Handwriting Generation Using Conditional GANs

Vansh Shah (SC22B034)
Saurabh Kumar (SC22B146)

May 5, 2025

Abstract

This report presents the design and implementation of a handwriting generation system using Conditional Generative Adversarial Networks (cGANs). By conditioning on character labels, the system generates grayscale handwritten letter images that mimic real handwriting styles. The approach leverages convolutional neural networks (CNNs) within both the generator and discriminator architectures. We detail the mathematical formulations, training objectives, and key architectural choices involved in building the model.

1 Introduction

The generation of realistic handwritten images has wide applications in fields like digital handwriting synthesis, personalization, and document generation. Traditional GANs generate data purely from random noise, but conditional GANs (cGANs) introduce label conditioning, enabling controlled generation tied to specific character classes.

This project focuses on generating handwritten characters conditioned on their labels using a cGAN framework. The dataset used consists of real handwritten letter images, providing the ground truth for supervised adversarial training.

2 Methodology

2.1 Conditional GAN Overview

While standard GANs learn a mapping $G(z)$ from noise to data, cGANs extend this to $G(z, y)$, where y is a label (e.g., character class). Both the generator and discriminator receive the label y as input, ensuring the generated images are guided by the desired conditioning.

2.2 Generator Architecture

The generator takes two inputs:

- A random noise vector $z \sim p_z(z)$.
- A label y embedded into a vector space.

The embedded label and noise vector are concatenated and passed through multiple `ConvTranspose2d` (deconvolution) layers to upsample the signal into a 64×64 grayscale image, producing a synthetic image $G(z, y)$.

Mathematically:

$$\text{input} = [z; E(y)], \quad G(z, y) = \text{ConvTranspose2d layers(input)},$$

where $E(y) \in R^d$ is the label embedding.

2.3 Discriminator Architecture

The discriminator receives:

- An image $x \in R^{1\times64\times64}$ (real or fake).
- A label y embedded and spatially expanded to a 64×64 map.

These are concatenated along the channel dimension, forming a 2-channel input. A CNN extracts features, which are flattened and passed through fully connected layers, culminating in a scalar probability after a sigmoid activation:

$$D(x, y) = \sigma(f(x, y)),$$

where $f(x, y)$ is the feature mapping.

2.4 Key CNN Operations

The system employs:

- `Conv2d`: For feature extraction.
- `ConvTranspose2d`: For upsampling in the generator.
- Batch Normalization (`BatchNorm2d`): To stabilize training.
- Activation functions (ReLU, LeakyReLU): For non-linearity.
- Fully Connected Layers: To map extracted features to final outputs.
- Sigmoid: To bound discriminator outputs between 0 and 1.

3 Training Objectives

3.1 Loss Functions

The discriminator loss:

$$L_D = -E_{x,y}[\log D(x, y)] - E_{z,y}[\log(1 - D(G(z, y), y))],$$

where the first term encourages correct real classifications, and the second penalizes incorrect fake classifications.

The generator loss:

$$L_G = -E_{z,y}[\log D(G(z, y), y)],$$

which encourages the generator to fool the discriminator.

3.2 Why Logarithmic Loss?

Logarithmic loss penalizes confident wrong predictions more heavily, steeply increasing when the predicted probability is near 1 but incorrect. This promotes more stable gradients and robust GAN training.

4 Final Training Algorithm

The training loop includes:

1. Sample a batch of real data (x, y).
2. Sample a batch of noise vectors z and labels y.
3. Generate fake images $G(z, y)$.
4. Compute discriminator loss on real and fake images, update discriminator.
5. Compute generator loss to fool discriminator, update generator.

5 Results and Discussion

The trained model successfully generates diverse and realistic handwritten character images conditioned on the specified labels. Example outputs and training loss graphs (if available) would be included here.
Following are the links related to our model :

- **Code link (Google Colab) :**
  `https://colab.research.google.com/drive/1M80m8lICa_LBAEhbHHOwOKlsblR8tdAY?usp=sharing`
- **Deployed model (Platform: Render) :**
  `https://dashboard.render.com/web/srv-d08leo3uibrs73b4ug2g`
- **Dataset (Google drive):**
  `https://drive.google.com/drive/folders/1c0jFxvA90aGB73MSFMu2KvFAR9_S2Vdl?usp=sharing`
- **Website repository (GitHub):**
  `https://github.com/saurabhkr132/handwriting-styles`
- **Live website (Deployed on Vercel):**
  `https://handwriting-styles.vercel.app/`

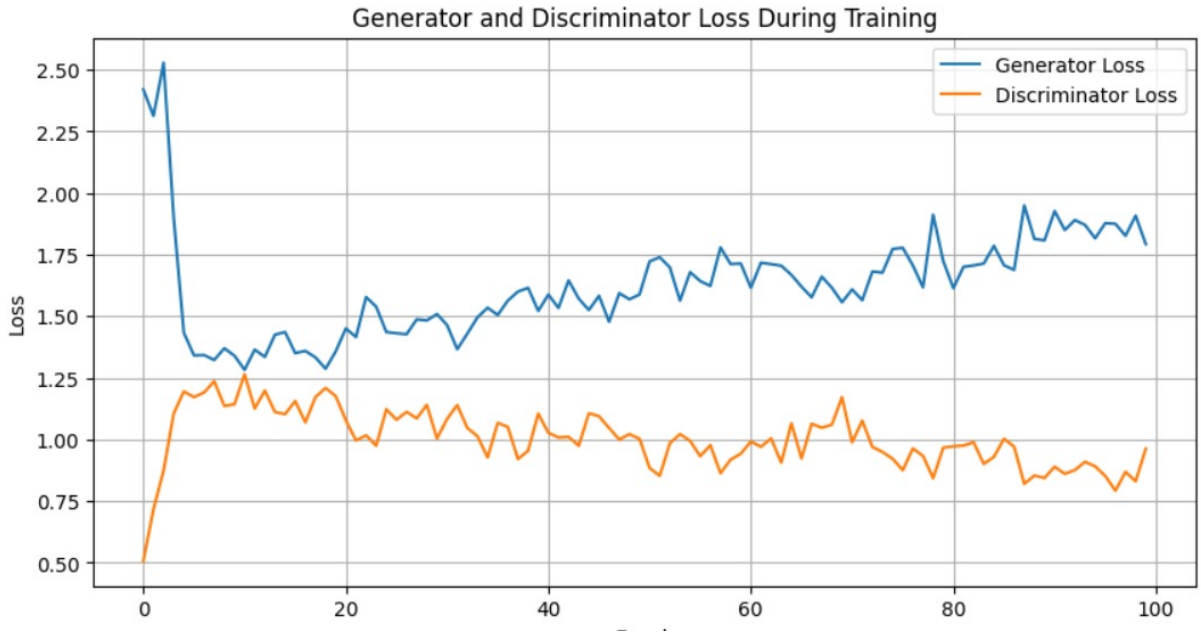

Figure 1: **Generator and Discriminator loss during training**

Figure 2: **Image generated by CGAN Model**

6 Conclusion

We implemented a cGAN-based handwriting generation system that produces realistic handwritten images conditioned on character labels. The combination of embedded label conditioning, CNN architectures, and adversarial training enables the model to learn fine-grained handwriting styles effectively.

References

- Goodfellow, I., et al. (2014). Generative Adversarial Nets.
- Mirza, M., & Osindero, S. (2014). Conditional Generative Adversarial Nets.
- Youtube , ChatGpt .