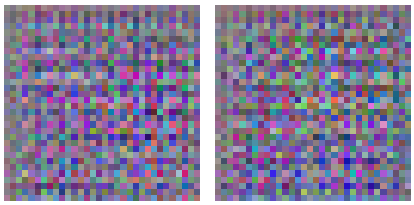


Deconv only in last layer.
Other layers use resize-convolution.
Artifacts of frequency 2.

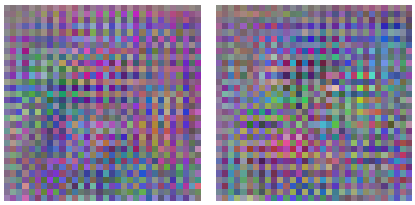


All layers use resize-convolution.
No artifacts.

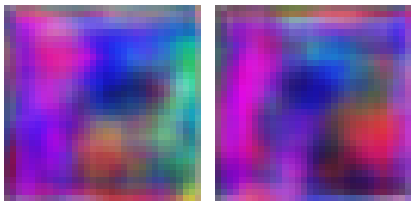
In fact, the difference in artifacts can be seen before any training occurs. If we look at the images the generator produces, initialized with random weights, we can already see the artifacts:



Deconvolution in last two layers.
Artifacts prior to any training.



Deconvolution only in last layer.
Artifacts prior to any training.



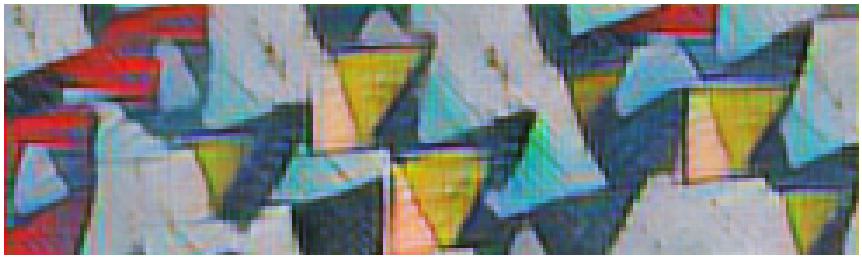
All layers use resize-convolution.
No artifacts before or after training.

This suggests that the artifacts are due to this method of generating images, rather than adversarial training. (It also suggests that we might be able to learn a lot about good generator design without the slow feedback cycle of training models.)

Another reason to believe these artifacts aren't GAN specific is that we see them in other kinds of models, and have found that they also go away when we switch to resize-convolution upsampling. For example, consider real-time artistic style transfer [10] where a neural net is trained to directly generate style-transferred images. We've found these to be vulnerable to checkerboard artifacts (especially when the cost doesn't explicitly resist them). However, switching deconvolutional layers for resize-convolution layers makes the artifacts disappear.

Using deconvolution.
Heavy checkerboard artifacts.

Using resize-convolution.
No checkerboard artifacts.



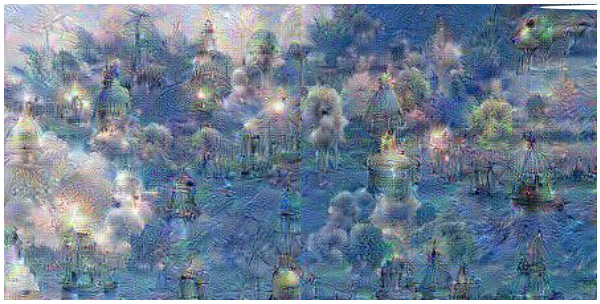
Forthcoming papers from the Google Brain team will demonstrate the benefits of this technique in more thorough experiments and state-of-the-art results. (We've chosen to present this technique separately because we felt it merited more detailed discussion, and because it cut across multiple papers.)

Artifacts in Gradients

Whenever we compute the gradients of a convolutional layer, we do deconvolution (transposed convolution) on the backward pass. This can cause checkerboard patterns in the gradient, just like when we use deconvolution to generate images.

The presence of high-frequency "noise" in image model gradients is already known in the feature visualization community, where it's a major challenge. Somehow, feature visualization methods must compensate for this noise.

For example, DeepDream [11] seems to cause destructive interference between artifacts in a number of ways, such as optimizing many features simultaneously, and optimizing at many offsets and scales. In particular, the "jitter" of optimizing at different offsets cancels out some of the checkerboard artifacts.



DeepDream only applying the neural network to a fixed position.
Severe artifacts.

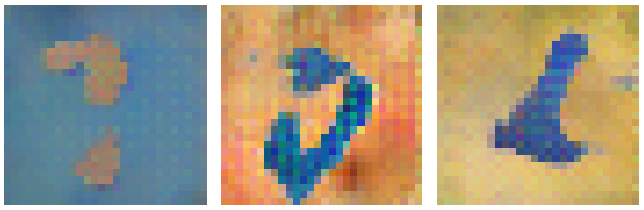
DeepDream applying the network to a different position each step.
Reduced artifacts.

(While some of the artifacts are our standard checkerboard pattern, others are a less organized high-frequency pattern. We believe these to be caused by max pooling. Max pooling was previously linked to high-frequency artifacts in [12].)

More recent work in feature visualization (eg. [13]), has explicitly recognized and compensated for these high-frequency gradient components. One wonders if better neural network architectures could make these efforts unnecessary.

Do these gradient artifacts affect GANs? If gradient artifacts can affect an image being optimized based on a neural networks gradients in feature visualization, we might also expect it to affect the family of images parameterized by the generator as they're optimized by the discriminator in GANs.

We've found that this does happen in some cases. When the generator is neither biased for or against checkerboard patterns, strided convolutions in the discriminator can cause them.



Discriminator has stride 2 convolution in first layer.
Strong frequency 2 artifacts.

Discriminator has regular convolution in first layer.
Very mild artifacts.

It's unclear what the broader implications of these gradient artifacts are. One way to think about them is that some neurons will get many times the gradient of their neighbors, basically arbitrarily. Equivalently, the network will care much more about some pixels in the input than others, for no good reason. Neither of those sounds ideal.

It seems possible that having some pixels affect the network output much more than others may exaggerate adversarial counter-examples. Because the derivative is concentrated on small number of pixels, small perturbations of those pixels may have outsized effects. We have not investigated this.

Conclusion

The standard approach of producing images with deconvolution — despite its successes! — has some conceptually simple issues that lead to artifacts in produced images. Using a natural alternative without these issues causes the artifacts to go away (Analogous arguments suggest that standard strided convolutional layers may also have issues).

This seems like an exciting opportunity to us! It suggests that there is low-hanging fruit to be found in carefully thinking through neural network architectures, even ones where we seem to have clean working solutions.

In the meantime, we've provided an easy to use solution that improves the quality of many approaches to generating images with neural networks. We look forward to seeing what people do with it, and whether it helps in domains like audio, where high frequency artifacts would be particularly problematic.

Acknowledgments

We are very grateful to Shan Carter for his wonderful improvements to the first interactive diagram, design advice, and editorial taste. We're also very grateful to David Dohan for providing us with an example of strided convolutions in discriminators causing artifacts and to Mike Tyka who originally pointed out the connection between jitter and artifacts in DeepDream to us.

Thank you also to Luke Vilnis, Jon Shlens, Luke Metz, Alex Mordvintsev, and Ben Poole for their feedback and encouragement.

This work was made possible by the support of the Google Brain team. Augustus Odena's work was done as part of the Google Brain Residency Program. Vincent Dumoulin did this while visiting the Brain Team as an intern.

Author Contributions

Augustus and Chris recognized the connection between deconvolution and artifacts. Augustus ran the GAN experiments. Vincent ran the artistic style transfer experiments. Chris ran the DeepDream experiments, created the visualizations and wrote most of the article.

References

1. **Unsupervised representation learning with deep convolutional generative adversarial networks** [PDF]
   Radford, A., Metz, L. and Chintala, S., 2015. arXiv preprint arXiv:1511.06434.
2. **Improved techniques for training gans** [PDF]
   Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X., 2016. Advances in Neural Information Processing Systems, pp. 2226—2234.
3. **Adversarial Feature Learning** [PDF]
   Donahue, J., Krahenbuhl, P. and Darrell, T., 2016. arXiv preprint arXiv:1605.09782.
4. **Adversarially Learned Inference** [PDF]
   Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O. and Courville, A., 2016. arXiv preprint arXiv:1606.00704.
5. **A guide to convolution arithmetic for deep learning** [PDF]
   Dumoulin, V. and Visin, F., 2016. arXiv preprint arXiv:1603.07285.
6. **Is the deconvolution layer the same as a convolutional layer?** [PDF]
   Shi, W., Caballero, J., Theis, L., Huszar, F., Aitken, A., Ledig, C. and Wang, Z., 2016. arXiv preprint arXiv:1609.07009.
7. **Conditional generative adversarial nets for convolutional face generation** [PDF]
   Gauthier, J., 2014. Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester, Vol 2014.
8. **Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network** [PDF]
   Shi, W., Caballero, J., Huszar, F., Totz, J., Aitken, A.P., Bishop, R., Rueckert, D. and Wang, Z., 2016. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1874—1883. DOI: 10.1109/cvpr.2016.207
9. **Image super-resolution using deep convolutional networks** [PDF]
   Dong, C., Loy, C.C., He, K. and Tang, X., 2014. arXiv preprint arXiv:1501.00092.
10. **Perceptual losses for real-time style transfer and super-resolution** [PDF]
    Johnson, J., Alahi, A. and Fei-Fei, L., 2016. arXiv preprint arXiv:1603.08155.
11. **Inceptionism: Going deeper into neural networks** [HTML]
    Mordvintsev, A., Olah, C. and Tyka, M., 2015. Google Research Blog. Retrieved June, Vol 20.
12. **Geodesics of learned representations** [PDF]
    Henaff, O.J. and Simoncelli, E.P., 2015. arXiv preprint arXiv:1511.06394.
13. **DeepDreaming with TensorFlow** [link]
    Mordvintsev, A., 2016.

Hyperspherical Variational Auto-Encoders

Article · April 2018

CITATIONS
62

READS
17,607

5 authors, including:

Nicola De Cao
University of Amsterdam
24 PUBLICATIONS **353** CITATIONS
SEE PROFILE

Thomas Kipf
University of Amsterdam
29 PUBLICATIONS **10,988** CITATIONS
SEE PROFILE

Jakub Mikolaj Tomczak
Vrije Universiteit Amsterdam
108 PUBLICATIONS **1,784** CITATIONS
SEE PROFILE

Hyperspherical Variational Auto-Encoders

Tim R. Davidson* **Luca Falorsi*** **Nicola De Cao*** **Thomas Kipf** **Jakub M. Tomczak**

University of Amsterdam

Abstract

The Variational Auto-Encoder (VAE) is one of the most used unsupervised machine learning models. But although the default choice of a Gaussian distribution for both the prior and posterior represents a mathematically convenient distribution often leading to competitive results, we show that this parameterization fails to model data with a latent hyperspherical structure. To address this issue we propose using a von Mises-Fisher (vMF) distribution instead, leading to a hyperspherical latent space. Through a series of experiments we show how such a hyperspherical VAE, or $\mathcal{S}$-VAE, is more suitable for capturing data with a hyperspherical latent structure, while outperforming a normal, $\mathcal{N}$-VAE, in low dimensions on other data types.

1 INTRODUCTION

First introduced by Kingma and Welling (2013); Rezende et al. (2014), the Variational Auto-Encoder (VAE) is an unsupervised generative model that presents a principled fashion for performing variational inference using an auto-encoding architecture. Applying the non-centered parameterization of the variational posterior (Kingma and Welling, 2014), further simplifies sampling and allows to reduce bias in calculating gradients for training. Although the default choice of a Gaussian prior is mathematically convenient, we can show through a simple example that in some cases it breaks the assumption of an *uninformative* prior leading to unstable results. Imagine a dataset on the circle $\mathcal{Z} \subset \mathcal{S}^1$, that is subsequently embedded in $\mathbb{R}^N$ using a transformation f to obtain $f : \mathcal{Z} \to \mathcal{X} \subset \mathbb{R}^N$. Given two hidden units, an autoencoder quickly discovers the latent circle, while a normal VAE becomes highly unstable. This is to be expected as a Gaussian prior is concentrated around the origin, while the KL-divergence tries to reconcile the differences between $\mathcal{S}^1$ and $\mathbb{R}^2$.

The fact that some data types like *directional data* are better explained through spherical representations is long known and well-documented (Mardia, 1975; Fisher et al., 1987), with examples spanning from protein structure, to observed wind directions. Moreover, for many modern problems such as text analysis or image classification, data is often first normalized in a preprocessing step to focus on the directional distribution. Yet, few machine learning methods explicitly account for the intrinsically spherical nature of some data in the modeling process. In this paper, we propose to use the *von Mises-Fisher* (vMF) distribution as an alternative to the Gaussian distribution. This replacement leads to a hyperspherical latent space as opposed to a hyperplanar one, where the Uniform distribution on the hypersphere is conveniently recovered as a special case of the vMF. Hence this approach allows for a truly uninformative prior, and has a clear advantage in the case of data with a hyperspherical interpretation. This was previously attempted by Hasnat et al. (2017), but crucially they do not learn the concentration parameter around the mean, κ.

In order to enable training of the concentration parameter, we extend the *reparameterization trick* for rejection sampling as recently outlined in Naesseth et al. (2017) to allow for n additional transformations. We then combine this with the rejection sampling procedure proposed by Ulrich (1984) to efficiently reparameterize the VAE [1].

We demonstrate the utility of replacing the normal distribution with the von Mises-Fisher distribution for generating latent representations by conducting a range of experiments in three distinct settings. First, we show that

*Equal contribution. . Correspondence to: Nicola De Cao <nicola.decao@student.uva.nl>.

[1] Code freely available on: `https://github.com/nicola-decao/s-vae`

our $\mathcal{S}$-VAEs outperform VAEs with the Gaussian variational posterior ($\mathcal{N}$-VAEs) in recovering a hyperspherical latent structure. Second, we conduct a thorough comparison with $\mathcal{N}$-VAEs on the MNIST dataset through an unsupervised learning task and a semi-supervised learning scenario. Finally, we show that $\mathcal{S}$-VAEs can significantly improve link prediction performance on citation network datasets in combination with a *Variational Graph Auto-Encoder* (VGAE) (Kipf and Welling, 2016).

2 VARIATIONAL AUTO-ENCODERS

2.1 FORMULATION

In the VAE setting, we have a latent variable model for data, where $\mathbf{z} \in \mathbb{R}^M$ denotes latent variables, $\mathbf{x}$ is a vector of D observed variables, and $p_\phi(\mathbf{x}, \mathbf{z})$ is a parameterized model of the joint distribution. Our objective is to optimize the log-likelihood of the data, $\log \int p_\phi(\mathbf{x}, \mathbf{z}) d\mathbf{z}$. When $p_\phi(\mathbf{x}, \mathbf{z})$ is parameterized by a neural network, marginalizing over the latent variables is generally intractable. One way of solving this issue is to maximize the Evidence Lower Bound (ELBO)

$$\log \int p_\phi(\mathbf{x}, \mathbf{z}) d\mathbf{z} \geq \mathbb{E}_{q(\mathbf{z})}[\log p_\phi(\mathbf{x}|\mathbf{z})] + - KL(q(\mathbf{z})||p(\mathbf{z})), \quad (1)$$

where $q(\mathbf{z})$ is the approximate posterior distribution, belonging to a family $\mathcal{Q}$. The bound is tight if $q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x})$, meaning $q(\mathbf{z})$ is optimized to approximate the true posterior. While in theory $q(\mathbf{z})$ should be optimized for every data point $\mathbf{x}$, to make inference more scalable to larger datasets the VAE setting introduces an inference network $q_\psi(\mathbf{z}|\mathbf{x}; \theta)$ parameterized by a neural network that outputs a probability distribution for each data point $\mathbf{x}$. The final objective is therefore to maximize

$$\mathcal{L}(\phi, \psi) = \mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{x};\theta)}[\log p_\phi(\mathbf{x}|\mathbf{z})] + - KL(q_\psi(\mathbf{z}|\mathbf{x}; \theta)||p(\mathbf{z})), \quad (2)$$

In the original VAE both the prior and the posterior are defined as normal distributions. We can further efficiently approximate the ELBO by Monte Carlo estimates, using the *reparameterization trick* (Kingma and Welling, 2013; Rezende et al., 2014). This is done by expressing a sample of $\mathbf{z} \sim q_\psi(\mathbf{z}|\mathbf{x}; \theta)$, as $\mathbf{z} = h(\theta, \varepsilon, \mathbf{x})$, where h is a reparameterization transformation and $\varepsilon \sim s(\varepsilon)$ is some noise random variable independent from θ.

2.2 THE LIMITATIONS OF A GAUSSIAN DISTRIBUTION PRIOR

Low dimensions: origin gravity In low dimensions, the Gaussian density presents a concentrated probability mass around the origin, encouraging points to cluster in the center. This is particularly problematic when the data is divided into multiple clusters. Although an ideal latent space should separate clusters for each class, the normal prior will encourage all the cluster centers towards the origin. An ideal prior would only stimulate the variance of the posterior without forcing its mean to be close to the center. A prior satisfying these properties is a uniform over the entire space. Such a uniform prior, however, is not well defined on the hyperplane.

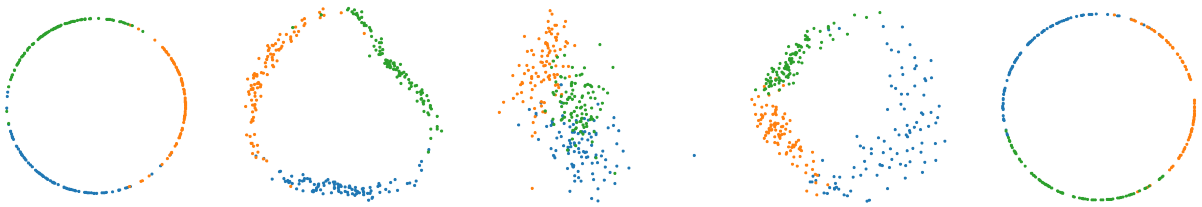
High dimensions: soap bubble effect It is a well-known phenomenon that the standard Gaussian distribution in high dimensions tends to resemble a uniform distribution on the surface of a hypersphere, with the vast majority of its mass concentrated on the hyperspherical shell. Hence it would appear interesting to compare the behavior of a Gaussian approximate posterior with an approximate posterior already naturally defined on the hypersphere. This is also motivated from a theoretical point of view, since the Gaussian definition is based on the L_2 norm that suffers from the curse of dimensionality.

2.3 BEYOND THE HYPERPLANE

Once we let go of the hyperplanar assumption, the possibility of a uniform prior on the hypersphere opens up. Mirroring our discussion in the previous subsection, such a prior would exhibit no pull towards the origin allowing clusters of data to evenly spread over the surface with no directional bias. Additionally, in higher dimensions, the cosine similarity is a more meaningful distance measure than the Euclidean norm.

Manifold mapping In general, exploring VAE models that allow a mapping to distributions in a latent space not homeomorphic to $\mathbb{R}^D$ is of fundamental interest. Consider data lying in a small M-dimensional manifold $\mathcal{M}$, embedded in a much higher dimensional space $\mathcal{X} = \mathbb{R}^N$. For most real data, this manifold will likely not be homeomorphic to $\mathbb{R}^M$. An encoder can be considered as a smooth map $enc : \mathcal{X} \to \mathcal{Z} = \mathbb{R}^D$ from the original space to $\mathcal{Z}$. The restriction of the encoder to $\mathcal{M}$, $enc|_{\mathcal{M}} : \mathcal{M} \to \mathcal{Z}$ will also be a smooth mapping. However since $\mathcal{M}$ is not homeomorphic to $\mathcal{Z}$ if $D \leq M$, then $enc|_{\mathcal{M}}$ cannot be a homeomorphism. That is, there exists no invertible and globally continuous mapping between the coordinates of $\mathcal{M}$ and the ones of $\mathcal{Z}$. Conversely if $D > M$ then $\mathcal{M}$ can be smoothly embedded in $\mathcal{Z}$ for D sufficiently big [2] , such that $enc|_{\mathcal{M}} : \mathcal{M} \to enc|_{\mathcal{M}}(\mathcal{M}) =: emb(\mathcal{M}) \subset \mathcal{Z}$ is a homeomorphism and $emb(\mathcal{M})$ denotes the embedding of

[2] By the Whitney embedding theorem any smooth real M-dimensional manifold can be smoothly embedded in $\mathbb{R}^{2M}$



(a) Original (b) Autoencoder (c) $\mathcal{N}$-VAE (d) $\mathcal{N}$-VAE, $\beta = 0.1$ (e) $\mathcal{S}$-VAE

Figure 1: Plots of the original latent space (a) and learned latent space representations in different settings, where β is a re-scaling factor for weighting the KL divergence. (Best viewed in color)

$\mathcal{M}$. Yet, since $D > M$, when taking random points in the latent space they will most likely *not* be in $emb(\mathcal{M})$ resulting in a poorly reconstructed sample.

The VAE tries to solve this problem by forcing $\mathcal{M}$ to be mapped into an approximate posterior distribution that has support in the entire $\mathcal{Z}$. Clearly, this approach is bound to fail since the two spaces have a fundamentally different structure. This can likely produce two behaviors: first, the VAE could just smooth the original embedding $emb(\mathcal{M})$ leaving most of the latent space empty, leading to bad samples. Second, if we increase the KL term the encoder will be pushed to occupy all the latent space, but this will create instability and discontinuity, affecting the convergence of the model. To validate our intuition we performed a small proof of concept experiment using $\mathcal{M} = \mathcal{S}^1$, which is visualized in Figure 1. Note that as expected the auto-encoder in Figure 1(b) mostly recovers the original latent space of Figure1(a) as there are no distributional restrictions. In Figure 1(c) we clearly observe for the $\mathcal{N}$-VAE that points collapse around the origin due to the KL, which is much less pronounced in Figure 1(d) when its contribution is scaled down. Lastly, the $\mathcal{S}$-VAE almost perfectly recovers the original circular latent space. The observed behavior confirms our intuition.

To solve this problem the best option would be to directly specify a $\mathcal{Z}$ homeomorphic to $\mathcal{M}$ and distributions on $\mathcal{M}$. However, for real data discovering the structure of $\mathcal{M}$ will often be a difficult inference task. Nevertheless, we believe this shows that investigating VAE architectures that map to posterior distributions defined on manifolds different than the Euclidean space is a topic worth to be explored. In that sense, this work represents an initial step in this research direction.

3 REPLACING GAUSSIAN WITH VON MISES-FISHER

3.1 VON MISES-FISHER DISTRIBUTION

The *von Mises-Fisher* (vMF) distribution is often described as the Normal Gaussian distribution on a hypersphere. Analogous to a Gaussian, it is parameterized by $\mu \in \mathbb{R}^m$ indicating the mean direction, and $\kappa \in \mathbb{R}_{\geq 0}$ the concentration around μ. For the special case of $\kappa = 0$, the vMF represents a Uniform distribution. The probability density function of the vMF distribution for a random unit vector $\mathbf{z} \in \mathbb{R}^m$ (or $\mathbf{z} \in \mathcal{S}^{m-1}$) is then defined as

$$q(\mathbf{z}|\mu, \kappa) = \mathcal{C}_m(\kappa) \exp\left(\kappa \mu^T \mathbf{z}\right) \tag{3}$$

$$\mathcal{C}_m(\kappa) = \frac{\kappa^{m/2-1}}{(2\pi)^{m/2} \mathcal{I}_{m/2-1}(\kappa)}, \tag{4}$$

where $||\mu||^2 = 1$, $\mathcal{C}_m(\kappa)$ is the normalizing constant, and $\mathcal{I}_v$ denotes the modified Bessel function of the first kind at order v.

3.2 KL DIVERGENCE

As previously emphasized, one of the main advantages of using the vMF distribution as an approximate posterior is that we are able to place a uniform prior on the latent space. The KL divergence term $KL(\text{vMF}(\mu, \kappa)||U(S^{m-1}))$ to be optimized is:

$$\kappa \frac{\mathcal{I}_{m/2}(k)}{\mathcal{I}_{m/2-1}(k)} + \log \mathcal{C}_m(\kappa) - \log \left(\frac{2(\pi^{m/2})}{\Gamma(m/2)} \right)^{-1}, \tag{5}$$

see Appendix B for complete derivation. Notice that since the KL term does not depend on μ, this is only optimized in the reconstruction term. The above expression cannot be handled by automatic differentiation packages because of the modified Bessel function in $\mathcal{C}_m(\kappa)$. Thus, to optimize this term we derive the gradient with respect to the

Algorithm 1 vMF sampling

Input: dimension m, mean μ, concentration κ
sample $\mathbf{v} \sim U(\mathcal{S}^{m-2})$
sample $\omega \sim g(\omega|\kappa, m) \propto \exp(\omega\kappa)(1-\omega^2)^{\frac{1}{2}(m-3)}$ {acceptance-rejection sampling}
$\mathbf{z}' \leftarrow (\omega; (\sqrt{1-\omega^2})\mathbf{v}^\top)^\top$
$U \leftarrow Householder(\mathbf{e}_1, \mu)$ {Householder transform}
Return: $U\mathbf{z}'$

concentration parameter $\nabla_\kappa KL(\text{vMF}(\mu,\kappa)||U(S^{m-1}))$:

$$\frac{1}{2}k\left(\frac{\mathcal{I}_{m/2+1}(k)}{\mathcal{I}_{m/2-1}(k)}+ - \frac{\mathcal{I}_{m/2}(k)\left(\mathcal{I}_{m/2-2}(k)+\mathcal{I}_{m/2}(k)\right)}{\mathcal{I}_{m/2-1}(k)^2}+1\right), \quad (6)$$

where the modified Bessel functions can be computed without numerical instabilities using the exponentially scaled modified Bessel function.

3.3 SAMPLING PROCEDURE

To sample from the vMF we follow the procedure of Ulrich (1984), outlined in Algorithm 1. We first sample from a vMF $q(\mathbf{z}|\mathbf{e}_1, \kappa)$ with modal vector $\mathbf{e}_1 = (1, 0, \cdots, 0)$. Since the vMF density is uniform in all the $m-2$ dimensional sub-hyperspheres $\{\mathbf{x} \in \mathcal{S}^{m-1}|\mathbf{e}_1^\top\mathbf{x} = \omega\}$, the sampling technique reduces to sampling the value ω from the univariate density $g(\omega|\kappa, m) \propto \exp(\kappa\omega)(1-\omega^2)^{(m-3)/2}, \quad \omega \in [-1, 1]$, using an acceptance-rejection scheme. After getting a sample from $q(\mathbf{z}|\mathbf{e}_1, \kappa)$ an orthogonal transformation $U(\mu)$ is applied such that the transformed sample is distributed according to $q(\mathbf{z}|\mu, \kappa)$. This can be achieved using a Householder reflection such that $U(\mu)\mathbf{e}_1 = \mu$. A more in-depth explanation of the sampling technique can be found in Appendix A.

It is worth noting that the sampling technique does not suffer from the curse of dimensionality, as the acceptance-rejection procedure is only applied to a univariate distribution. Moreover in the case of $\mathcal{S}^2$, the density $g(\omega|\kappa, 3)$ reduces to $g(\omega|\kappa, 3) \propto \exp(k\omega)\mathbb{1}_{[-1,+1]}(\omega)$ which can be directly sampled without rejection.

3.4 N-TRANSFORMATION REPARAMETERIZATION TRICK

While the *reparameterization trick* is easily implementable in the normal case, unfortunately it can only be applied to a handful of distributions. However a recent technique introduced by Naesseth et al. (2017) allows to extend the reparameterization trick to the wide class of distributions that can be simulated using rejection sampling. Dropping the dependence from $\mathbf{x}$ for simplicity, assume the approximate posterior is of the form $g(\omega|\theta)$ and that it can be sampled by making proposals from $r(\omega|\theta)$. If *the proposal distribution can be reparameterized* we can still perform the reparameterization trick. Let $\varepsilon \sim s(\varepsilon)$, and $\omega = h(\varepsilon, \theta)$, a reparameterization of the proposal distribution, $r(\omega|\theta)$. Performing the reparameterization trick for $g(\omega|\theta)$ is made possible by the fundamental lemma proven in (Naesseth et al., 2017):

Lemma 1. *Let f be any measurable function and $\varepsilon \sim \pi(\varepsilon|\theta) = s(\varepsilon)\dfrac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)}$ the distribution of the accepted sample. Then:*

$$\mathbb{E}_{\pi(\varepsilon|\theta)}[f(h(\varepsilon,\theta))] = \int f(h(\varepsilon,\theta))\pi(\varepsilon|\theta)d\varepsilon = \int f(\omega)g(\omega|\theta)d\omega = \mathbb{E}_{g(\omega|\theta)}[f(\omega)], \quad (7)$$

Then the gradient can be taken using the log derivative trick:

$$\nabla_\theta\mathbb{E}_{g(\omega|\theta)}[f(\omega)] = \nabla_\theta\mathbb{E}_{\pi(\varepsilon|\theta)}[f(h(\varepsilon,\theta))] = \mathbb{E}_{\pi(\varepsilon|\theta)}[\nabla_\theta f(h(\varepsilon,\theta))] + + \mathbb{E}_{\pi(\varepsilon|\theta)}\left[f(h(\varepsilon,\theta))\nabla_\theta \log\frac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)}\right], \quad (8)$$

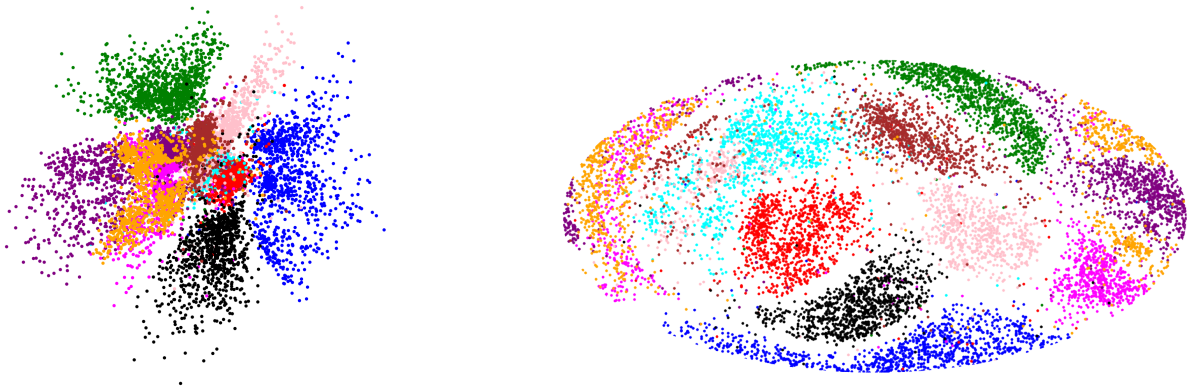
However, in the case of the vMF a different procedure is required. After performing the transformation $h(\varepsilon, \theta)$ and accepting/rejecting the sample, we sample *another* random variable $\mathbf{v} \sim \pi_2(\mathbf{v})$, and then apply a transformation $\mathbf{z} = \mathcal{T}(h(\varepsilon,\theta), \mathbf{v}; \theta)$, such that $\mathbf{z} \sim q_\psi(\mathbf{z}|\theta)$ is distributed as the approximate posterior (in our case a vMF). Effectively this entails applying another reparameterization trick after the acceptance/rejection step. To still be able to perform the reparameterization we show that Lemma 1 fundamentally still holds in this case as well.

Lemma 2. *Let f be any measurable function and $\varepsilon \sim \pi_1(\varepsilon|\theta) = s(\varepsilon)\dfrac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)}$ the distribution of the accepted sample. Also let $\mathbf{v} \sim \pi_2(v)$, and $\mathcal{T}$ a transformation that depends on the parameters such that if $\mathbf{z} = \mathcal{T}(\omega, v; \theta)$ with $\omega \sim g(\omega|\theta)$, then $\sim q(\mathbf{z}|\theta)$:*

$$\mathbb{E}_{(\varepsilon,\mathbf{v})\sim\pi_1(\varepsilon|\theta)\pi_2(\mathbf{v})}\left[f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right)\right] = \int f(\mathbf{z})q(\mathbf{z}|\theta)d\mathbf{z} = \mathbb{E}_{q(\mathbf{z}|\theta)}[f(\mathbf{z})], \quad (9)$$

Proof. See Appendix C. □

With this result we are able to derive a gradient expression similarly as done in equation 8. We refer to Appendix D for a complete derivation.



(a) $\mathbb{R}^2$ latent space of the $\mathcal{N}$-VAE. (b) Hammer projection of $\mathcal{S}^2$ latent space of the $\mathcal{S}$-VAE.

Figure 2: Latent space visualization of the 10 MNIST digits in 2 dimensions of both $\mathcal{N}$-VAE (left) and $\mathcal{S}$-VAE (right). (Best viewed in color)

3.5 BEHAVIOR IN HIGH DIMENSIONS

The surface area of a hypersphere is defined as

$$S(m-1) = r^m \frac{2(\pi^{m/2})}{\Gamma(m/2)} \tag{10}$$

where m is the dimensionality and r the radius. Notice that $S(m-1) \to 0$, as $m \to \infty$. However, even for $m > 20$ we observe a *vanishing surface problem* (see Figure 6 in Appendix E). This could thus lead to unstable behavior of hyperspherical models in high dimensions.

4 RELATED WORK

Extending the VAE The majority of VAE extensions focus on increasing the flexibility of the approximate posterior. This is usually achieved through *normalizing flows* (Rezende and Mohamed, 2015), a class of invertible transformations applied sequentially to an initial reparameterizable density $q_0(\mathbf{z}_0)$, allowing for more complex posteriors. Normalizing flows can be considered orthogonal to our proposed approach. In fact, while allowing for a more flexible posterior, they do not modify the standard normal prior assumption. They could be perfectly combined with $\mathcal{S}$-VAEs allowing for more flexible distributions on the hypersphere.

One approach to obtain a more flexible prior is to use a simple mixture of Gaussians (MoG) prior (Dilokthanakul et al., 2016). The recently introduced VampPrior model (Tomczak and Welling, 2017) outlines several advantages over the MoG and instead tries to learn a more flexible prior by expressing it as a mixture of approximate posteriors. A non-parametric prior is proposed in Nalisnick and Smyth (2017), utilizing a truncated stick-breaking process. Opposite to these approaches, we aim at using a non-informative prior to simplify the inference.

The closest approach to ours is a VAE with a vMF distribution in the latent space used for a sentence generation task by (Guu et al., 2017). While formally this approach is cast as a variational approach, the proposed model does not reparameterize and learn the concentration parameter κ, treating it as a constant value that remains the same for every approximate posterior instead. Critically, as indicated in Equation 5, the KL divergence term only depends on κ therefore leaving κ constant means never explicitly optimizing the KL divergence term in the loss. The method then only optimizes the reconstruction error by adding vMF noise to the encoder output in the latent space to still allow generation. Moreover, using a fixed global κ for *all* the approximate posteriors severely limits the flexibility and the expressiveness of the model.

Non-Euclidean Latent Space In Liu and Zhu (2017), a general model to perform Bayesian inference in Riemannian Manifolds is proposed. Following other Stein-related approaches, the method does not explicitly define a posterior density but approximates it with a number of particles. Despite its generality and flexibility, it requires the choice of a kernel on the manifold and multiple particles to have a good approximation of the posterior distribution. The former is not necessarily straightforward, while the latter quickly becomes computationally unfeasible.

Another approach by Nickel and Kiela (2017), capitalizes on the hierarchical structure present in some data types. By learning the embeddings for a graph in a non-euclidean negative curvature hyperbolical space, they show this topology has clear advantages over embedding these objects in a Euclidean space. Although they did not use a VAE-based approach, that is, they did not build a

Table 1: Summary of results (mean and standard-deviation over 10 runs) of unsupervised model on MNIST. RE and KL correspond respectively to the reconstruction and the KL part of the ELBO. Best results are highlighted only if they passed a student t-test with $p < 0.01$.

Method	$\mathcal{N}$**-VAE**				$\mathcal{S}$**-VAE**			
	LL	$\mathcal{L}[q]$	RE	KL	LL	$\mathcal{L}[q]$	RE	KL
$d = 2$	-135.73$\pm$.83	-137.08$\pm$.83	-129.84$\pm$.91	7.24$\pm$.11	**-132.50**$\pm$.73	-133.72$\pm$.85	-126.43$\pm$.91	7.28$\pm$.14
$d = 5$	-110.21$\pm$.21	-112.98$\pm$.21	-100.16$\pm$.22	12.82$\pm$.11	**-108.43**$\pm$.09	-111.19$\pm$.08	-97.84$\pm$.13	13.35$\pm$.06
$d = 10$	-93.84$\pm$.30	-98.36$\pm$.30	-78.93$\pm$.30	19.44$\pm$.14	**-93.16**$\pm$.31	-97.70$\pm$.32	-77.03$\pm$.39	20.67$\pm$.08
$d = 20$	-88.90$\pm$.26	-94.79$\pm$.19	-71.29$\pm$.45	23.50$\pm$.31	-89.02$\pm$.31	-96.15$\pm$.32	-67.65$\pm$.43	28.50$\pm$.22
$d = 40$	**-88.93**$\pm$.30	-94.91$\pm$.18	-71.14$\pm$.56	23.77$\pm$.49	-90.87$\pm$.34	-101.26$\pm$.33	-67.75$\pm$.70	33.50$\pm$.45

probabilistic generative model of the data interpreting the embeddings as latent variables, this approach shows the merit of explicitly adjusting the choice of latent topology to the data used.

A Hyperspherical Perspective As noted before, a distinction must be made between models dealing with the challenges of intrinsically hyperspherical data like omnidirectional video, and those attempting to exploit some latent hyperspherical manifold. A recent example of the first can be found in Cohen et al. (2018), where *spherical* CNNs are introduced. While flattening a spherical image produces unavoidable distortions, the newly defined convolutions take into account its geometrical properties.

The most general implementation of the second model type was proposed by Gopal and Yang (2014), who introduced a suite of models to improve cluster performance of high-dimensional data based on mixture of vMF distributions. They showed that reducing an object representation to its directional components increases clusterability over standard methods like K-Means or Latent Dirichlet Allocation (Blei et al., 2003).

Specific applications of the vMF can be further found ranging from computer vision, where it is used to infer structure from motion (Guan and Smith, 2017) in spherical video, or structure from texture (Wilson et al., 2014), to natural language processing, where it is utilized in text analysis (Banerjee et al., 2003, 2005) and topic modeling (Banerjee and Basu, 2007; Reisinger et al., 2010).

Additionally, modeling data by restricting it to a hypersphere provides some natural regularizing properties as noted in (Liu et al., 2017). Finally Aytekin et al. (2018) show on a variety of deep auto-encoder models that adding L2 normalization to the latent space during training, i.e. forcing the latent space on a hypersphere, improves clusterability.

5 EXPERIMENTS

In this section, we first perform a series of experiments to investigate the theoretical properties of the proposed $\mathcal{S}$-VAE compared to the $\mathcal{N}$-VAE. In a second experiment, we show how $\mathcal{S}$-VAEs can be used in semi-supervised tasks to create a better separable latent representation to enhance classification. In the last experiment, we show that the $\mathcal{S}$-VAE indeed presents a promising alternative to $\mathcal{N}$-VAEs for data with a non-Euclidean latent representation of low dimensionality, on a link prediction task for three citation networks. All architecture and hyperparameter details are given in Appendix F.

5.1 RECOVERING HYPERSPHERICAL LATENT REPRESENTATIONS

In this first experiment we build on the motivation developed in Subsection 2.3, by confirming with a synthetic data example the difference in behavior of the $\mathcal{N}$-VAE and $\mathcal{S}$-VAE in recovering latent hyperspheres. We first generate samples from a mixture of three vMFs on the circle, $\mathcal{S}^1$, as shown in Figure 1(a), which subsequently are mapped into the higher dimensional $\mathbb{R}^{100}$ by applying a noisy, non-linear transformation. After this, we in turn train an auto-encoder, a $\mathcal{N}$-VAE, and a $\mathcal{S}$-VAE. We further investigate the behavior of the $\mathcal{N}$-VAE, by training a model using a scaled down KL divergence.

Results The resulting latent spaces, displayed in Figure 1, clearly confirm the intuition built in Subsection 2.3. As expected, in Figure 1(b) the auto-encoder is perfectly capable to embed in low dimensions the original underlying data structure. However, most parts of the latent space are not occupied by points, critically affecting the ability to generate meaningful samples.

In the $\mathcal{N}$-VAE setting we observe two types of behaviours, summarized by Figures 1(c) and 1(d). In the first we observe that if the prior is too strong it will force the

Table 2: Summary of results (mean accuracy and standard-deviation over 20 runs) of semi-supervised K-NN on MNIST. Best results are highlighted only if they passed a student t-test with $p < 0.01$.

Method	100		600		1000	
	$\mathcal{N}$-VAE	$\mathcal{S}$-VAE	$\mathcal{N}$-VAE	$\mathcal{S}$-VAE	$\mathcal{N}$-VAE	$\mathcal{S}$-VAE
$d = 2$	$72.6_{\pm 2.1}$	**77.9**$_{\pm 1.6}$	$80.8_{\pm 0.5}$	**84.9**$_{\pm 0.6}$	$81.7_{\pm 0.5}$	**85.6**$_{\pm 0.5}$
$d = 5$	$81.8_{\pm 2.0}$	**87.5**$_{\pm 1.0}$	$90.9_{\pm 0.4}$	**92.8**$_{\pm 0.3}$	$92.0_{\pm 0.2}$	**93.4**$_{\pm 0.2}$
$d = 10$	$75.7_{\pm 1.8}$	**80.6**$_{\pm 1.3}$	$88.4_{\pm 0.5}$	**91.2**$_{\pm 0.4}$	$90.2_{\pm 0.4}$	**92.8**$_{\pm 0.3}$
$d = 20$	$71.3_{\pm 1.9}$	**72.8**$_{\pm 1.6}$	$88.3_{\pm 0.5}$	**89.1**$_{\pm 0.6}$	$90.1_{\pm 0.4}$	**91.1**$_{\pm 0.3}$
$d = 40$	**72.3**$_{\pm 1.6}$	$67.7_{\pm 2.3}$	$88.0_{\pm 0.5}$	$87.4_{\pm 0.7}$	$90.3_{\pm 0.5}$	$90.4_{\pm 0.4}$

posterior to match the prior shape, concentrating the samples in the center. However, this prevents the $\mathcal{N}$-VAE to correctly represent the true shape of the data and creates instability problems for the decoder around the origin. On the contrary, if we scale down the KL term, we observe that the samples from the approximate posterior maintain a shape that reflects the $\mathcal{S}^1$ structure smoothed with Gaussian noise. However, as the approximate posterior differs strongly from the prior, obtaining meaningful samples from the latent space again becomes problematic.

The $\mathcal{S}$-VAE on the other hand, almost perfectly recovers the original dataset structure, while the samples from the approximate posterior closely match the prior distribution. This simple experiment confirms the intuition that having a prior that matches the true latent structure of the data, is crucial in constructing a correct latent representation that preserves the ability to generate meaningful samples.

5.2 EVALUATION OF EXPRESSIVENESS

To compare the behavior of the $\mathcal{N}$-VAE and $\mathcal{S}$-VAE on a data set that does not have a clear hyperspherical latent structure, we evaluate both models on a reconstruction task using dynamically binarized MNIST (Salakhutdinov and Murray, 2008). We analyze the ELBO, KL, negative reconstruction error, and marginal log-likelihood (LL) for both models on the test set. The LL is estimated using importance sampling with 500 sample points (Burda et al., 2015).

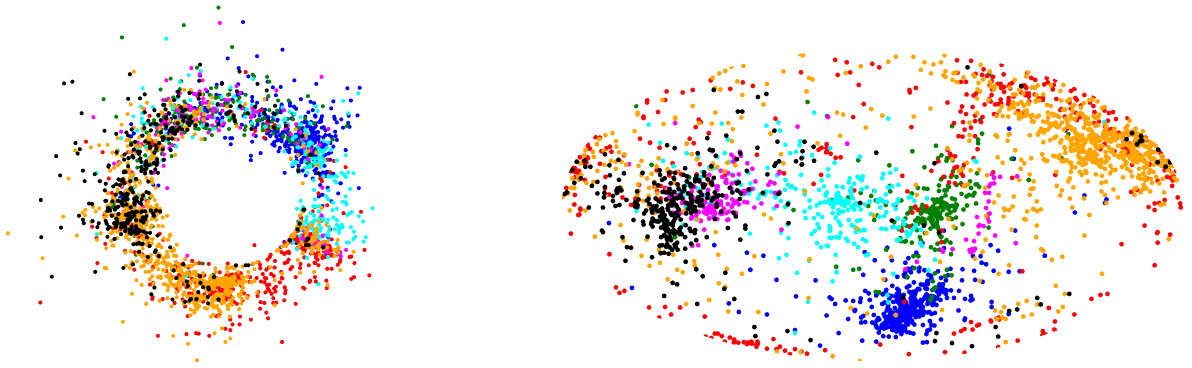
Results Results are shown in Table 1. We first note that in terms of negative reconstruction error the $\mathcal{S}$-VAE outperforms the $\mathcal{N}$-VAE in all dimensions. Since the $\mathcal{S}$-VAE uses a uniform prior, the KL divergence increases more strongly with dimensionality, which results in a higher ELBO. However in terms of log-likelihood (LL) the $\mathcal{S}$-VAE clearly has an edge in low dimensions ($d = 2, 5, 10$) and performs comparable to the $\mathcal{N}$-VAE in $d = 20$. This empirically confirms the hypothesis of Subsection 2.2, showing the positive effect of having a uniform prior in low dimensions. In the absence of any origin pull, the data is able to cluster naturally, utilizing the entire latent space which can be observed in Figure 2. Note that in Figure 2(a) all mass is concentrated around the center, since the prior mean is zero. Conversely, in Figure 2(b) all available space is evenly covered due to the uniform prior, resulting in more separable clusters in $\mathcal{S}^2$ compared to $\mathbb{R}^2$. However, as dimensionality increases, the Gaussian distribution starts to approximate a hypersphere, while its posterior becomes more expressive than the vMF due to the higher number of variance parameters. Simultaneously, as described in Subsection 3.5, the surface area of the vMF starts to collapse limiting the available space.

In Figure 7 and 8 of Appendix G, we present randomly generated samples from the $\mathcal{N}$-VAE and the $\mathcal{S}$-VAE, respectively. Moreover, in Figure 9 of Appendix G, we show 2-dimensional manifolds for the two models. Interestingly, the manifold given by the $\mathcal{S}$-VAE indeed results in a latent space where digits occupy the entire space and there is a sense of continuity from left to right.

5.3 SEMI-SUPERVISED LEARNING

Having observed the $\mathcal{S}$-VAE's ability to increase clusterability of data points in the latent space, we wish to further investigate this property using a semi-supervised classification task. For this purpose we re-implemented the M1 and M1+M2 models as described in (Kingma et al., 2014), and evaluate the classification accuracy of the $\mathcal{S}$-VAE and the $\mathcal{N}$-VAE on dynamically binarized MNIST. In the M1 model, a classifier utilizes the latent features obtained using a VAE as in experiment 5.2. The M1+M2 model is constructed by stacking the M2 model on top of M1, where M2 is the result of augmenting the VAE by introducing a partially observed variable $\mathbf{y}$, and combining the ELBO and classification objective. This concatenated model is trained end-to-end [3].

[3]It is worth noting that in the original implementation by Kingma et al. (2014) the stacked model did not converge well using end-to-end training, and used the extracted features of the



(a) $\mathbb{R}^2$ latent space of the $\mathcal{N}$-VGAE. (b) Hammer projection of $\mathcal{S}^2$ latent space of the $\mathcal{S}$-VGAE.

Figure 3: Latent space of unsupervised $\mathcal{N}$-VGAE and $\mathcal{S}$-VGAE models trained on Cora citation network. Colors denote documents classes which are not provided during training. (Best viewed in color)

This last model also allows for a combination of the two topologies due to the presence of two distinct latent variables, $\mathbf{z}_1$ and $\mathbf{z}_2$. Since in the M2 latent space the class assignment is expressed by the variable $\mathbf{y}$, while $\mathbf{z}_2$ only needs to capture the style, it naturally follows that the $\mathcal{N}$-VAE is more suited for this objective due to its higher number of variance parameters. Hence, besides comparing the $\mathcal{S}$-VAE against the $\mathcal{N}$-VAE, we additionally run experiments for the M1+M2 model by modeling $\mathbf{z}_1$, $\mathbf{z}_2$ respectively with a vMF and normal distribution.

Results As can be see in Table 2, for M1 the $\mathcal{S}$-VAE outperforms the $\mathcal{N}$-VAE in all dimensions up to $d = 40$. This result is amplified for a low number of observed labels. Note that for both models absolute performance drops as the dimensionality increases, since K-NN used as the classifier suffers from the curse of dimensionality. Besides reconfirming superiority of the $\mathcal{S}$-VAE in $d < 20$, its better performance than the $\mathcal{N}$-VAE for $d = 20$ was unexpected. This indicates that although the log-likelihood might be comparable(see Table 1) for higher dimensions, the $\mathcal{S}$-VAE latent space better captures the cluster structure.

In the concatenated model M1+M2, we first observe in Table 3 that either the pure $\mathcal{S}$-VAE or the $\mathcal{S}$+$\mathcal{N}$-VAE model yields the best results, where the $\mathcal{S}$-VAE almost always outperforms the $\mathcal{N}$-VAE. Our hypothesis regarding the merit of a $\mathcal{S}$+$\mathcal{N}$-VAE model is further confirmed, as displayed by the stable, strong performance across all different dimensions. Furthermore, the clear edge in clusterability of the $\mathcal{S}$-VAE in low dimensional $\mathbf{z}_1$ as already observed in Table 2, is again evident. As the dimensionality of $\mathbf{z}_1, \mathbf{z}_2$ increases, the accuracy of the $\mathcal{N}$-VAE improves, reducing the performance gap with the $\mathcal{S}$-VAE. As previously noticed the $\mathcal{S}$-VAE performance drops when $dim\ \mathbf{z}_2 = 50$, with the best result being obtained for $dim\ \mathbf{z}_1 = dim\ \mathbf{z}_2 = 10$. In fact, it is worth noting that for this setting the $\mathcal{S}$-VAE obtains comparable results to the original settings of (Kingma et al., 2014), while needing a considerably smaller latent space. Finally, the end-to-end trained $\mathcal{S}$+$\mathcal{N}$-VAE model is able to reach a significantly higher classification accuracy than the original results reported by Kingma et al. (2014), 96.7$_{\pm.1}$.

The M1+M2 model allows for conditional generation. Similarly to (Kingma et al., 2014), we set the latent variable $\mathbf{z}_2$ to the value inferred from the test image by the inference network, and then varied the class label $\mathbf{y}$. In Figure 10 of Appendix H we notice that the model is able to disentangle the style from the class.

Table 3: Summary of results of semi-supervised model M1+M2 on MNIST.

Method		**100**		
$dim\ \mathbf{z}_1$	$dim\ \mathbf{z}_2$	$\mathcal{N}$+$\mathcal{N}$	$\mathcal{S}$+$\mathcal{S}$	$\mathcal{S}$+$\mathcal{N}$
5	5	90.0$_{\pm.4}$	**94.0**$_{\pm.1}$	93.8$_{\pm.1}$
	10	90.7$_{\pm.3}$	94.1$_{\pm.1}$	**94.8**$_{\pm.2}$
	50	90.7$_{\pm.1}$	92.7$_{\pm.2}$	**93.0**$_{\pm.1}$
10	5	90.7$_{\pm.3}$	91.7$_{\pm.5}$	**94.0**$_{\pm.4}$
	10	92.2$_{\pm.1}$	**96.0**$_{\pm.2}$	**95.9**$_{\pm.3}$
	50	92.9$_{\pm.4}$	95.1$_{\pm.2}$	**95.7**$_{\pm.1}$
50	5	92.0$_{\pm.2}$	91.7$_{\pm.4}$	**95.8**$_{\pm.1}$
	10	93.0$_{\pm.1}$	95.8$_{\pm.1}$	**97.1**$_{\pm.1}$
	50	93.2$_{\pm.2}$	94.2$_{\pm.1}$	**97.4**$_{\pm.1}$

M1 model as inputs for the M2 model instead.

5.4 LINK PREDICTION ON GRAPHS

In this experiment, we aim at demonstrating the ability of the $\mathcal{S}$-VAE to learn meaningful embeddings of nodes in a graph, showing the advantages of embedding objects in a non-Euclidean space. We test hyperspherical reparameterization on the recently introduced Variational Graph Auto-Encoder (VGAE) (Kipf and Welling, 2016), a VAE model for graph-structured data. We perform training on a link prediction task on three popular citation network datasets (Sen et al., 2008): Cora, Citeseer and Pubmed.

Dataset statistics and further experimental details are summarized in Appendix F.3. The models are trained in an unsupervised fashion on a masked version of these datasets where some of the links have been removed. All node features are provided and efficacy is measured in terms of average precision (AP) and area under the ROC curve (AUC) on a test set of previously removed links. We use the same training, validation, and test splits as in Kipf and Welling (2016), i.e. we assign 5% of links for validation and 10% of links for testing.

Table 4: Results for link prediction in citation networks.

Method		$\mathcal{N}$**-VGAE**	$\mathcal{S}$**-VGAE**
Cora	AUC	$92.7_{\pm.2}$	$\mathbf{94.1}_{\pm.1}$
	AP	$93.2_{\pm.4}$	$\mathbf{94.1}_{\pm.3}$
Citeseer	AUC	$90.3_{\pm.5}$	$\mathbf{94.7}_{\pm.2}$
	AP	$91.5_{\pm.5}$	$\mathbf{95.2}_{\pm.2}$
Pubmed	AUC	$\mathbf{97.1}_{\pm.0}$	$96.0_{\pm.1}$
	AP	$\mathbf{97.1}_{\pm.0}$	$96.0_{\pm.1}$

Results In Table 4, we show that our model outperforms the $\mathcal{N}$-VGAE baseline on two out of the three datasets by a significant margin. The log-probability of a link is computed as the dot product of two embeddings. In a hypersphere, this can be interpreted as the cosine similarity between vectors. Indeed we find that the choice of a dot product scoring function for link prediction is problematic in combination with the normal distribution on the latent space. If embeddings are close to the zero-center, noise during training can have a large destabilizing effect on the angle information between two embeddings. In practice, the model finds a solution where embeddings are "pushed" away from the zero-center, as demonstrated in Figure 3(a). This counteracts the pull towards the center arising from the standard prior and can overall lead to poor modeling performance. By constraining the embeddings to the surface of a hypersphere, this effect is mitigated, and the model can find a good separation of the latent clusters, as shown in Figure 3(b).

On Pubmed, we observe that the $\mathcal{S}$-VAE converges to a lower score than the $\mathcal{N}$-VAE. The Pubmed dataset is significantly larger than Cora and Citeseer, and hence more complex. The $\mathcal{N}$-VAE has a larger number of variance parameters for the posterior distribution, which might have played an important role in better modeling the relationships between nodes. We further hypothesize that not all graphs are necessarily better embedded in a hyperspherical space and that this depends on some fundamental topological properties of the graph. For instance, the already mentioned work from Nickel and Kiela (2017) shows that hyperbolical space is better suited for graphs with a hierarchical, tree-like structure. These considerations prefigure an interesting research direction that will be explored in future work.

6 CONCLUSION

With the $\mathcal{S}$-VAE we set an important first step in the exploration of hyperspherical latent representations for variational auto-encoders. Through various experiments, we have shown that $\mathcal{S}$-VAEs have a clear advantage over $\mathcal{N}$-VAEs for data residing on a known hyperspherical manifold, and are competitive or surpass $\mathcal{N}$-VAEs for data with a non-obvious hyperspherical latent representation in lower dimensions. Specifically, we demonstrated $\mathcal{S}$-VAEs improve separability in semi-supervised classification and that they are able to improve results on state-of-the-art link prediction models on citation graphs, by merely changing the prior and posterior distributions as a simple drop-in replacement.

We believe that the presented research paves the way for various promising areas of future work, such as exploring more flexible approximate posterior distributions through normalizing flows on the hypersphere, or hierarchical mixture models combining hyperspherical and hyperplanar space. Further research should be done in increasing the performance of $\mathcal{S}$-VAEs in higher dimensions; one possible solution of which could be to dynamically learn the radius of the latent hypersphere in a full Bayesian setting.

Acknowledgements

We would like to thank Rianne van den Berg, Jonas Köhler, Pim de Haan, Taco Cohen, Marco Federici, and Max Welling for insightful discussions. T.K. is supported by the SAP Innovation Center Network. J.M.T. was funded by the European Commission within the Marie Skłodowska-Curie Individual Fellowship (Grant No. 702666, Deep learning and Bayesian inference for medical imaging).

References

Aytekin, C., Ni, X., Cricri, F., and Aksu, E. (2018). Clustering and unsupervised anomaly detection with l2 normalized deep auto-encoder representations. *arXiv preprint arXiv:1802.00187*.

Banerjee, A. and Basu, S. (2007). Topic models over text streams: A study of batch and online unsupervised learning. *ICDM*, pages 431–436.

Banerjee, A., Dhillon, I., Ghosh, J., and Sra, S. (2003). Generative model-based clustering of directional data. *SIGKDD*, pages 19–28.

Banerjee, A., Dhillon, I. S., Ghosh, J., and Sra, S. (2005). Clustering on the unit hypersphere using von mises-fisher distributions. *Journal of Machine Learning Research*, 6(Sep):1345–1382.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

Cohen, T. S., Geiger, M., Khler, J., and Welling, M. (2018). Spherical CNNs. *ICLR*.

Dilokthanakul, N., Mediano, P. A. M., Garnelo, M., Lee, M. C. H., Salimbeni, H., Arulkumaran, K., and Shanahan, M. (2016). Deep unsupervised clustering with gaussian mixture variational autoencoders. *CoRR*, abs/1611.02648.

Fisher, N. I., Lewis, T., and Embleton, B. J. (1987). *Statistical analysis of spherical data*. Cambridge university press.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, pages 249–256.

Gopal, S. and Yang, Y. (2014). Von mises-fisher clustering models. *ICML*, pages 154–162.

Guan, H. and Smith, W. A. (2017). Structure-from-motion in spherical video using the von mises-fisher distribution. *IEEE Transactions on Image Processing*, 26(2):711–723.

Guu, K., Hashimoto, T. B., Oren, Y., and Liang, P. (2017). Generating sentences by editing prototypes. *arXiv preprint arXiv:1709.08878*.

Hasnat, M., Bohné, J., Milgram, J., Gentric, S., Chen, L., et al. (2017). von mises-fisher mixture model-based deep learning: Application to face verification. *arXiv preprint arXiv:1706.04264*.

Kingma, D. and Welling, M. (2014). Efficient gradient-based inference through transformations between bayes nets and neural nets. *ICML*, pages 1782–1790.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. *NIPS*, pages 3581–3589.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

Kipf, T. N. and Welling, M. (2016). Variational Graph Auto-Encoders. *NIPS Bayesian Deep Learning Workshop*.

Liu, C. and Zhu, J. (2017). Riemannian Stein Variational Gradient Descent for Bayesian Inference. *ArXiv e-prints*.

Liu, W., Zhang, Y.-M., Li, X., Yu, Z., Dai, B., Zhao, T., and Song, L. (2017). Deep hyperspherical learning. *NIPS*, pages 3953–3963.

Mardia, K. V. (1975). Statistics of directional data. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 349–393.

Naesseth, C., Ruiz, F., Linderman, S., and Blei, D. (2017). Reparameterization Gradients through Acceptance-Rejection Sampling Algorithms. *AISTATS*, pages 489–498.

Nalisnick, E. and Smyth, P. (2017). Stick-breaking variational autoencoders. *ICLR*.

Nickel, M. and Kiela, D. (2017). Poincaré embeddings for learning hierarchical representations. *NIPS*, pages 6341–6350.

Reisinger, J., Waters, A., Silverthorn, B., and Mooney, R. J. (2010). Spherical topic models. *ICML*.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. *ICML*, 37:1530–1538.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Salakhutdinov, R. and Murray, I. (2008). On the quantitative analysis of deep belief networks. *ICML*, pages 872–879.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI magazine*, 29(3):93.

Tomczak, J. M. and Welling, M. (2017). VAE with a VampPrior. *arXiv preprint arXiv:1705.07120*.

Ulrich, G. (1984). Computer generation of distributions on the m-sphere. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 33(2):158–163.

Wilson, R. C., Hancock, E. R., Pekalska, E., and Duin, R. P. (2014). Spherical and hyperbolic embeddings of data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2255–2269.

A SAMPLING PROCEDURE

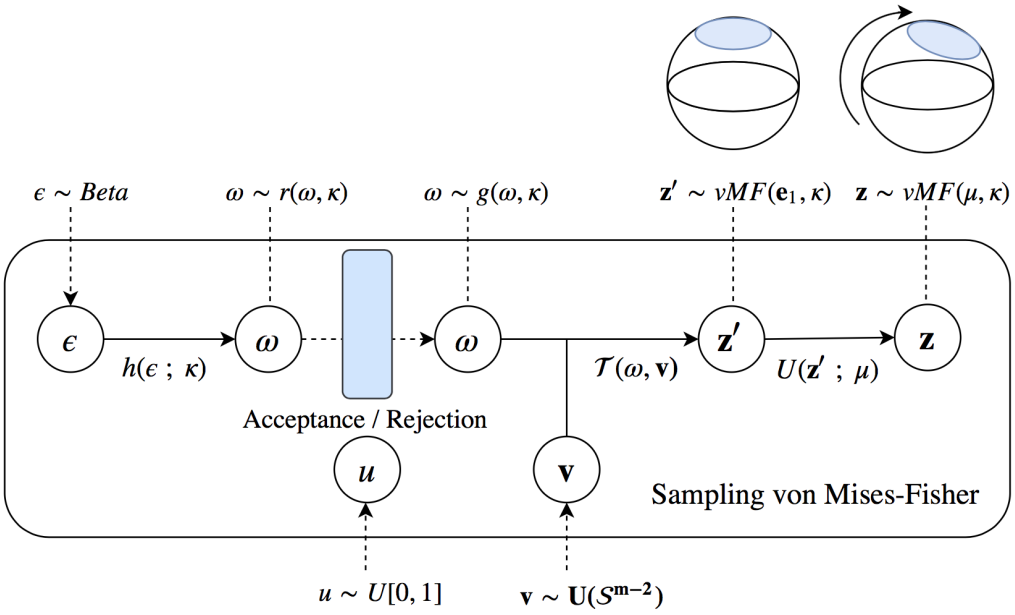


Figure 4: Overview of von Mises-Fisher sampling procedure. Note that as ω is a scalar, the procedure does not suffer from the curse of dimensionality.

The general algorithm for sampling from a vMF has been outlined in Algorithm 1. The exact form of the distribution of the univariate distribution $g(\omega|k)$ is:

$$g(\omega|k) = \frac{2(\pi^{m/2})}{\Gamma(m/2)} \mathcal{C}_m(k) \frac{\exp(\omega k)(1-\omega^2)^{\frac{1}{2}(m-3)}}{B(\frac{1}{2}, \frac{1}{2}(m-1))}, \tag{11}$$

Samples from this distribution are drawn using an acceptance/rejection algorithm when $m \neq 3$. The complete procedure is described in Algorithm 2. The $Householder$ reflection (see Algorithm 3 for details) simply finds an orthonormal transformation that maps the modal vector $\mathbf{e}_1 = (1, 0, \cdots, 0)$ to μ. Since an orthonormal transformation preserves the distances all the points in the hypersphere will stay in the surface after mapping. Notice that even the transform $U\mathbf{z}' = (\mathbb{I} - 2\mathbf{u}\mathbf{u}^\top)\mathbf{z}'$, can be executed in $\mathcal{O}(m)$ by rearranging the terms.

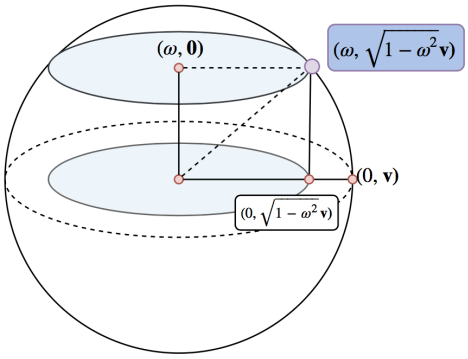


Figure 5: Geometric representation of a single sample in $\mathcal{S}^2$, where $\omega \sim g(\omega|k)$ and $\mathbf{v} \sim U(\mathcal{S}^1)$.

Algorithm 2 $g(\omega|k)$ acceptance-rejection sampling

Input: dimension m, concentration κ
Initialize values:
$b \leftarrow \dfrac{-2k + \sqrt{4k^2 + (m-1)^2}}{m-1}$
$a \leftarrow \dfrac{(m-1) + 2k + \sqrt{4k^2 + (m-1)^2}}{4}$
$d \leftarrow \dfrac{4ab}{(1+b)} - (m-1)\ln(m-1)$
repeat
  Sample $\varepsilon \sim$ Beta$(\frac{1}{2}(m-1), \frac{1}{2}(m-1))$
  $\omega \leftarrow h(\varepsilon, k) = \dfrac{1-(1+b)\varepsilon}{1-(1-b)\varepsilon}$
  $t \leftarrow \dfrac{2ab}{1-(1-b)\varepsilon}$
  Sample $u \sim \mathcal{U}(0,1)$
until $(m-1)\ln(t) - t + d \geq \ln(u)$
Return: ω

Algorithm 3 Householder transform

Input: mean μ, modal vector $\mathbf{e}_1$
$\mathbf{u}' \leftarrow \mathbf{e}_1 - \mu$
$\mathbf{u} \leftarrow \frac{\mathbf{u}'}{||\mathbf{u}'||}$
$U \leftarrow \mathbb{I} - 2\mathbf{u}\mathbf{u}^\top$
Return: U

Table 5: Expected number of samples needed before acceptance, computed using Monte Carlo estimate with 1000 samples varying dimensionality and concentration parameters. Notice that the sampling complexity increases in κ, but decreases as the dimensionality, d, increases.

	$\kappa=1$	$\kappa=5$	$\kappa=10$	$\kappa=50$	$\kappa=100$	$\kappa=500$	$\kappa=1000$	$\kappa=5000$	$\kappa=10000$
$d=5$	1.020	1.171	1.268	1.398	1.397	1.426	1.458	1.416	1.440
$d=10$	1.008	1.094	1.154	1.352	1.411	1.407	1.369	1.402	1.419
$d=20$	1.001	1.031	1.085	1.305	1.342	1.367	1.409	1.410	1.407
$d=40$	1.000	1.011	1.027	1.187	1.288	1.397	1.433	1.402	1.423
$d=100$	1.000	1.000	1.006	1.092	1.163	1.317	1.360	1.398	1.416

B KL DIVERGENCE DERIVATION

The KL divergence between a von-Mises-Fisher distribution $q(\mathbf{z}|\mu,k)$ and an uniform distribution in the hypersphere (one divided by the surface area of $\mathcal{S}^{m-1}$) $p(\mathbf{x}) = \left(\dfrac{2(\pi^{m/2})}{\Gamma(m/2)}\right)^{-1}$ is:

$$\mathcal{KL}[q(\mathbf{z}|\mu,k) \;||\; p(\mathbf{z})] = \int_{\mathcal{S}^{m-1}} q(\mathbf{z}|\mu,k)\log\frac{q(\mathbf{z}|\mu,k)}{p(\mathbf{z})}d\mathbf{z} \tag{12}$$

$$= \int_{\mathcal{S}^{m-1}} q(\mathbf{z}|\mu,k)\left(\log\mathcal{C}_m(k) + k\mu^T\mathbf{z} - \log p(\mathbf{z})\right)d\mathbf{z} \tag{13}$$

$$= k\mu\,\mathbb{E}_q[\mathbf{z}] + \log\mathcal{C}_m(k) - \log\left(\frac{2(\pi^{m/2})}{\Gamma(m/2)}\right)^{-1} \tag{14}$$

$$= k\frac{\mathcal{I}_{m/2}(k)}{\mathcal{I}_{m/2-1}(k)} + \left((m/2-1)\log k - (m/2)\log(2\pi) - \log\mathcal{I}_{m/2-1}(k)\right) \tag{15}$$

$$+ \frac{m}{2}\log\pi + \log 2 - \log\Gamma(\frac{m}{2}),$$

B.1 GRADIENT OF KL DIVERGENCE

Using

$$\nabla_k \mathcal{I}_v(k) = \frac{1}{2}\left(\mathcal{I}_{v-1}(k) + \mathcal{I}_{v+1}(k)\right), \tag{16}$$

and

$$\nabla_k \log \mathcal{C}_m(k) = \nabla_k \left((m/2-1)\log k - (m/2)\log(2\pi) - \log \mathcal{I}_{m/2-1}(k)\right) \tag{17}$$

$$= -\frac{\mathcal{I}_{m/2}(k)}{\mathcal{I}_{m/2-1}(k)}, \tag{18}$$

then

$$\nabla_\kappa \mathcal{KL}[q(\mathbf{z}|\mu,k) \mid\mid p(\mathbf{z})] = \nabla_\kappa k \frac{\mathcal{I}_{m/2}(k)}{\mathcal{I}_{m/2-1}(k)} + \nabla_k \log \mathcal{C}_m(k) \tag{19}$$

$$= \frac{\mathcal{I}_{m/2}(k)}{\mathcal{I}_{m/2-1}(k)} + k\nabla_k \frac{\mathcal{I}_{m/2}(k)}{\mathcal{I}_{m/2-1}(k)} - \frac{\mathcal{I}_{m/2}(k)}{\mathcal{I}_{m/2-1}(k)} \tag{20}$$

$$= \frac{1}{2}k\left(\frac{\mathcal{I}_{m/2+1}(k)}{\mathcal{I}_{m/2-1}(k)} - \frac{\mathcal{I}_{m/2}(k)\left(\mathcal{I}_{m/2-2}(k) + \mathcal{I}_{m/2}(k)\right)}{\mathcal{I}_{m/2-1}(k)^2} + 1\right), \tag{21}$$

Notice that we can use $\mathcal{I}^{exp}_{m/2} = \exp(-k)\mathcal{I}_{m/2}$ for numerical stability.

C PROOF OF LEMMA 2

Lemma 3 (2). *Let f be any measurable function and $\varepsilon \sim \pi_1(\varepsilon|\theta) = s(\varepsilon)\frac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)}$ the distribution of the accepted sample. Also let $\mathbf{v} \sim \pi_2(\mathbf{v})$, and $\mathcal{T}$ a transformation that depends on the parameters such that if $\mathbf{z} = \mathcal{T}(\omega, \mathbf{v}; \theta)$ with $\omega \sim g(\omega|\theta)$, then $\mathbf{z} \sim q(\mathbf{z}|\theta)$:*

$$\mathbb{E}_{(\varepsilon,\mathbf{v})\sim\pi_1(\varepsilon|\theta)\pi_2(\mathbf{v})}\left[f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right)\right] = \int f(\mathbf{z})q(\mathbf{z}|\theta)d\mathbf{z} = \mathbb{E}_{q(\mathbf{z}|\theta)}[f(\mathbf{z})], \tag{22}$$

Proof.

$$\mathbb{E}_{(\varepsilon,\mathbf{v})\sim\pi_1(\varepsilon|\theta)\pi_2(\mathbf{v})}\left[f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right)\right] = \iint f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right)\pi_1(\varepsilon|\theta)\pi_2(\mathbf{v})d\varepsilon d\mathbf{v}, \tag{23}$$

Using the same argument employed by Naesseth et al. (2017) we can apply the change of variables $\omega = h(\varepsilon,\theta)$ rewrite the expression as:

$$= \iint f\left(\mathcal{T}(\omega,\mathbf{v};\theta)\right) g(\omega|\theta)\pi_2(\mathbf{v})d\omega d\mathbf{v} =^* \int f(\mathbf{z})q(\mathbf{z}|\theta)d\mathbf{z} \tag{24}$$

Where in * we applied the change of variables $\mathbf{z} = \mathcal{T}(\omega,\mathbf{v};\theta)$. □

D REPARAMETRIZATION GRADIENT DERIVATION

D.1 GENERAL EXPRESSION DERIVATION

We can then proceed as in 8 using Lemma 2 and the the log derivative trick to compute the gradient of the expectation term $\nabla_\theta \mathbb{E}_{q(\mathbf{z}|\theta)}[f(\mathbf{z})]$:

$$\nabla_\theta \mathbb{E}_{q(\mathbf{z}|\theta)}[f(\mathbf{z})] = \nabla_\theta \iint f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right)\pi_1(\varepsilon|\theta)\pi_2(\mathbf{v})d\varepsilon d\mathbf{v} \tag{25}$$

$$= \nabla_\theta \iint f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right) s(\varepsilon) \frac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)} \pi_2(\mathbf{v}) d\varepsilon d\mathbf{v} \tag{26}$$

$$= \iint s(\varepsilon)\pi_2(\mathbf{v}) \nabla_\theta \left(f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right) \frac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)} \right) d\varepsilon d\mathbf{v} \tag{27}$$

$$= \iint s(\varepsilon)\pi_2(\mathbf{v}) \frac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)} \nabla_\theta \left(f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right)\right) d\varepsilon d\mathbf{v} \tag{28}$$

$$+ \iint s(\varepsilon)\pi_2(\mathbf{v}) f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right) \nabla_\theta \left(\frac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)} \right) d\varepsilon d\mathbf{v}$$

$$= \iint \pi_1(\varepsilon|\theta)\pi_2(\mathbf{v}) \nabla_\theta \left(f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right)\right) d\varepsilon dv \tag{29}$$

$$+ \iint s(\varepsilon)\pi_2(\mathbf{v}) f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right) \nabla_\theta \left(\frac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)} \right) d\varepsilon d\mathbf{v}$$

$$= \underbrace{\mathbb{E}_{(\varepsilon,\mathbf{v})\sim\pi_1(\varepsilon|\theta)\pi_2(\mathbf{v})} \left[\nabla_\theta f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right)\right]}_{g_{rep}} \tag{30}$$

$$+ \underbrace{\mathbb{E}_{(\varepsilon,\mathbf{v})\sim\pi_1(\varepsilon|\theta)\pi_2(\mathbf{v})} \left[f\left(\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right) \nabla_\theta \log \left(\frac{g(h(\varepsilon,\theta)|\theta)}{r(h(\varepsilon,\theta)|\theta)} \right) \right]}_{g_{cor}},$$

where g_{rep} is the reparameterization term and g_{cor} the correction term. Since h is invertible in ε, Naesseth et al. (2017) show that $\nabla_\theta \log \frac{q(h(\varepsilon,\theta),\theta)}{r((h(\varepsilon,\theta),\theta)}$ in g_{cor} simplifies to:

$$\nabla_\theta \log \frac{g(h(\varepsilon,\theta),\theta)}{r((h(\varepsilon,\theta),\theta)} = \nabla_\theta \log g(h(\varepsilon,\theta),\theta) + \nabla_\theta \log |\frac{\partial h(\varepsilon,\theta)}{\partial \varepsilon}|, \tag{31}$$

D.2 GRADIENT CALCULATION

In our specific case we want to take the gradient w.r.t. θ of the expression:

$$\mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{x};\theta)}[\log p_\phi(\mathbf{x}|\mathbf{z})] \quad \text{where } \theta = (\mu, \kappa), \tag{32}$$

The gradient can be computed using the Lemma 2 and the subsequent gradient derivation with $f(\mathbf{z}) = p_\phi(\mathbf{x}|\mathbf{z})$. As specified in Section 3.4 we optimize unbiased Monte Carlo estimates of the gradient. Therefore fixed one datapoint $\mathbf{x}$ and sampled $(\varepsilon, \mathbf{v}) \sim \pi_1(\varepsilon|\theta)\pi_2(\mathbf{v})$ the gradient is:

$$\nabla_\theta \mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{x};\theta)}[\log p_\phi(\mathbf{x}|\mathbf{z})] = g_{rep} + g_{cor}, \tag{33}$$

With

$$g_{rep} \approx \nabla_\theta \log p_\phi\left(\mathbf{x}|\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right), \tag{34}$$

$$g_{cor} \approx p_\phi\left(x|\mathcal{T}(h(\varepsilon,\theta),\mathbf{v};\theta)\right) \left(\nabla_\theta \log g(h(\varepsilon,\theta)|\theta) + \nabla_\theta \log |\frac{\partial h(\varepsilon,\theta)}{\partial \varepsilon}| \right), \tag{35}$$

where g_{rep} is simply the gradient of the reconstruction loss w.r.t θ and can be easily handled by automatic differentiation packages.

For what concerns g_{cor} we notice that the terms $g()$ and $h()$ do not depend on μ. Thus the g_{cor} term w.r.t. μ is 0 an all the following calculations can will be only w.r.t. κ. We therefore have:

$$\frac{\partial h(\varepsilon,k)}{\partial \varepsilon} = \frac{-2b}{((b-1)\varepsilon+1)^2} \quad \text{where } b = \frac{-2k + \sqrt{4k^2 + (m-1)^2}}{m-1}, \tag{36}$$

and

$$\nabla_\kappa \log g(\omega|k) = \nabla_\kappa \left(\log \mathcal{C}_m(k) + \omega k + \frac{1}{2}(m-3)\log(1-\omega^2) \right) \tag{37}$$

$$= \nabla_k \log \mathcal{C}_m(k) + \nabla_\kappa \left(\omega k + \frac{1}{2}(m-3)\log(1-\omega^2) \right). \tag{38}$$

So, putting everything together we have:

$$g_{cor} = \log p_\phi(x|z) \cdot \left[-\frac{\mathcal{I}_{m/2}}{\mathcal{I}_{m/2-1}} + \nabla_\kappa \left(\omega\kappa + \frac{1}{2}(m-3)\log(1-\omega^2) + \log|\frac{-2b}{((b-1)\varepsilon+1)^2}| \right) \right], \tag{39}$$

where

$$b = \frac{-2k + \sqrt{4k^2 + (m-1)^2}}{m-1} \tag{40}$$

$$\omega = h(\varepsilon, \theta) = \frac{1-(1+b)\varepsilon}{1-(1-b)\varepsilon} \tag{41}$$

$$z = \mathcal{T}(h(\varepsilon, \theta), \mathbf{v}; \theta), \tag{42}$$

And the term $\nabla_\kappa \left(\omega\kappa + \frac{1}{2}(m-3)\log(1-\omega^2) + \log|\frac{-2b}{((b-1)\varepsilon+1)^2}| \right)$ can be computed by automatic differentiation packages.

E COLLAPSE OF THE SURFACE AREA

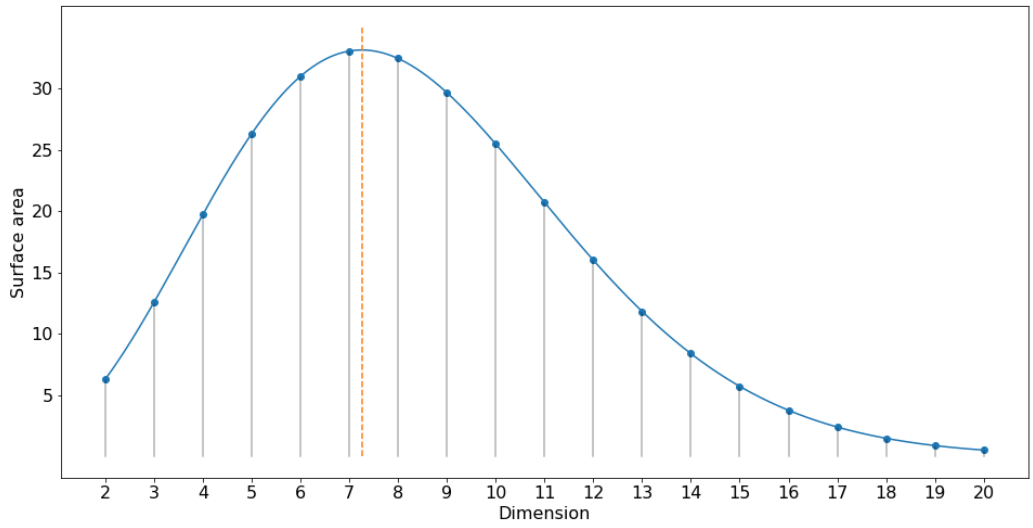


Figure 6: Plot of the unit hyperspherical surface area against dimensionality. The surface area has a maximum for $m = 7$.

F EXPERIMENTAL DETAILS: ARCHITECTURE AND HYPERPARAMETERS

F.1 EXPERIMENT 5.2

Architecture and hyperparameters For both the encoder and the decoder we use MLPs with 2 hidden layers of respectively, [256, 128] and [128, 256] hidden units. We trained until convergence using early-stopping with a look

ahead of 50 epochs. We used the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 1e-3, and mini-batches of size 64. Additionally, we used a linear *warm-up* for 100 epochs (Bowman et al., 2015). The weights of the neural network were initialized according to (Glorot and Bengio, 2010).

F.2 EXPERIMENT 5.3

Architecture and Hyperparameters For M1 we reused the trained models of the previous experiment, and used K-nearest neighbors (K-NN) as a classifier with $k = 5$. In the $\mathcal{N}$-VAE case we used the Euclidean distance as a distance metric. For the $\mathcal{S}$-VAE the geodesic distance $\arccos(\mathbf{x}^\top \mathbf{y})$ was employed. The performance was evaluated for $N = [100, 600, 1000]$ observed labels.

The stacked M1+M2 model uses the same architecture as outlined by Kingma et al. (2014), where the MLPs utilized in the generative and inference models are constructed using a single hidden layer, each with 500 hidden units. The latent space dimensionality of $\mathbf{z}_1$, $\mathbf{z}_2$ were both varied in $[5, 10, 50]$. We used the rectified linear unit (ReLU) as an activation function. Training was continued until convergence using early-stopping with a look ahead of 50 epochs on the validation set. We used the Adam optimizer with a learning rate of 1e-3, and mini-batches of size 100. All neural network weight were initialized according to (Glorot and Bengio, 2010). N was set to 100, and the α parameter used to scale the classification loss was chosen between $[0.1, 1.0]$. Crucially, we train this model end-to-end instead of by parts.

F.3 EXPERIMENT 5.4

Architecture and Hyperparameters We are training a Variational Graph Auto-encoder (VGAE) model, a state-of-the-art link prediction model for graphs, as proposed in Kipf and Welling (2016). For a fair comparison, we use the same architecture as in the original paper and we just change the way the latent space is generated using the vMF distribution instead of a normal distribution. All models are trained for 200 epochs on Cora and Citeseer, and 400 epochs on Pubmed with the Adam optimizer. Optimal learning rate $lr \in \{0.01, 0.005, 0.001\}$, dropout rate $p_{do} \in \{0, 0.1, 0.2, 0.3, 0.4\}$ and number of latent dimensions $d_z \in \{8, 16, 32, 64\}$ are determined via grid search based on validation AUC performance. For $\mathcal{S}$-VGAE, we omit the $d_z = 64$ setting as some of our experiments ran out of memory. The model is trained with a single hidden layer with 32 units and with document features as input, as in Kipf and Welling (2016). The weights of the neural network were initialized according to (Glorot and Bengio, 2010). For testing, we report performance of the model selected from the training epoch with highest AUC score on the validation set. Different from (Kipf and Welling, 2016), we train both the $\mathcal{N}$-VGAE and the $\mathcal{S}$-VGAE models using negative sampling in order to speed up training, i.e. for each positive link we sample, uniformly at random, one negative link during every training epoch. All experiments are repeated 5 times, and we report mean and standard error values.

F.3.1 FURTHER EXPERIMENTAL DETAILS

Dataset statistics are summarized in Table 6. Final hyperparameter choices found via grid search on the validation splits are summarized in Table 7.

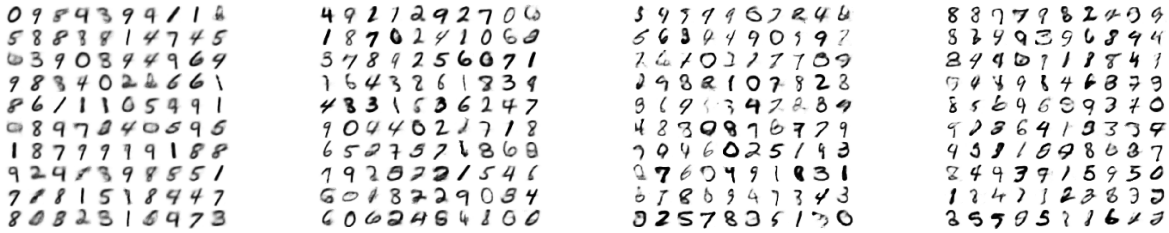
Table 6: Dataset statistics for citation network datasets.

Dataset	Nodes	Edges	Features
Cora	2,708	5,429	1,433
Citeseer	3,327	4,732	3,703
Pubmed	19,717	44,338	500

Table 7: Best hyperparameter settings found for citation network datasets.

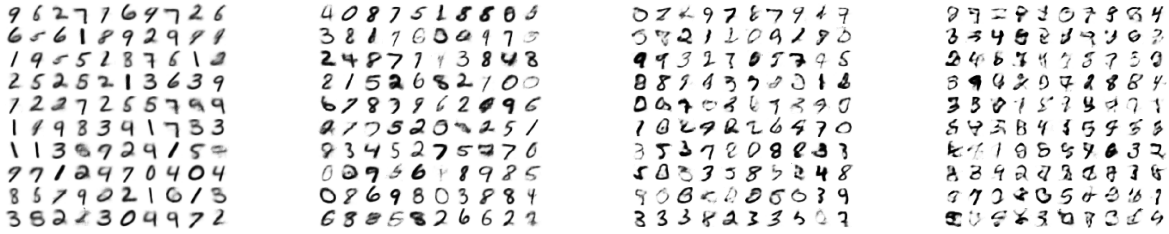
Dataset	Model	lr	p_{do}	d_z
Cora	$\mathcal{N}$-VAE	0.005	0.4	64
	$\mathcal{S}$-VAE	0.001	0.1	32
Citeseer	$\mathcal{N}$-VAE	0.01	0.4	64
	$\mathcal{S}$-VAE	0.005	0.2	32
Pubmed	$\mathcal{N}$-VAE	0.001	0.2	32
	$\mathcal{S}$-VAE	0.01	0.0	32

G VISUALIZATION OF SAMPLES AND LATENT SPACES



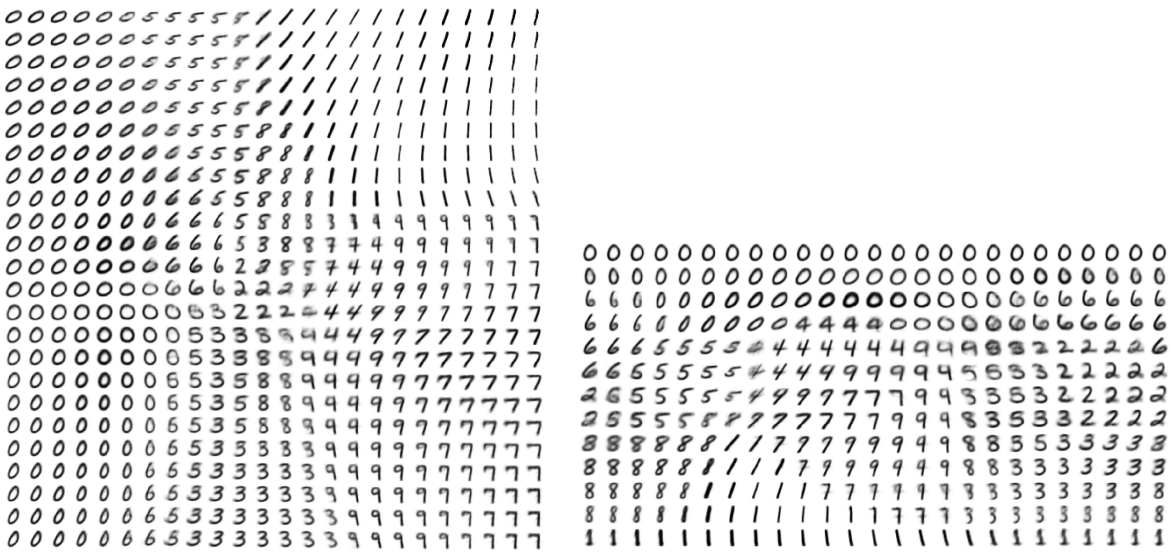
(a) $d = 2$ (b) $d = 5$ (c) $d = 10$ (d) $d = 20$

Figure 7: Random samples from $\mathcal{N}$-VAE of MNIST for different dimensionalities of latent space.



(a) $d = 2$ (b) $d = 5$ (c) $d = 10$ (d) $d = 20$

Figure 8: Random samples from $\mathcal{S}$-VAE of MNIST for different dimensionalities of latent space.



(a) $\mathcal{N}$-VAE (b) $\mathcal{S}$-VAE

Figure 9: Visualization of the 2 dimensional manifold of MNIST for both the $\mathcal{N}$-VAE and $\mathcal{S}$-VAE. Notice that the $\mathcal{N}$-VAE has a clear center and all digits are spread around it. Conversely, in the $\mathcal{S}$-VAE instead all digits occupy the entire space and there is a sense of continuity from left to right.

H VISUALIZATION OF CONDITIONAL GENERATION

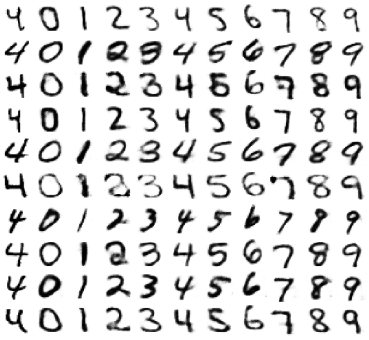


Figure 10: Visualization of handwriting styles learned by the model, using conditional generation on MNIST of M1+M2 with $dim(\mathbf{z}_1) = 50$, $dim(\mathbf{z}_2) = 50$, $\mathcal{S}$+$\mathcal{N}$. Following Kingma et al. (2014), the left most column shows images from the test set. The other columns show analogical fantasies of $\mathbf{x}$ by the generative model, where in each row the latent variable $\mathbf{z}_2$ is set to the value inferred from the test image by the inference network and the class label $\mathbf{y}$ is varied per column.