Training Self-supervised Class-conditional GANs with Classifier Gradient Penalty and Dynamic Prior

Jeongik Cho
jeongik.jo.01@gmail.com

Abstract

Class-conditional GAN generates class-conditional data from continuous latent distribution and categorical distribution. Typically, a class-conditional GAN can be trained only when the label, which is the conditional categorical distribution of the target data, is given.

In this paper, we propose a novel GAN that allows the model to perform self-supervised class-conditional data generation and clustering without knowing labels, optimal prior categorical probability, or metric function. The proposed method uses a discriminator, a classifier, and a generator. The classifier is trained with cross-entropy loss to predict the conditional vector of the fake data. Also, the conditional vector of real data predicted by the classifier is used to train the class-conditional GAN. When training class-conditional GAN with this classifier, the decision boundary of the classifier falls to the local optima where the density of the data is minimized. The proposed method adds a classifier gradient penalty loss to the classifier loss to prevent the classifier's decision boundary from falling into narrow a range of local optima. It regulates the gradient of the classifier's output to prevent the gradient near the decision boundary from becoming too large. As the classifier gradient penalty loss weight increases, the decision boundary falls into a wider range of local optima. It means that the sensitivity of each class can be adjusted by the weight of the gradient penalty loss. Additionally, the proposed method updates the prior categorical probability with the categorical probability of real data predicted by the classifier. As training progresses, the entropy of the prior categorical probability decreases and converges according to the classifier gradient penalty loss weight.

1 Introduction

Among recent deep generative models, generative adversarial network (GAN) [1] and diffusion model [13] have shown state-of-the-art performance.

In general, the generative performance of diffusion models is known to be higher [14]. However, analyzing the latent space of diffusion models can be challenging due to their high-dimensional latent spaces and recurring inference. On the other hand, GANs have a lower-dimensional latent space and require only a single inference for sampling, making it easier to analyze the latent space and apply it to applications. Especially, the lower-dimensional latent space of GAN can be utilized in many applications through generative model inversion. For example, InterFaceGAN [16] showed that the label of data is linearly separable in latent space. It also proposed a method to continuously change the attributes of the input face image using these characteristics. Several applications utilizing GAN inversion can be seen in the GAN inversion survey paper [20].

Among variations of GANs, a conditional GAN [2] (CGAN) is a GAN that can generate conditional data distribution. CGAN's generator takes an unconditional latent vector and a conditional vector as input and generates conditional data corresponding to the conditional vector. In general, the training dataset for CGAN should consist of a target (real) data vector and the corresponding conditional vector, which can be continuous, discrete, or both. Pix2Pix [17] is an example of a conditional GAN using a continuous conditional vector. Pix2Pix takes an image that is a continuous conditional vector and generates a corresponding conditional image. For example, Pix2Pix can be trained to take a grayscale image as input and output a corresponding color image.

Class-conditional GAN is a conditional GAN where the conditional vector is a discrete categorical vector. Auxiliary Classifier GAN (ACGAN) [3] and Conditional Activation GAN (CAGAN) [4] are examples of class-conditional GANs. ACGAN and CAGAN take one or multiple discrete categorical vectors as input and generate data corresponding to the categorical vectors. In ACGAN, a classifier is trained to predict the label of real data, and a generator is trained so that the fake data generated with the discrete categorical vector is correctly classified by the classifier. CAGAN is a composite of multiple GANs, where each GAN is trained to generate each class. Therefore, CAGAN does not use a classifier, but only multiple adversarial losses to generate class-conditional data. However, these class-conditional GANs can only be trained given the labels (class-conditional vector) of the data. Therefore, these methods cannot be used with unlabeled datasets.

Unlike ACGAN or CAGAN, class-conditional InfoGAN [5] can generate class-conditional data even if the data is not labeled. In class-conditional InfoGAN, the classifier and generator are trained so that the conditional vector of the generated data is correctly classified by the classifier. InfoGAN has shown that it is possible to generate class-conditional data without knowing the conditional vector of the real data if the generator and the classifier are trained with classification loss. This is because the generator tries to generate class-conditional data that is easy to be classified by the classifier. For example, the MNIST handwritten digits dataset [9] consists of handwritten images of 10 different digits, each with a proportion of 0.1. If a class-conditional InfoGAN is trained on the MNIST dataset using a 10-dimensional categorical conditional vector, with each category assigned a probability of 0.1, then each conditional vector will uniquely represent one of the ten digits. Although InfoGAN does not require a conditional vector of the real data, it can only be trained given the optimal categorical probability.

Elastic InfoGAN [12] proposed a method for class-conditional data generation even when the optimal prior categorical probability is not known. In elastic InfoGAN, the categorical latent probability is updated to minimize generator loss through gradient descent. Elastic InfoGAN also restricted each class to have the same identity by using contrastive loss [15] with identity-preserving transformations.

In this paper, we analyze the problems of previous works and propose a novel self-supervised (unsupervised) class-conditional GAN, Classifier Gradient Penalty GAN (CGPGAN) to address them. CGPGAN can be used under the following conditions:

1. The labels of all data are unknown.

2. Optimal categorical latent distribution is unknown.

3. Metric to measure the distance between the data is unknown.

CGPGAN uses a discriminator, a classifier, and a generator. The classifier of CGPGAN is trained with cross-entropy loss to predict the categorical vector of the fake data. Also, the categorical vector of real data predicted by the classifier is used to train the class-conditional GAN. When training class-conditional GAN with the classifier, the decision boundary of the classifier falls to the local optima where the density of the data is minimized. This is because the density of data near the decision boundary of the classifier should be low to minimize the classification loss. However, if the probability density function of the data estimated by the model is lumpy, the decision boundary of the classifier may fall on a very narrow range of local optima. CGPGAN uses a classifier gradient penalty loss for the classifier to prevent the classifier's decision boundary from falling into narrow a range of local optima. It regulates the gradient of the classifier's output to prevent the gradient near the decision boundary from becoming too large. As the classifier gradient penalty loss weight increases, the decision boundary falls into a wider range of local optima. It means that the sensitivity of each class can be adjusted by the weight of the gradient penalty loss in CGPGAN. Additionally, CGPGAN updates the prior categorical probability with the categorical probability of real data predicted by the classifier. As training progresses, the entropy of the prior categorical probability decreases and converges according to the classifier gradient penalty loss weight.

InfoGAN cannot be used under condition 2 because it requires an optimal categorical latent distribution. Elastic InfoGAN cannot be used under condition 3 because it requires a metric for identity-preserving transformation. Also, Elastic InfoGAN cannot adjust the sensitivity of each category, while CGPGAN can adjust the sensitivity of each category through the classifier gradient penalty loss weight.

2 Class-conditional Data Generation

Typically, when training a GAN, everything is assumed to be continuous. It means that the data distribution and latent distribution are assumed to be continuous, and the generator and discriminator

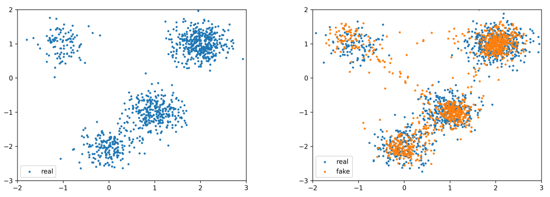

Figure 1: Left plot: Two-dimensional dataset consisting of four Gaussian clusters. The centers and probabilities of each cluster are $(-1, 2), (0, -2), (1, -1), (2, 1)$ and $[0.1, 0.2, 0.3, 0.4]$, respectively. The standard deviation for all clusters is 0.3. Right plot: Samples generated by GAN trained only with a continuous latent distribution.

of GAN are assumed to be continuous functions (in general, deep learning models are differentiable continuous functions for error backpropagation). However, the data distribution is not necessarily continuous. When data distribution includes a discrete part and latent distribution is continuous, a sufficiently complex deep generative model can approximate the discrete part of the data distribution. However, approximating the discrete part of the data distribution is still not easy for most deep generative models, which is a continuous function.

The left plot of Fig. 1 shows a data distribution example consisting of four Gaussian clusters. There is no perfect discrete part in this data distribution (i.e., the probability density function is still continuous), but it is easier and better to represent this data distribution with a 4-dimensional discrete categorical latent distribution.

The right plot of Fig. 1 shows generated data with GAN trained only with a continuous latent distribution. One can see that the model generates lines connecting the clusters. This is because the latent distribution is continuous, and the generator is a continuous function, making it difficult to represent the discrete part of the data distribution. As training progresses, the probability density of the line connecting the clusters decreases, but it requires a long training period, and it is hard to say that the continuous latent distribution correctly represents data distribution (i.e., entangled data representation).

For datasets with discrete parts, using a discrete latent distribution is more appropriate for model training and disentangled data representation. Class-conditional generative models, such as ACGAN [3] or CAGAN [4], take both continuous latent distribution and discrete categorical latent distribution as inputs and generate class-conditional data distribution. Training model with discrete categorical latent random variable allows the model to represent discrete parts of the dataset appropriately. However, ACGAN or CAGAN can only be trained when class-conditional vectors (labels) of real data are given.

Class-conditional InfoGAN [5] can perform class-conditional data generation and inversion (clustering) by maximizing mutual information of generator input categorical latent distribution and classifier output distribution, even if the data is not labeled. The following equations show losses for (class-conditional) InfoGAN.

$$L_q = \lambda_{cls} L_{cls} \tag{1}$$

$$L_d = L^d_{adv} \tag{2}$$

$$L_g = L^g_{adv} + \lambda_{cls} L_{cls} \tag{3}$$

$$L_{cls} = \mathbb{E}_{z,c_f}\left[-c_f \cdot \log(Q(G(z, c_f)))\right] \tag{4}$$

$$L^d_{adv} = \mathbb{E}_{x,z,c_f}\left[A_d(D(x), D(G(z, c_f)))\right] \tag{5}$$

$$L^g_{adv} = \mathbb{E}_{z,c_f}\left[A_g(D(G(z, c_f)))\right] \tag{6}$$

In Eqs. 1, 2, and 3, L_q, L_d, and L_g represent classifier loss, discriminator loss, and generator loss of InfoGAN, respectively. L_{cls} and λ_{cls} represent classification loss and classification loss weight, respectively. In Eqs. 4, 5 and 6, Q, D, and G represent classifier, discriminator, and generator, respectively. In Eq. 4, L_{cls} is categorical cross entropy between categorical latent vector c and predicted probability of generated data $Q(G(z, c_f))$. c_f and z represent the categorical latent vector and continuous latent vector sampled from the categorical latent distribution C and continuous latent distribution Z, respectively. Operation "$\cdot$" represents the inner product. In Eqs. 5 and 6, A_d and A_g represent adversarial loss function [6] for GAN training. In InfoGAN, a classifier Q can share hidden layers with a discriminator D for efficiency.

From the above equations, one can see that a classifier Q and a generator G are trained to minimize classification loss L_{cls}. InfoGAN has shown that, given an appropriate categorical latent distribution C, it can perform class-conditional data generation and clustering (inversion) even when the data is unlabeled. However, InfoGAN still needs prior probability of categorical latent distribution C. Without knowing the appropriate categorical latent distribution C, InfoGAN cannot perform class-conditional data generation and clustering appropriately. Additionally, InfoGAN's generator is trained with both adversarial loss and classification loss like ACGAN. It means that InfoGAN shares the same problems as ACGAN: adversarial loss and classification loss conflict with each other in generator loss, resulting in a decrease in the generative performance of the model. Specifically, the density of fake data is always lower than the density of real data near the decision boundary of classifier Q. This is because the generator is trained to move the generated data away from the decision boundary due to the classification loss.

Elastic InfoGAN [12] proposed a method for class-conditional data generation when the prior probability of a categorical latent distribution is not known. In Elastic InfoGAN, the categorical latent distribution probability is updated to minimize generator loss through gradient descent. To allow the gradient to flow up to a categorical latent distribution probability, Elastic InfoGAN uses Gumbel softmax [18, 19]. Elastic InfoGAN also used contrastive loss [15] to ensure that each class has the same identity. Contrastive loss allows augmented data with identity-preserving transformations to be classified in the same class. By training the classifier with contrastive loss, generators are constrained to generate data with the same identity if they are of the same class. For example, Elastic InfoGAN has used rotation, zoom, flip, crop, and gamma change as identity-preserving transformations for image data.

However, there are still several problems in Elastic InfoGAN. First, contrastive loss can only be used if a good transformation that preserves the identity of the data is known. Therefore, it cannot be used in data domains where good identity-preserving transformations are not known. Second, clustering only can be performed based on identity-preserving transformations. For example, on the MNIST handwritten digits dataset, Elastic InfoGAN will consider digits 6 and 9 as the same class if 180-degree rotation is used for identity-preserving transformation. Also, like InfoGAN, Elastic InfoGAN uses classification loss for the generator like ACGAN, which causes conflict between adversarial loss and classification loss and decreases the generative performance of the model.

3 Classifier Gradient Penalty GAN

In this paper, we introduce Classifier Gradient Penalty GAN (CGPGAN) which can perform class-conditional data generation and clustering under more general conditions than previous works. CGPGAN can be used under the following very general conditions:

1. The labels of all data are unknown.
2. Optimal categorical latent distribution is unknown.
3. Metric to measure the distance between the data is unknown.

InfoGAN cannot be used under condition 2 because it requires an optimal categorical latent distribution, and Elastic InfoGAN cannot be used under condition 3 because it requires a metric for identity-preserving transformation.

A CGPGAN consists of a discriminator D, classifier Q, and (class-conditional) generator G. The generator G takes d_z-dimensional continuous latent distribution and d_c-dimensional categorical latent distribution as inputs to generate class-conditional data. The classifier Q is trained to predict the label of the generated data, and the label of the real data predicted by the classifier is used for adversarial training of the discriminator and generator for class-conditional data generation. The generator G and discriminator D are trained with class-conditional GAN loss to generate class-conditional data. Instead of ACGAN loss [3], CGPGAN uses CAGAN loss [4] for better generative performance.

The following equations show the losses for CGPGAN.

$$L_q = \lambda_{cls} L_{cls} + \lambda_{creg} L_{creg} \tag{7}$$

$$L_d = L^d_{adv} \tag{8}$$

$$L_g = L^g_{adv} \tag{9}$$

$$L_{creg} = \mathbb{E}_{z,c_f}\left[\|\nabla_{G(z,c_f)}\left((1 - Q(G(z,c_f)) \cdot c_f)^2\right)\|_2^2\right] \tag{10}$$

$$L^d_{adv} = \mathbb{E}_{x,z,c_f}\left[A_d(D(x) \cdot argmax\ onehot(Q(x)), D(G(z,c_f)) \cdot c_f)\right] \tag{11}$$

$$L^g_{adv} = \mathbb{E}_{z,c_f}\left[A_g(D(G(z,c_f)) \cdot c_f)\right] \tag{12}$$

In Eq. 7, λ_{creg} and L_{creg} represent classifier gradient penalty loss weight and classifier gradient penalty loss, respectively. Classification loss L_{cls} is cross-entropy loss, which is the same as InfoGAN's classification loss (Eq. 4). In Eq. 9, one can see that there is no classification loss L_{cls} in generator loss L_g. Since CGPGAN's generator is trained with adversarial losses only, there is no conflict between L_{cls} and L^g_{adv} as in InfoGAN.

Eqs. 11 and 12 show CAGAN adversarial losses for CGPGAN. Since the true label c_f of the fake data $G(z, c_f)$ is known, the adversarial loss for fake data in CGPGAN is the same as CAGAN loss. However, the label of the real data x is unknown. Instead, in CGPGAN, $argmax\ onehot(Q(x))$ is used as the label of the real data x. The $argmax\ onehot$ function replaces the maximum value of the vector with 1 and all other values with 0 (e.g., $argmax\ onehot([0.2, 0.5, 0.3]) = [0.0, 1.0, 0.0]$).

Eq. 10 shows classifier gradient penalty loss for CGPGAN. L_{creg} is the gradient of cross-entropy loss L_{cls} with respect to generated data $G(z, c_f)$. The classification loss L_{cls} can be minimized by simply classifying the generated data well, but it can also be minimized by increasing the slope of the decision boundary or by moving the decision boundary near the generated data with lower density. Therefore, when training the classifier with cross-entropy loss, we assumed that the decision boundary will naturally move to the local optimum which minimizes the density of the generated data, and the slope of the decision boundary will increase. If the slope of the decision boundary is very large and the probability density function of the generated data is lumpy, the decision boundary will converge to a local optimum in a very narrow region that minimizes the density of the data. To avoid classifier decision boundary converging local optimum in a narrow region that minimizes the probability density of generated data $P(G(X, C))$, CGPGAN uses a classifier gradient penalty loss L_{creg}. By relaxing the slope of the classifier's decision boundary, the decision boundary can converge to a local optimum in a wider region. One can see that the classifier gradient penalty loss L_{creg} becomes higher when the generated data point $G(z, c_f)$ is near the decision boundary in Eq. 10. In the classifier, the larger the classifier gradient penalty loss L_{creg}, the more the decision boundary converges to the local optimum in a larger region. Therefore, CGPGAN can adjust the sensitivity of each category (cluster) through the weight of the classifier gradient penalty loss λ_{creg}.

Additionally, CGPGAN updates the probability of the categorical latent distribution $P(C)$ during the training with $\mathbb{E}_x[Q(x)]$ (i.e., $P(C) \approx \mathbb{E}_x[Q(x)]$). Through this approximation, CGPGAN can approximate $P(C)$ without knowing the optimal prior probability. However, updating $P(C)$ early in the training can make CGPGAN converge to a trivial solution (i.e., one category has a probability of 1 and the other has a probability of 0). To avoid converging a trivial solution and ensure that the ratio of real to fake data in each category is similar, CGPGAN normalizes $Q(x)$ by the batch distribution only at the beginning of training.

Algo. 1 shows the training step of CGPGAN.

The training step of CGPGAN requires X (data random variable), Z (continuous latent random variable), C (categorical latent random variable), D (discriminator), Q (classifier), and G (generator).

In lines 1-3, the *sample* function represents the sampling function from a random variable. x (real data point), z (continuous latent vector), and c_f (fake categorical latent vector) are sampled from X, Z, and C, respectively.

In line 4, G generates fake data x' with z and c_f. In lines 5 and 6, D and Q takes a fake data point x' as input and outputs a_f (fake adversarial vector) and c'_f (fake categorical latent vector prediction), respectively. Similarly, in lines 7-8, D and Q take a real data point x as input and outputs a_r (real adversarial vector) and c'_r (real categorical latent vector prediction).

Algorithm 1 Algorithm to train CGPGAN

```
Require: $X, Z, C, D, Q, G$
                                                   ▷ update $D$ and $Q$
 1: $x \leftarrow sample(X)$
 2: $z \leftarrow sample(Z)$
 3: $c_f \leftarrow sample(C)$
 4: $x' \leftarrow G(z, c_f)$

 5: $a_r \leftarrow D(x)$
 6: $c'_r \leftarrow Q(x)$
 7: $a_f \leftarrow D(x')$
 8: $c'_f \leftarrow Q(x')$

 9: if $early\ in\ training$ then
10:     $c'_r = prob\,normalize(c'_r)$
11: end if
12: $c_r \leftarrow argmax\ onehot(c'_r)$

13: $L_{cls} \leftarrow -c_f \cdot \log(c'_f)$
14: $L_{creg} \leftarrow \|gradient((1 - c'_f \cdot c_f)^2, x')\|_2^2$

15: $L_d \leftarrow A_d(a_r \cdot c_r, a_f \cdot c_f)$
16: $L_q \leftarrow \lambda_{cls} L_{cls} + \lambda_{creg} L_{creg}$

17: $D \leftarrow minimize(D, L_d)$
18: $Q \leftarrow minimize(Q, L_q)$
                                                   ▷ update $G$
19: $z \leftarrow sample(Z)$
20: $c_f \leftarrow sample(C)$
21: $x' \leftarrow G(z, c_f)$
22: $a_f \leftarrow D(x')$

23: $L_g \leftarrow A_g(a_f \cdot c_f)$

24: $G \leftarrow minimize(G, L_g)$

25: $P(C) \leftarrow update(P(C), c'_r)$             ▷ update $P(C)$
```

In lines 9-11, the real categorical latent vector prediction c'_r is normalized for stable training only at the beginning of the training. In line 10, the *prob normalize* function forces the real categorical latent vector c_r to approach a uniform distribution. This ensures that the ratio of real data to fake data in each category is similar, allowing for stable training in the early stages of CGPGAN training.

$$prob\,normalize(\mathbf{c}) = \mathbf{c} - batch\,average(\mathbf{c}) + \frac{1}{d_c} \tag{13}$$

Eq. 13 shows the function to normalize the categorical latent vectors $\mathbf{c}$, where $\mathbf{c}$ is a $b \times d_c$ matrix, and b represents the batch size. $batchaverage$ is a function that computes the element-wise average vector of $\mathbf{c}$. Therefore, $batchaverage(\mathbf{c})$ is d_c-dimensional vector. One can see that $batchaverage(probnormalize(\mathbf{c}))$ is always $[\frac{1}{d_c}, \frac{1}{d_c}, ..., \frac{1}{d_c}]$. However, the *prob normalize* function restricts the representation of the real categorical latent vector c_r, so it is disabled after some training. In line 12, real categorical latent vector c_r is calculated from c'_r.

In line 13, L_{cls} represents classification loss for c_f prediction. L_{cls} is categorical cross-entropy loss. In line 14, $gradient(y, x)$ function calculates gradient dy/dx.

In lines 15 and 16, L_d and L_q represent discriminator loss and classifier loss, respectively. A_d represents GAN adversarial loss functions for the discriminator. When training GAN with R1 or R2 regularization [7], we recommend using R2 regularization because the true label of generated data is known, unlike real data.

In lines 17 and 18, discriminator D and classifier Q are updated to minimize its loss. In lines 19-24, generator G is updated to minimize its loss.

In line 25, $P(C)$ is updated with predicted real categorical latent vector c'_r. The *update* function can be a simple moving average, an exponential moving average, or others. In CGPGAN, $P(C)$ is initialized with $[\frac{1}{d_c}, \frac{1}{d_c}, ..., \frac{1}{d_c}]$, and c'_r is normalized at the beginning of the training (line 10). Thus, at the beginning of training, $P(C)$ will always be $[\frac{1}{d_c}, \frac{1}{d_c}, ..., \frac{1}{d_c}]$. This makes C to not converge to a trivial solution.

4 Experiments

We trained the models to generate two-dimensional Gaussian clusters, MNIST dataset [9], and AFHQ dataset [21]. In Gaussian clusters experiments, we compare the performance of Vanilla GAN [1], InfoGAN [5], Elastic InfoGAN [12], and our proposed CGPGAN. In MNIST experiments, we compared the clustering of CGPGAN according to classifier gradient penalty loss weight λ_{creg}. In AFHQ experiments, we trained a CGPGAN to generate the AFHQ dataset.

4.1 Gaussian Clusters Experiments

In Gaussian clusters experiments, we used the dataset consisting of four 2-dimensioanl Gaussian clusters as a training dataset. The left plot of Fig. 1 shows data distribution for the experiments. One can see that there are four Gaussian clusters with different probabilities in data distribution. The generator, discriminator, and classifier consisting of four fully connected hidden layers with 512 units were used for training. The following hyperparameters were used for experiments.

$$Z \sim N(0, I_{256})$$
$$optimizer = AdamW \begin{pmatrix} learning\ rate = 0.0001 \\ weight\ decay = 0.0001 \\ \beta_1 = 0.0 \\ \beta_2 = 0.99 \end{pmatrix}$$
$$batch\ size = 16$$
$$\lambda_{r2} = 1$$
$$train\ step\ per\ epoch = 2000$$
$$epoch = 100$$
$$activation\ function = Leaky\ ReLU$$

λ_{r2} represents R2 regularization [7] loss weight $(\lambda_{r2} L_{r2} = \lambda_{r2} \mathbb{E}_{z,c_f} \left[\|\nabla_{G(z,c_f)}(D(G(z,c_f)) \cdot c_f)\|_2^2 \right])$.

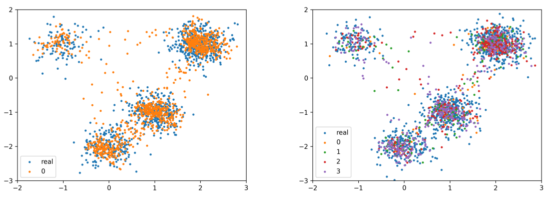

Figure 2: Vanilla GAN (trained only with adversarial loss) is trained with both continuous latent distribution and categorical latent distribution. Left plot: 1-dimensional categorical latent distribution ($P(C) = [1.0]$). Right plot: 4-dimensional optimal categorical latent distribution ($P(C) = [0.1, 0.2, 0.3, 0.4]$).

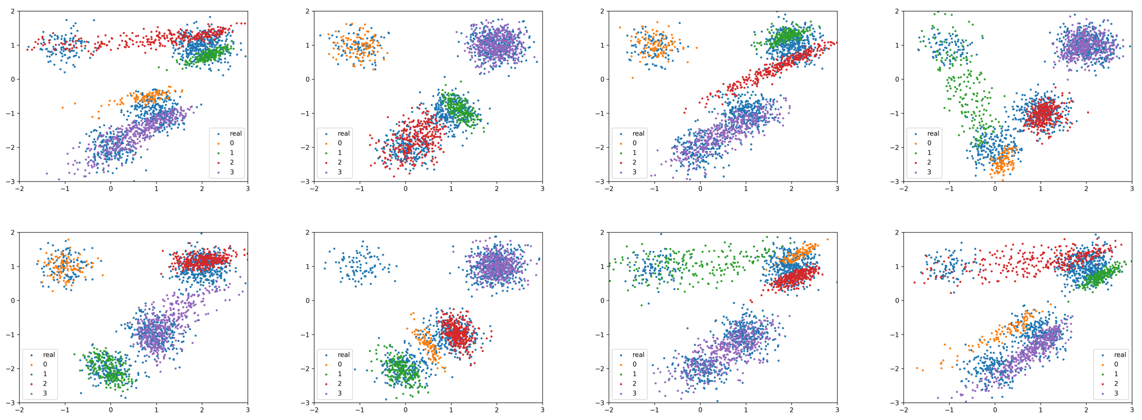

Figure 3: InfoGAN trained with 4-dimensional optimal categorical latent distribution ($P(C) = [0.1, 0.2, 0.3, 0.4]$). Eight times repeated.

Classification loss weight $\lambda_{cls} = 1.0$ was used for InfoGAN, elastic InfoGAN, and CGPGAN. We used exponential moving average with *decay rate* $= 0.999$ as *update* function for CGPGAN. In InfoGAN, Elastic InfoGAN, and CGPGAN, classifier Q and discriminator D do not share hidden layers. Equalized learning rate [8] was used for all trainable weights. Also, exponential moving average [8] with $decayrate = 0.999$ was used for generator weights. In CGPGAN and Elastic InfoGAN, $d_c = 16$ was used, and $P(C)$ was updated after epoch 30 (i.e., in CGPGAN, $early\,in\,training$ in line 9 of Algo. 1 was $True$ until epoch 30). Since we assumed that there is no good metric to measure the distance between data, we did not use identity preserving transformations in Elastic InfoGAN. Only gradient descent on a categorical latent distribution with Gumbel softmax was used for Elastic InfoGAN.

Fig. 2 shows samples generated with vanilla GAN (trained only with adversarial loss). The left plot of Fig. 2 shows data generated by a vanilla GAN with a one-dimensional categorical latent distribution. Since there is no discrete part in the latent distribution, the vanilla GAN generates lines between clusters. This shows that when training a GAN with only continuous latent vectors, the latent space is entangled and not suitable for representing data with discrete parts.

The right plot of Fig. 2 shows data generated by a vanilla GAN trained with optimal categorical latent distribution ($P(C) = [0.1, 0.2, 0.3, 0.4]$). One can see that the generator of vanilla GAN did not use the information of categorical latent distribution, and training was exclusively performed only with a continuous latent distribution. Therefore, the generator output was also continuous, which caused a line generation between each cluster. This means that even if the generator takes a discrete categorical latent distribution as input, additional loss is required to make the generator use it meaningful.

Fig. 3 shows data generated by InfoGAN trained with the optimal categorical latent distribution. Unlike the Vanilla GAN, one can see that the model generates class-conditional distribution with the categorical latent distribution. However, one can still see the problems of InfoGAN in this figure.

The first problem was that even though the categorical latent distribution was optimal, sometimes each conditional vector was not mapped to the correct cluster. We repeated the InfoGAN training

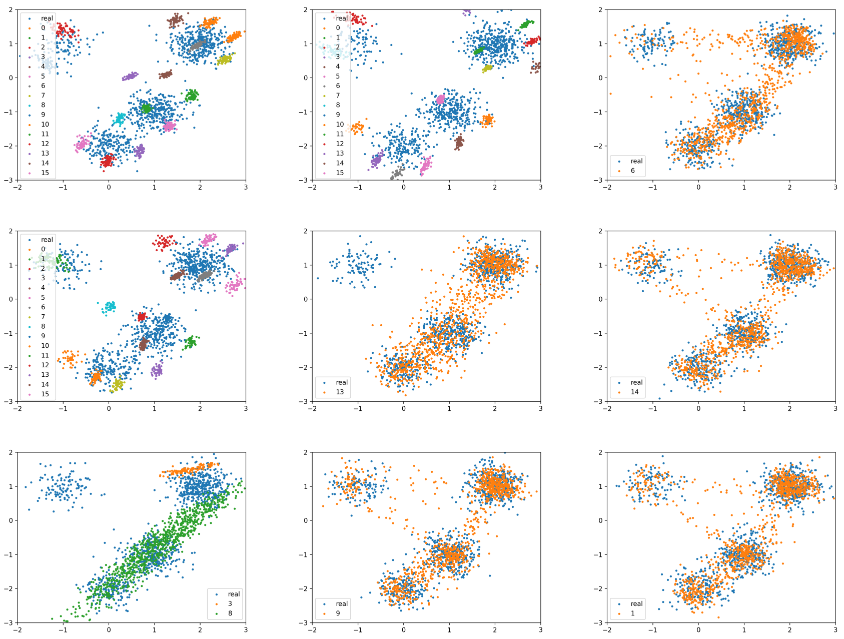

Figure 4: Elastic InfoGAN trained with different temperatures and learning rates. Contrastive loss was not used. Row 1: $t = 0.1$, Row 2: $t = 0.3$, Row 3: $t = 1.0$, Column 1: $lr = 0.0001$, Column 2: $lr = 0.0003$, Column 3: $lr = 0.001$. Categories with a probability of less than 1% were omitted.

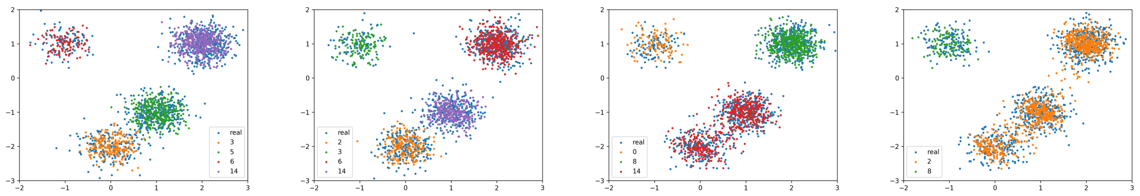

Figure 5: CGPGAN trained with different λ_{creg}. Categories with a probability of less than 1% were omitted. Plot 1: $\lambda_{creg} = 0.0$, $P(C) = [0.2003, 0.3029, 0.0994, 0.3973]$. Plot 2: $\lambda_{creg} = 1.0$, $P(C) = [0.1973, 0.0956, 0.4054, 0.3018]$. Plot 3: $\lambda_{creg} = 10.0$, $P(C) = [0.0992, 0.4002, 0.5006]$. Plot 4: $\lambda_{creg} = 100.0$, $P(C) = [0.9002, 0.0998]$.

eight times, one can see that in some iterations, some classes were assigned correctly, but never all classes were assigned correctly. For example, in the second and fourth plots, class 3 was assigned correctly. In the fifth plot, classes 0 and 1 were assigned correctly. In the sixth plot, classes 1 and 3 were assigned correctly. However, there was never an iteration when all four classes were assigned correctly. This is because when the decision boundary of the InfoGAN classifier is not initialized very ideally, the decision boundary of the classifier converges to a local optimum, resulting in inaccurate cluster assignments. And since the probability of the classifier being initialized very ideally is very low, InfoGAN was not able to assign all classes correctly.

The second problem is that the generator does not generate data near the decision boundary of the classifier. In Fig. 3, one can see that the generator of InfoGAN does not generate data near the decision boundary of the classifier. This is because classification loss and adversarial loss conflict with the generator of InfoGAN.

Fig. 4 shows data generated by elastic InfoGAN. We tested several combinations of hyperparameters (temperature t and learning rate for the categorical latent distribution lr), but Elastic InfoGAN could not generate class-conditional data correctly.

Fig. 5 shows samples generated by CGPGAN trained with different λ_{creg}. In the first and second plots of Fig. 5, each category is assigned to each cluster correctly. Because the probability density

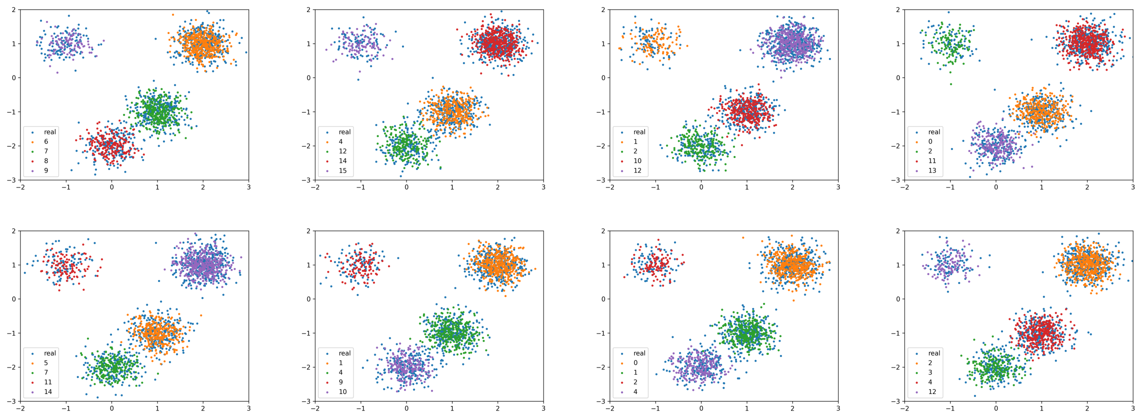

Figure 6: CGPGAN trained with $\lambda_{creg} = 0.1$. Categories with a probability of less than 1% were omitted. Eight times repeated.

function of the generated data distribution is smooth due to the simplicity of the data and small model, each class is assigned correctly, even though there was no classifier gradient penalty in the first plot. Also, the probability of each category is also very accurate, and there was a natural division between clusters, unlike InfoGAN. This is because CGPGAN's generator is only trained with adversarial losses of CAGAN, not classification loss, so there is no conflict between those losses. In the third and fourth plots, multiple clusters were assigned to the same category. This is because λ_{creg} was too high, causing the classifier to converge to a local optimum in a wide region. This shows that CGPGAN can adjust the sensitivity of each category via λ_{creg}.

Fig. 6 shows the results of eight iterations of CGPGAN training with $\lambda_{creg} = 0.1$. One can see that CGPGAN is generating class-conditional data correctly over multiple iterations.

4.2 MNIST Experiments

In this experiment, we trained CGPGAN to generate the MNIST handwritten digits dataset [9]. The MNIST dataset consists of 10 digits from 0 to 9, with each digit representing about 10% of the total.

The generator, discriminator, and classifier simply consist of CNNs. $learningrate = 0.001$, $d_c = 32$, $epoch = 300$ were used for the experiments. The categorical latent distribution of CGPGAN was updated after epoch 100. Other hyperparameters are the same as in section 4.1. We used FID [10], precision & recall [11] for generative performance evaluation. All evaluation methods used the Inception model. 32k training samples were used for generative performance evaluation.

Figs. 7, 8, 9,10 show the difference in class-conditional data generation of CGPGAN according to λ_{creg}.

First, in Fig. 7, since λ_{creg} was too low, the classifier decision boundary converged on a local optimum in a narrow region. Thus, some digits were split into multiple categories, but it does not mean that CGPGAN performed an incorrect class-conditional data generation. If we ignore the human knowledge of each digit, digit 2 with a loop and without a loop, and digit 7 with a horizontal line in the center and without a horizontal line can be considered different categories. In Fig. 7, the digit 2 was divided into categories 4 (column 4) and 11 (column 11). However, this division is still meaningful. One can see that the digit 2 in category 4 has a loop, but category 11 has no loop. Also, the sum of the probabilities of those two categories is $0.074 + 0.0274 = 0.1014$, which is similar to the proportion of the digit 2 in the MNIST digits dataset. Digit 7 was divided into categories 6 and 15. One can see that the digit 7 in category 6 does not have a horizontal line, but category 15 has it. The sum of the probability of those two categories is $0.0902 + 0.0138 = 0.1040$, which is about 10%. This shows that the optimal clustering (and class-conditional data generation) may depend on the sensitivity of each cluster.

As λ_{creg} increases, the classifier decision boundary converges to the local optimum over a wider region. In Fig. 10, $\lambda_{creg} = 120$ was used for training. One can see that multiple digits are clustered into one category, except for digit 1. For example, digits 3, 5, and 8 were clustered in category 2. Also,

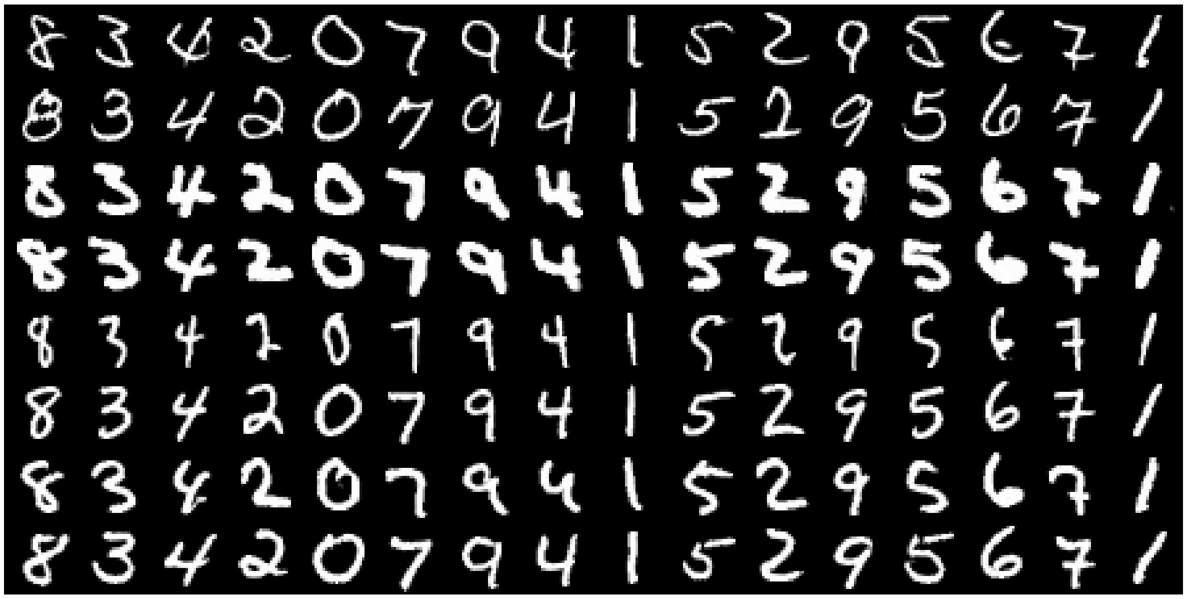

Figure 7: MNIST generated data with $\lambda_{creg} = 50$. Each row has the same continuous latent vector, and each column has the same categorical latent vector. Out of d_c categories, those with a probability less than 1% were omitted. The probabilities for each category are $[0.0975, 0.1059, 0.0384, 0.0740, 0.0974, 0.0902, 0.0395, 0.0601, 0.0657, 0.0385, 0.0274, 0.0563, 0.0504, 0.1002, 0.0138, 0.0447]$. FID: 1.3140, precision: 0.8158, recall: 0.6763.

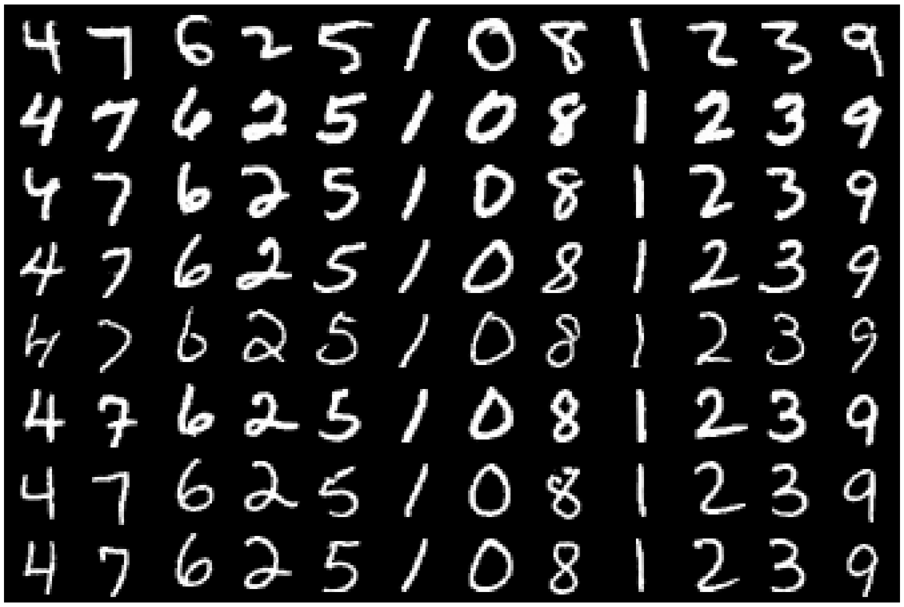

Figure 8: MNIST generated data with $\lambda_{creg} = 70$. Each row has the same continuous latent vector, and each column has the same categorical latent vector. Out of d_c categories, those with a probability less than 1% were omitted. The probabilities for each category are $[0.0975, 0.1051, 0.0993, 0.0765, 0.0901, 0.0454, 0.0988, 0.0964, 0.0668, 0.0270, 0.1012, 0.0957]$. FID: 1.4724, precision: 0.8122, recall: 0.6728.

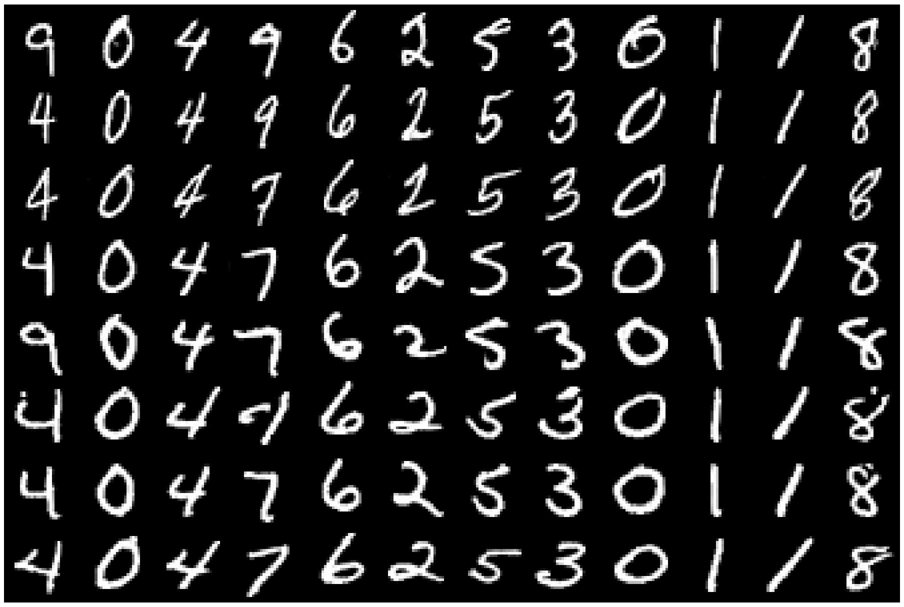

Figure 9: MNIST generated data with $\lambda_{creg} = 100$. Each row has the same continuous latent vector, and each column has the same categorical latent vector. Out of d_c categories, those with a probability less than 1% were omitted. The probabilities for each category are $[0.1060, 0.0467, 0.0359, 0.1566, 0.0982, 0.1014, 0.0900, 0.1064, 0.0522, 0.0706, 0.0395, 0.0965]$. FID: 1.3847, precision: 0.8084, recall: 0.6802.

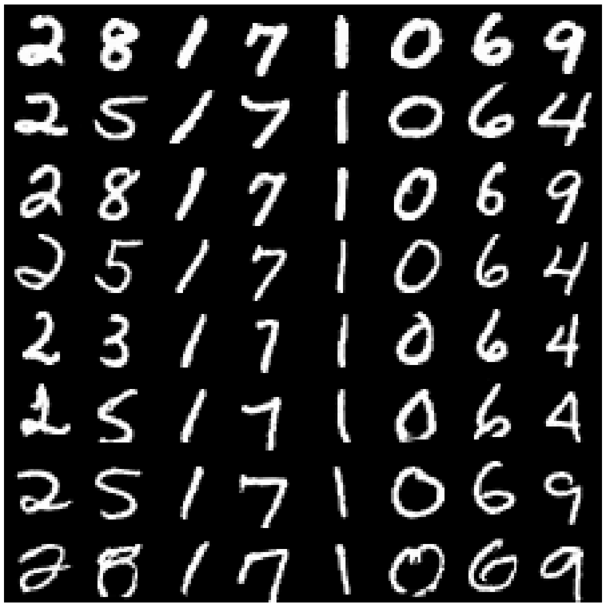

Figure 10: MNIST generated data with $\lambda_{creg} = 120$. Each row has the same continuous latent vector, and each column has the same categorical latent vector. Out of d_c categories, those with a probability less than 1% were omitted. The probabilities for each category are $[0.1013, 0.2940, 0.0491, 0.0908, 0.0586, 0.1004, 0.0947, 0.2110]$. FID: 1.6374, precision: 0.8076, recall: 0.6734.

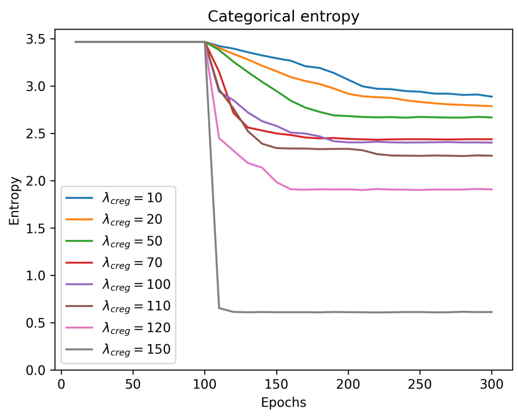

Figure 11: The entropy of a categorical latent distribution over epochs.

digits 4 and 9 were clustered in category 8. It means that the distance between those digits is closer than other digits for the model. The digit 1 was divided into two categories based on whether it was tilted or not in Figs. 7, 8, 9, 10. It means that for the model, it is easy to distinguish between a tilted digit 1 and a vertical digit 1.

Fig. 11 shows the entropy of a categorical latent distribution over epochs. Entropy was calculated every 10 epochs. One can see that the larger λ_{creg} is, the faster the entropy decreases.

In this experiment, we showed that CGPGAN can properly generate the class-conditional data of the MNIST handwritten digit by adjusting λ_{creg}. In particular, when λ_{creg} is low so the sensitivity of each cluster is high, CGPGAN found that there are different patterns within some digits (e.g., digits 1, 2, and 7 in Fig. 7). When λ_{creg} is high so the sensitivity of each cluster is low, CGPGAN found that some digits have a similar pattern (e.g., digits 3, 5, 8, and 4, 9). This means that CGPGAN can generate class-conditional data by adjusting the sensitivity of each cluster according to λ_{creg}. Separately, all three CGPGANs have good unconditional generative performance from FID and precision & recall.

4.3 AFHQ Experiments

In this section, we trained CGPGAN to generate AFHQ dataset [21]. The AFHQ dataset consists of animal face images. We resized the images to 256×256 resolution and used them for training.

The following hyperparameters were used to train the model.

$$\begin{gathered}
Z \sim N(0, I_{512}) \\
optimizer = AdamW\begin{pmatrix} learning\ rate = 0.003 \\ weight\ decay = 0.0001 \\ \beta_1 = 0.0 \\ \beta_2 = 0.99 \end{pmatrix} \\
batch\ size = 8 \\
\lambda_{r2} = 10.0 \\
epoch = 150 \\
\lambda_{creg} = 50.0 \\
d_c = 16 \\
decay\ rate = 0.999 \\
activation\ function = Leaky\ ReLU
\end{gathered}$$

The model simply consists of CNN residual blocks, and $P(C)$ was updated after epoch 50.

Fig. 12 shows generated samples of CGPGAN trained with the AFHQ dataset. In Fig. 12, each row has the same continuous latent vector, and each column has the same categorical latent vector. One can see that each column has similar patterns. For example, the animals in the second column have an overall yellow fur, while the animals in the third column have striped or spotted patterns. The animals in the seventh column have black muzzles, and the animals in the twelfth column have black fur. In addition to these, each column exhibits a similar pattern.

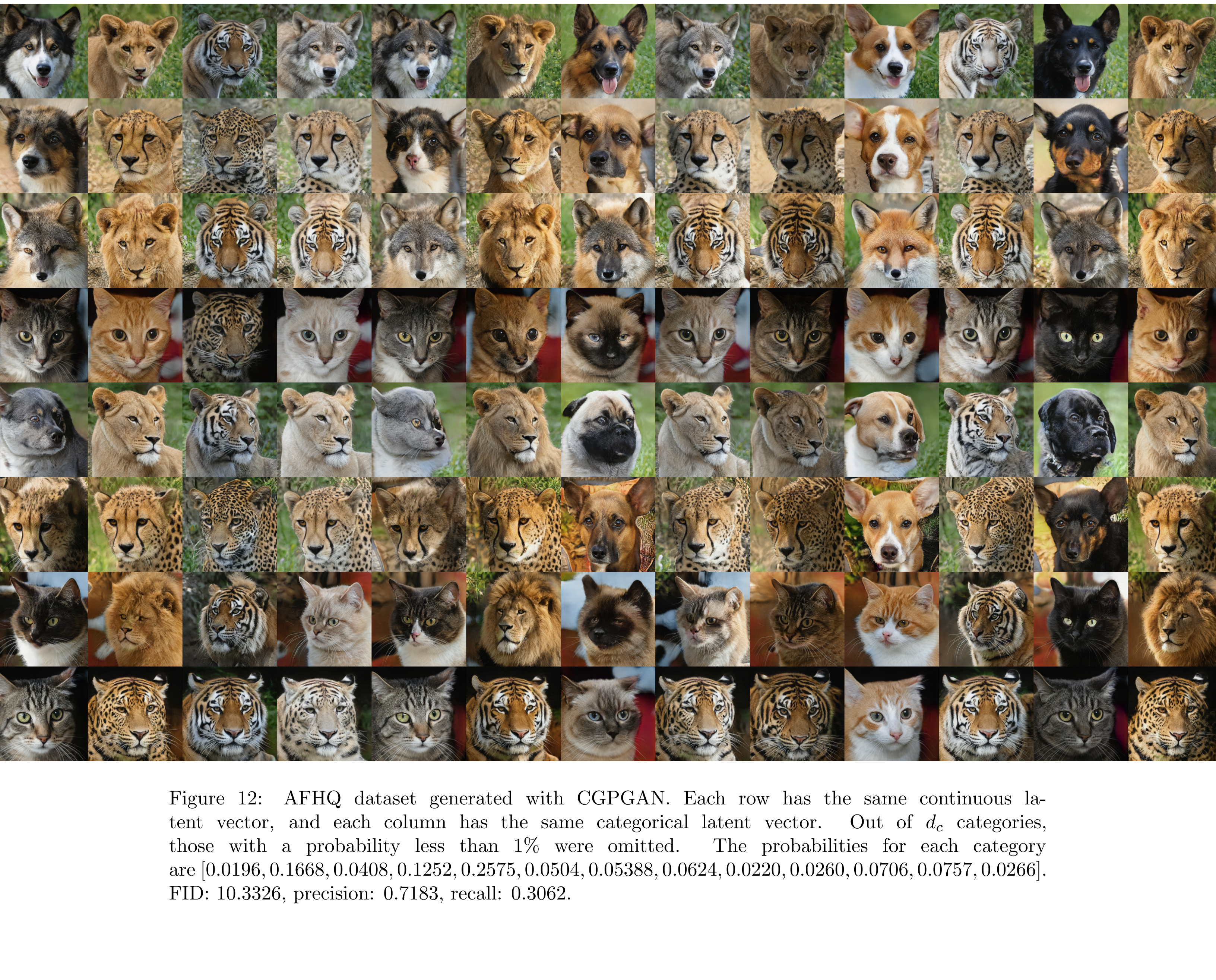

Figure 12: AFHQ dataset generated with CGPGAN. Each row has the same continuous latent vector, and each column has the same categorical latent vector. Out of d_c categories, those with a probability less than 1% were omitted. The probabilities for each category are $[0.0196, 0.1668, 0.0408, 0.1252, 0.2575, 0.0504, 0.05388, 0.0624, 0.0220, 0.0260, 0.0706, 0.0757, 0.0266]$. FID: 10.3326, precision: 0.7183, recall: 0.3062.

5 Conclusion

In this paper, we introduced CGPGAN, a self-supervised class-conditional GAN. Unlike previous works, CGPGAN does not require a label of data, optimal categorical latent distribution, and a good metric to calculate the distance between data. Therefore, CGPGAN can be used in more general situations regardless of the data domain.

CGPGAN uses CAGAN loss, classification loss, and classifier gradient penalty loss. Also, the categorical latent distribution is updated to approximate the classifier output distribution of the real data. The classifier gradient penalty loss weight of CGPGAN controls the sensitivity of each cluster. The higher the classifier gradient penalty loss weight, the wider the decision boundary of the classifier converges to the local optima, so the sensitivity of each cluster decreases. The entropy of the categorical latent distribution gradually decreases and converges to the appropriate value according to the classifier gradient penalty loss weight.

CGPGAN showed better performance than Vanilla GAN, InfoGAN, and Elastic InfoGAN in Gaussian cluster generation experiments. We also showed that CGPGAN could perform self-supervised class-conditional data generation on the MNIST and AFHQ experiments.

References

[1] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Nets. In Commun. ACM, vol. 63, no. 11, pp. 139-144, Nov. 2020. https://doi.org/10.1145/3422622

[2] Mirza, M., Osindero, S.: Conditional Generative Adversarial Nets. In arXiv preprint, 2014, arXiv:1411.1784. https://arxiv.org/abs/1411.1784

[3] A. Odena, C. Olah, J. Shlens, "Conditional Image Synthesis with Auxiliary Classifier GANs," in proceedings of the 34th International Conference on Machine Learning, PMLR 70:2642-2651, 2017. https://proceedings.mlr.press/v70/odena17a.html

[4] Cho, J., Yoon, K.: Conditional Activation GAN: Improved Auxiliary Classifier GAN. In IEEE Access, vol. 8, pp. 216729-216740, 2020. https://doi.org/10.1109/ACCESS.2020.3041480

[5] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P.: InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In NIPS proceedings, 2016. https://papers.nips.cc/paper/2016/hash/7c9d0b1f96aebd7b5eca8c3edaa19ebb-Abstract.html

[6] Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O.: Are GANs Created Equal? A Large-Scale Study. In NIPS, 2018. https://papers.nips.cc/paper/2018/hash/e46de7e1bcaaced9a54f1e9d0d2f800d-Abstract.html

[7] Mescheder, L., Geiger, A., Nowozin S.: Which Training Methods for GANs do actually Converge? In PMLR, 2018. https://proceedings.mlr.press/v80/mescheder18a.html

[8] Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive Growing of GANs for Improved Quality, Stability, and Variation. In ICLR conference, Vancouver, Canada, Apr. 30-May 3, 2018. https://openreview.net/forum?id=Hk99zCeAb

[9] Lecun, Y., Cortes, C., and Burges, C.: MNIST handwritten digit database, 2010. In ATT Labs. http://yann.lecun.com/exdb/mnist

[10] 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. https://papers.nips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html

[11] Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., Aila, T.: Improved precision and recall metric for assessing generative models. In NIPS proceedings, 2019. https://proceedings.neurips.cc/paper/2019/hash/0234c510bc6d908b28c70ff313743079-Abstract.html

[12] Utkarsh Ojha, Krishna Kumar Singh, Cho-Jui Hsieh, Yong Jae Lee, ”Elastic-InfoGAN: Unsupervised Disentangled Representation Learning in Class-Imbalanced Data,” in NIPS, 2020. https://proceedings.neurips.cc/paper/2020/hash/d1e39c9bda5c80ac3d8ea9d658163967-Abstract.html

[13] Jonathan Ho, Ajay Jain, Pieter Abbeel, ”Denoising Diffusion Probabilistic Models,” in NIPS, 2020.https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html

[14] Zhisheng Xiao, Karsten Kreis, Arash Vahdat, ”Tackling the Generative Learning Trilemma with Denoising Diffusion GANs”, in ICLR 2022. https://openreview.net/pdf?id=JprM0p-q0Co

[15] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, ICML 2020. https://dl.acm.org/doi/abs/10.5555/3524938.3525087

[16] Yujun Shen, Jinjin Gu, Xiaoou Tang, Bolei Zhou, ”Interpreting the Latent Space of GANs for Semantic Face Editing,” in CVPR 2020.https://openaccess.thecvf.com/content_CVPR_2020/papers/Shen_Interpreting_the_Latent_Space_of_GANs_for_Semantic_Face_Editing_CVPR_2020_paper.pdf

[17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros, in CVPR 2017. https://openaccess.thecvf.com/content_cvpr_2017/papers/Isola_Image-To-Image_Translation_With_CVPR_2017_paper.pdf

[18] Eric Jang, Shixiang Gu, and Ben Poole, ”Categorical reparameterization with gumbel-softmax,” in ICLR, 2017. https://openreview.net/pdf?id=rkE3y85ee

[19] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh, ”The concrete distribution: A continuous relaxation of discrete random variables,” in ICLR, 2017. https://openreview.net/forum?id=S1jE5L5gl

[20] W. Xia, Y. Zhang, Y. Yang, J. -H. Xue, B. Zhou and M. -H. Yang, ”GAN Inversion: A Survey,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022, doi: 10.1109/TPAMI.2022.3181070. https://ieeexplore.ieee.org/abstract/document/9792208

[21] Y. Choi, Y. Uh, J. Yoo, J. Ha, ”StarGAN v2: Diverse Image Synthesis for Multiple Domains,” in CVPR 2020. https://openaccess.thecvf.com/content_CVPR_2020/papers/Choi_StarGAN_v2_Diverse_Image_Synthesis_for_Multiple_Domains_CVPR_2020_paper.pdf