Progress Report 3: GAGAN Implementation

Group 40 - Basil Ali Khan & Ahsan Siddiqui

April 20, 2025

Implementation Summary

Our project focuses on implementing a Generative Adversarial Network (GAN) with Genetic Algorithms (GAGAN) for image generation. The implementation is divided into the following components:

- **Generator and Discriminator Networks:** The generator and discriminator architectures have been implemented using PyTorch. The generator uses transposed convolutions to upsample latent vectors into images, while the discriminator uses standard convolutional layers to classify real and fake images. Both networks have been tested for basic functionality.
- **Genetic Algorithm for Discriminator Evolution:** A population of discriminators is maintained, and their fitness is evaluated based on their ability to distinguish real and fake images. Genetic operations such as selection, crossover, and mutation have been implemented.
- **Training Loop:** The training loop alternates between training the generator and evolving the discriminator population. Label smoothing and noise injection have been added to improve training stability.
- **Visualization and Model Saving:** Sample images are generated at regular intervals during training and saved for visualization. The generator and the best discriminator are saved periodically for checkpointing.

Omissions: We have not yet implemented advanced regularization techniques like gradient penalties or spectral normalization due to time constraints. These implementations are anyways out of scope of our project as our task was to focus on optimizing discriminator weights using Evolutionary Algorithms.

Correctness Testing

To verify the correctness of our implementation, we performed the following tests:

- **Unit Tests:** Verified that the generator and discriminator produce outputs of the correct dimensions.
- **Sample Inputs and Outputs:** Tested the generator with random noise inputs to confirm that it produces images with pixel values in the expected range.

- **Edge Cases:** Tested the genetic algorithm with extreme values for mutation rate and elite ratio to ensure stability.
- **Baseline Comparisons:** Compared the generator's outputs with those from a standard GAN implementation to ensure similar behavior during early training epochs.

Complexity & Runtime Analysis

- **Theoretical Complexity:** The generator and discriminator training steps have a complexity of $O(n)$, where n is the number of images in a batch. The genetic algorithm introduces additional overhead with a complexity of $O(p)$, where p is the population size.
- **Empirical Performance:** Training on a subset of the CelebA dataset (10% of the total dataset) with a batch size of 32 and a population size of 10 takes approximately 5 minutes per epoch on an NVIDIA RTX 3060 GPU.
- **Scaling Issues:** Increasing the population size significantly impacts runtime due to the linear scaling of fitness evaluations. Using the full CelebA dataset ($10\times$ more data) like in the research paper will increase the number of iterations per epoch by $10\times$. This directly translates to approximately $10\times$ longer training time per epoch and storage and data loading pipelines may become bottlenecks. Increasing batch size will also consume more GPU memory, potentially hitting hardware limits

Challenges & Solutions

- **Training Instability:** Training instability due to overly powerful discriminators was resolved by introducing label smoothing and noise injection.
- **High Computational Cost:** Limited the population size and reduced the frequency of genetic evolution steps to address computational overhead.
- **Hyperparameter Tuning:** Conducted grid search experiments to identify optimal values for mutation rate, elite ratio, and mutation strength.

Enhancements

- Added label smoothing and noise injection to improve training stability.
- Tested the implementation on a subset of the CelebA dataset to validate performance on real-world data.
- Added functionality to save and visualize generated images at regular intervals.

Training Progress

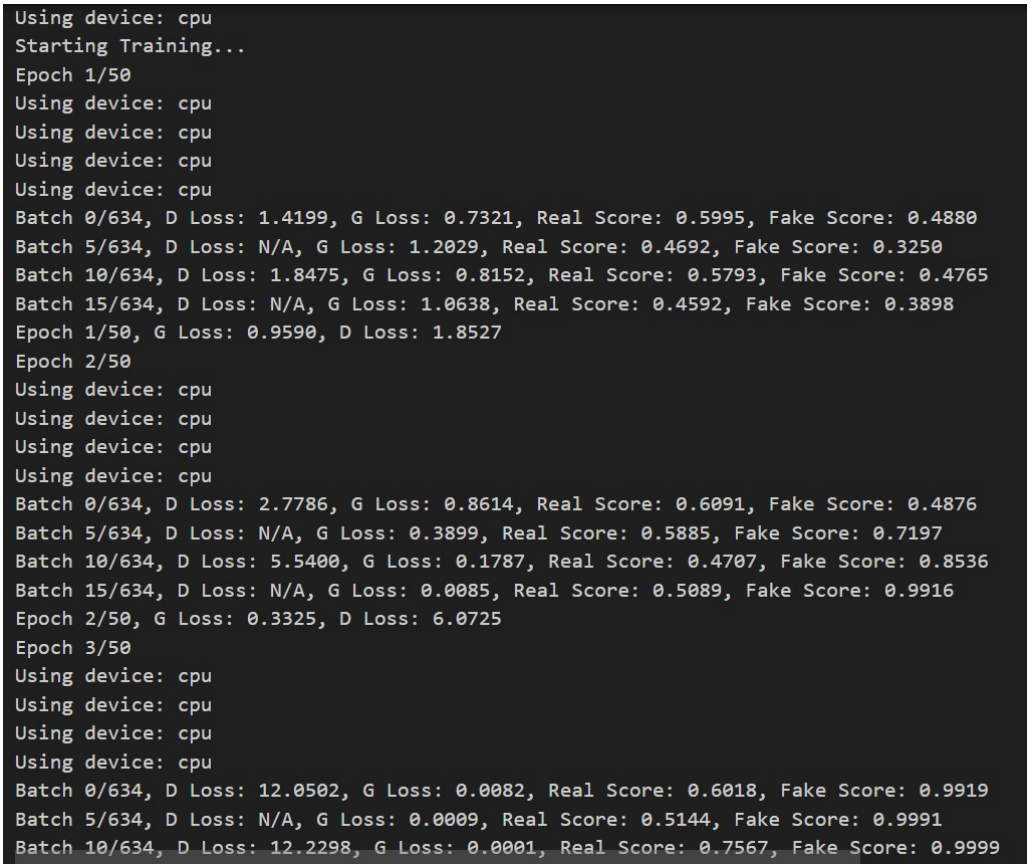

Figure 1: Epoch-wise training progress showing generator and discriminator loss curves.