EMPIRICAL ANALYSIS OF OVERFITTING AND MODE DROP IN GAN TRAINING

Yasin Yazici[⋆†] *Chuan-Sheng Foo*[†] *Stefan Winkler*[‡] *Kim-Hui Yap*[⋆] *Vijay Chandrasekhar*[†*]

[⋆] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore
[†] Institute for Infocomm Research, A*STAR, Singapore
[‡] School of Computing, National University of Singapore

ABSTRACT

We examine two key questions in GAN training, namely overfitting and mode drop, from an empirical perspective. We show that when stochasticity is removed from the training procedure, GANs can overfit and exhibit almost no mode drop. Our results shed light on important characteristics of the GAN training procedure. They also provide evidence against prevailing intuitions that GANs do not memorize the training set, and that mode dropping is mainly due to properties of the GAN objective rather than how it is optimized during training.

***Index Terms*—** Generative Adversarial Networks, Generative Models, Deep Learning

1. INTRODUCTION

We perform empirical analyses to address key questions relating to overfitting, generalization and mode dropping in the training of generative adversarial networks (GAN). We hypothesize that these phenomena are related to stochasticity in GAN training and provide experimental evidence to support this intuition. We show that specific GAN architectures and optimization methods can overfit to the training set. We define overfitting as the case when the generator produces images that are nearly indistinguishable from the (training) data samples and "covers" all of them. By definition such a generator does not exhibit mode drop as it has complete recall on the training data distribution. Our experiments provide an answer to whether the GAN objective and its variants are sufficient to match the support of the training data distribution as questioned by [1]. It leads to new insights into how GANs generalize and why mode drop occurs.

While our observation that GANs overfit when stochasticity is removed may appear obvious by analogy to the behavior of models trained with maximum log-likelihood objectives, GANs are trained by optimizing a saddle-point objective and exhibit very different training dynamics. In particular, variants of gradient descent on the saddle-point objective may not converge without regularization even in simple cases [2].

In summary, our contribution is to empirically show that in GAN training (a) the generator tends to overfit to a large extent as stochasticity decreases, and similarly (b) the generator shows limited mode drop behavior, which reveals the relationship between mode drop and stochasticity.

[*] Contributed in 2019.

2. RELATED WORK

Overfitting: Do GANs overfit or memorize the data in the training set? This question is answered negatively in many papers, by searching for nearest-neighbors of generated images in training dataset [3–5]. Using a different approach, [6, 7] analyzed overfitting in GANs and other generative models through searching in the code space of the generator. While the latter approaches retrieve generated samples closer to ones in the training set, there are significant differences between the images, especially in the fine details. [8] suggests that memorization may not be happening as the generator does not directly learn from the training samples but from the feedback of the discriminator. These works suggest that GANs do not overfit to the training set.[1]

Mode drop (or collapse): This is a notorious behavior of GANs; instead of covering the full support of the data distribution, GANs tend to cover only parts of it. Some link this behavior to the underlying divergence of the GAN objective (non-saturating version) which is related to the reverse-KL divergence [9, 10]. Theoretical studies suggest that the low capacity of the discriminator is related to mode dropping [9]. [1] analyzed this empirically using a birthday paradox test and concluded that GANs are prone to leaving modes out. [11] related mode drop to the mismatch between the multimodality of data distribution and the unimodality of the prior distribution of the generator. Motivated by disconnected manifolds, [12] also blames the multimodality of the data distribution and suggests a mixture distribution of multiple generators. [13] interprets mode drop with catastrophic forgetting as a result of continuously changing generative distribution. By contrast, we show that mode drop is avoidable to a large extent by simply removing stochasticity from the training, while keeping the prior distribution and objective function the same.

[1] [4] claims that the discriminator overfits by showing the discrepancy between the logits of training set and validation set. In this paper, we are looking at overfitting/memorization from the generator side not the discriminator.

 © IEEE 2020

[14] suggested methods to reduce variance in gradient calculations without using larger mini-batches. In essence, we also reduce stochasticity by means of larger batch sizes. However the aim of our paper is different and more generally shows how overfitting and mode dropping are linked to stochasticity in training. [15] provides a theoretical analysis of GAN memorization, which is defined there as the case when the generator distribution matches the empirical data distribution over all samples from the prior, whereas our definition requires a match only on limited samples from the prior; [15] also does not aim to answer questions about mode dropping.

3. METHOD

GAN is a two player zero-sum game between a discriminator and a generator:

$$\min_G \max_D E_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})]+E_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]$$

It utilizes a discriminator to assess a peudo-divergence between the true data distribution, $p_{data}(\boldsymbol{x})$, and the generator's distribution, $p_g(\boldsymbol{x})$. The discriminator maximizes the divergence, while the generator minimizes it. In this way, the generator learns to mimic the data distribution implicitly.

In practice, the objective function is approximated with empirical averages:

$$\min_G \max_D \frac{1}{n}\sum_{i=1}^{n} \log D(\boldsymbol{x}_i) + \frac{1}{k}\sum_{i=1}^{k} \log(1 - D(G(\boldsymbol{z}_i)))$$

where $\boldsymbol{x}_i$ denotes the i^{th} sample from a fixed set $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ sampled from $p_{data}(\boldsymbol{x})$, and $\boldsymbol{z}_i$ is the i^{th} sample from a fixed set $\boldsymbol{Z} = \{\boldsymbol{z}_1, ..., \boldsymbol{z}_k\}$ sampled from input noise $p_z(\boldsymbol{z})$. In practice $k >> n$, as $p_z(\boldsymbol{z})$ is readily available, while data samples are limited.

During training, gradients of the objective are further approximated with stochastic gradients using a mini-batch of size m, leading to the following parameter updates:

$$\phi^{(t+1)} = \phi^{(t)} + \frac{\alpha}{m}\sum_{i=1}^{m} \frac{\partial(\log D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i))))}{\partial\phi^{(t)}}$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\alpha}{m}\sum_{i=1}^{m} \frac{\partial(\log(1 - D(G(\boldsymbol{z}_i))))}{\partial\theta^{(t)}}$$

where m samples are drawn from $\boldsymbol{X}$ with replacement and from $\boldsymbol{Z}$ without replacement,[2] $\phi^{(t)}$ and $\theta^{(t)}$ are the parameters of D and G at the t^{th} iteration respectively.

There are two sources of stochasticity in GAN training: (i) random samples from the prior and (ii) stochastic gradient updates.[3] The former is considered as a necessary step to capture support of the prior distribution, while the latter is used to reduce computation time by approximating the true gradient. To remove both sources of stochasticity, we select $k = n$ and $m = n$. Under these settings, G generates exactly n samples during training, and D only distinguishes them from n samples of the empirical data distribution. As G only generates a finite number of samples, this enables us to analyze pairwise distances between generated samples and data samples (or vice versa).

After analyzing the deterministic setting, we re-introduce varying amounts of stochasticity to investigate its effect on overfitting, generalization, and mode drop. We re-introduce stochastic gradients by reducing the mini-batch size m to $\frac{n}{2^l}, l = 2, 4, 6, 8$. We also re-introduce noise into the input (code) space of the generator. We did not consider sampling more from the prior as it modifies the ratio of real/fake samples. Each time, a single type of stochasticity is considered to study its effect in isolation.

Our intuition is that stochastic optimization leads the generator to leave out some modes and reduces the fidelity of image generation. In the initial training phase, stochasticity from the gradients due to sampling error is smaller than the estimated difference between the true and learnt distributions which enables GAN training to progress. However as training progresses, the generator manifold gets closer to the data manifold and stochasticity dominates the training, forcing the discriminator to find superficial explanations for the difference between the distributions. This prevents further progress in learning the data distribution and leads to non-convergence over the parameters. Indeed, the performance of features extracted from the GAN's discriminator deteriorates for a downstream classification task in later stages of training [16]. Also, large batch size training shows significant improvements in image generation [4].

4. EXPERIMENTS

We perform experiments on the SVHN [17], CIFAR-10 [18] and FFHQ [19] datasets commonly used for evaluating GANs. FFHQ images are scaled to 32×32 to make the analysis possible with large batch sizes and sufficiently large data sizes. We use 12,800 images from each dataset for training. In all our experiments, we use the non-saturating GAN loss [3] with alternating gradient descent and a 1:1 ratio for discriminator/generator updates.

We use a DCGAN [20] like architecture with details in the Apendix. In the discriminator we apply spectral normalization [21] which improves the results considerably. ADAM optimizer [22] with $\alpha = 0.0001$, $\beta_1 = 0$ and $\beta_2 = 0.9$ is used. Exponential moving average over generator parameters [5, 23] is used for evaluation and visualization. $z \in \mathbb{R}^{512}$ are sampled from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and fixed during training.

We include two types of randomness analysis in this paper. We show the effects of mini-batch size as the first type of

[2] As new samples are drawn from $p_z(\boldsymbol{z})$ at each iteration, and it is unlikely to draw the exact same sample again.

[3] We did not use stochastic components in the network (like batch-norm).

stochasticity. Then, by fixing the mini-batch size, we include noise to the fixed input samples of the generator as a second type of stochasticity. For this we replace $G(z)$ with $G(\hat{z})$ where $\hat{z} = z + \epsilon, \epsilon \sim \mathcal{N}(\mathbf{0}, 0.5\boldsymbol{I})$ during training. This corresponds to using an equally weighted mixture of n Gaussians with fixed mean and co-variance. During the inference stage, when analyzing and visualizing the generator, $G(z)$ is used, which corresponds to the mean of each Gaussian. This can also be considered as most likely mode of truncation trick [4].

To quantify overfitting and mode drop we utilize two types of metrics: (a) pixel-wise loss in data space with ℓ_1-norm, and (b) semantic loss in a latent space with ℓ_2-norm. The main reason to use pixel-wise loss is to show the generator can produce data samples closely. As this loss may not find semantically similar samples in certain cases [24], we also utilize semantic loss. With these metrics, nearest neighbor algorithm (1-NN) is used to evaluate and visualize the distance to the first nearest neighbor. For overfitting, this analysis is done by searching the nearest neighbor of each generated samples in all data samples, while for mode drop analysis it is done in the reverse order: searching the nearest neighbor of each data sample in all generated samples. The former can be interpreted as precision and the latter as recall.

Because search for the nearest sample is an expensive operation for large datasets, we approximate it with 200 samples. For example, for the overfitting analysis, the nearest neighbor of 200 generated samples are searched in all data samples. For the semantic loss, we use *InceptionV3* [25] architecture's penultimate layer activations similar to FID [26]. For pixel-wise loss, images are scaled to $[0, 255]$ to be able to interpret the results more easily.

The above metrics average over the samples, which does not clearly demonstrate the worst cases in that average. To include those, we also report the top 10% and 5% samples with highest first nearest neighbor (Table 1).

5. RESULTS

We begin the analysis from the visual part which motivated this work. Our qualitative results are best interpreted together with the quantitative results as we can only show limited samples. For qualitative results we show both pixel-wise loss and semantic loss, while for quantitative results we only show the pixel-wise loss as it is sufficient to support our hypothesis.

In our early experimentation we used full-batch setting without a significant difference from a batch size of 3,200 in terms of performance, thus we stick with highest mini-batch size of 3,200 to approximate the deterministic case. We train each case till convergence of our metrics.

5.1. Mini-Batch Size

In Fig. 1, larger batch sizes exhibit more overfitting (nearly indistinguishable samples), while smaller ones miss certain data points and produce more artifacts over the samples. Even though some of the artifacts are not easy to see, they influence the nearest neighbor in latent space. Similarly, Fig. 2 shows mode drop behavior for various mini-batch sizes. The pattern closely resembles the overfitting results, where smaller batch sizes (more stochasticity) exhibit more mode drop, i.e. nearest neighbor search from data samples to generated ones does not return to similar samples. In both analyses, SVHN is affected less with respect to batch size change compared to CIFAR-10 and FFHQ, which are harder to model.

Quantitative results (pixel-wise loss) are shown in Table 1. Scores for overfitting and mode drop exhibit similar patterns and supports our qualitative observations: as stochasticity decreases overfitting occurs and mode drop diminishes. This shows that modeling all parts of the distribution (no mode drop) is not sacrificed for image generation quality (overfitting). However there is a trade-off between generalization and mode drop. Table 1 also shows the scores for worst-case samples, which increase with lower percentages. As in the Figures, SVHN scores are less affected when batch size changes than the other datasets.

Table 1. Pixel-wise loss for overfitting and mode drop as a function of mini-batch size m and noisy latent code (*Noise*).

Dataset	m	*Noise*	Overfitting			Mode Drop		
			Avg	10%	5%	Avg	10%	5%
SVHN	3,200	No	1.55	2.82	3.58	1.55	2.50	2.80
SVHN	800	No	3.05	4.87	5.55	3.26	6.35	7.88
SVHN	200	No	7.43	11.59	12.42	7.84	12.66	13.99
SVHN	50	No	9.06	16.63	18.6	9.36	20.8	25.98
SVHN	3,200	Yes	6.09	10.71	11.85	7.15	18.07	22.76
CIFAR10	3,200	No	4.90	7.40	8.59	4.88	6.90	7.39
CIFAR10	800	No	8.04	12.96	14.51	8.43	13.73	15.6
CIFAR10	200	No	24.47	37.01	38.87	24.71	40.44	43.36
CIFAR10	50	No	28.31	42.42	44.35	29.49	47.37	50.89
CIFAR10	3,200	Yes	34.39	49.57	51.41	32.16	48.25	51.31
FFHQ	3,200	No	6.39	9.71	11.81	6.57	12.55	17.41
FFHQ	800	No	10.7	16.2	18.96	10.72	17.64	21.74
FFHQ	200	No	27.81	38.78	40.78	28.38	39.57	41.59
FFHQ	50	No	32.84	43.79	45.19	32.57	44.37	46.27
FFHQ	3,200	Yes	35.56	45.83	47.53	33.9	44.51	46.04

5.2. Noisy Latent Code

We now show the effect of the second type of stochasticity, namely noisy latent code, to the generator. In our experiment one model receives noise to its input (as explained in Section 4), while the other does not. Otherwise they are the same, with a mini-batch size of 3,200. Both models are trained for the same amount of iterations, but the results are drastically different, especially for CIFAR10 and FFHQ (see Figs. 1 and 2, and Table 1). Overfitting analysis of noisy input (Fig. 1 bottom row) can only show limited similarity between generated and training data (some ships and dogs in case of CIFAR10), while noise-free counterpart shows exact matching in each column.

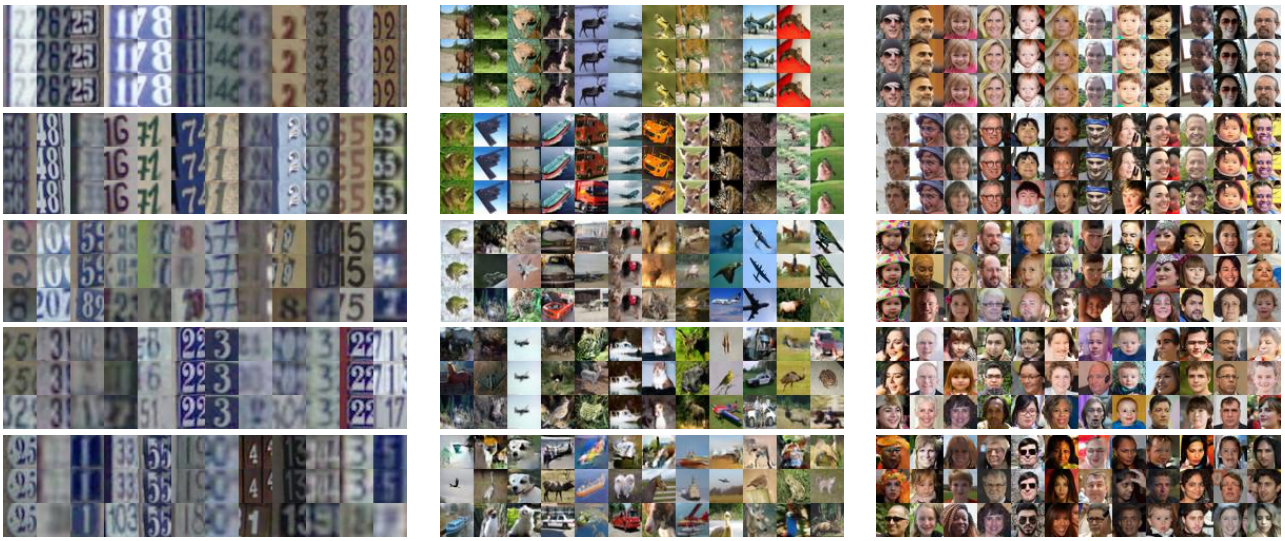

Fig. 1. Analysis of overfitting on SVHN (left), CIFAR10 (middle) and FFHQ (right) as a function of mini-batch size (3,200/800/200/50 from the top) and noisy latent space (bottom row). For each subfigure, the first row shows generated samples, while the second and third rows show the closest data samples in pixel space and latent space, respectively.

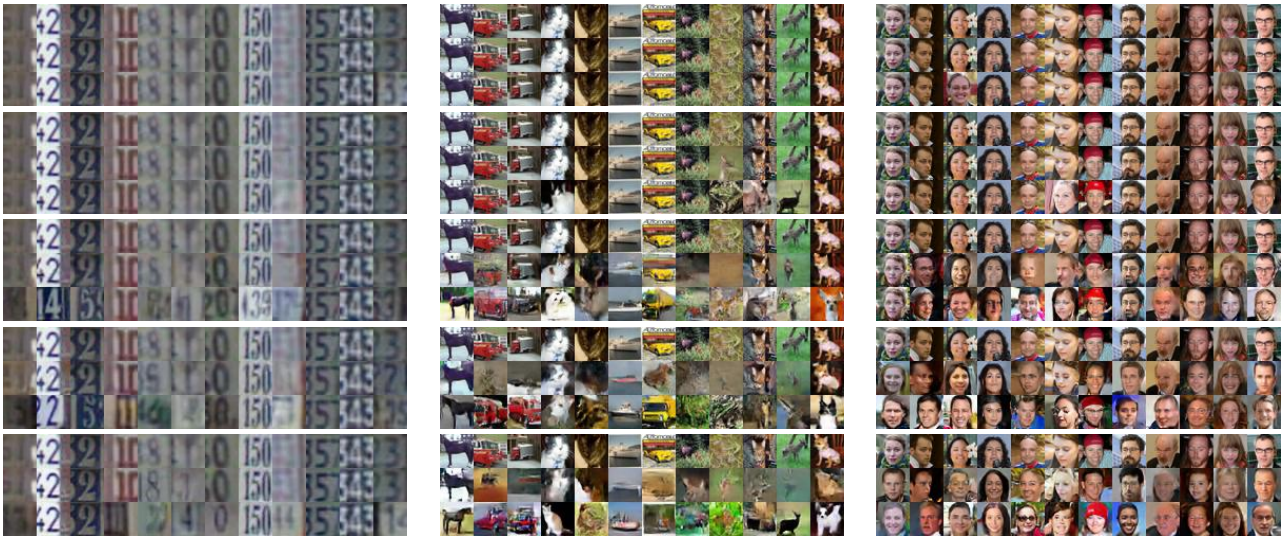

Fig. 2. Analysis of mode drop on SVHN (left), CIFAR10 (middle) and FFHQ (right) as a function of mini-batch size (3,200/800/200/50 from the top) and noisy latent space (bottom row). For each subfigure, the first row shows data samples, while the second and third rows show the closest generated samples in pixel space and latent space, respectively.

Similar behavior can be seen from the mode drop analysis (Fig. 2). SVHN results seem to be robust against the change in both analyses visually, but a significant difference can still be seen in Table 1.

A comparison of the two types of stochasticity (Table 1) shows that noisy input influences overfitting and mode drop more than the noise coming from a batch-size of 50 for CIFAR10 and FFHQ. Nevertheless, both types of stochasticity support our hypothesis. Furthermore, image generation does not seem to reach a satisfactory level of fidelity when noise is included, especially in CIFAR10.

6. CONCLUSIONS

We have shown empirically that GANs can overfit and show little to no mode drop when stochasticity is removed from training. Moreover, we have shown the trade-off between generalization and mode drop; it can be adjusted by changing the amount of stochasticity. We believe our observations on the effects of stochasticity in GAN training can benefit GAN design criteria and future research on this topic.

Acknowledgement: The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg).

7. REFERENCES

[1] S. Arora, A. Risteski, and Y. Zhang, "Do GANs learn the distribution? Some theory and empirics," in *Proc. ICLR*, 2018.

[2] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?" in *Proc. ICML*, 2018.

[3] 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 27*, 2014, pp. 2672–2680.

[4] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *Proc. ICLR*, 2019.

[5] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," in *Proc. ICLR*, 2018.

[6] R. Webster, J. Rabin, L. Simon, and F. Jurie, "Detecting overfitting of deep generative networks via latent recovery," *CoRR*, vol. abs/1901.03396, 2019.

[7] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," in *Proc. ICLR*, 2017.

[8] O. Bousquet, R. Livni, and S. Moran, "Passing tests without memorizing: Two models for fooling discriminators," *CoRR*, vol. abs/1902.03468, 2019.

[9] S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang, "Generalization and equilibrium in generative adversarial nets (GANs)," in *Proc. 34th ICML*, 2017.

[10] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *Proc. 5th ICLR*, 2017.

[11] C. Xiao, P. Zhong, and C. Zheng, "Bourgan: Generative networks with metric embeddings," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 2269–2280.

[12] M. Khayatkhoei, M. K. Singh, and A. Elgammal, "Disconnected manifold learning for generative adversarial networks," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 7343–7353.

[13] H. Thanh-Tung, T. Tran, and S. Venkatesh, "On catastrophic forgetting and mode collapse in generative adversarial networks," *CoRR*, vol. abs/1807.04015, 2018.

[14] T. Chavdarova, G. Gidel, F. Fleuret, and S. Lacoste-Julien, "Reducing noise in GAN training with variance reduced extragradient," *CoRR*, vol. abs/1904.08598, 2019.

[15] V. Nagarajan, C. Raffel, and I. Goodfellow, "Theoretical insights into memorization in GANs," in *Proc. NeurIPS Workshop*, 2018.

[16] T. Chen, X. Zhai, M. Ritter, M. Lucic, and N. Houlsby, "Self-supervised generative adversarial networks," *CoRR*, vol. abs/1811.11212, 2018.

[17] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop*, 2011.

[18] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 dataset." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[19] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *CoRR*, vol. abs/1812.04948, 2018.

[20] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015.

[21] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *Proc. ICLR*, 2018.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[23] Y. Yazici, C.-S. Foo, S. Winkler, K.-H. Yap, G. Piliouras, and V. Chandrasekhar, "The unusual effectiveness of averaging in GAN training," in *Proc. ICLR*, 2019.

[24] L. Theis, A. van den Oord, and M. Bethge, "A note on the evaluation of generative models," in *Proc. ICLR*, 2016.

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[26] 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 30*, 2017, pp. 6626–6637.

Appendix: Network Architectures

The models in the tables are the ones used for the experiments.
"LReLU" is LeakyReLU with $\alpha = 0.2$.

Table 2. Generator Architecture

Layers	Activation	Output Shape
Latent vector	-	512 x 1 x 1
Conv 4 x 4, st=1, pad=3	LReLU	1024 x 4 x 4
Upsample	-	1024 x 8 x 8
Conv 3 x 3, st=1, pad=1	LReLU	512 x 8 x 8
Upsample	-	512 x 16 x 16
Conv 3 x 3, st=1, pad=1	LReLU	256 x 16 x 16
Upsample	-	256 x 32 x 32
Conv 3 x 3, st=1, pad=1	Tanh	3 x 32 x 32

Table 3. Discriminator Architecture

Layers	Activation	Output Shape
Input image	-	3 x 32 x 32
Conv 5 x 5, st=2, pad=2	LReLU	128 x 16 x 16
Conv 5 x 5, st=2, pad=2	LReLU	256 x 8 x 8
Conv 5 x 5, st=2, pad=2	LReLU	512 x 4 x 4
Conv 5 x 5, st=1, pad=2	LReLU	1024 x 4 x 4
Conv 4 x 4, st=1, pad=0	Squeeze	1