IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 13, 2020, accepted November 24, 2020, date of publication December 1, 2020,
date of current version December 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3041480

Conditional Activation GAN: Improved Auxiliary Classifier GAN

JEONGIK CHO[1] AND KYOUNGRO YOON[2], (Senior Member, IEEE)
[1]Department of Computer Science, Concordia University, Montréal, H3G 1M8 QC, Canada
[2]Department of Smart ICT Convergence Engineering, Konkuk University, 05029 Seoul, South Korea

Corresponding author: Kyoungro Yoon (yoonk@konkuk.ac.kr)

This work was supported by "University Innovation Grant" from the Ministry of Education and National Research Foundation of Korea.

ABSTRACT A conditional generative adversarial network (cGAN) is a generative adversarial network (GAN) that generates data with a desired condition from a latent vector. Among the different types of cGAN, the auxiliary classifier GAN (ACGAN) is the most frequently used. In this study, we describe the problems of an AC-GAN and propose replacing it with a conditional activation GAN (CAGAN) to reduce the number of hyperparameters and improve the training speed. The loss function of a CAGAN is defined as the sum of the loss of each GAN created for each condition. The proposed CAGAN is an integration of multiple GANs, where each GAN shares all hidden layers, and their integration can be considered as a single GAN. Therefore, the structure of the integrated GANs does not significantly increase the number of computations. Additionally, to prevent the conditions given in the discriminator of a cGAN from being ignored with batch normalization, we propose mixed batch training, in which every batch for the discriminator keeps the ratio of the real and generated data consistent.

INDEX TERMS Artificial neural networks, auxiliary classifier GAN, batch normalization, conditional GAN, deep learning, generative adversarial networks, loss function.

I. INTRODUCTION

A conditional generative adversarial network (cGAN) [1] is a generative adversarial network (GAN) [2] that can generate data with a desired condition from a latent vector. Among the various cGANs [3], [4], the most frequently used is the auxiliary classifier GAN (ACGAN) [5]–[11]. Some studies have used variations of ACGAN [10], [11], without giving any details on the rationalization of the variations made. In this study, we describe the reasons for modifying an ACGAN and its disadvantages.

In an ACGAN, when the real and generated data distributions are the same, the auxiliary classifier of the discriminator and generator can be considered as a group of GANs, wherein each GAN trains condition using cross-entropy adversarial loss and shares all hidden layers. Considering an ACGAN as a set of GANs, the generated data classification loss of the ACGAN discriminator loss interferes with the training of each GAN and hence is removed in the modified ACGAN.

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Sharif.

Each GAN can only be trained when the real and generated data distributions are the same; hence, a problem might arise wherein each individual GAN may not be trained at the beginning of the ACGAN training.

Additionally, to use the advanced adversarial loss, as applied in the least squares GAN (LSGAN) [12] or the Wasserstein GAN-gradient penalty (WGAN-GP) [13] in ACGAN, it is necessary to determine a hyperparameter that adjusts the ratio of adversarial loss to classification loss.

We propose a conditional activation GAN (CAGAN) that can replace an ACGAN to reduce the number of hyperparameters and improve the training speed (i.e., performance increase per epoch) to overcome the ACGAN problems mentioned above. The loss of a CAGAN is the sum of the losses of each GAN when it is created for each condition. Each GAN shares all hidden layers, and thus, a CAGAN created through the conceptual aggregation of an individual GAN can be considered a single GAN.

Unlike an ACGAN, which uses two losses (i.e., adversarial and classification losses), a CAGAN uses a conditional activation loss, and thus, there is no need to find the proper adversarial to classification loss ratio.

In ACGAN, it begins to train each condition when the real data distribution is the same as the generated data distribution; whereas a CAGAN always trains all conditions simultaneously, which indicates that it always produces meaningful gradients, even during the early training stage. Therefore, the CAGAN's performance is better than that of ACGAN.

A cGAN is trained by applying batch normalization [14] to the discriminator that causes the generator to distort the distribution of the input conditions.

When batch normalization is applied to the discriminator, and when the distribution of the real and generated data differs, the discriminator may use the distribution conditions of the batch to discriminate between the real and generated data. Further, the distribution conditions of the generated data follows that of the real data, rather than the distribution of the input target conditions.

To prevent the generator from ignoring the distribution of the input target conditions, we suggest applying a mixed batch training technique, which is used to configure each batch for the discriminator under the same ratio of real to generated data, such that each batch always has a mixture of real and generated data, in the same ratio

However, if mixed batch training technique is applied prior training, it is essential to classify the generated and real data in a single batch; consequently, training of the discriminator is not required. Therefore, to apply mixed batch training, each batch's ratio of real data and generated data is gradually changed to the target ratio as training progresses. After the ratio of each batch reaches the target ratio, training proceeds without changing the ratio.

II. RELATED WORKS

cGAN [1] was proposed to generate data with the desired conditions. The generator of cGAN generates data with the desired condition by receiving a pair consisting of a latent vector and a condition vector. The discriminator discriminates a pair consisting of the real data and real data condition vector(labels) as real, and a pair consisting of generated data and a corresponding condition vector as fake. Subsequently, ACGAN, which is an extended version of cGAN, was proposed.

There is an auxiliary classifier in the ACGAN discriminator. This classifier is trained to classify the conditions of real data and generated data correctly. Additionally, generators are trained to classify the generated data correctly from the auxiliary classifiers. It is possible to change the image attribute using an image instead of a latent vector as input to the ACGAN generator and adding an additional reconstruction loss to the generator [10], [11]. Vanilla GAN [2] uses cross-entropy as an adversarial loss, which reduces the Jensen-Shannon divergence between real data and generated data. However, because the Jensen-Shannon divergence always has almost the same value when the distance between the real data distribution and the generated data distribution is similar to the values prior training; consequently, this indicates that there is almost no gradient [25]. Therefore, results obtained from the GAN training is insufficient. Different divergences or distances between the two distributions or adversarial losses using different methods [22]–[25] were proposed to solve this problem. Among these, LSGAN and WGAN-GP are widely used. Mario *et al.* [20] compared the performance of several adversarial losses. Sebastian *et al.* [22] proposed a method to improve the performance of GAN, wherein each batch was composed of randomly selected real data and data generated by a Bernoulli trial. Subsequently, proposed a GAN with unique adversarial loss that trains the discriminator to infer the ratio of real data in each batch.

III. ANALYSIS OF AUXILIARY CLASSIFIER GAN

The loss of an ACGAN is defined as follows:

$$L_d = L^d_{adv} + L^r_{cls} + L^g_{cls} \tag{1}$$

$$L_g = L^g_{adv} + L^r_{cls} + L^g_{cls} \tag{2}$$

$$L^r_{cls} = E_{cnd,x}\left[-\log D_{cls}\left(cnd \mid x\right)\right] \tag{3}$$

$$L^g_{cls} = E_{cnd',z}\left[-\log D_{cls}\left(cnd' \mid G\left(cnd', z\right)\right)\right] \tag{4}$$

$$\begin{aligned} L^d_{adv} = {} & E_x\left[-\log D_{adv}\left(x\right)\right] \\ & + E_{cnd',z}\left[-\log\left(1 - D_{adv}\left(G\left(cnd', z\right)\right)\right)\right] \end{aligned} \tag{5}$$

$$L^g_{adv} = E_{cnd',z}\left[\log\left(1 - D_{adv}\left(G\left(cnd', z\right)\right)\right)\right] \tag{6}$$

In (1) and (2), L_d is the loss of the discriminator, and L_g is the loss of the generator. L^d_{adv} and L^g_{adv} are the adversarial loss of the discriminator and generator, respectively. Similarly, L^r_{cls} and L^g_{cls} are the classification loss of real data and generated data, respectively.

In (3), E is the expectation of the given variable; x is the real data, and cnd is a binary vector that expresses the conditions of real data x. $D_{cls}(k)$ is the probability distribution of input data k within the auxiliary classifier of the discriminator, and $-\log D_{cls}(b \mid k)$ is the cross-entropy loss between binary vectors b and $D_{cls}(k)$. Minimizing $-\log D_{cls}(b \mid k)$ implies that D_{cls} is being trained to estimate binary vector b when the input data is k.

In (4), cnd' is the target condition binary vector, and z is the latent vector. $G\left(cnd', z\right)$ is the generated data distribution by generator G with the target condition binary vector cnd' and latent vector z.

In (5) and (6), D_{adv} is the probability distribution function of the data in the adversarial module, and $D_{adv}(k)$ is the probability distribution of k, which is given as the input to the adversarial module.

Note that L^r_{cls} in L_g does not have any role, because the generator does not affect the calculation of L^r_{cls}.

In ACGAN, when the real and generated data distributions are the same, the auxiliary classifier of the discriminator and the generator can be considered as a group of GANs, wherein each GAN trains each condition using cross-entropy adversarial loss and shares all hidden layers, as depicted in Fig. 1.

Assume that an ACGAN training three independent conditions (A, B, and C) does so using only the adversarial loss, and that the real and generated data distributions are the same.

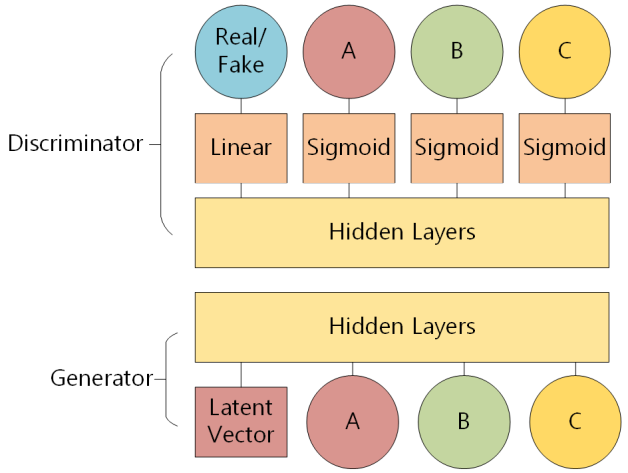

FIGURE 1. ACGAN training conditions A, B, and C.

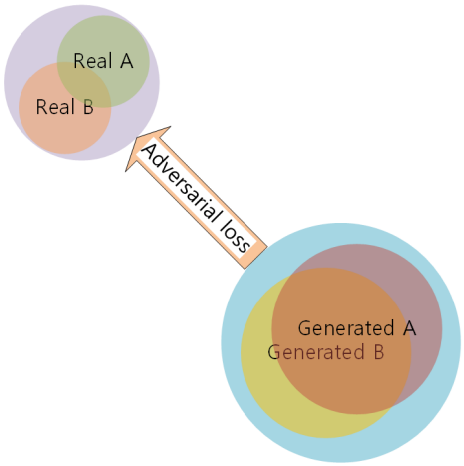

FIGURE 2. Data distribution at the start of the training using ACGAN.

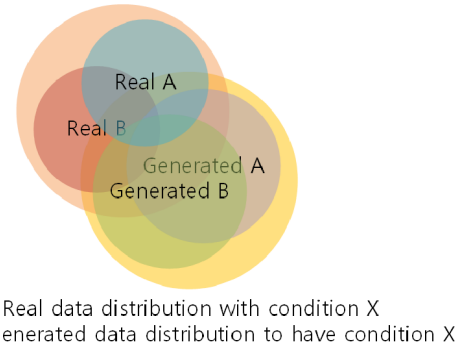

FIGURE 3. Distribution of generated data after training using ACGAN.

Node A of the discriminator is trained by $L^r_{cls}[A]$ in L_d to output 1 to represent real data when it receives real data under condition A, and 0 to represent generated data under a not-A condition.

When the generator receives 1 as its node A input, it attempts to generate data using $L^g_{cls}[A]\,L_g$ under condition A and trains node A of the discriminator to output 1.

If the generator attempts to generate data under condition A but fails, the generated data distribution will be close to the real data distribution with a not-A condition; consequently, it is assumed that the real and generated data distributions are the same.

Thus, the hidden layers of the discriminator and node A, hidden layers of the generator and the latent vector input, and node A itself can be considered as a single GAN A that generates data with condition A, trained by $L^r_{cls}[A]$ in L_d and $L^g_{cls}[A]$ in L_g. However, $L^g_{cls}[A]$ in L_d trains node A of the discriminator to output 1, which represents a real value—when the discriminator receives the generated data. Therefore, $L^g_{cls}[A]$ in L_d interferes with the training of GAN A.

Additionally, when the generator receives 0 as its node A input, it can be considered as a GAN that generates data under a not-A condition.

An ACGAN uses cross-entropy loss as an adversarial loss. However, to use an advanced adversarial loss, such as that of LSGAN or WGAN-GP, a hyperparameter is needed to adjust the ratio of adversarial loss to classification loss.

To solve these problems, the loss of the modified ACGANs used in StarGAN [10] or AttGAN [11] and is modified as follows:

$$L_d = L^d_{adv} + \lambda_{cls} L^r_{cls} \tag{7}$$

$$L_g = L^g_{adv} + \lambda_{cls} L^g_{cls} \tag{8}$$

In (7) and (8), L_d is the loss of the discriminator, and L_g is the loss of the generator. L^d_{adv} is the advanced adversarial loss of the discriminator, and L^g_{adv} is the advanced adversarial loss of the generator. L^r_{cls} and L^g_{cls} are the same as in (3) and (4), respectively. λ_{cls} is a hyperparameter representing the classification loss weight.

As described above, a modified ACGAN can also be considered as a group of GANs. However, each GAN can only be trained as a GAN for each condition if the real and generated data distributions for the corresponding condition are the same.

In other words, if the real and generated data distributions differ at early stages of training, the training does not proceed with the classification loss but only with the adversarial loss, as shown in Fig. 2.

By training using adversarial loss, the real and generated data distributions become closer. As these distributions move closer to each other, the classification loss gradually acts as the cross-entropy adversarial loss of each GAN, producing meaningful gradients, and training is performed to generate data under each condition.

An ACGAN has the disadvantage of requiring one additional hyperparameter to adjust the ratio of adversarial to classification loss in both the discriminator and generator and does not produce meaningful gradients during the early training stage.

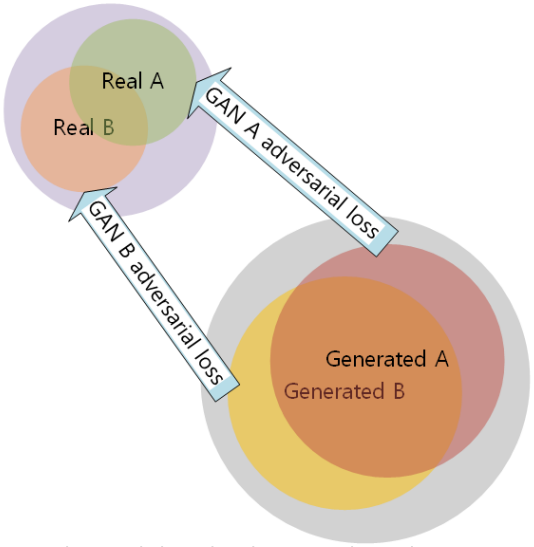

FIGURE 4. Data distribution at the beginning of training using CAGAN.

IV. CONDITIONAL ACTIVATION GAN (CAGAN)

To solve these ACGAN problems, we propose a CAGAN that is similar to having multiple GANs, each of which is defined to train the corresponding condition.

The loss of a CAGAN is the sum of each loss of a GAN, wherein each GAN trains only one condition, as expressed by the following:

$$L_d = \sum\nolimits_{\forall c \in cnd} L_{d_c} \tag{9}$$

$$L_g = \sum\nolimits_{\forall c \in cnd} L_{g_c} \tag{10}$$

$$L_{d_c} = E_{c,x}\left[c \times f_r^d\left(D_c\left(x\right)\right)\right] + E_z\left[f_g^d\left(D_c\left(G_c\left(1,z\right)\right)\right)\right] \tag{11}$$

$$L_{g_c} = E_z\left[f^g\left(D_c\left(G_c\left(1,z\right)\right)\right)\right] \tag{12}$$

In (9) and (10), L_d and L_g represent the discriminator and generator losses of the CAGAN, respectively. In $\sum_{\forall c \in cnd}()$, c is a specific binary condition scalar in the binary condition vector cnd, and GAN c is an individual GAN trained for only condition c.

In addition, G_c and D_c are the generator and discriminator of GAN c, respectively, where G_c receives a binary activation value with a latent vector. If G_c receives 1 as an activation value, G_c tries to trick D_c, and consequently D_c tries to discriminate the generated data from G_c as fake. If G_c receives 0 as the activation value, the generated data does not depend on G_c and D_c. D_c is only concerned with discriminating real data, which have conditions c, and does not pay attention to other real data, including real data with a not-condition c.

In (11), c is a binary scalar that expresses the condition of real data x, and $G_c(1, z)$ is the generated data distribution by G_c when it receives latent vector z with 1 as the activation value.

Here, f_r^d is a function used to calculate the adversarial loss of the discriminator corresponding to the real data, and f_g^d is a function that calculates the adversarial loss of the discriminator corresponding to the generated data. In (14), f^g is a function that calculates the adversarial loss of the generator.

The following equation is an example of the adversarial loss of GAN c, which uses the adversarial loss given in LSGAN [12].

$$L_{d_c} = E_{c,x}\left[c \times \left(D_c\left(x\right) - 1\right)^2\right] + E_z\left[D_c\left(G\left(1,z\right)\right)^2\right] \tag{13}$$

$$L_{g_c} = E_z\left[\left(D_c\left(G\left(1,z\right)\right) - 1\right)^2\right] \tag{14}$$

In CAGAN, because each GAN shares all hidden layers, a conditional activation loss can be changed through the following equations:

$$L_d = E_{cnd,x}\left[f_r^d\left(D\left(x\right)\right) \cdot cnd\right] + E_{cnd',z}\left[f_g^d\left(D\left(G\left(cnd',z\right)\right)\right) \cdot cnd'\right] \tag{15}$$

$$L_g = E_{cnd',z}\left[f^g\left(D\left(G\left(cnd',z\right)\right)\right) \cdot cnd'\right] \tag{16}$$

In (15) and (16), "$\cdot$" is an inner product (element-wise sum of products).

The following equation expresses the loss of CAGAN when using the adversarial loss of LSGAN.

$$L_d = E_{cnd,x}\left[\left(D\left(x\right) - 1\right)^2 \cdot cnd\right] + E_{cnd',z}\left[\left(D\left(G\left(cnd',z\right)\right)\right)^2 \cdot cnd'\right] \tag{17}$$

$$L_g = E_{cnd',z}\left[\left(D\left(G\left(cnd',z\right)\right) - 1\right)^2 \cdot cnd'\right] \tag{18}$$

Likewise, the loss of CAGAN when using the adversarial loss of WGAN-GP can be defined by the following equation:

$$L_d = E_{x,cnd\sim P_r(x,cnd)}\left[-D\left(x\right) \cdot cnd\right] + E_{x',cnd'\sim P_g(x',cnd')}\left[D\left(x'\right) \cdot cnd'\right] + \lambda_{gp} gp\_loss \tag{19}$$

$$gp\_loss = E_{\hat{x}}\left[\left(\left\|\nabla_{\hat{x}} average\left(D\left(\hat{x}\right)\right)\right\|_2 - 1\right)^2\right] \tag{20}$$

$$L_g = E_{x',cnd'\sim P_g(x',cnd')}\left[-D\left(x'\right) \cdot cnd'\right] \tag{21}$$

In (19), λ_{gp} and gp_loss represent the gradient penalty loss weight and gradient penalty loss, respectively. gp_loss is the average of each GAN's gradient penalty loss. In (20), $\hat{x}$ is data uniformly sampled from a straight line between x and x'. $average$ is a function that calculates the average of the input vector.

In ACGAN, GAN A, which trains condition A, generates data with the not-A condition as well as those with condition A.

However, in a CAGAN, because GAN A disregards the not-A condition when training with condition A, a new GAN must be added to train the not-A condition.

$Assuming P(Black\ hair) + P(Blond\ hair) + P(Bald) = 1$ and $P(Male) + P(Female) = 1$, Figs. 5–8 show the input

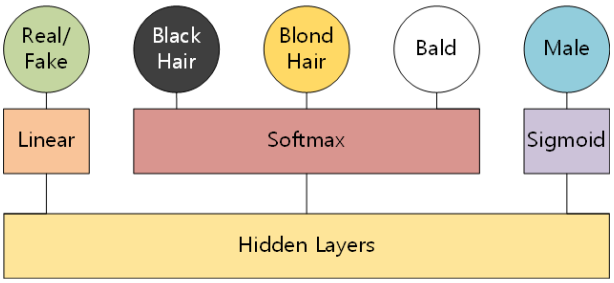

FIGURE 5. ACGAN discriminator output part.

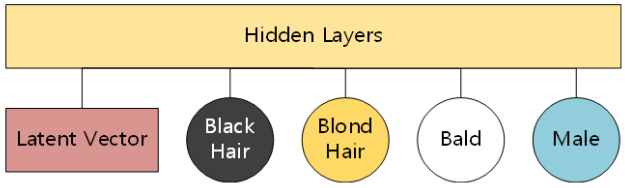

FIGURE 6. ACGAN generator input part.

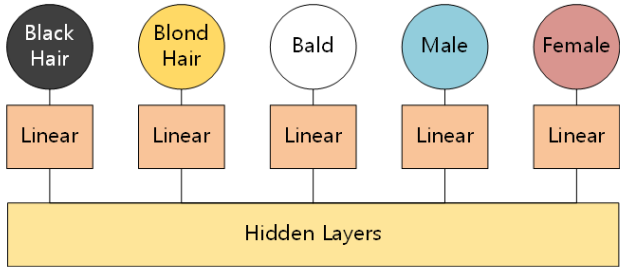

FIGURE 7. CAGAN discriminator output part.

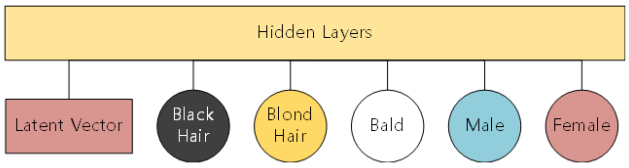

FIGURE 8. CAGAN generator input part.

and output parts of ACGAN and CAGAN that train these conditions.

In CAGAN, meaningful gradients are generated even at the early stages of training because each GAN can be trained through an advanced adversarial loss that generates meaningful gradients, even if the real and generated data distributions differ. For this reason, the performance of CAGAN is better than that of ACGAN.

Additionally, unlike ACGAN, which uses two losses (adversarial and classification losses), CAGAN uses only one loss (conditional activation loss), and thus, there is no need to find the proper adversarial to classification loss ratio. This indicates that it takes less time to search for an important hyperparameter, that is, the ratio of adversarial to classification loss.

V. MIXED BATCH TRAINING

A cGAN is trained by applying batch normalization to the discriminator that may cause the generator to distort the distribution of input conditions.

When batch normalization is applied to the discriminator and the distribution of target conditions used for training is different from that of the real data, the discriminator may use the distribution conditions on the batch to discriminate between real and generated data. Further, the distribution conditions in the generated data to follow that of real data. To prevent the generator from ignoring the distribution of conditions in the input target, we suggest using mixed batch training.

Mixed batch training technique is used to configure each batch, allowing the discriminator to maintain the same ratio of real to generated data continuously. Each training batch is configured to maintain the same ratio of real to generated data; thus, the discriminator will not discriminate between the real and generated data through a distribution of conditions, and the generator will not attempt to follow the distribution of conditions in real data.

Additionally, if the ratio of real data or generated data in each batch is not constant, for example, when $\frac{real\ data\ size}{batch\ size} \sim U(0, 1)$ because the generator cannot determine the ratio of real data or generated data of each batch, it is advantageous for the generator to follow the distribution of conditions in real data. This causes the generator to ignore the input condition as if mixed batch training is not applied. Therefore, to apply mixed batch training, each batch should always consist of the same ratio of real data and generated data.

However, if mixed batch training is applied prior training, it is easy for the discriminator to discriminate the generated data from the real data in the batch, and the training hardly proceeds.

Therefore, the ratio of real data and generated data of each batch gradually changes to the target ratio as training progresses. For example, when the target ratio of ''real data: generated data'' is 50:50, at the beginning of training, the ratio of each batch is ''100:0 and 0:100.'' As training progresses, the ratio changes to ''20:80 and 80:20,'' ''40:60 and 60:40,'' and finally becomes ''50:50 and 50:50.'' After the ratio of each batch reaches the target ratio, training proceeds without changing the ratio. The ratio of ''real data: generated data'' that changes for each epoch or iteration is an additional hyperparameter used for mixed batch training. This hyperparameter exists to prevent training from failing early, so it has little impact on the model's final performance and is very easy to find. Additionally, when the training generator and discriminator are unbalanced, the target ratio may not be 50:50, but in general, 50:50 is used.

VI. MATERIAL AND METHODS

We conducted experiments on the MNIST handwriting number dataset [15] and Celeb A dataset [26]. Tensorflow2 was used [18].

To evaluate the proposed network, the average Fréchet inception distance (FID) [19] in overall conditions were used. The better the GAN's performance, the lower the FID. In all experiments, *Adam optimizer* [17] with $beta1 = 0.9$, $beta2 = 0.999$, and $batch\ size = 32$ were used. The size of the generated dataset was the same as the size of each test dataset during the evaluation.

All experiments were conducted three times, and the results were averaged.

The same row and column of the generated images have the same latent vector and condition vector, respectively.

A. MNIST EXPERIMENT

In this experiment, we used the MNIST handwriting number dataset, which has 60,000 training images and 10,000 test images, with a pixel resolution of 28×28 and a channel size of 1. The conditions to train are the type of number (0–9). The average of the FIDs for each number was used for the evaluation.

The basic design of a Deep Convolutional GAN(DCGAN) [16], applying instance normalization to both the generator and discriminator, was used for the model architecture. The adversarial loss of LSGAN was used.

For both ACGAN and CAGAN, the generator received a 10-dimensional condition vector and a 256-dimensional latent vector following a normal distribution.

The output dimension of the ACGAN discriminator is $1 + 10 = 11$. The output dimension of the CAGAN discriminator was 10. We used $learning\ rate = 10^{-5}$ for both generator and discriminator training.

In the original MNIST handwriting number training dataset, the number of images for each number is approximately the same. For the mixed batch training experiment, we intentionally used a dataset consisting of 5,500 zeros and 500 other numbers from 1 to 9 (each from the MNIST handwriting number training dataset) to create an unbalanced dataset. Zeros in the dataset occupy 55% of the 10,000 images, and each of the remaining numbers (1–9) accounts for 5%. Because the size of the training data was reduced to 1/6, the epoch was doubled ($epoch = 100$), and the learning rate was tripled ($learning\ rate = 3 \times 10^{-5}$) for this experiment. Since the epoch doubled, the FID was measured every two epochs. We replaced all instance normalization layers in the discriminator with batch normalization layers. For mixed batch training, we used $batch\ ratio\ change\ per\ epoch = 1\%p$. This means that the batch ratio gradually changes to the target ratio by $1\%p$ for epochs 1–50. Additionally, epochs 51–100 were trained with $real\ data : generated\ data = 50 : 50$.

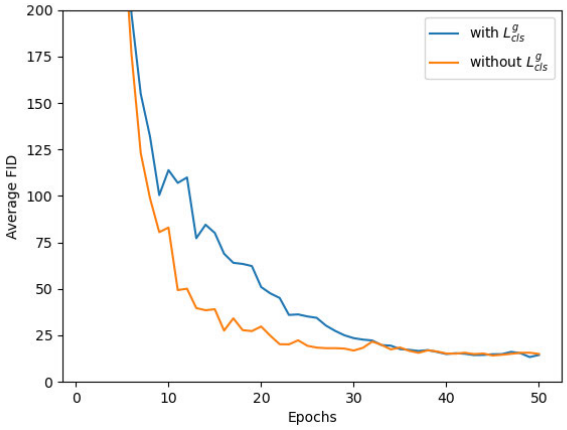

FIGURE 9. Effect of L^g_{cls} on modified ACGAN performance using MNIST.

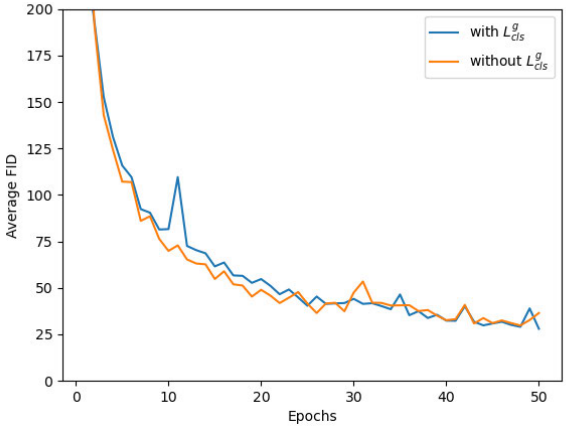

FIGURE 10. Effect of L^g_{cls} on modified ACGAN performance using Celeb A.

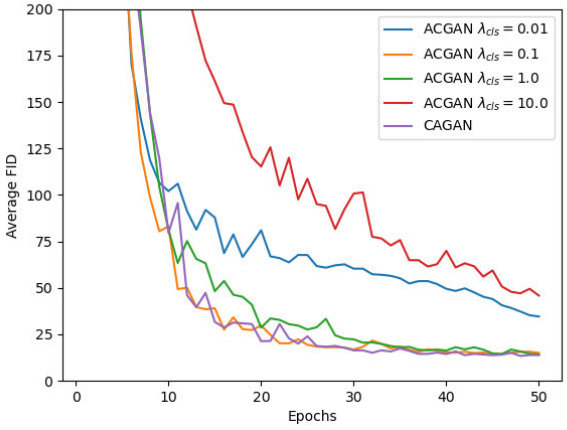

FIGURE 11. Performance comparison between modified ACGAN with different classification loss weights and CAGAN, using MNIST.

B. CELEB A DATASET

In this experiment, we used the Celeb A dataset, which has 162,770 training images, 19,867 validation images, and 19,962 test images. We used validation images for hyperparameter tuning and test images for evaluation. We cropped the 128×128 pixels at the center of the image and then downscaled it to 64×64. The channel size of the images was 3. There were three conditions to train: "Black or Brown hair," "Male," "Smiling." The average of the $2^3 = 8$ possible combinations of FIDs was used for the evaluation.

The basic design of a DCGAN, with instance normalization on the discriminator and batch normalization on the

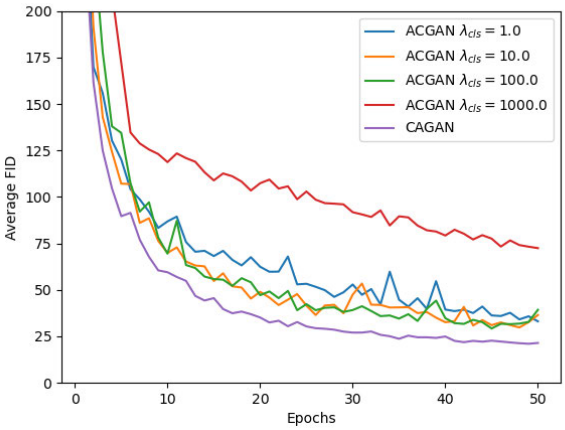

FIGURE 12. Performance comparison between modified ACGAN with different classification loss weights and CAGAN, using Celeb A.

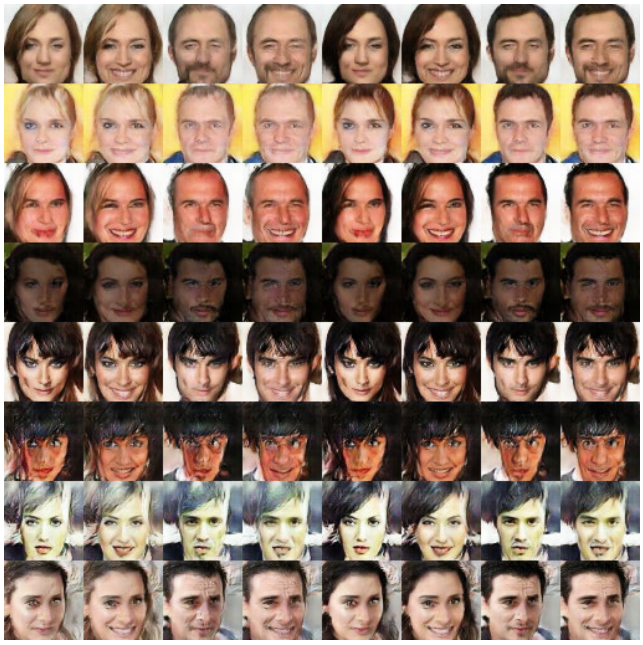

FIGURE 13. Data generated by CAGAN in the Celeb A experiment after 50 epochs.

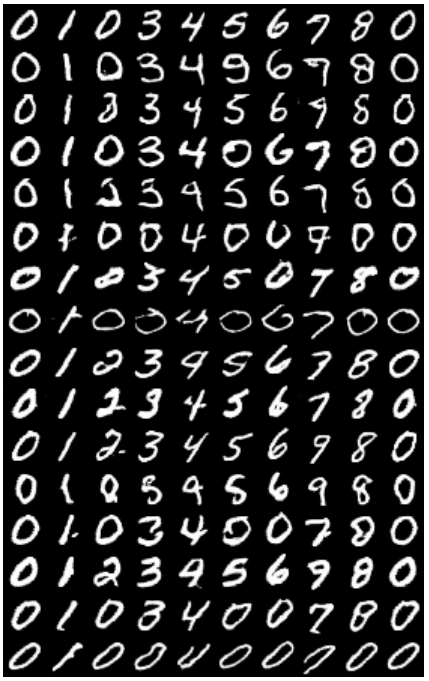

FIGURE 14. Data generated by modified ACGAN after 100 epochs, without mixed batch training using MNIST.

FIGURE 15. Data generated by modified ACGAN after 100 epochs, with mixed batch training using MNIST.

generator, was used for the model architecture. An adversarial loss of WGAN-GP $\lambda_{gp} = 0.1$ was used.

For ACGAN, the generator received a 3-D condition vector and a 512-D latent vector following a normal distribution. The output dimension of the discriminator of ACGAN was $1 + 3 = 4$.

The CAGAN generator received a $2 \times 3 = 6$-D condition vector. The output dimension of the CAGAN discriminator was $2 \times 3 = 6$.

In ACGAN and CAGAN experiments, we used *learning rate* $= 10^{-5}$, *epoch* $= 50$.

In the mixed batch training experiment, we trained the unbalanced conditions: "Black hair," "Bangs," and "Young." These conditions occupy 23.90%, 15.17%, and 77.89% of the training data, respectively. We replaced all instance normalization layers in the discriminator with batch normalization layers. Because WGAN-GP cannot apply batch normalization to the discriminator [13], we changed adversarial loss to LSGAN adversarial loss. Because the adversarial loss and model architecture changed WGAN-GP to LSGAN and instance normalization to batch normalization; subsequently, we also changed the λ_{cls} and learning rate. We used $\lambda_{cls} = 0.1$ and *learning rate* $= 3 \times 10^{-5}$ to generator optimizer, and *learningrate* $= 3 \times 10^{-6}$ to discriminator

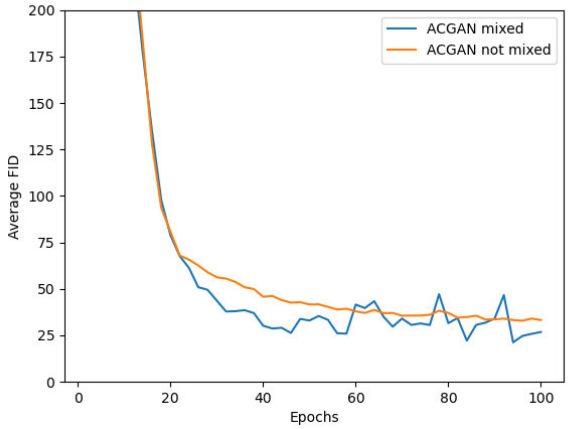

FIGURE 16. Comparison of mixed batch training for modified ACGAN using MNIST.

FIGURE 17. Data generated by CAGAN after 100 epochs, without mixed batch training using MNIST.

FIGURE 18. Data generated by CAGAN after 100 epochs, with mixed batch training using MNIST.

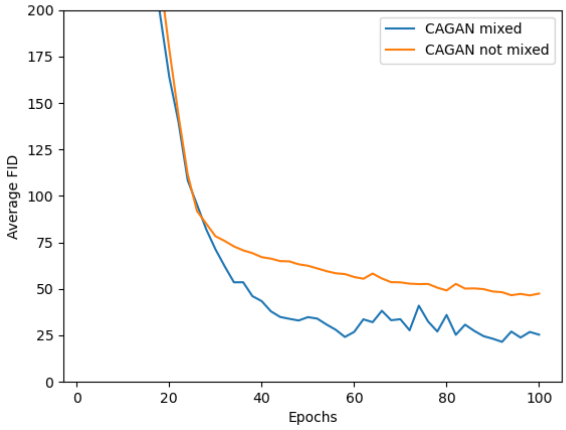

FIGURE 19. Mixed batch training performance comparison for CAGAN using MNIST.

optimizer for mixed batch training experiments. We used *batch ratio change per epoch* $= 2.5\%p$ for mixed batch training. Therefore, the batch ratio gradually changed to the target ratio by 2.5% p for epochs 1–20. Epochs 21–50 were trained with *real data* : *generated data* $= 50 : 50$.

VII. EXPERIMENTAL RESULTS AND DISCUSSION

A. ACGAN AND CAGAN

First, we compared the performance of a modified ACGAN, with and without L^g_{cls} discriminator loss, when an adversarial loss occurred.

In Figs. 9 and 10, the blue line shows the average FID of the modified ACGAN, with $L_d = L^d_{adv} + \lambda_{cls}\left(L^r_{cls} + L^g_{cls}\right)$, and the orange line shows the average FID of the modified ACGAN, with $L_d = L^d_{adv} + \lambda_{cls}L^r_{cls}$. The MNIST experiment used $\lambda_{cls} = 0.1$, and the Celeb A experiment used $\lambda_{cls} = 10.0$. Both λ_{cls} are hyperparameters that show the best performance in each ACGAN.

As both graphs indicate, the performance of the network without L^g_{cls} is better.

The next experiments compare the performance of ACGAN with different λ_{cls} and CAGAN.

Figs. 11 and 12 show the FID when classification loss weight λ_{cls} varies with the adversarial loss weight fixed at 1.0. The changes in training speed and the quality of the results as the ratio of adversarial to classification loss weight

FIGURE 20. Data generated by modified ACGAN after 50 epochs, without mixed batch training using Celeb A.

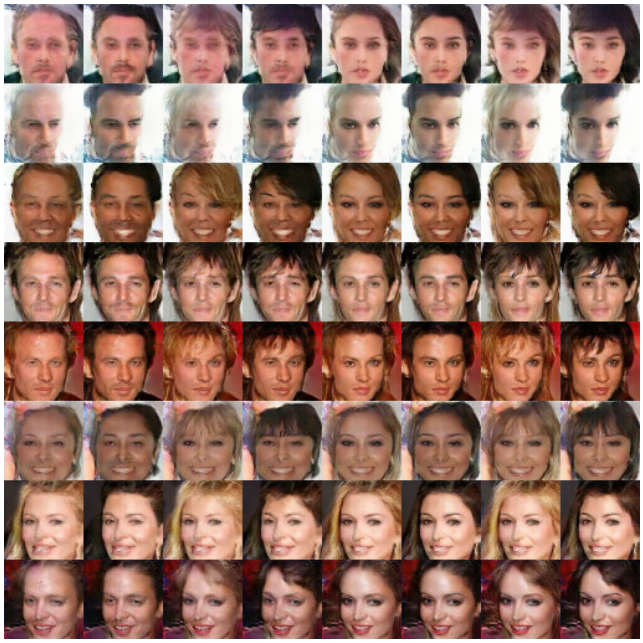

FIGURE 21. Data generated by modified ACGAN after 50 epochs, with mixed batch training using Celeb A.

changes are clearly shown in this graph. Additionally, both figures show that the performance of CAGAN is similar to or better than that of the modified ACGAN when we use a good hyperparameter (λ_{cls}).

Fig. 13 shows the data generated by CAGAN in the Celeb A experiment after 50 epochs. Columns1–4 have the condition ''Not black or brown hair,'' and 5–8 have the condition ''Black or brown hair.'' Columns 1, 2, 5, and 6 have the

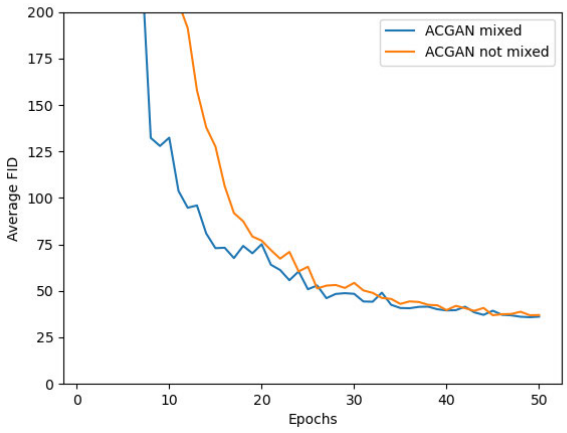

FIGURE 22. Performance comparison of mixed batch training for modified ACGAN using Celeb A.

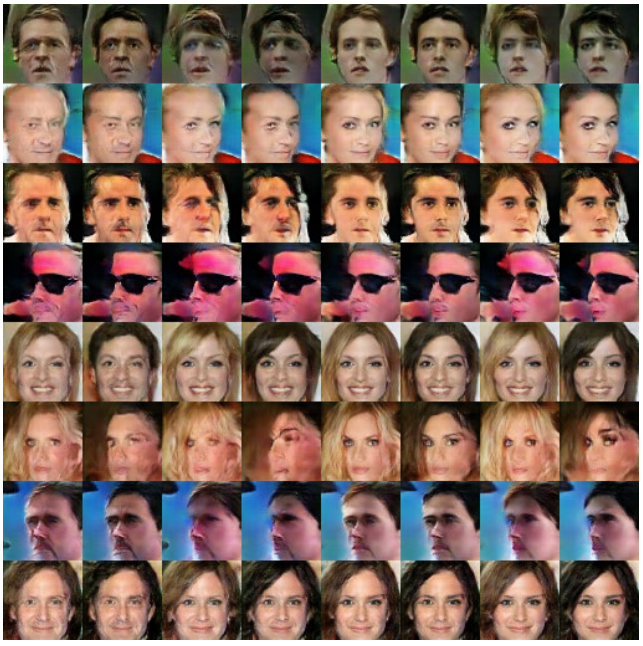

FIGURE 23. Data generated by CAGAN after 50 epochs, without mixed batch training using Celeb A.

condition ''Not Male,'' and columns 3, 4, 7, and 8 have the condition ''Male.'' Columns 1, 3, 5, and 7 have the condition ''Not smiling,'' and columns 2, 4, 6, and 8 have the condition ''Smiling.''

B. MIXED BATCH TRAINING

Figs. 14 and 15 show the data generated by the modified ACGAN after 100 epochs without and with mixed batch training in the MNIST experiment, respectively. Fig. 16 compares these two cases based on the FID. Figs. 17–19 show the same experiment.

These figures clearly show the effectiveness of mixed batch training in a modified ACGAN.

Figs. 17 and 18 show the data generated by the CAGAN after 100 epochs, without and with mixed batch training, respectively. Fig. 19 compares these two cases based on FID.

FIGURE 24. Data generated by CAGAN after 50 epochs, with mixed batch training using Celeb A.

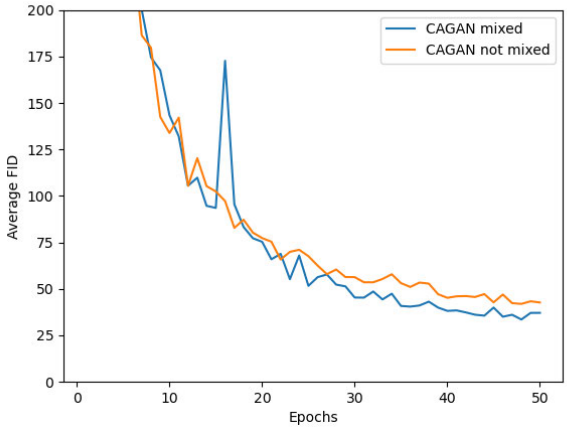

FIGURE 25. Mixed batch training performance comparison for CAGAN in the Celeb A experiment.

Condition vector ([3]), Latent vector ([512]) for ACGAN	Condition vector ([6]), Latent vector ([512]) for CAGAN
Fully connected ([4 × 4 × 1024])	
Reshape ([4, 4, 1024])	
Transposed convolution (filters=512, kernel_size=5, strides=2, normalization=Batch norm, activation=ReLU)	
Transposed convolution (filters=256, kernel_size=5, strides=2, normalization=Batch norm, activation=ReLU)	
Transposed convolution (filters=128, kernel_size=5, strides=2, normalization=Batch norm, activation=ReLU)	
Transposed convolution (filters=64, kernel_size=5, strides=2, normalization=Batch norm, activation=ReLU)	
Convolution (filters=3, kernel_size=1, activation=Tanh)	

FIGURE 26. Generator architecture for Celeb A experiments.

Input image ([64, 64, 3])	
Convolution (filters=64, kernel_size=5, normalization=Instance or Batch norm, activation=LeakyReLU)	
Convolution (filters=128, kernel_size=5, strides=2, normalization=Instance or Batch norm, activation=LeakyReLU)	
Convolution (filters=256, kernel_size=5, strides=2, normalization=Instance or Batch norm, activation=LeakyReLU)	
Convolution (filters=512, kernel_size=5, strides=2, normalization=Instance or Batch norm, activation=LeakyReLU)	
Convolution (filters=1024, kernel_size=5, strides=2, normalization=Instance or Batch norm, activation=LeakyReLU)	
Flatten ()	
Fully connected (1, activation=Linear), Fully connected (3, activation=Sigmoid) for ACGAN	Fully connected (6, activation=Linear) for CAGAN

FIGURE 27. Discriminator architecture for Celeb A experiments.

The generated data clearly show that both the modified ACGAN and CAGAN without mixed batch training ignore the conditional vectors and generate many zeros, following the distribution of conditions in the training data. The results shown in Figs. 14–19 clearly demonstrate that the performance with mixed batch training is better than that without such training in both the modified ACGAN and CAGAN.

In particular, Figs. 15 and 18 show that mixed batch training in a conditional GAN can prevent the conditional vector from being ignored.

The same experiment was repeated on the Celeb A dataset.

Figs. 20–25 show the results of the Celeb A experiment.

In all generated data in Figs. 20–25, columns 1–4 have condition "Not young," and 5–8 have condition "Young." Columns 1, 2, 5, and 6 have condition "Not Bangs," and columns 3, 4, 7, and 8 have condition "Bangs." Columns 1, 3, 5, and 7 have condition "Not black hair," and columns 2, 4, 6, and 8 have condition "Black hair."

The results shown in Figs. 20–25 indicate that mixed batch training improves the performance of modified ACGAN and CAGAN.

VIII. CONCLUSION

In this study, we interpreted an ACGAN as a set of GANs, described why the generated data classification loss of the discriminator loss in an ACGAN interferes with the training, and confirmed this theory experimentally.

Condition vector ([10]), Latent vector ([256])
Fully connected layer ([7 × 7 × 256], activation=Leaky ReLU)
Reshape ([7, 7, 256])
Instance norm ()
Up sampling 2D ()
Convolution (filters=128, kernel_size=3, activation=Leaky ReLU)
Instance norm ()
Convolution (filters=128, kenrel_size=3, activation=Leaky ReLU)
Instance norm ()
Convolution (filters=128, kernel_size=3, activation=Leaky ReLU)
Instance norm ()
Up sampling 2D ()
Convolution (filters=64, kernel_size=3, activation=Leaky ReLU)
Instance norm ()
Convolution (filters=64, kernel_size=3, activation=Leaky ReLU)
Instance norm ()
Convolution (filters=64, kernel_size=3, activation=Leaky ReLU)
Instance norm ()
Convolution (filters=1, kernel_size=1, activation=Tanh)

FIGURE 28. Generator architecture for MNIST experiments.

Input image ([28, 28, 1])	
Convolution (filters=64, kernel_size=3, activation=Leaky ReLU)	
Instance or Batch norm ()	
Average pooling ()	
Convolution (filters=128, kernel_size=3, activation=Leaky ReLU)	
Instance or Batch norm ()	
Convolution (filters=128, kernel_size=3, activation=Leaky ReLU)	
Instance or Batch norm ()	
Convolution (filters=128, kernel_size=3, activation=Leaky ReLU)	
Instance or Batch norm ()	
Average pooling ()	
Convolution (filters=256, kernel_size=3, activation=Leaky ReLU)	
Instance or Batch norm ()	
Convolution (filters=256, kernel_size=3, activation=Leaky ReLU)	
Instance or Batch norm ()	
Convolution (filters=256, kernel_size=3, activation=Leaky ReLU)	
Instance or Batch norm ()	
Flatten ()	
Fully connected (1, activation=Linear), Fully connected (10, activation=Softmax) for ACGAN	Fully connected (10, activation=Linear) for CAGAN

FIGURE 29. Discriminator architecture for MNIST experiments.

Based on this interpretation, we proposed a novel CAGAN, which can be interpreted as an integration of GANs in which each individual GAN trains only one condition. Unlike the modified ACGAN, CAGAN generates a meaningful gradient even at the early stages of training; consequently, the training is significantly faster, as demonstrated by our experiments.

CAGAN is expected to be used as a replacement for the modified ACGAN in many GAN applications because the former has fewer hyperparameters and trains faster while remaining compatible with ACGAN.

We also predicted that the discriminator, with batch normalization, might use a distribution of conditions in each batch to discriminate between real and generated data in a cGAN, which degrades the performance.

To prevent this degradation, we propose the use of mixed batch training, which is configured for each batch for a discriminator with the same ratio of real to generated data such that each batch always has the same distribution of conditions. Based on the experiments, the performance improvement of cGANs (i.e., modified ACGANs and CAGANs) were confirmed through the results of mixed batch training.

Mixed batch training is expected to help train cGANs using batch normalization for discriminators.

In conclusion, the proposed CAGAN provides better performance than ACGAN in terms of the training speed (performance increase per epoch) and hyperparameter search. Mixed batch training also improves the performance of a somewhat trained conditional GAN by inducing healthy competition between the generator and discriminator.

APPENDIX

All codes used in the MNIST and Celeb A experiments are available at https://github.com/jeongik-jo/CAGAN-MNIST and https://github.com/jeongik-jo/CAGAN-CelebA, respectively.

REFERENCES

[1] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*. [Online]. Available: http://arxiv.org/abs/1411.1784

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 2672–2680. [Online]. Available: https://papers.nips.cc/paper/5423-generative-adversarial-nets

[3] T. Kaneko, K. Hiramatsu, and K. Kashino, "Generative attribute controller with conditional filtered generative adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6089–6098. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2017/html/Kaneko_Generative_Attribute_Controller_CVPR_2017_paper.html

[4] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2172–2180. [Online]. Available: http://papers.nips.cc/paper/6399-infogan-interpretable-representation

[5] A. Odena, C. Olah, C. Olah, J. B. Shlens, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2642–2651. [Online]. Available: https://dl.acm.org/doi/10.5555/3305890.3305954.
[6] L. Zhang, Y. Ji, X. Lin, and C. Liu, "Style transfer for anime sketches with enhanced residual U-net and auxiliary classifier GAN," in *Proc. 4th IAPR Asian Conf. Pattern Recognit. (ACPR)*, Nanjing, China, Nov. 2017, pp. 506–511. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8575875
[7] X. Xia, R. Togneri, F. Sohel, and D. Huang, "Auxiliary classifier generative adversarial network with soft labels in imbalanced acoustic event detection," *IEEE Trans. Multimedia*, vol. 21, no. 6, pp. 1359–1371, Jun. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8523637
[8] X. Chen, C. Xu, X. Yang, L. Song, and D. Tao, "Gated-GAN: Adversarial gated networks for multi-collection style transfer," *IEEE Trans. Image Process.*, vol. 28, no. 2, pp. 546–560, Feb. 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8463508
[9] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification," *Neurocomputing*, vol. 321, pp. 321–331, Dec. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0925231218310749
[10] Z. He, W. Zuo, M. Kan, S. Shan, and X. Chen, "AttGAN: Facial attribute editing by only changing what you want," *IEEE Trans. Image Process.*, vol. 28, no. 11, pp. 5464–5478, Nov. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8718508
[11] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8789–8797. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Choi_StarGAN_Unified_Generative_CVPR_2018_paper.html
[12] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2794–2802. [Online]. Available: https://ieeexplore.ieee.org/document/8237566
[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5767–5777. [Online]. Available: http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans
[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, vol. 37. Jul. 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html
[15] Y. LeCun, C. Cortes, and C. J. C. Burges *The Mnist Database of Handwritten Digits*. Accessed: Dec. 2, 2020. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[16] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015, *arXiv:1511.06434*. [Online]. Available: http://arxiv.org/abs/1511.06434
[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[18] *Tensorflow 2*. Accessed: Dec. 2, 2020. [Online]. Available: http://www.tensorflow.org
[19] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 6626–6637. [Online]. Available: http://papers.nips.cc/paper/7240-gans-trained-by-a-two-t
[20] L. Mario, K. Karol, M. Marcin, G. Sylvain, and B. Olivier, "Are GANs created equal? A large-scale study," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 700–709. [Online]. Available: https://papers.nips.cc/paper/7350-are-gans-created-equal-a-large-scale-study
[21] L. Thomas, T. Corentin, O. Yann, and V. Jakob, "Mixed batches and symmetric discriminators for GAN training," in *Proc. Mach. Learn. Res. (PMLR)*, 2018, pp. 2844–2853. [Online]. Available: http://proceedings.mlr.press/v80/lucas18a.html
[22] N. Sebastian, C. Botond, and T. Ryota, "F-GAN: Training generative neural samplers using variational divergence minimization," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 271–279. [Online]. Available: https://papers.nips.cc/paper/6066-f-gan-training-generative-neural-samplers-using-variational-divergence-minimization
[23] D. Berthelot, T. Schumm, and L. Metz, "BEGAN: Boundary equilibrium generative adversarial networks," 2017, *arXiv:1703.10717*. [Online]. Available: http://arxiv.org/abs/1703.10717
[24] N. Kodali, J. Abernethy, J. Hays, and Z. Kira, "On convergence and stability of GANs," 2017, *arXiv:1705.07215*. [Online]. Available: http://arxiv.org/abs/1705.07215
[25] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. Mach. Learn. Res. (PMLR)*, vol. 70, 2017, pp. 214–223. [Online]. Available: http://proceedings.mlr.press/v70/arjovsky17a.html
[26] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 3730–3738. [Online]. Available: http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

JEONGIK CHO received the B.S. degree in computer science and engineering from Konkuk University, Seoul, South Korea, in 2020. He is currently pursuing the degree with Concordia University, QC, Canada.

KYOUNGRO YOON (Senior Member, IEEE) received the B.S. degree in computer and electronic engineering from Yonsei University, Seoul, South Korea, in 1987, the M.S.E. degree in electrical engineering/systems from the University of Michigan, Ann Arbor, in 1989, and the Ph.D. degree in computer and information science from Syracuse University in 1999. He was a Principal Researcher and a Group Leader with the Mobile Multimedia Research Laboratory, LG Electronics Institute of Technology, from 1999 to 2003. He joined the School of Computer Science and Engineering, Konkuk University, Seoul, in 2003, as an Assistant Professor and became a Full Professor in 2012. He has been with the Department of Smart ICT Convergence, since 2017. His main research interests include smart media systems, image processing, and multimedia information and metadata processing. He has also served as the Co-Chair of the Ad Hoc Group on User Preferences and the Chair of the Ad Hoc Group on MPEG Query Format and Ad Hoc Group on MPEG-V of ISO/IEC JTC1 SC29 WG11 (MPEG). He also served as the Chair of the Metadata Subgroup and JPSearch Ad Hoc Group of ISO/IEC JTC1 SC29 WG1 (i.e., JPEG). He currently serves as the Chair of IEEE-SA 2888 WG. He is an Editor of various international standards, such as ISO IS 15938-12, 23005-1, 23005-2, 23005-5, 23005-6, 23093-1, 24800-3, 24800-5, and 24800-6.