Generative AI Theory & Applications
Report 3: Generative Adversarial Networks

Syed Abraham Ahmed

March 17, 2025

Contents

1 DCGAN vs. WGAN-GP

1.1 DCGAN vs. WGAN-GP on the Same Dataset

To show the effectiveness of WGAN-GP in comparison to DCGAN, both models were trained on CelebA-like using identical architectures but different loss functions (binary cross-entropy for DCGAN and Wasserstein loss with gradient penalty for WGAN-GP).

Both models were trained on $64 \times 64 \times 3$ images with a 128-dimensional latent space, using a batch size of 16 for 30 epochs. The only difference between GAN architectures were the loss functions: DCGAN used binary cross-entropy, while WGAN-GP used Wasserstein loss with gradient penalty. [2]

1.2 Training Performance

Figure 1 shows the generator and discriminator losses. DCGAN shows higher generator loss which is typical for cross-entropy, while WGAN-GP shows negative generator loss due to the Wasserstein distance. Overall, WGAN-GP was more stable and suffered fewer training collapses. The gradient penalty in WGAN-GP helps well-behaved gradients even when the discriminator is active. This reduces the risk of mode collapse and provides the generator with more stable, continuous feedback throughout training.

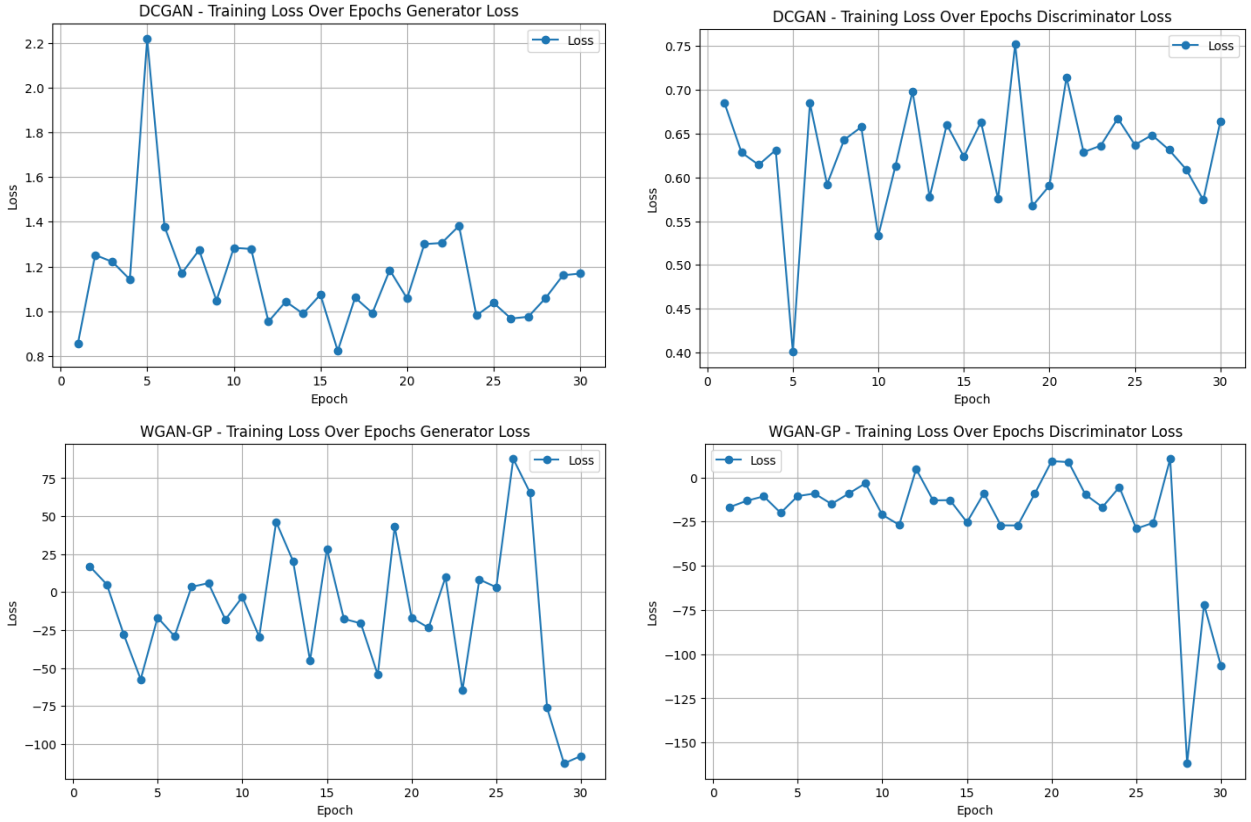

Figure 1: Training loss curves

1.3 Sample Outputs

In Figure 2, we show one sample from each model at different epochs which are most visually appealing.

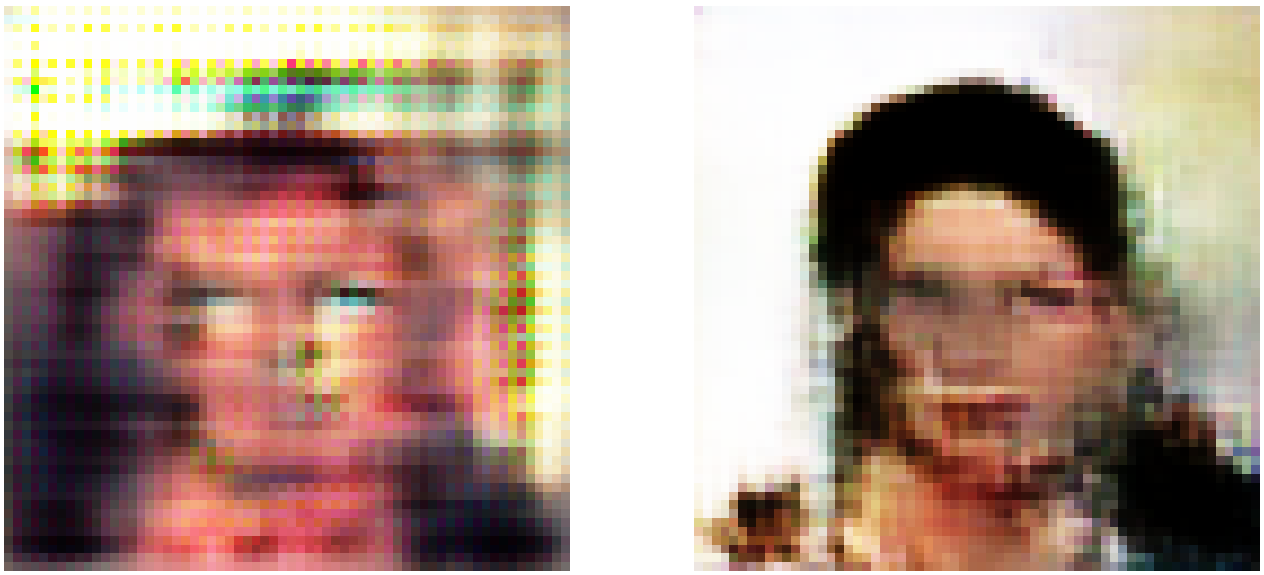

Figure 2: DCGAN (left) vs. WGAN-GP (right) Generated Samples

With the same architectures and hyperparameters, WGAN-GP demonstrated more stable training and slightly higher quality generated images than DCGAN, supporting the claim that it can be a significant improvement when applied to the same dataset. By visual comparison, WGAN-GP samples typically contain fewer artifacts and sharper details. Apologies for the nightmare fuel.

2 Conditional GANs

2.1 High-Level Overview of the Conditional GAN

Conditional GANs build upon the Vanilla GAN framework by changing both the generator and discriminator inputs with a "class". Specifically:

1. **Generator Input:** A random noise vector $\mathbf{z}$ is concatenated with the one hot label, producing a combined input that the generator reshapes and de-convolves into an image.

2. **Discriminator Input:** The real or fake image is concatenated along the channel dimension with the same one-hot label. This combined tensor is fed into the discriminator, which outputs a single logit, telling if it's real or fake.

2.2 Loss Function and Attribute Handling

The loss is the same binary cross-entropy used in standard GANs. The discriminator is trained to classify real images vs. generated images while the generator is trained to trick the discriminator into predicting incorrectly for the fake images. However, because both models see the class attributes, the generated images are conditioned on the chosen label, allowing class-specific image "influence".

2.3 Flavour of Conditional GAN

This approach follows the CGAN paradigm shown by Mirza & Osindero [1], in which the conditioning is applied by adding label information to both the generators' noise input and the discriminators' image input. The class labels are essentially conditional inputs without a classification loss defined.

3 GycleGAN

3.1 CycleGAN Overview

CycleGAN aims to translate images from one domain to another without paired training data. It uses two generators (G: $X \rightarrow Y$,and F: $Y \rightarrow X$) and two discriminators (D_x, D_y) to ensure the generated images look realistic in their respective domains. [3]

3.2 Loss Function & Component Analysis

1. **Adversarial (GAN) Loss:** Each generator tries to produce images that trick its domain's discriminator.
2. **Cycle Consistency Loss:** If we transform a image (x) into implied ($\hat{y}$) and then transform it back, we want to recover the original image (x). The same applies in the opposite direction ($y \rightarrow x \rightarrow y$). This cycle constraint ensures that the model truly learns to capture the core features of each domain, rather than just matching blindly.
3. **Identity Loss:** When a generator receives an image already from its own target domain, it's encouraged to act as an identity function so that the content is maintained. This helps the model avoid unnecessary changes when an image is already in the correct domain.

The final generator loss is a combination of these three terms (adversarial, cycle consistency, and identity), weighted by coefficients depending on the architecture. The discriminators each use a standard real/fake classification objective, rewarding correct predictions for real images and penalizing fake images. [3]

3.3 Training Process

1. **Forward Pass:** We feed (x) to generator G to get fake samples, then pass those to generator F to check if we recover the original data. Likewise, we transform real samples (y) with F and then try to reconstruct (y) with G.
2. **Discriminator Checks:** Each discriminator receives both real images and fake images created by the generator.
3. **Loss Evaluation:** Evaluate all 3 loss terms on the generator, and evaluate real/fake classification loss on the discriminator.
4. **Backpropagation and Updates:** We update the generators' parameters using the combined generator loss, then update each discriminator's parameters using its real/fake classification loss. This update continues for all epochs, improving both sides of the model.

CycleGAN generally learns to do image translation by using 2 adversarial objectives (one for each domain), while cycle consistency ensures it preserves the original content, and identity constraints stop it from modifying images already in the correct domain. [3]

3.4 Training Performance

Over training which took approximately 17 minutes for one epoch, both the generator and discriminator losses were measured. By the end of training, the discriminator losses were around 0.15 (D_x) and 0.12 (D_y), while generator F and G losses were approximately 3.62 and 4.15, respectively.

These losses combine the adversarial term, the cycle consistency term, and the identity term. In practice, the cycle loss often dominates early in training because the model initially struggles to map one domain to the other and back again. [3]

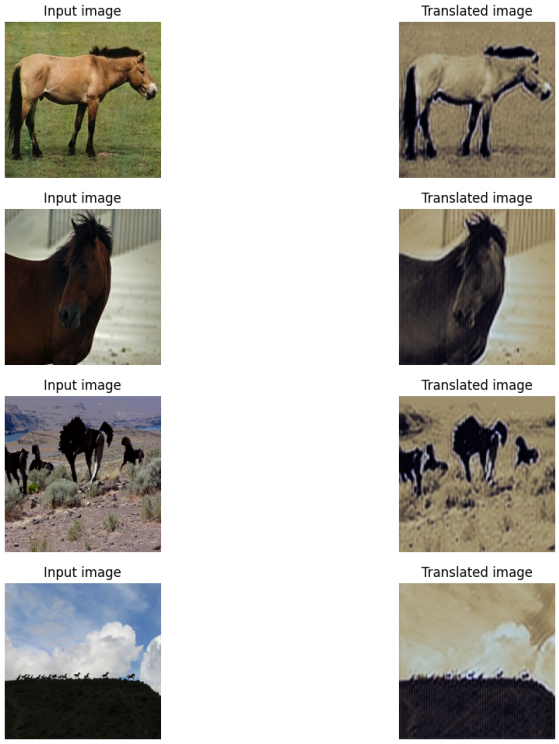

Figure 3: Translated Images Showing the Outputs of F and G

4 InfoGAN Discussion

InfoGAN extends a Vanilla GAN by splitting the latent space into standard "noise" and an additional code vector. This code is designed to capture distinct factors of variation in the data (e.g., digit shape or rotation). The generator takes both noise and the code to produce images, while a separate network attempts to recover the code from those generated images. By maximizing the mutual information between the code and output, InfoGAN tries to ensure that each code dimension corresponds to a specific property of the generated samples. [4]

5 References

[1] M. Mirza, S. Osindero, Conditional Generative Adversarial Nets, University of Montreal, Yahoo, 2014.DIRECT LINK

[2] F.Chollet, A.K.Nain, DCGAN/WGAN-GP/CycleGAN, Keras, 2024.WGAN-GP DCGAN CycleGAN

[3] J. Zhu, T. Park, P. Isola, A. Efros, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, Berkeley AI Research Laboratory, UC Berkely, 2020. DIRECT LINK

[4] S. Prince, Understanding Deep Learning, November 21, 2024. DIRECT LINK