ECE 5831
Pattern Recognition And Neural Networks

Name:
Kshitij Dhannoda
Srishti Kachhara

Problem Statement:

Generate realistic images of birds from textual descriptions using a Generative Adversarial Network (GAN).

What is Text-to-Image Generation?

- Bridging the gap between visual and natural language data.
- Example: *"A small yellow bird with black wings"* → *Realistic Image*.

Motivation:

- **Applications**: Content generation, AI art, and accessibility tools.
- **Challenges**: Understanding and integrating text and image data.

Dataset

Dataset Used: **CUB-200-2011 Birds Dataset**

- **Total Images**: 11,788
- **Classes**: 200 bird species
- **Text Descriptions**: 10 descriptions per image

Preprocessing:

- Images resized to **64x64 pixels**
- Text descriptions converted to **text embeddings**
- Stored efficiently in **HDF5 format** for fast loading

GAN Training Pipeline

GAN Architecture:

1. **Generator**:
   - Inputs: Text embeddings + Random noise
   - Output: 64x64 realistic bird images
2. **Discriminator**:
   - Inputs: Image + Text embedding
   - Output: Real or Fake

- **Loss Functions**:
  1. **Adversarial Loss**: Binary Cross-Entropy Loss for training GANs.
  2. **L1 Loss** and **L2 Loss**
- **Training Pipeline**:
  1. Train the **Discriminator** to classify real and fake images.
  2. Train the **Generator** to produce images that fool the Discriminator.

GAN Training Pipeline

Training Steps

1. **Load Data**: Images and text embeddings from the dataset.
2. **Train Discriminator**:
   - Real image + text → Real label
   - Generated image + text → Fake label
3. **Train Generator**:
   - Input: Noise + Text embeddings
   - Output: Fake image to fool the Discriminator.
4. **Update Parameters**: Use **Adam Optimizer** to update both networks.

Tools and Frameworks:

- **PyTorch**: Model training and optimization
- **DataLoader**: Efficient batching of data
- **Loss Functions**: Adversarial Loss, L1, and L2

Implementation

Tools and Libraries:

- **Framework**: PyTorch
- **Data Handling**: h5py for HDF5 files, NumPy
- **Image Processing**: PIL (Pillow)
- **Text Embeddings**: Precomputed text embeddings for conditioning

Tools and Libraries:

Module	Description
txt2image_dataset.py	Handles data loading and preprocessing
gan_cls.py	Defines Generator and Discriminator
trainer.py	Implements the training pipeline
utils.py	Utility functions (e.g., save checkpoints)
runtime.py	Entry point for training and inference

Results

Training Progress:

- **Discriminator Loss**: Reduced over epochs (better at distinguishing real/fake).
- **Generator Loss**: Improved over time (producing better images).

Results

Observations:

a. Early images were noisy and lacked structure.
b. Over time, the GAN started learning basic features like **colors** and **shapes** of birds.

Challenges and Limitations

Hardware Constraints:

- MPS backend on MacBook Air M1 has limited support compared to CUDA.
- Training speed was slower due to hardware limitations.

Training Instability:

- GAN training is inherently unstable.
- Loss oscillations and mode collapse observed during early epochs.

Challenges and Limitations:

Low Image Resolution:

- Generated images are limited to **64x64 pixels**.
- Higher resolution requires more advanced architectures like StackGAN.

Dataset Variability:

- Text descriptions vary in length and detail, affecting model performance.

Limited Training Time:

- Due to time constraints, the model did not fully converge.

Future Work:

1. **Train for More Epochs**:
   a. Extended training will help the Generator produce sharper and more detailed images.
2. **2. Higher Resolution Outputs**:
   a. Implement advanced GAN architectures like:
      i. **StackGAN**: Two-stage refinement process for 256x256 images.
      ii. **AttnGAN**: Attention mechanisms to focus on specific text details.
3. **3. Improved Text Embeddings**:
   a. Use pre-trained models like **BERT** or **CLIP** for better text representations.

Future Work:

1. **Stabilize Training**:
    a. Techniques like **Wasserstein GAN (WGAN)** and **Gradient Penalty** to reduce loss oscillations and mode collapse.
2. **Quantitative Evaluation**:
    a. Include metrics like:
        i. **Inception Score (IS)**
        ii. **Fréchet Inception Distance (FID)**
3. **Hardware Upgrades**:
    a. Use **NVIDIA GPUs** or cloud-based platforms like **Google Colab** or **AWS** for faster training.

Conclusion:

- **Summary of Achievements**:
  - Successfully implemented a **GAN** for text-to-image generation.
  - Demonstrated the ability to generate images conditioned on text descriptions.
  - Set up a robust training pipeline using PyTorch and the CUB-200-2011 dataset.
- **Key Challenges**:
  - Hardware limitations and training instability.
  - Limited resolution (64x64) and time constraints.

Conclusion:

- **Future Directions**:
  - Train longer for convergence and improve resolution using advanced GANs.
  - Integrate pre-trained text embeddings and evaluation metrics.
- **Final Remarks**:
  - The project serves as a foundation for further exploration in **text-to-image synthesis**.

Thank You