MGAN: TRAINING GENERATIVE ADVERSARIAL NETS WITH MULTIPLE GENERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose in this paper a new approach to train the Generative Adversarial Nets (GANs) with a mixture of generators to overcome the mode collapsing problem. The main intuition is to employ multiple generators, instead of using a single one as in the original GAN. The idea is simple, yet proven to be extremely effective at covering diverse data modes, easily overcoming the mode collapsing problem and deliver state-of-the-art results. A minimax formulation was able to establish among a classifier, a discriminator, and a set of generators in a similar spirit with GAN. Generators create samples that are intended to come from the same distribution as the training data, whilst the discriminator determines whether samples are true data or generated by generators, and the classifier specifies which generator a sample comes from. The distinguishing feature is that internal samples are created from multiple generators, and then one of them will be randomly selected as final output similar to the mechanism of a probabilistic mixture model. We term our method *Mixture Generative Adversarial Nets* (MGAN). We develop theoretical analysis to prove that, at the equilibrium, the Jensen-Shannon divergence (JSD) between the mixture of generators' distributions and the empirical data distribution is minimal, whilst the JSD among generators' distributions is maximal, hence effectively avoiding the mode collapsing problem. By utilizing parameter sharing, our proposed model adds minimal computational cost to the standard GAN, and thus can also efficiently scale to large-scale datasets. We conduct extensive experiments on synthetic 2D data and natural image databases (CIFAR-10, STL-10 and ImageNet) to demonstrate the superior performance of our MGAN in achieving state-of-the-art Inception scores over latest baselines, generating diverse and appealing recognizable objects at different resolutions, and specializing in capturing different types of objects by the generators.

1 INTRODUCTION

Generative Adversarial Nets (GANs) (Goodfellow et al., 2014) are a recent novel class of deep generative models that are successfully applied to a large variety of applications such as image, video generation, image inpainting, semantic segmentation, image-to-image translation, and text-to-image synthesis, to name a few (Goodfellow, 2016). From the game theory metaphor, the model consists of a discriminator and a generator playing a two-player minimax game, wherein the generator aims to generate samples that resemble those in the training data whilst the discriminator tries to distinguish between the two as narrated in (Goodfellow et al., 2014). Training GAN, however, is challenging as it can be easily trapped into the mode collapsing problem where the generator only concentrates on producing samples lying on a few modes instead of the whole data space (Goodfellow, 2016).

Many GAN variants have been recently proposed to address this problem by modifying the objective function of the discriminator. The main approaches are to redefine the distance between the model and data distributions (Arjovsky et al., 2017), or to modify the discriminator to yield better gradient signals (Nguyen et al., 2017), (Warde-Farley & Bengio, 2016), (Zhao et al., 2016), (Metz et al., 2016). The common theme in these variants is that generators are shown, at equilibrium, to be able to recover the unknown data-generating distribution, but convergence remains elusive in practice. Most experiments were conducted on toy datasets or on narrow-domain datasets such as LSUN (Yu et al., 2015) or CelebA (Liu et al., 2015). To our knowledge, only (Warde-Farley & Bengio, 2016)

and (Nguyen et al., 2017) performed quantitative evaluation of models trained on much more diverse datasets such as STL-10 (Coates et al., 2011) and ImageNet (Russakovsky et al., 2015).

Given current limitations in the training of single-generator GANs, some very recent attempts were made following the mixture approach. (Tolstikhin et al., 2017) applied boosting techniques to train a mixture of generators by sequentially train and add new generators to the mixture. Sequentially training many generators is computational expensive. Moreover, this approach is built on the implicit assumption that a single-generator GAN can generate very good images of some modes so reweighing the training data and incrementally training new generators will result in a mixture that covers the whole data space. This assumption is not true in practice as current single-generator GANs trained on diverse data sets such as ImageNet (Russakovsky et al., 2015) tend to generate images of unrecognizable objects. (Arora et al., 2017) trained a mixture of generators and discriminators, and optimized the minimax game with the reward function being the weighted average reward function between any pair of generator and discriminator. This model is computationally expensive and lacks a mechanism to enforce the divergence among generators. (Ghosh et al., 2017) trained many generators by using a multi-class discriminator that, in addition to detecting whether a data sample is fake, predicts which generator generates the data sample. The objective function in this model punishes generators for generating samples that are detected as fake but does not directly encourage generators to specialize in generating different types of data.

We propose in this paper a novel approach to train a mixture of generators. Instead of training a single generator as in standard GANs, we train a set of generators with the objective that the mixture of them would approximate the data distribution. In order to force generators to generate different types of data, we use a classifier that estimates the probability that a sample comes from each of the generators. The result is a novel generative adversarial architecture formulated as a minimax game among three parties: a classifier, a discriminator, and a set of generators. We term our proposed model as *Mixture Generative Adversarial Nets (MGAN)*. We provide analysis that our model is optimized towards minimizing the Jensen-Shannon Divergence (JSD) between the mixture of generators' distributions and the unknown data-generating distribution while maximizing the JSD among generators.

Empirically, our proposed model can be trained efficiently by utilizing parameter sharing among generators, and between the classifier and the discriminator. In addition, simultaneously training many generators while enforcing JSD among generators helps them focus on some modes of the data space and learn better. Trained on CIFAR-10, each generator learned to specialize in generating samples from a different class such as horse, car, ship, dog, bird or airplane. Overall, the models trained on the CIFAR-10, STL-10 and ImageNet datasets successfully generated diverse, recognizable objects and achieved state-of-the-art Inception scores (Salimans et al., 2016). The model trained on the CIFAR-10 even outperformed GANs trained in a semi-supervised fashion (Salimans et al., 2016), (Odena et al., 2016).

In short, our main contributions are: (i) a novel adversarial model to efficiently train a mixture of generators while enforcing the JSD among the generators; (ii) a theoretical analysis that our objective function is optimized towards minimizing the JSD between the mixture of all generators' distributions and the real data distribution, while maximizing the JSD among generators; and (iii) a comprehensive evaluation on the performance of our method on both synthetic and diverse datasets including CIFAR-10, STL-10, and ImageNet.

2 Generative Adversarial Nets

Given the discriminator D and generator G, both parameterized via neural networks, training GAN can be formulated as the following minimax objective function:

$$\min_{G} \max_{D} \mathbb{E}_{\mathbf{x} \sim P_{data}(\mathbf{x})} \left[\log D\left(\mathbf{x}\right) \right] + \mathbb{E}_{\mathbf{z} \sim P_{\mathbf{z}}} \left[\log \left(1 - D\left(G\left(\mathbf{z}\right)\right) \right) \right] \tag{1}$$

where $\mathbf{x}$ is drawn from data distribution P_{data}, $\mathbf{z}$ is drawn from a prior distribution $P_{\mathbf{z}}$. The mapping $G\left(\mathbf{z}\right)$ induces a generator distribution P_{model} in data space. GAN alternatively optimizes D and G using stochastic gradient-based learning. As a result, the optimization order in 1 can be reversed, causing the minimax formulation to become maximin. G is therefore incentivized to map every $\mathbf{z}$ to a single $\mathbf{x}$ that is most likely to be classified as true data, leading to mode collapsing problem. Another commonly asserted cause of generating less diverse samples in GAN is that, at the optimal point of

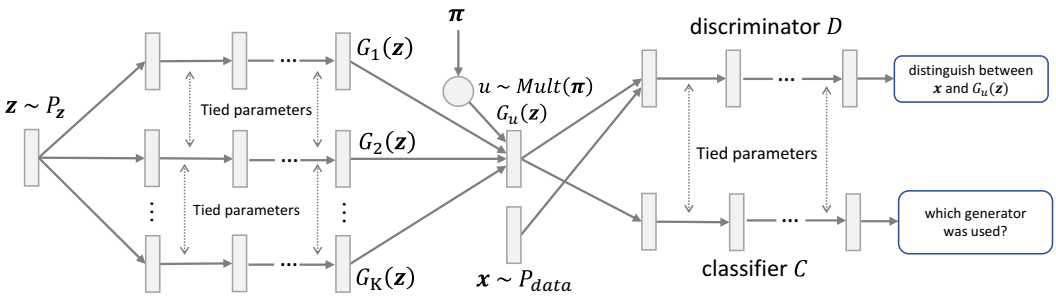


Figure 1: MGAN architecture with K generators, a binary discriminator, a multi-class classifier.

D, minimizing G is equivalent to minimizing the JSD between the data and model distributions, which has been empirically proven to prefer to generate samples around only a few modes whilst ignoring other modes (Huszár, 2015), (Theis et al., 2015).

3 Proposed Mixture GANs

We now present our main contribution of a novel approach that can effectively tackle mode collapse in GAN. Our idea is to use a mixture of many distributions rather than a single one as in the standard GAN, to approximate the data distribution, and simultaneously we enlarge the divergence of those distributions so that they cover different data modes.

To this end, an analogy to a game among K generators $G_{1:\mathrm{K}}$, a discriminator D and a classifier C can be formulated. Each generator G_k maps $\mathbf{z}$ to $\mathbf{x} = G_k(\mathbf{z})$, thus inducing a single distribution P_{G_k}; and K generators altogether induce a mixture over K distributions, namely P_{model} in the data space. An index u is drawn from a multinomial distribution $Mult(\boldsymbol{\pi})$ where $\boldsymbol{\pi} = [\pi_1, \pi_2, ..., \pi_{\mathrm{K}}]$ is the coefficients of the mixture; and then the sample $G_u(\mathbf{z})$ is used as the output. The discriminator D aims to distinguish between this sample and the training samples. The classifier C performs multi-class classification to classify samples labeled by the indices of their corresponding generators. We term this whole process and our model the *Mixture Generative Adversarial Nets* (MGAN).

Fig. 1 illustrates the general architecture of our proposed MGAN, where all components are parameterized by neural networks. G_ks tie their parameters together except the input layer, whilst C and D share parameters except the output layer. This parameter sharing scheme enables the networks to leverage their common information such as features at low-level layers that are close to the data layer, hence helps to train model effectively. In addition, it also minimizes the number of parameters and adds minimal complexity to the standard GAN, thus the whole process is still very efficient.

More formally, D, C and $G_{1:\mathrm{K}}$ now play the following multi-player minimax optimization game:

$$\min_{G_{1:\mathrm{K}}, C} \max_{D} \mathcal{J}(G_{1:\mathrm{K}}, C, D) = \mathbb{E}_{\mathbf{x}\sim P_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim P_{model}}[\log(1 - D(\mathbf{x}))] - \beta \left\{ \sum_{k=1}^{\mathrm{K}} \pi_k \mathbb{E}_{\mathbf{x}\sim P_{G_k}}[\log C_k(\mathbf{x})] - \sum_{k=1}^{\mathrm{K}} \pi_k \log \pi_k \right\} \quad (2)$$

where $C_k(\mathbf{x})$ is probability that $\mathbf{x}$ is generated by G_k and $\beta > 0$ is the diversity hyper-parameter. This should be recognized as a standard softmax loss for a multi-classification setting. The first two terms show the interaction between generators and the discriminator as in the standard GAN. The last term represents the interaction between generators and the classifier, which encourages each generator to produce data separable from those produced by other generators. The strength of this interaction is controlled by β. Similar to GAN, our proposed network can be trained by alternatively updating D, C and $G_{1:\mathrm{K}}$. We refer to Appendix A for the pseudo-code and algorithms for parameter learning for our proposed MGAN.

3.1 THEORETICAL ANALYSIS

Assuming all C, D and $G_{1:\mathrm{K}}$ have enough capacity, we show below that at the equilibrium, the JSD between the mixture induced by $G_{1:\mathrm{K}}$ and data distributions is minimal, i.e. $p_{data} = p_{model}$, and the JSD among K generators is maximal, i.e. two arbitrary generators almost never produce the same data. In what follows we present our mathematical statement and the sketch of their proofs. We refer to Appendix B for full derivations.

Proposition 1. *For fixed generators G_1, G_2, ..., G_{K} and their mixture weights $\pi_1, \pi_2, ..., \pi_{\mathrm{K}}$, the optimal solution $C^*_{1:\mathrm{K}}$ and D^* for $\mathcal{J}\left(G_{1:\mathrm{K}}, C, D\right)$ in Eq. (2) are:*

$$C^*_k\left(\mathbf{x}\right) = \frac{\pi_k p_{G_k}\left(\mathbf{x}\right)}{\sum_{j=1}^{\mathrm{K}} \pi_j p_{G_j}\left(\mathbf{x}\right)} \quad \text{and} \quad D^*\left(\mathbf{x}\right) = \frac{p_{data}\left(\mathbf{x}\right)}{p_{data}\left(\mathbf{x}\right) + p_{model}\left(\mathbf{x}\right)}$$

Proof. It can be seen that the solution C^*_k is a general case of D^* when D classifies samples from two distributions with equal weight of $1/2$. We refer the proofs for D^* to Proposition 1 in (Goodfellow et al., 2014), and our proof for C^* to Appendix B in this manuscript. □

Substituting $C^*_{1:\mathrm{K}}$ and D^* into Eq. (2), we reformulate the objective function for $G_{1:\mathrm{K}}$ as follows:

$$\begin{aligned}\mathcal{L}\left(G_{1:\mathrm{K}}\right) = &\mathbb{E}_{\mathbf{x}\sim P_{data}}\left[\log \frac{p_{data}\left(\mathbf{x}\right)}{p_{data}\left(\mathbf{x}\right) + p_{model}\left(\mathbf{x}\right)}\right] + \mathbb{E}_{\mathbf{x}\sim P_{model}}\left[\log \frac{p_{model}\left(\mathbf{x}\right)}{p_{data}\left(\mathbf{x}\right) + p_{model}\left(\mathbf{x}\right)}\right] \\ &- \beta\left\{\sum_{k=1}^{\mathrm{K}} \pi_k \mathbb{E}_{\mathbf{x}\sim P_{G_k}}\left[\log \frac{\pi_k p_{G_k}\left(\mathbf{x}\right)}{\sum_{j=1}^{\mathrm{K}} \pi_j p_{G_j}\left(\mathbf{x}\right)}\right] - \sum_{k=1}^{\mathrm{K}} \pi_k \log \pi_k\right\}\end{aligned} \tag{3}$$

The sum of the first two terms in Eq. (3) was shown in (Goodfellow et al., 2014) to be $2 \cdot \mathrm{JSD}\left(P_{data} \| P_{model}\right) - \log 4$. Appendix B shows the last term is $\beta \cdot \mathrm{JSD}_\pi\left(P_{G_1}, P_{G_2}, ..., P_{G_{\mathrm{K}}}\right) = \beta \cdot H\left(\sum_{k=1}^{\mathrm{K}} \pi_k p_{G_k}\right) - \beta \sum_{k=1}^{\mathrm{K}} \pi_k H\left(p_{G_k}\right)$, where $H\left(P\right)$is the Shannon entropy for distribution P. Thus, $\mathcal{L}\left(G_{1:\mathrm{K}}\right)$ can be rewritten as:

$$\mathcal{L}\left(G_{1:\mathrm{K}}\right) = -\log 4 + 2 \cdot \mathrm{JSD}\left(P_{data} \| P_{model}\right) - \beta \cdot \mathrm{JSD}_\pi\left(P_{G_1}, P_{G_2}, ..., P_{G_{\mathrm{K}}}\right) \tag{4}$$

Eq. (4) shows that $\mathcal{L}\left(G_{1:\mathrm{K}}\right)$ can be minimized by minimizing $\mathrm{JSD}\left(P_{data} \| P_{model}\right)$ or maximizing $\mathrm{JSD}_\pi\left(P_{G_1}, P_{G_2}, ..., P_{G_{\mathrm{K}}}\right)$ or both. The following lemma shows that $\mathrm{JSD}_\pi\left(P_{G_1}, P_{G_2}, ..., P_{G_{\mathrm{K}}}\right)$ is maximal when generators almost never produce the same data.

Lemma 2. $\mathrm{JSD}_\pi\left(P_1, ..., P_{\mathrm{K}}\right)$ *reaches its upper bound of* $\log K$ *if and only if* $\mathbb{E}_{P_i}\left[P_j\left(\mathbf{x}\right)\right] = 0 \; \forall i, j \in \left(1, ..., \mathrm{K}\right)$ *and* $\pi_1 = \pi_2 = ... = \pi_{\mathrm{K}} = \frac{1}{\mathrm{K}}$.

Proof. Refer to Appendix B. □

Theorem 3. *The global minimum of $\mathcal{L}\left(G_{1:\mathrm{K}}\right)$ is achieved if and only if $p_{data} = p_{model}$ and the Jensen-Shannon divergence $\mathrm{JSD}_\pi\left(P_{G_1}, P_{G_2}, ..., P_{G_{\mathrm{K}}}\right)$ is maximized. At that point, $\pi_1 = \pi_2 = ... = \pi_{\mathrm{K}} = \frac{1}{\mathrm{K}}$ and $\mathcal{L}\left(G_{1:\mathrm{K}}\right) = -\log 4 - \beta \log \mathrm{K}$.*

Proof. From Eq. (4) and Lemma 2, we have $\mathcal{L}\left(G_{1:\mathrm{K}}\right) \geq -\log 4 - \beta \log \mathrm{K}$. The global minimum is achieved if and only if $p_{data} = p_{model}$, $\mathbb{E}_{P_i}\left[P_j\left(\mathbf{x}\right)\right] = 0 \; \forall i, j \in \left(1, ..., \mathrm{K}\right)$, and $\pi_1 = \pi_2 = ... = \pi_{\mathrm{K}} = \frac{1}{\mathrm{K}}$. □

Theorem 3 shows that progressing towards equilibrium is equivalently to minimizing $\mathrm{JSD}\left(P_{data} \| P_{model}\right)$ while maximizing $\mathrm{JSD}_\pi\left(P_{G_1}, P_{G_2}, ..., P_{G_{\mathrm{K}}}\right)$. In practice, C, D, and $G_{1\mathrm{K}}$ are parameterized by neural networks and are optimized in the parameter space rather than in the function space. As all generators $G_{1:\mathrm{K}}$ share the same objective function, we can efficiently update their weights using the same backpropagation passes. Empirically, we set the parameter $\pi_k = \frac{1}{\mathrm{K}}, \forall k \in \left\{1, ..., \mathrm{K}\right\}$, which can be the global minimum of $\mathcal{L}\left(G_{1:\mathrm{K}}\right)$ in Theorem 3. To simplify the computational graph, we assume that each generator is sampled the same number of times in each minibatch. In addition, we adopt the non-saturating heuristic to train $G_{1:\mathrm{K}}$ by maximizing $\log D\left(G_k\left(\mathbf{z}\right)\right)$ instead of minimizing $\log D\left(1 - G_k\left(\mathbf{z}\right)\right)$ (Goodfellow et al., 2014).

4 RELATED WORK

Recent attempts to address the mode collapse by modifying the discriminator include minibatch discrimination (Salimans et al., 2016), Unrolled GAN (Metz et al., 2016) and Denoising Feature Matching (DFM) (Warde-Farley & Bengio, 2016). The idea of minibatch discrimination is to allow the discriminator to detect samples that are noticeably similar to other generated samples. Although this method can generate visually appealing samples, it is computationally expensive, thus normally used in the last hidden layer of discriminator. Unrolled GAN improves the learning by unrolling computational graph to include additional optimization steps of the discriminator. It could effectively reduce the mode collapsing problem, but the unrolling step is expensive, rendering it unscalable up to large-scale datasets. DFM augments the objective function of generator with one of a Denoising AutoEncoder (DAE) that minimizes the reconstruction error of activations at the penultimate layer of the discriminator. The idea is that gradient signals from DAE can guide the generator towards producing samples whose activations are close to the manifold of real data activations. DFM is surprisingly effective at avoiding mode collapse, but the involvement of a deep DAE adds considerable computational cost to the model.

An alternative approach is to train additional discriminators. D2GAN (Nguyen et al., 2017) employs two discriminators to minimize both Kullback-Leibler (KL) and reverse KL divergences, thus placing a fair distribution across the data modes. This method can avoid the mode collapsing problem to a certain extent, but still could not outperform DFM. Another work uses many discriminators to boost the learning of generator (Durugkar et al., 2016). The authors state that this method is robust to mode collapse, but did not provide experimental results to support that claim.

Another direction is to train multiple generators. The so-called MIX+GAN (Arora et al., 2017) is related to our model in the use of mixture but the idea is very different. Based on min-max theorem (Neumann, 1928), the MIX+GAN trains a mixture of multiple generators and discriminators with *different parameters* to play mixed strategies in a min-max game. The total reward of this game is computed by weighted averaging rewards over all pairs of generator and discriminator. The lack of parameter sharing renders this method computationally expensive to train. Moreover, there is no mechanism to enforce the divergence among generators as in ours.

The most closely related to ours is MAD-GAN (Ghosh et al., 2017) which trains many generators and uses a multi-class classifier as the discriminator. In this work, two strategies are proposed to address the mode collapse: (i) augmenting generator's objective function with a user-defined similarity based function to encourage different generators to generate diverse samples, and (ii) modifying discriminator's objective functions to push different generators towards different identifiable modes by separating samples of each generator. Our approach is different in that, rather than modifying the discriminator, we use an additional classifier that discriminates samples produced by each generator from those by others under multi-class classification setting. This nicely results in an optimization problem that maximizes the JSD among generators, thus naturally enforcing them to generate diverse samples and effectively avoiding mode collapse.

5 EXPERIMENTS

In this section, we conduct experiments on both synthetic data and real-world large-scale datasets. The aim of using synthetic data is to visualize, examine and evaluate the learning behaviors of our proposed MGAN, whilst using real-world datasets to quantitatively demonstrate its efficacy and scalability of addressing the mode collapse in a much larger and wider data space. For fair comparison, we use experimental settings that are identical to previous work, and hence we quoted the results from the latest state-of-the-art GAN-based models to compare with ours.

We use TensorFlow (et al., 2015) to implement our model and will release the code after publication. For all experiments, we use: (i) shared parameters among generators in all layers except for the weights from the input to the first hidden layer; (ii) shared parameters between the discriminator and classifier in all layers except for the weights from the penultimate layer to the output; (iii) Adam optimizer (Kingma & Ba, 2014) with learning rate of 0.0002 and the first-order momentum of 0.5; (iv) minibatch size of 64 samples for training the discriminators; (v) ReLU activations (Nair & Hinton, 2010) for the generators; (vi) Leaky ReLU (Maas et al., 2013) with slope of 0.2 for the discriminator and classifier; and (vii) weights initialized from an isotropic Gaussian: $\mathcal{N}(0, 0.02)$ and

zero biases. We refer to the appendix for more information about model architectures and additional experimental results.

5.1 SYNTHETIC DATA

In the first experiment, following (Nguyen et al., 2017) we reuse the experimental design proposed in (Metz et al., 2016) to investigate how well our MGAN can explore and capture multiple data modes. The training data is sampled from a 2D mixture of 8 isotropic Gaussian distributions with a covariance matrix of $0.02\boldsymbol{I}$ and means arranged in a circle of zero centroid and radius of 2.0. Our purpose of using such small variance is to create low density regions and separate the modes.

We employ 8 generators, each with a simple architecture of an input layer with 256 noise units drawn from isotropic multivariate Gaussian distribution $\mathcal{N}(0, \boldsymbol{I})$, and two fully connected hidden layers with 128 ReLU units each. For the discriminator and classifier, one hidden layer with 128 ReLU units is used. The diversity hyperparameter β is set to 0.125.

Fig. 2c shows the evolution of 512 samples generated by our model and baselines through time. It can be seen that the regular GAN generates data collapsing into a *single* mode hovering around the valid modes of data distribution, thus reflecting the mode collapse in GAN as expected. At the same time, UnrolledGAN (Metz et al., 2016), D2GAN (Nguyen et al., 2017) and our MGAN distribute data around *all* 8 mixture components, and hence demonstrating the abilities to successfully learn multimodal data in this case. Our proposed model, however, converges much faster than the other two since it successfully explores and neatly covers all modes at the early step 15K, whilst two baselines produce samples cycling around till the last steps. At the end, our MGAN captures data modes more precisely than UnrolledGAN and D2GAN as, in each mode, the UnrolledGAN generates data that concentrate only on several points around the mode's centroid, thus seems to produce fewer samples than ours whose samples fairly spread out the entire mode, but not exceed the boundary whilst the D2GAN still generates many points scattered between two adjacent modes.

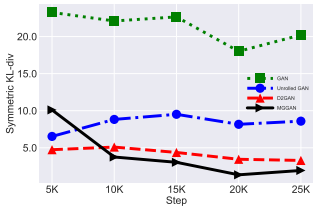
Next we further quantitatively compare the quality of generated data. Since we know the true distribution p_{data} in this case, we employ two measures, namely symmetric Kullback-Leibler (KL) divergence and Wasserstein distance. These measures compute the distance between the normalized histograms of 10,000 points generated from the model to true p_{data}. Figs. 2a and 2b again clearly demonstrate the superiority of our approach over GAN, UnrolledGAN and D2GAN w.r.t both distances (lower is better); notably the Wasserstein distances from ours and D2GAN's to the true distribution almost reduce to zero, and at the same time, our symmetric KL metric is significantly better than that of D2GAN. These figures also show the stability of our MGAN (black curves) and D2GAN (red curves) during training as it is much less fluctuating compared with GAN (green curves) and UnrolledGAN (blue curves).

Lastly, we perform experiments with different numbers of generators. The MGAN models with 2, 3, and 4 generators all successfully explore 8 modes but the models with more generators generate fewer points scattered between adjacent modes. We also examine the behavior of the diversity coefficient β by training the 4-generator model with different values of β. Without the JSD force ($\beta = 0$), generated samples cluster around one mode. When $\beta = 0.25$, the JSD force is weak and generated data cluster near 4 different modes. When $\beta = 0.75$ or 1.0, the JSD force is too strong and causes the generators to collapse, generating 4 increasingly tight clusters. When $\beta = 0.5$, generators successfully cover all of the 8 modes. To save space, please refer to Appendix C.1 for experiment details.

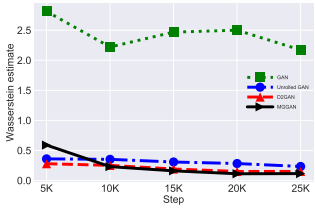
5.2 REAL-WORLD DATASETS

Next we train our proposed method on real-world databases from natural scenes to investigate its performance and scalability on much more challenging large-scale image data.

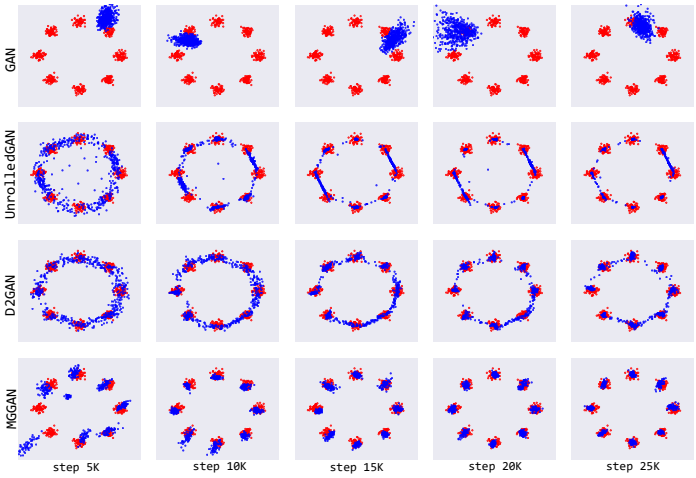
Datasets. We use three widely-adopted datasets: CIFAR-10 (Krizhevsky & Hinton, 2009), STL-10 (Coates et al., 2011) and ImageNet (Russakovsky et al., 2015). CIFAR-10 contains 50,000 32×32 training images of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. STL-10, subsampled from ImageNet, is a more diverse dataset than CIFAR-10, containing about 100,000 unlabeled 96×96 images. ImageNet (2012 release) presents the largest and most diverse



(a) Symmetric KL divergence.



(b) Wasserstein distance.



(c) Evolution of data (in blue) generated by GAN, UnrolledGAN, D2GAN and our MGAN from the top row to the bottom, respectively. Data sampled from the true mixture of 8 Gaussians are red.

Figure 2: The comparison of our MGAN and GAN's variants on 2D synthetic dataset.

consisting of over 1.2 million images from 1,000 classes. In order to facilitate fair comparison with baselines (Warde-Farley & Bengio, 2016), we follow the procedure of (Krizhevsky et al., 2012) to resize the STL-10 and ImageNet images down to 48×48 and 32×32, respectively.

Evaluation protocols. For quantitative evaluation, we adopt the Inception score proposed in (Salimans et al., 2016), which computes $\exp\left(\mathbb{E}_{\mathbf{x}}\left[KL\left(p\left(y|\mathbf{x}\right)\|p\left(y\right)\right)\right]\right)$ where $p\left(y|\mathbf{x}\right)$ is the conditional label distribution for the image $\mathbf{x}$ estimated by the reference Inception model (Szegedy et al., 2015). This metric rewards good and varied samples and is found to be well-correlated with human judgment (Salimans et al., 2016). We use the code provided in (Salimans et al., 2016) to compute the Inception scores for 10 partitions of 50,000 randomly generated samples. For qualitative demonstration of image quality obtained by our proposed model, we show samples generated by the mixture as well as samples generated by each generator. Samples are randomly drawn rather than cherry-picked.

Model architectures. Our generator and discriminator architectures closely follow the DCGAN's design (Radford et al., 2015). The only difference is we apply batch normalization (Ioffe & Szegedy, 2015) to all layers in the networks except for the output layer. Regarding the classifier, we empirically find that our proposed MGAN achieves the best performance (i.e., fast convergence rate and high inception score) when the classifier shares parameters of all layers with the discriminator except for the output layer. The reason is that this parameter sharing scheme would allow the classifier and discriminator to leverage their common features and representations learned at every layer, thus helps to improve and speed up the training progress. When the parameters are not tied, the model learns slowly and eventually yields lower performance.

During training we observe that the percentage of active neurons chronically declined (see Appendix C.2). One possible cause is that the batch normalization center (offset) is gradually shifted to the negative range, causing up to 45% of ReLU units of the generator networks to die. Our ad-hoc solution for this problem is to fix the offset at zero for all layers in the generator networks. The rationale is that for each feature map, the ReLU gates will open for about 50% highest inputs in a minibatch across all locations and generators, and close for the rest.

We also experiment with other activation functions of generator networks. First we use Leaky ReLU and obtain similar results with using ReLU. Then we use MaxOut units (Goodfellow et al., 2013) and achieves good Inception scores but generates unrecognizable samples. Finally, we try SeLU (Klambauer et al., 2017) but fail to train our model.

Table 1: Inception scores on different datasets. "–" denotes unavailable result.

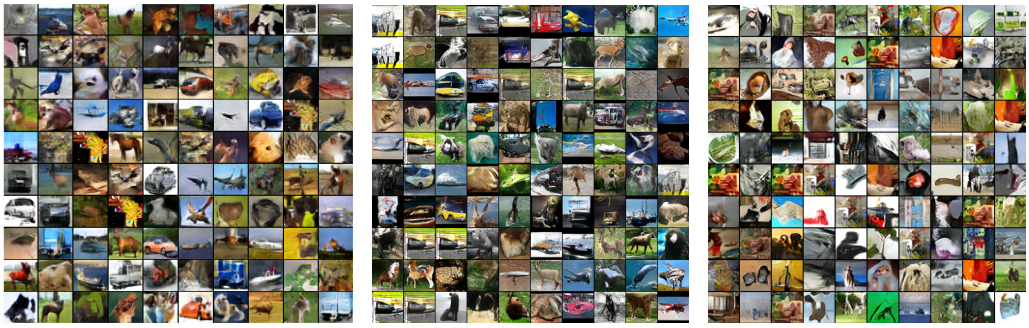
Model	CIFAR-10	STL-10	ImageNet
Real data	11.24±0.16	26.08±0.26	25.78±0.47
WGAN (Arjovsky et al., 2017)	3.82±0.06	–	–
MIX+WGAN (Arora et al., 2017)	4.04±0.07	–	–
Improved-GAN (Salimans et al., 2016)	4.36±0.04	–	–
ALI (Dumoulin et al., 2016)	5.34±0.05	–	–
BEGAN (Berthelot et al., 2017)	5.62	–	–
MAGAN (Wang et al., 2017)	5.67	–	–
GMAN (Durugkar et al., 2016)	6.00±0.19	–	–
DCGAN (Radford et al., 2015)	6.40±0.05	7.54	7.89
DFM (Warde-Farley & Bengio, 2016)	7.72±0.13	8.51±0.13	9.18±0.13
D2GAN (Nguyen et al., 2017)	7.15±0.07	7.98	8.25
MGAN	**8.33±0.10**	**9.22±0.11**	**9.32±0.10**

Hyperparameters. Three key hyperparameters of our model are: number of generators K, diversity controlling coefficient β and the minibatch size. We use a minibatch size of $[^{128}/_{\mathrm{K}}]$ for each generator, so that the total number of samples for training all generators is about 128. We train models with 4 generators and 10 generators corresponding with minibatch sizes of 32 and 12 each, and find that models with 10 generators performs better. For ImageNet, we try an additional setting with 32 generators and a minibatch size of 4 for each. The batch of 4 samples is too small for updating sufficient statistics coefficients of a batch-norm layer, thus we drop batch-norm in the input layer of each generator. This 32-generator model, however, does not obtain significantly better results than the 10-generator one. Therefore in what follows we only report the results of models with 10 generators. For the diversity coefficient β, we observe no significant difference in Inception scores when varying the value of β but the quality of generated images declines when β is too low or too high. Generated samples by each generator vary more when β is low, and vary less but become less realistic when β is high. We found a reasonable range for β to be $(0.01, 1.0)$ We finally set to 0.01 for CIFAR-10, 0.1 for ImageNet and 1.0 for STL-10.

Inception results. We now report the Inception scores obtained by our MGAN and baselines in Tab. 1 for a quantitative evaluation. It is worthy to note that only models trained in a completely unsupervised manner *without label* information are included for fair comparison; and DCGAN's and D2GAN's results on STL-10 are available only for the models trained on 32×32 resolution. Overall, our proposed model outperforms the baselines by large margins and achieves state-of-the-art performance on all datasets. Moreover, we would highlight that our MGAN obtains a score of 8.33 on CIFAR-10 that is even better than those of models trained *with labels* such as 8.09 of Improved GAN (Salimans et al., 2016) and 8.25 of AC-GAN (Odena et al., 2016). In addition, we train our model on the original 96×96 resolution of STL-10 and achieve a score of 9.79±0.08. This suggests the MGAN can be successfully trained on higher resolution images and achieve the higher Inception score.

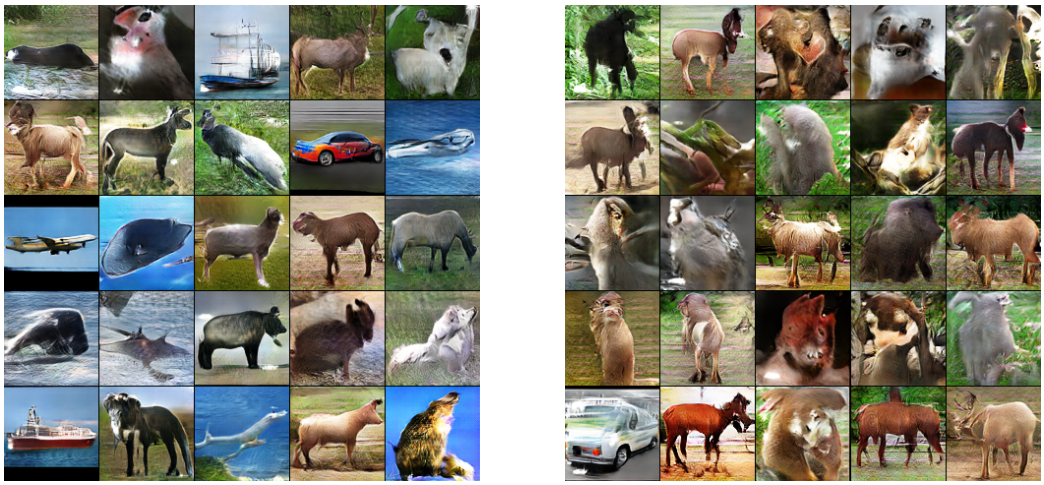
Image generation. Next we present samples randomly generated by our proposed model trained on the 3 datasets for qualitative assessment. Fig. 3a shows CIFAR-10 32×32 images containing a wide range of objects in such as airplanes, cars, trucks, ships, birds, horses or dogs. Similarly, STL-10 48×48 generated images in Fig. 3b include cars, ships, airplanes and many types of animals, but with wider range of different themes such as sky, underwater, mountain and forest. Images generated for ImageNet 32×32 are diverse with some recognizable objects such as lady, old man, birds, human eye, living room, hat or slippers, etc. Fig. 4a shows several cherry-picked STL-10 96×96 images, which demonstrate that the MGAN is capable of generating visually appealing images with complicated details. However, many samples are still incomplete and unrealistic as shown in Fig. 4b, leaving plenty of room for improvement.

Finally, we investigate samples generated by each generator as well as the evolution of these samples through numbers of training epochs. Fig. 5 shows images generated by each of the 10 generators in our MGAN trained on CIFAR-10 at epoch 20, 50, and 250 of training. Samples in each row correspond to a different generator. Generators start to specialize in generating different types of objects



(a) CIFAR-10 32×32. (b) STL-10 48×48. (c) ImageNet 32×32.

Figure 3: Images generated by our proposed MGAN trained on natural image datasets. Due to the space limit, please refer to the appendix for larger plots.



(a) Cherry-picked samples. (b) Incomplete, unrealistic samples.

Figure 4: Images generated by our MGAN trained on the original 96×96 STL10 dataset.

as early as epoch 20 and become more and more consistent: generator 2 and 3 in flying objects (birds and airplanes), generator 4 in full pictures of cats and dogs, generator 5 in portraits of cats and dogs, generator 8 in ships, generator 9 in car and trucks, and generator 10 in horses. Generator 6 seems to generate images of frog or animals in a bush. Generator 7, however, collapses in epoch 250. One possible explanation for this behavior is that images of different object classes tend to have different themes. Lastly, (Wang et al., 2016) noticed one of the causes for non-convergence in GANs is that the generators and discriminators constantly vary; the generators at two consecutive epochs of training generate significantly different images. This experiment demonstrates the effect of the JSD force in preventing generators from moving around the data space.

6 Conclusion

We have presented a novel adversarial model to address the mode collapse in GANs. Our idea is to approximate data distribution using a mixture of multiple distributions wherein each distribution captures a subset of data modes separately from those of others. To achieve this goal, we propose a minimax game of one discriminator, one classifier and many generators to formulate an optimization problem that minimizes the JSD between P_{data} and P_{model}, i.e., a mixture of distributions induced by the generators, whilst maximizes JSD among such generator distributions. This helps our model

(a) Epoch #20. (b) Epoch #50. (c) Epoch #250.

Figure 5: Images generated by our MGAN trained on CIFAR10 at different epochs. Samples in each row correspond to a different generator.

generate diverse images to better cover data modes, thus effectively avoids mode collapse. We term our proposed model *Mixture GAN* (MGAN).

The MGAN can be efficiently trained by sharing parameters between its discriminator and classifier, and among its generators, thus our model is scalable to be evaluated on real-world large-scale datasets. Comprehensive experiments on synthetic 2D data, CIFAR-10, STL-10 and ImageNet databases demonstrate the following capabilities of our model: (i) achieving state-of-the-art Inception scores; (ii) generating diverse and appealing recognizable objects at different resolutions; and (iv) specializing in capturing different types of objects by the generators.

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. 1, 1

Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (gans). *arXiv preprint arXiv:1703.00573*, 2017. 1, 4, 1

David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017. 1

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011. 1, 5.2

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016. 1

Ishan Durugkar, Ian Gemp, and Sridhar Mahadevan. Generative multi-adversarial networks. *arXiv preprint arXiv:1611.01673*, 2016. 4, 1

Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org. 5

Arnab Ghosh, Viveka Kulharia, Vinay Namboodiri, Philip HS Torr, and Puneet K Dokania. Multi-agent diverse generative adversarial networks. *arXiv preprint arXiv:1704.02906*, 2017. 1, 4

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016. 1, B

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*, pp. 2672–2680, 2014. 1, 3.1, 3.1, 3.1, B

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013. 5.2

Ferenc Huszár. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*, 2015. 2

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015. 5.2

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017. 5.2

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 5.2

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012. 5.2

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3730–3738, 2015. 1

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013. 5

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016. 1, 4, 5.1

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010. 5

J v Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320, 1928. 4

Tu Dinh Nguyen, Trung Le, Hung Vu, and Dinh Phung. Dual discriminator generative adversarial nets. In *Advances in Neural Information Processing Systems 29 (NIPS)*, pp. accepted, 2017. 1, 4, 5.1, 1

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016. 1, 5.2

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 5.2, 1

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1, 5.2

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*, pp. 2234–2242, 2016. 1, 4, 5.2, 1, 5.2

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015. 5.2

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015. 2

Ilya Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann Simon-Gabriel, and Bernhard Schölkopf. Adagan: Boosting generative models. *arXiv preprint arXiv:1701.02386*, 2017. 1

Ruohan Wang, Antoine Cully, Hyung Jin Chang, and Yiannis Demiris. Magan: Margin adaptation for generative adversarial networks. *arXiv preprint arXiv:1704.03817*, 2017. 1

Yaxing Wang, Lichao Zhang, and Joost van de Weijer. Ensembles of generative adversarial networks. *arXiv preprint arXiv:1612.00991*, 2016. 5.2

David Warde-Farley and Yoshua Bengio. Improving generative adversarial networks with denoising feature matching. 2016. 1, 4, 5.2, 1

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015. 1

Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016. 1

A Appendix: Framework

In our proposed method, generators G_1, G_2, ... G_{K} are deep convolutional neural networks parameterized by θ_G. These networks share parameters in all layers except for the input layers. The input layer for generator G_k is parameterized by the mapping $f_{\theta_G,k}(\mathbf{z})$ that maps the sampled noise $\mathbf{z}$ to the first hidden layer activation $\mathbf{h}$. The shared layers are parameterized by the mapping $g_{\theta_G}(\mathbf{h})$ that maps the first hidden layer to the generated data. The pseudo-code of sampling from the mixture is described in Alg. 1. Classifier C and classifier D are also deep convolutional neural networks that are both parameterized by θ_{CD}. They share parameters in all layers except for the last layer. The pseudo-code of learning θ_G and θ_{CD} is described in 2. θ_G and θ_{CD} are alternatively updated using stochastic gradient descend.

Algorithm 1 Sampling from MGAN's mixture of generators.

1: Sample noise $\mathbf{z}$ from the prior $p_{\mathbf{z}}$.
2: Sample a generator index from $[1, 2, ..., K]$ using the mixing probability π
3: $\mathbf{h} = f_{\theta_G,k}(\mathbf{z})$
4: $\mathbf{x} = g_{\theta_G}(\mathbf{h})$
5: Return generated data $\mathbf{x}$ and label k

Algorithm 2 Alternative training of MGAN using stochastic gradient descent.

1: **for** number of training iterations **do**
2: Sample a minibatch of M data points $\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(M)}\right)$ from the data distribution p_{data}.
3: Sample a minibatch of N generated data points $\left(\mathbf{x'}^{(1)}, \mathbf{x'}^{(2)}, ..., \mathbf{x'}^{(N)}\right)$ and N labels $(k_1, k_2, ..., k_N)$ from the current mixture.
4: $\mathcal{L}_C = -\frac{1}{N}\sum_{m=1}^{N} \log C_{k_m}\left(\mathbf{x'}^{(m)}\right)$
5: $\mathcal{L}_D = -\frac{1}{M}\sum_{m=1}^{M} \log D\left(\mathbf{x}^{(m)}\right) - \frac{1}{N}\sum_{m=1}^{N} \log\left[1 - D\left(\mathbf{x'}^{(m)}\right)\right]$
6: Update classifier C and discriminator D by descending along their gradient: $\nabla_{\theta_{CD}}\left(\mathcal{L}_C + \mathcal{L}_D\right)$
7: Sample a minibatch of N generated data points $\left(\mathbf{x'}^{(1)}, \mathbf{x'}^{(2)}, ..., \mathbf{x'}^{(N)}\right)$ and N labels $(k_1, k_2, ..., k_N)$ from the current mixture.
8: $\mathcal{L}_G = -\frac{1}{N}\sum_{m=1}^{N} \log D\left(\mathbf{x'}^{(m)}\right) - \frac{\beta}{N}\sum_{m=1}^{N} \log C_{k_m}\left(\mathbf{x'}^{(m)}\right)$
9: Update the mixture of generators G by ascending along its gradient: $\nabla_{\theta_G}\mathcal{L}_{\mathcal{G}}$
10: **end for**

B APPENDIX: PROOFS FOR SECTION 3.1

Proposition 1 (Proposition 1 restated). For fixed generators G_1, G_2, ..., G_K and mixture weights $\pi_1, \pi_2, ..., \pi_K$, the optimal discriminators C^* and D^* are:

$$C_k^*(\mathbf{x}) = \frac{\pi_k p_{G_k}(\mathbf{x})}{\sum_{j=1}^K \pi_j p_{G_j}(\mathbf{x})}$$
$$D^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_{model}(\mathbf{x})}$$

Proof. The optimal D^* was proved in (Goodfellow, 2016). This section shows a similar proof for the optimal C^*. Assuming that C^* can be optimized in the functional space, we can calculate the functional derivatives of $\mathcal{J}(G, C, D)$with respect to each $C_k(\mathbf{x})$ for $k \in (2, ..., K)$ and set them equal to zero:

$$\begin{aligned}\frac{\delta \mathcal{J}}{\delta C_k(\mathbf{x})} &= -\beta \frac{\delta}{\delta C_k(\mathbf{x})} \int \left(\pi_1 p_{G_1}(\mathbf{x}) \log \left(1 - \sum_{k=2}^K C_k(\mathbf{x})\right) + \sum_{k=2}^K \pi_k p_{G_k}(\mathbf{x}) \log C_k(\mathbf{x}) \right) dx \\ &= -\beta \left(\frac{\pi_k p_{G_k}(\mathbf{x})}{C_k(\mathbf{x})} - \frac{\pi_1 p_{G_1}(\mathbf{x})}{C_1(\mathbf{x})} \right) \end{aligned} \tag{5}$$

Setting $\frac{\delta \mathcal{J}(G,C,D)}{\delta C_k(\mathbf{x})}$ to 0 for $k \in (2, ..., K)$, we get:

$$\frac{\pi_1 p_{G_1}(\mathbf{x})}{C_1^*(\mathbf{x})} = \frac{\pi_2 p_{G_2}(\mathbf{x})}{C_2^*(\mathbf{x})} = ... = \frac{\pi_K p_{G_K}(\mathbf{x})}{C_K^*(\mathbf{x})} \tag{6}$$

$C_k^*(\mathbf{x}) = \frac{\pi_k p_{G_k}(\mathbf{x})}{\sum_{j=1}^K \pi_j p_{G_j}(\mathbf{x})}$ results from 6 due to the fact that $\sum_{k=1}^K C_k^*(\mathbf{x}) = 1$. □

Reformulation of $\mathcal{L}(G)$. Replacing the optimal C^* and D^* into 2, we can reformulate the objective function for the generator:

$$\begin{aligned}\mathcal{L}(G) &= \mathcal{J}(G, C^*, D^*) \\ &= \mathbb{E}_{p_{data}}\left[\log \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_{model}(\mathbf{x})}\right] + \mathbb{E}_{p_{model}}\left[\log \frac{p_{model}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_{model}(\mathbf{x})}\right] \\ &\quad - \beta \left\{ \sum_{k=1}^K \pi_k \mathbb{E}_{p_{G_k}}\left[\log \frac{\pi_k p_{G_k}(\mathbf{x})}{\sum_{j=1}^K \pi_j p_{G_j}(\mathbf{x})}\right] - \sum_{k=1}^K \pi_k \log \pi_k \right\} \end{aligned} \tag{7}$$

The sum of the first two terms in 7 was shown in (Goodfellow et al., 2014) to be $2 \cdot JSD(p_{data} \| p_{model}) - \log 4$. The last term $\beta\{*\}$ of 7 is the JSD for the K distributions:

$$\begin{aligned} * &= \sum_{k=1}^K \pi_k \mathbb{E}_{p_{G_k}}\left[\log \frac{\pi_k p_{G_k}(\mathbf{x})}{\sum_{j=1}^K \pi_j p_{G_j}(\mathbf{x})}\right] - \sum_{k=1}^K \pi_k \log \pi_k \\ &= \sum_{k=1}^K \pi_k \mathbb{E}_{p_{G_k}}\left[\log p_{G_k}(\mathbf{x})\right] - \sum_{k=1}^K \pi_k \mathbb{E}_{p_{G_k}}\left[\log \sum_{j=1}^K \pi_j p_{G_j}(\mathbf{x})\right] \\ &= -\sum_{k=1}^K \pi_k H(p_{G_k}) + H\left(\sum_{j=1}^K \pi_j p_{G_j}(\mathbf{x})\right) \\ &= JSD_\pi(p_{G_1}, p_{G_2}, ..., p_{G_k}) \end{aligned} \tag{8}$$

Thus, $\mathcal{L}(G)$ can be rewritten as:

$$\mathcal{L}(G) = -\log 4 + 2 \cdot JSD(p_{data} \| p_{model}) - \beta JSD_\pi(p_{G_1}, p_{G_2}, ..., p_{G_K})$$

Lemma 2 (Lemma 2 restated). $JSD_\pi(P_1, ..., P_K)$ reaches its upper bound of $\log K$ if and only if $\mathbb{E}_{P_i}[P_j(\mathbf{x})] = 0 \; \forall i, j \in (1, ..., K)$ and $\pi_1 = \pi_2 = ... = \pi_K = \frac{1}{K}$.

Proof. From 8, we have:

$$JSD_{\pi}\left(P_1, ..., P_K\right) = \sum_{i=1}^{K} \pi_i \mathbb{E}_{P_i}\left[\log \frac{\pi_i P_i\left(\mathbf{x}\right)}{\sum_{j=1}^{K} \pi_j P_j\left(\mathbf{x}\right)}\right] - \sum_{k=1}^{K} \pi_i \log \pi_i \tag{9}$$

Consider each of the expectations in 9:

$$\begin{aligned}\mathbb{E}_{P_i}\left[\log \frac{\pi_i P_i\left(\mathbf{x}\right)}{\sum_{j=1}^{K} \pi_j P_j\left(\mathbf{x}\right)}\right] &= \mathbb{E}_{P_i}\left[\mathbb{I}_{\sum_{j\neq i} \pi_j P_j(\mathbf{x})>0} \log \frac{\pi_i P_i\left(\mathbf{x}\right)}{\sum_{j=1}^{K} \pi_j P_j\left(\mathbf{x}\right)}\right] \\ &\quad + \mathbb{E}_{P_i}\left[\mathbb{I}_{\sum_{j\neq i} \pi_j P_j(\mathbf{x})=0} \log \frac{\pi_i P_i\left(\mathbf{x}\right)}{\sum_{j=1}^{K} \pi_j P_j\left(\mathbf{x}\right)}\right] \\ &= \mathbb{E}_{P_i}\left[\mathbb{I}_{\sum_{j\neq i} \pi_j P_j(\mathbf{x})>0} \log \frac{\pi_i P_i\left(\mathbf{x}\right)}{\sum_{j=1}^{K} \pi_j P_j\left(\mathbf{x}\right)}\right]\end{aligned}$$

where $\mathbb{I}\left(\cdot\right)$ be the indicator function. As $\log \frac{\pi_i P_i(\mathbf{x})}{\sum_{j=1}^{K} \pi_j P_j(\mathbf{x})} < 0$ where $\sum_{j\neq i} \pi_j P_j\left(\mathbf{x}\right) > 0$, $\mathbb{E}_{P_i}\left[\log \frac{\pi_i P_i(\mathbf{x})}{\sum_{j=1}^{K} \pi_j P_j(\mathbf{x})}\right]$ reaches its upper bound of 0 if and only if $\mathbb{E}_{P_i}\left[\mathbb{I}_{\sum_{j\neq i} \pi_j P_j(\mathbf{x})>0}\right] = \mathbb{E}_{P_i}\left[\sum_{j\neq i} \pi_j P_j\left(\mathbf{x}\right)\right] = \sum_{j\neq i} \pi_j \mathbb{E}_{P_i}\left[P_j\left(\mathbf{x}\right)\right] = 0$. The last equation occurs if and only if $\mathbb{E}_{P_i}\left[P_j\left(\mathbf{x}\right)\right] = 0\ \forall j$. Thus, $JSD_{\pi}\left(P_1, ..., P_K\right) \leq -\sum_{k=1}^{K} \pi_i \log \pi_i$. By log sum inequality, $-\sum_{k=1}^{K} \pi_i \log \pi_i \leq -\log \frac{1}{K} = \log K$. Therefore, $JSD_{\pi}\left(P_1, ..., P_K\right) \leq \log K$. The equality occurs if and only if $\mathbb{E}_{P_i}\left[P_j\left(\mathbf{x}\right)\right] = 0\ \forall i, j$ and $\pi_1 = \pi_2 = ... = \pi_K = \frac{1}{K}$. □

C Appendix: Additional Experiments

C.1 Synthetic 2D Gaussian data

The true data is sampled from a 2D mixture of 8 Gaussian distributions with a covariance matrix $0.02\boldsymbol{I}$ and means arranged in a circle of zero centroid and radius 2.0. We use a simple architecture of 8 generators with two fully connected hidden layers and a classifier and a discriminator with one shared hidden layer. All hidden layers contain the same number of 128 ReLU units. The input layer of generators contains 256 noise units sampled from isotropic multivariate Gaussian distribution $\mathcal{N}\left(0, \boldsymbol{I}\right)$. We do not use batch normalization in any layer. We refer to Table 2 for more specifications of the network and hyperparameters. "Shared" is short for parameter sharing among generators or between the classifier and the discriminator. Feature maps of 8/1 in the last layer for C and D means that two separate fully connected layers are applied to the penultimate layer, one for C that outputs 8 logits and another for D that outputs 1 logit.

The effect of numbers of generators on generated samples.

Figure 6 shows generated samples by the MGAN models with different number of generators trained on synthetic data for 25,000 epochs. The model with 1 generator behaves similarly to the standard GAN. The models with 2, 3 and 4 generators all successfully cover 8 modes but the models with more generators generate fewer points scattered between adjacent modes. Finally, the model with 10 generators also cover 8 modes with 2 generators sharing one mode and one generator hovering around another mode.

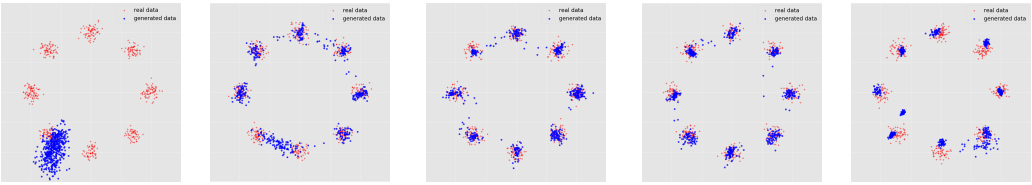
The effect of β on generated samples. To examine the behavior of the diversity coefficient β, figure 7 compares generated samples by 4 generators after 25,000 epochs of training with different values of β. Without the JSD force ($\beta = 0$), generated samples cluster around one mode. When $\beta = 0.25$, generated data cluster near 4 different modes. When $\beta = 0.75$ or 1.0, the JSD force is too strong and causes the generators to collapse, generating 4 increasingly tight clusters. When $\beta = 0.5$, generators successfully cover all of the 8 modes.

C.2 Real-world datasets

Fixing Batch Normalization Center

Table 2: Network architecture and hyperparameters for 2D Gaussian data.

Operation	Feature maps	Nonlinearity	Shared?
$G(\mathbf{z}) : \mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$	256		
Fully connected	128	ReLU	$\times$
Fully connected	128	ReLU	$\surd$
Fully connected	2	Linear	$\surd$
$C(\mathbf{x}), D(\mathbf{x})$	2		
Fully connected	128	ReLU	$\surd$
Fully connected	8/1	Linear	$\times$
Number of generators	8		
Batch size for real data	512		
Batch size for each generator	128		
Number of iterations	20,000		
Leaky ReLU slope	0.2		
Learning rate	0.0002		
Regularization constants	$\beta = 0.125$		
Optimizer	Adam($\beta_1 = 0.5, \beta_2 = 0.999$)		
Weight, bias initialization	$\mathcal{N}(\mu = 0, \sigma = 0.02)$, 0		



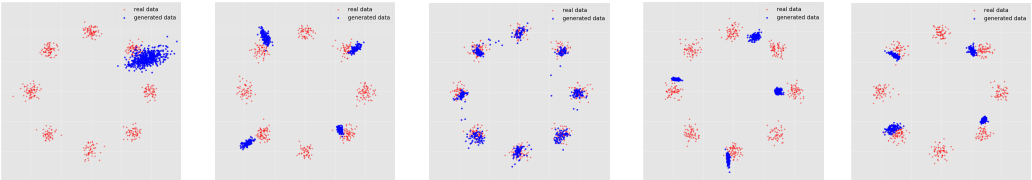
(a) 1 generator. (b) 2 generators. (c) 3 generators. (d) 4 generators. (e) 10 generators.

Figure 6: Samples generated by MGAN models trained on synthetic data with 2, 3 and 4 generators. Generated data are in blue and data samples from the 8 Gaussians are in red.

During training we observe that the percentage of active neurons, which we define as ReLU units with positive activation for at least 10% of samples in the minibatch, chronically declined. Figure 8a shows the percentage of active neurons in generators trained on CIFAR-10 declined consistently to 55% in layer 2 and 60% in layer 3. Therefore, the quality of generated images, after reaching the peak level, started declining. One possible cause is that the batch normalization center (offset) is gradually shifted to the negative range as shown in the histogram 8b. We also observe the same problem in DCGAN. Our ad-hoc solution for this problem, i.e., we fix the offset at zero for all layers in the generator networks. The rationale is that for each feature map, the ReLU gates will open for about 50% highest inputs in a minibatch across all locations and generators, and close for the rest. Therefore, batch normalization can keep ReLU units alive even when most of their inputs are otherwise negative, and introduces a form of competition that encourages generators to "specialize" in different features. This measure significantly improves performance but does not totally solve the dying ReLUs problem. We found that late in the training, the input to generators' ReLU units became more and more right-skewed, causing the ReLU gates to open less and less often.

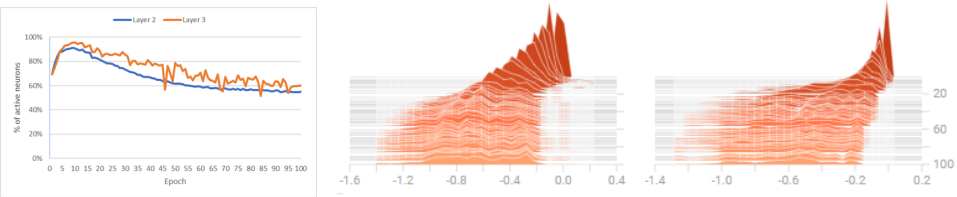
Experiment Settings

For the experiments on three large-scale natural scene datasets (CIFAR-10, STL-10, ImageNet), we closely followed the network architecture and training procedure of DCGAN. The specifications of our models trained on CIFAR-10, STL-10 48×48, STL-10 96×96 and ImageNet datasets are described in Tables. (3, 4, 5, 6), respectively. "BN" is short for batch normalization and "BN center" is short for whether to learn batch normalization's center or set it at zero. "Shared" is short for parameter sharing among generators or between the classifier and the discriminator. Feature maps of 10/1 in the last layer for C and D means that two separate fully connected layers are applied to the penultimate layer, one for C that outputs 10 logits and another for D that outputs 1 logit. Finally,



(a) $\beta = 0.0$ (b) $\beta = 0.25$ (c) $\beta = 0.5$ (d) $\beta = 0.75$ (e) $\beta = 1.0$

Figure 7: Samples generated by MGAN models trained on synthetic data with different values of the diversity coefficient β. Generated data are in blue and data samples from the 8 Gaussians are in red.



(a) % of active neurons in layer 2 and 3.

(b) Histogram of batch normalization centers in layer 2 (left) and 3 (right).

Figure 8: Observation of activate neuron rates and batch normalization centers in MGAN's generators trained on CIFAR-10.

Figures (9, 10, 11, 12, 13) respectively are the enlarged version of Figures (3a, 3b, 3c, 4a, 4b) in the main manuscript.

Table 3: Network architecture and hyperparameters for the CIFAR-10 dataset.

Operation	Kernel	Strides	Feature maps	BN?	BN center?	Nonlinearity	Shared?
$G(\mathbf{z}) : \mathbf{z} \sim \text{Uniform}[-1, 1]$			100				
Fully connected			4×4×512	√	×	ReLU	×
Transposed convolution	5×5	2×2	256	√	×	ReLU	√
Transposed convolution	5×5	2×2	128	√	×	ReLU	√
Transposed convolution	5×5	2×2	3	×	×	Tanh	√
$C(\mathbf{x}), D(\mathbf{x})$			32×32×3				
Convolution	5×5	2×2	128	√	√	Leaky ReLU	√
Convolution	5×5	2×2	256	√	√	Leaky ReLU	√
Convolution	5×5	2×2	512	√	√	Leaky ReLU	√
Fully connected			10/1	×	×	Softplus	×
Number of generators	10						
Batch size for real data	64						
Batch size for each generator	12						
Number of iterations	250						
Leaky ReLU slope	0.2						
Learning rate	0.0002						
Regularization constants	$\beta = 0.01$						
Optimizer	Adam($\beta_1 = 0.5, \beta_2 = 0.999$)						
Weight, bias initialization	$\mathcal{N}(\mu = 0, \sigma = 0.01), 0$						

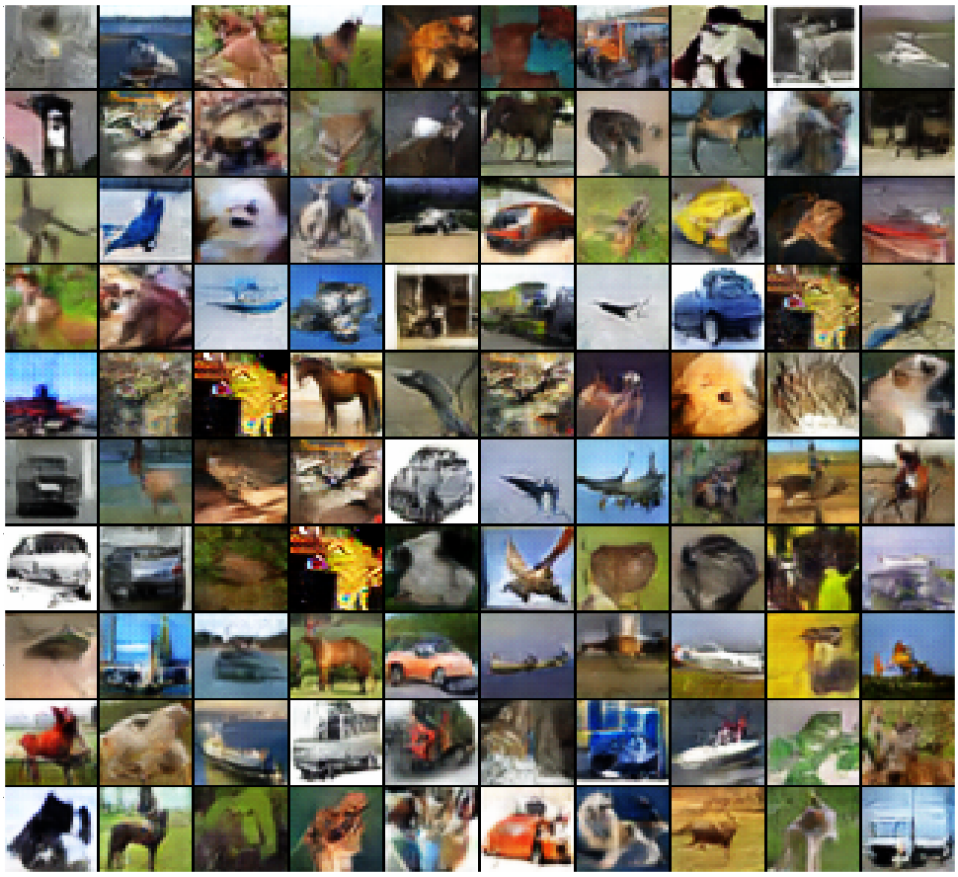


Figure 9: Images generated by MGAN trained on the CIFAR-10 dataset.

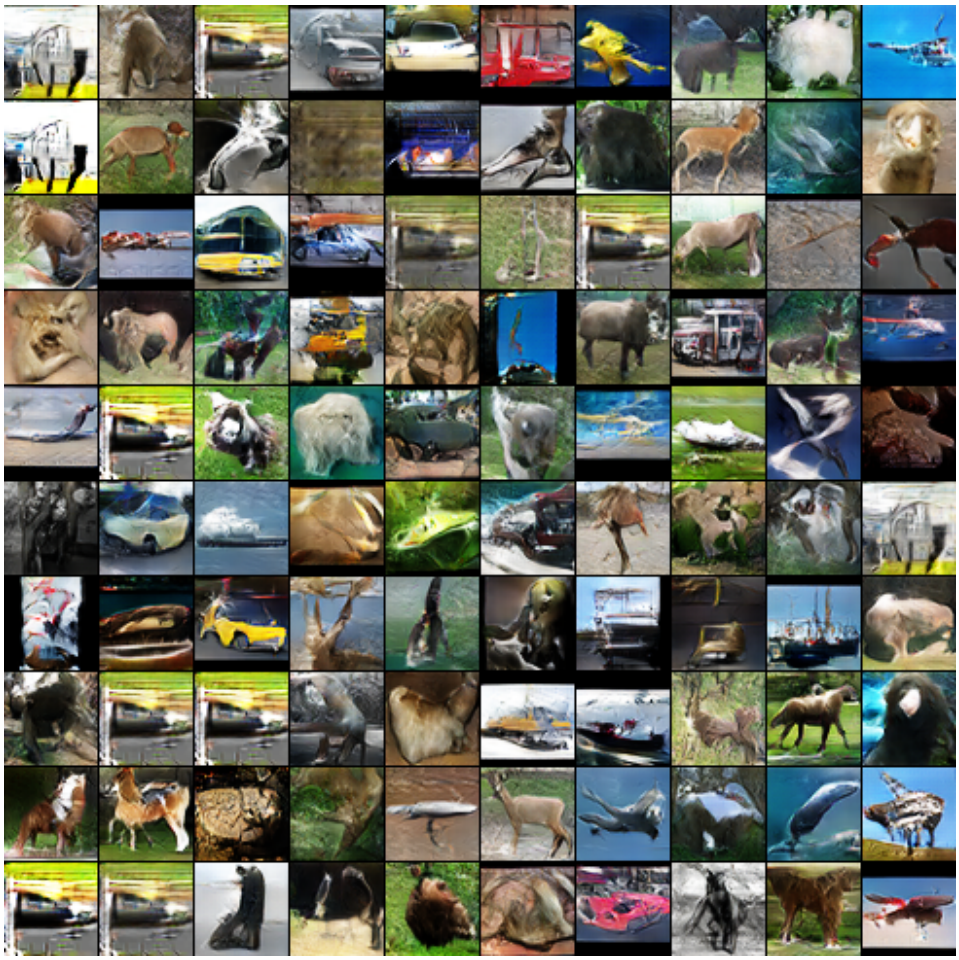


Figure 10: Images generated by MGAN trained on the rescaled 48×48 STL-10 dataset.

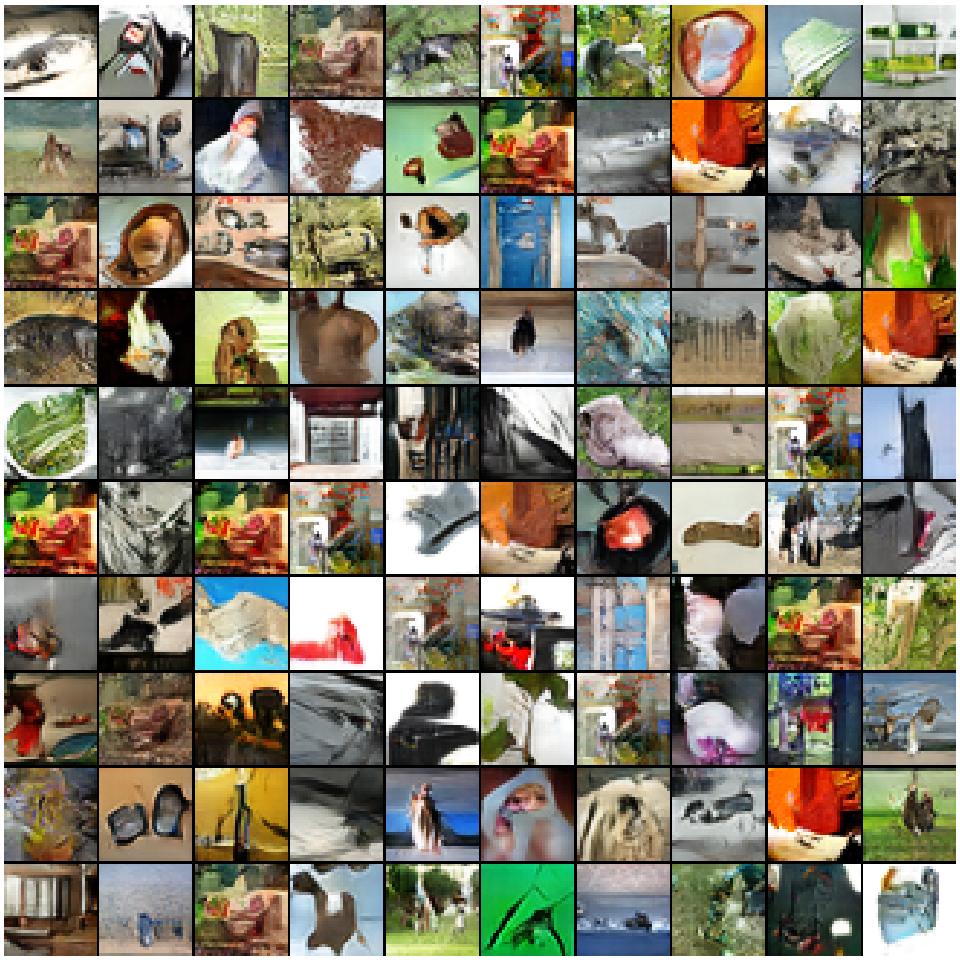


Figure 11: Images generated by MGAN trained on the rescaled 32×32 ImageNet dataset.

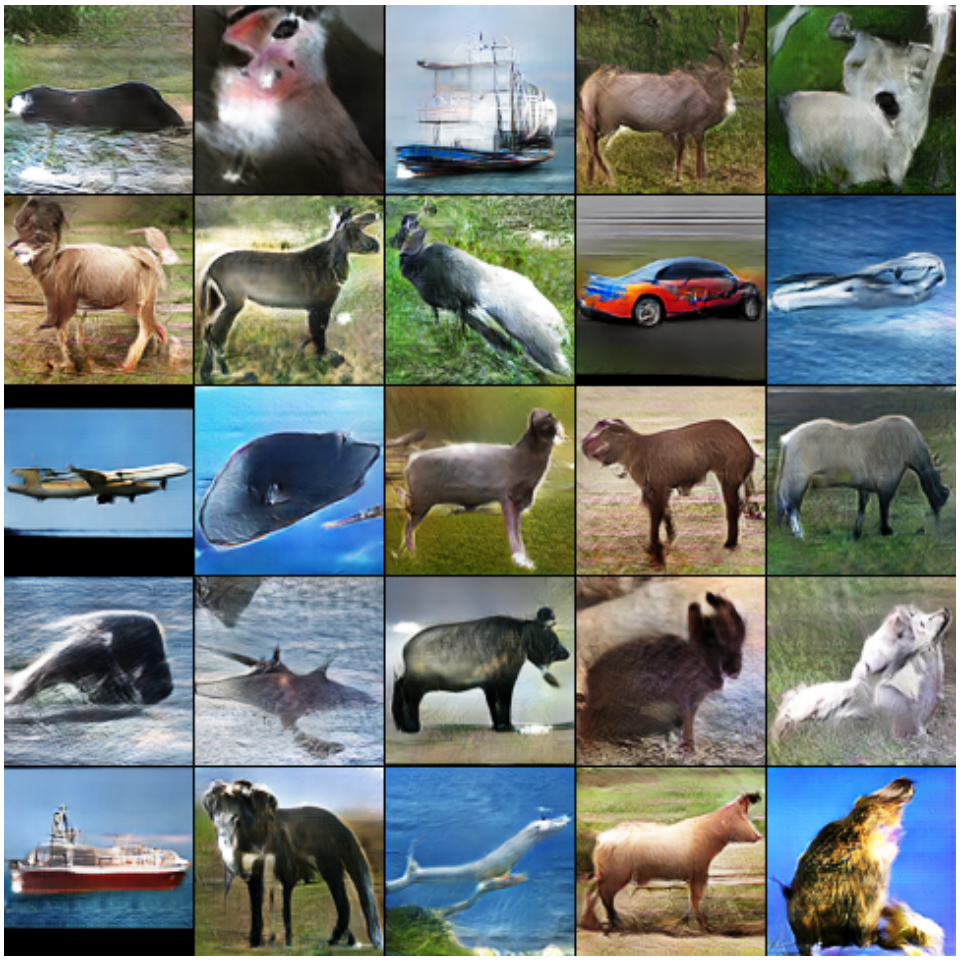


Figure 12: Cherry-picked samples generated by MGAN trained on the 96×96 STL-10 dataset.

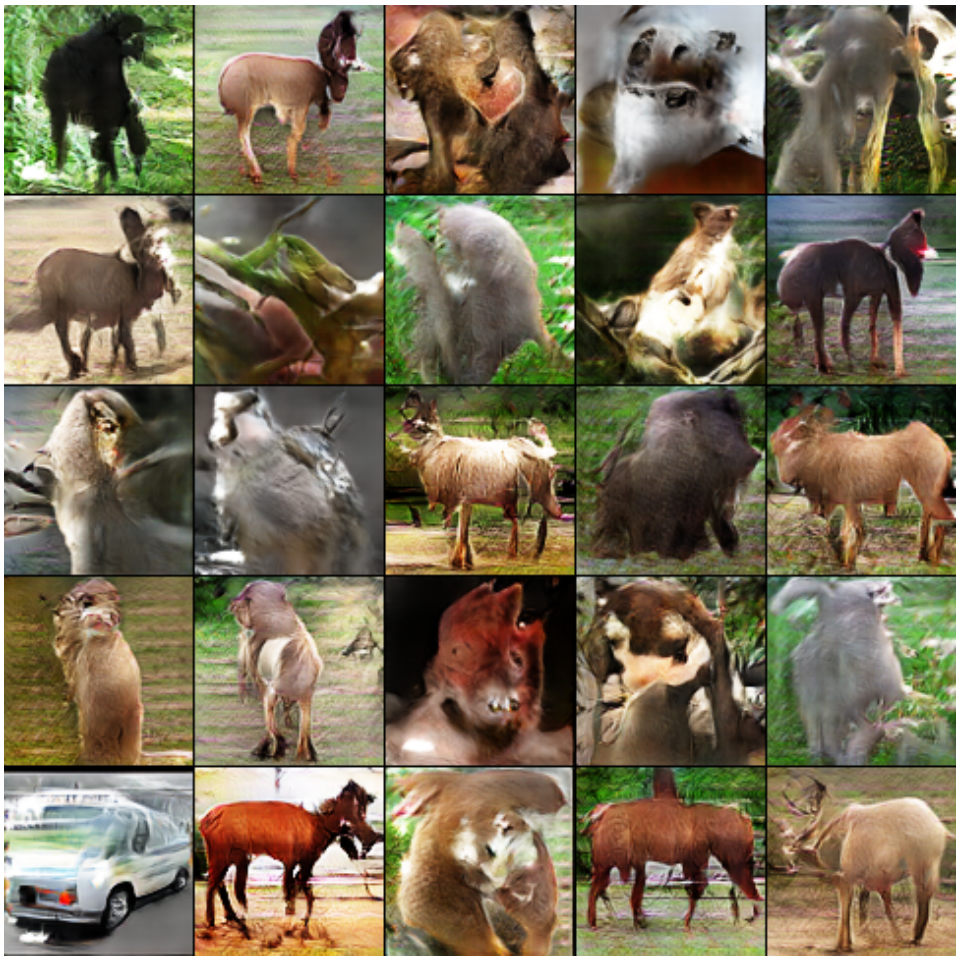


Figure 13: Incomplete, unrealistic samples generated by MGAN trained on the 96×96 STL-10 dataset.

Table 4: Network architecture and hyperparameters for the STL-10 48×48 dataset.

Operation	Kernel	Strides	Feature maps	BN?	BN center?	Nonlinearity	Shared?
$G(\mathbf{z}) : \mathbf{z} \sim$ Uniform $[-1, 1]$			100				
Fully connected			4×4×1024	√	×	ReLU	×
Transposed convolution	5×5	2×2	512	√	×	ReLU	√
Transposed convolution	5×5	2×2	256	√	×	ReLU	√
Transposed convolution	5×5	2×2	128	√	×	ReLU	√
Transposed convolution	5×5	2×2	3	×	×	Tanh	√
$C(\mathbf{x}), D(\mathbf{x})$			48×48×3				
Convolution	5×5	2×2	128	√	√	Leaky ReLU	√
Convolution	5×5	2×2	256	√	√	Leaky ReLU	√
Convolution	5×5	2×2	512	√	√	Leaky ReLU	√
Convolution	5×5	2×2	1024	√	√	Leaky ReLU	√
Fully connected			10/1	×	×	Softplus	×
Number of generators	10						
Batch size for real data	64						
Batch size for each generator	12						
Number of iterations	250						
Leaky ReLU slope	0.2						
Learning rate	0.0002						
Regularization constants	$\beta = 1.0$						
Optimizer	Adam($\beta_1 = 0.5, \beta_2 = 0.999$)						
Weight, bias initialization	$\mathcal{N}(\mu = 0, \sigma = 0.01), 0$						

Table 5: Network architecture and hyperparameters for the STL96×96 dataset.

Operation	Kernel	Strides	Feature maps	BN?	BN center?	Nonlinearity	Shared?
$G(\mathbf{z}) : \mathbf{z} \sim$ Uniform $[-1, 1]$			100				
Fully connected			4×4×2046	√	×	ReLU	×
Transposed convolution	5×5	2×2	1024	√	×	ReLU	√
Transposed convolution	5×5	2×2	512	√	×	ReLU	√
Transposed convolution	5×5	2×2	256	√	×	ReLU	√
Transposed convolution	5×5	2×2	128	√	×	ReLU	√
Transposed convolution	5×5	2×2	3	×	×	Tanh	√
$C(\mathbf{x}), D(\mathbf{x})$			32×32×3				
Convolution	5×5	2×2	128	√	√	Leaky ReLU	√
Convolution	5×5	2×2	256	√	√	Leaky ReLU	√
Convolution	5×5	2×2	512	√	√	Leaky ReLU	√
Convolution	5×5	2×2	1024	√	√	Leaky ReLU	√
Convolution	5×5	2×2	2048	√	√	Leaky ReLU	√
Fully connected			10/1	×	×	Softplus	×
Number of generators	10						
Batch size for real data	64						
Batch size for each generator	12						
Number of iterations	250						
Leaky ReLU slope	0.2						
Learning rate	0.0002						
Regularization constants	$\beta = 1.0$						
Optimizer	Adam($\beta_1 = 0.5, \beta_2 = 0.999$)						
Weight, bias initialization	$\mathcal{N}(\mu = 0, \sigma = 0.01), 0$						

Table 6: Network architecture and hyperparameters for the ImageNet dataset.

Operation	Kernel	Strides	Feature maps	BN?	BN center?	Nonlinearity	Shared?
$G(\mathbf{z}) : \mathbf{z} \sim$ Uniform $[-1, 1]$			100				
Fully connected			4×4×512	√	×	ReLU	×
Transposed convolution	5×5	2×2	256	√	×	ReLU	√
Transposed convolution	5×5	2×2	128	√	×	ReLU	√
Transposed convolution	5×5	2×2	3	×	×	Tanh	√
$C(\mathbf{x}), D(\mathbf{x})$			32×32×3				
Convolution	5×5	2×2	128	√	√	Leaky ReLU	√
Convolution	5×5	2×2	256	√	√	Leaky ReLU	√
Convolution	5×5	2×2	512	√	√	Leaky ReLU	√
Fully connected			10/1	×	×	Softplus	×
Number of generators	10						
Batch size for real data	64						
Batch size for each generator	12						
Number of iterations	50						
Leaky ReLU slope	0.2						
Learning rate	0.0002						
Regularization constants	$\beta = 0.1$						
Optimizer	Adam($\beta_1 = 0.5, \beta_2 = 0.999$)						
Weight, bias initialization	$\mathcal{N}(\mu = 0, \sigma = 0.01), 0$						