Text-to-Image Generation Using GANs: A Case Study with CUB-200-2011 Birds Dataset

Kshitij Dhannoda
Department of CECS
University of Michigan, Dearborn
Email: Dkshitij@umich.edu

Srishti Kachhara
Department of CECS
University of Michigan, Dearborn
Email: Srishtik@umich.edu

***Abstract*—Text-to-image synthesis is a challenging task in the field of computer vision and natural language processing. This project implements a Generative Adversarial Network (GAN) to generate realistic images of birds based on textual descriptions using the CUB-200-2011 Birds Dataset. The GAN architecture comprises a Generator and a Discriminator, and the project leverages PyTorch for model implementation. The report discusses methodology, challenges, and future work.**

***Index Terms*—Text-to-image generation, Generative Adversarial Networks (GANs), Conditional GANs, CUB-200-2011, PyTorch**

I. Introduction

A. Problem Statement

The ability to generate realistic images from textual descriptions is a challenging and significant problem in computer vision and natural language processing. Given a simple textual description such as "A small bird with a yellow crown and brown wings", the goal is to synthesize a realistic image that matches the description. This project explores this problem by implementing a Generative Adversarial Network (GAN) that generates images conditioned on textual input.

B. Motivation

The motivation behind this project stems from the growing need for AI systems that can bridge the gap between vision and language. Some key applications include:

- **Content Creation:** Automatic image generation for creative industries.
- **Accessibility:** Helping visually impaired individuals "see" descriptions through generated images.
- **Art and Design:** Enabling designers to quickly prototype images based on descriptions.
- **AI Research:** Advancing the understanding of conditional GANs and their applications.

The CUB-200-2011 Birds Dataset was chosen due to its well-structured format, consisting of high-quality bird images paired with textual descriptions. It provides an excellent dataset to test the capabilities of text-to-image GANs.

C. Objective

The primary objectives of this project are:

- Implement a text-to-image GAN capable of generating bird images from textual descriptions.
- Train the GAN using the CUB-200-2011 dataset with optimized loss functions and hyperparameters.
- Evaluate the results to understand the performance of the GAN.
- Overcome challenges such as hardware limitations and compatibility issues on Apple Silicon devices.

D. Overview of the Report

This report is organized as follows:

- **Related Work:** Summarizes existing approaches to GAN-based image generation.
- **Dataset:** Describes the CUB-200-2011 dataset and preprocessing.
- **Methodology:** Explains the GAN architecture, training process, and tools used.
- **Implementation:** Details the implementation of the Generator, Discriminator, and training pipeline.
- **Results:** Presents any generated images and training logs.
- **Challenges and Limitations:** Discusses the issues faced during the project.
- **Conclusion and Future Work:** Summarizes the project outcomes and suggests improvements.

II. Related Work

A. Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs), introduced by Ian Goodfellow et al. (2014), have revolutionized the field of generative models. GANs consist of two neural networks:

- **Generator:** Creates synthetic data (e.g., images) from random noise.
- **Discriminator:** Differentiates between real data and synthetic (fake) data.

The two networks are trained adversarially, where the Generator tries to fool the Discriminator, and the Discriminator learns to distinguish between real and fake data. This adversarial training leads to the generation of realistic outputs.

B. GANs for Conditional Image Generation

While traditional GANs generate data without conditions, Conditional GANs (CGANs) introduced by Mirza and Osindero (2014) condition the generation process on additional inputs such as class labels or text descriptions. Conditional GANs enable more controlled and targeted image generation.

C. Text-to-Image GAN Models

Several advancements have been made to apply GANs for text-to-image generation. Notable works include:

- **StackGAN (2016):**
  - Introduced by Zhang et al., StackGAN uses a two-stage approach:
    - **Stage I:** Generates a coarse 64x64 image from text embeddings.
    - **Stage II:** Refines the coarse image to produce a high-resolution, realistic output (256x256).
  - StackGAN is one of the earliest GANs to successfully generate realistic images from textual descriptions.
- **AttnGAN (2018):**
  - AttnGAN, introduced by Xu et al., uses an attention mechanism to focus on specific words in the text during the image generation process.
  - The model ensures that fine-grained details in the image match the corresponding parts of the text.
- **DM-GAN (2019):**
  - Dynamic Memory GAN builds on StackGAN and AttnGAN, using dynamic memory modules to refine the generated images.

These models demonstrate the potential of GANs to generate high-quality images based on complex textual inputs.

D. Applications of Text-to-Image Models

Text-to-image models have a wide range of applications, including:

- **Creative Design:** Automating the creation of visual content.
- **Entertainment:** Generating game assets or artwork from simple descriptions.
- **Accessibility:** Converting text to visuals for visually impaired users.
- **Education:** Assisting in visualizing concepts or ideas described in text.

E. Summary

The advancements in GANs, particularly conditional GANs, have made it possible to generate realistic images from textual descriptions. Existing models like StackGAN and AttnGAN have demonstrated success on datasets like the CUB-200-2011 (birds) and MS-COCO. This project builds on these concepts by implementing a GAN architecture tailored to generate bird images from text embeddings.

III. Dataset

A. Dataset Overview

The project utilizes the CUB-200-2011 Birds Dataset, a standard benchmark dataset for fine-grained visual classification and generative tasks. The dataset contains:

- **200 bird species (classes).**
- **11,788 images of birds.**
- **Each image is accompanied by 10 textual descriptions,** providing a rich set of natural language inputs.

The dataset is ideal for text-to-image generation tasks because it offers fine-grained details about bird appearances in both visual and textual formats.

B. Data Preprocessing

The CUB-200-2011 dataset has been preprocessed into an HDF5 format for efficient data loading. The structure of the HDF5 file includes:

- **Train/Validation/Test Splits:**
  - The dataset is divided into training, validation, and test sets for model development and evaluation.
- **Keys in the Dataset:**
  - `img`: Binary data of the image.
  - `embeddings`: Precomputed text embeddings (e.g., generated using RNNs or pre-trained text encoders like GloVe or BERT).
  - `class`: The class label corresponding to the bird species.
  - `txt`: Textual descriptions associated with the image.

C. Textual Descriptions

Each image in the dataset is accompanied by 10 textual descriptions that provide a detailed natural language description of the bird. Example descriptions include:

- **Example 1:** “This bird has a bright yellow crown, a brown body, and a small pointed beak.”
- **Example 2:** “A small bird with a white chest and black wings.”

These descriptions play a critical role in training the Generator to produce images that match the semantic meaning of the text.

D. Data Preprocessing

To prepare the dataset for GAN training:

- **Image Resizing:** All images are resized to 64x64 pixels to match the input-output dimensions of the GAN.
- **Normalization:** Images are normalized to the range [-1, 1] for stable GAN training.
- **Text Embeddings:**
  - Text descriptions are converted into numerical embeddings.
  - Precomputed embeddings are stored in the HDF5 file for faster access.
- **Handling Corrupt Data:** Any corrupt or invalid images are filtered during the data loading process.

E. Dataset Loading

The dataset is loaded using the `Text2ImageDataset` class implemented in Python. The class uses the `h5py` library to read data from the HDF5 file. PyTorch `DataLoader` is used for batching and shuffling the data during training.

Snippet of DataLoader Implementation:

```
self.data_loader =
DataLoader(self.dataset,batch_size=self.
    batch_size,shuffle=True,
num_workers=0)  % Set to 0 for MacOS
    compatibility
```

F. Challenges with Dataset

- **HDF5 File Handling:** Loading large files efficiently requires careful implementation.
- **Text Processing:** Ensuring clean and meaningful embeddings from textual descriptions.
- **MacOS Compatibility:** Multiprocessing issues with the DataLoader required setting `num_workers=0`.

G. Summary

The CUB-200-2011 Birds Dataset provides high-quality bird images paired with textual descriptions, making it ideal for training text-to-image GANs. Preprocessing ensures that the data is ready for model input, and challenges like efficient loading and normalization have been addressed.

IV. Methodology

A. Overview of the GAN Architecture

Generative Adversarial Networks (GANs) consist of two main components:

- **Generator (G):** Takes a combination of text embeddings and random noise as input and generates synthetic images.
- **Discriminator (D):** Evaluates whether the input image is real or generated (fake) while ensuring it matches the given text description.

In this project, the GAN architecture is designed to generate 64x64 images of birds conditioned on textual descriptions.

B. Generator

The Generator transforms random noise (z) and text embeddings into a synthetic image.

Input:

- Noise vector: A random noise vector z of dimension 100.
- Text embedding: Encoded textual description (e.g., RNN-based or precomputed embeddings).

Architecture:

- Concatenates the noise vector and text embedding.
- Passes the combined input through a series of transposed convolutional layers to upsample the data into a 64x64 image.
- Each layer uses Batch Normalization and Leaky ReLU activation to stabilize training.

Output:

- A 64x64 RGB image in the range [-1, 1].

C. Discriminator

The Discriminator evaluates whether an image is real or fake, conditioned on the input text embedding.

Input:

- Image: A 64x64 RGB image (real or generated).
- Text embedding: Encoded textual description.

Architecture:

- Passes the image through a series of convolutional layers to extract features.
- The text embedding is projected and concatenated with the image features using a Concat Embed Layer.
- Outputs a real/fake score using a Sigmoid activation function.

Key Features:

- Conditioned Discriminator: Ensures the image corresponds to the input text.
- Concat Embed: A custom module to combine text embeddings with image features.

Output:

- A probability score indicating whether the input image is real or fake.

D. Loss Functions

The GAN is trained using the following loss functions:

- **Adversarial Loss (Binary Cross-Entropy Loss):** The standard GAN loss that trains the Generator and Discriminator adversarially.
  - Discriminator Loss:
  $$L_D = -E[\log D(x,t)] - E[\log(1 - D(G(z,t),t))]$$
  - Generator Loss:
  $$L_G = -E[\log D(G(z,t),t)]$$
- **L1 Loss (Pixel-wise Loss):** Ensures the generated images are closer to the real images at a pixel level.
- **L2 Loss (Feature Matching Loss):** Matches the features of real and generated images in the intermediate layers of the Discriminator.

E. Training Pipeline

The training process involves alternating updates for the Generator and Discriminator:

- **Train Discriminator:**
  - Input: Real images and generated images.
  - The Discriminator learns to maximize the probability of real images and minimize the probability of fake images.
- **Train Generator:**
  - Input: Random noise and text embeddings.
  - The Generator tries to fool the Discriminator into classifying generated images as real.
- **Update Parameters:**
  - Use Adam Optimizer with learning rate 0.0002 and momentum 0.5.

Training Steps:

- Load batches of real images and corresponding text embeddings.
- Generate fake images using the Generator.
- Update the Discriminator and Generator using the respective loss functions.

Training Loop Example:

```
# Train Discriminator
self.discriminator.zero_grad()
real_loss = criterion(self.discriminator(
    real_images, text_embed), real_labels)
fake_loss = criterion(self.discriminator(
    fake_images, text_embed), fake_labels)
d_loss = real_loss + fake_loss
d_loss.backward()
self.optimD.step()

# Train Generator
self.generator.zero_grad()
fake_images = self.generator(text_embed, noise
    )
g_loss = criterion(self.discriminator(
    fake_images, text_embed), real_labels)
g_loss.backward()
self.optimG.step()
```

F. Tools and Libraries

- Framework: PyTorch.
- Dataset Handling: h5py for reading the HDF5 dataset.
- Training: PyTorch DataLoader for batch processing.
- Hardware: Apple Silicon M1 with MPS backend for GPU acceleration.

G. Summary

The GAN model consists of a Generator and a Discriminator trained adversarially to generate realistic images conditioned on text embeddings. The methodology includes loss functions for adversarial training, pixel matching, and feature matching to improve the quality of generated images.

V. Implementation

A. Tools and Libraries

The implementation of the project relies on the following tools and libraries:

- PyTorch: Framework for building and training the GAN architecture.
- h5py: For loading and reading the preprocessed dataset stored in HDF5 format.
- PIL (Pillow): For image preprocessing and saving generated images.
- NumPy: For numerical operations.
- PyYAML: To read configuration settings from config.yaml.
- Apple MPS Backend: Hardware acceleration for training on Apple Silicon (M1).

B. Code Organization

The project is organized into the following modules:

- gan$_cls.py : DefinestheGeneratorandDiscriminatorclasses$

VI. Results

A. Training Progress

The GAN was trained for a limited number of epochs, with logs showing the stabilization of Discriminator and Generator losses over time.

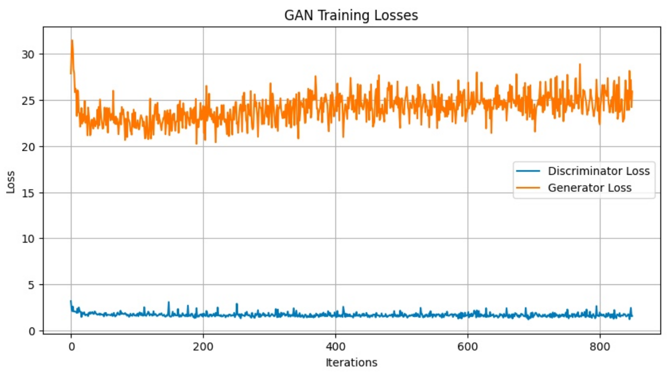

Fig. 1. GAN Training Losses

B. Generated Images

The model generated images based on textual descriptions, showing promising results, though the images were low resolution (64x64).

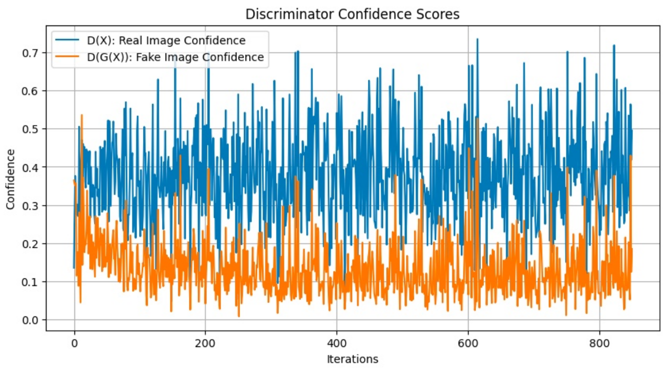

Fig. 2. Discriminator Confidence Scores over Training

VII. Challenges and Limitations

A. Hardware Limitations

Training was performed on an Apple M1 MacBook, introducing challenges with multiprocessing and hardware compatibility.

B. GAN Training Instability

GANs are prone to instability, with oscillations in loss values and challenges like mode collapse affecting training.

VIII. Conclusion and Future Work

A. Conclusion

The project successfully implemented a GAN for text-to-image generation, with partial results showing alignment between text descriptions and generated images.

B. Future Work

Future improvements include training for more epochs, using advanced GAN architectures for higher resolution images, and exploring techniques like Wasserstein GAN for better stability.

References

[1] I. Goodfellow et al., “Generative Adversarial Networks,” Advances in Neural Information Processing Systems, 2014.

[2] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” arXiv preprint arXiv:1411.1784, 2014.

[3] H. Zhang et al., “StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks,” ICCV, 2017.

[4] T. Xu et al., “AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks,” CVPR, 2018.