MNIST GAN with FastAPI and HTML UI

A PROJECT REPORT ON A DEPLOYABLE GENERATIVE AI APPLICATION

Executive Summary

Objective: To build and deploy a Generative Adversarial Network (GAN) as a web service for generating handwritten digits.

Achievement: Successfully developed a full-stack application with a FastAPI backend and an HTML frontend that uses a trained Generator model to create new images. A separate Classifier model was integrated to predict the digit and confidence of the generated images.

Impact: Created a reproducible, end-to-end pipeline for deploying a generative model, demonstrating key MLOps principles for turning a model into a functional application.

Problem & Methodology

PROBLEM: MAKING A TRAINED GENERATIVE MODEL ACCESSIBLE TO END-USERS THROUGH A SIMPLE, INTERACTIVE WEB INTERFACE.

BACKEND: A FASTAPI SERVER WAS BUILT TO LOAD BOTH THE GENERATOR AND CLASSIFIER MODELS AND HANDLE IMAGE GENERATION REQUESTS. IT RETURNS A BASE64 ENCODED IMAGE AND THE TOP-3 PREDICTIONS.

FRONTEND: A MINIMALIST HTML/JAVASCRIPT UI ALLOWS USERS TO GENERATE NEW DIGITS AND SEE REAL-TIME PREDICTIONS AND CONFIDENCE SCORES.

DATA: THE STANDARD MNIST DATASET WAS USED FOR TRAINING BOTH THE GAN AND THE CLASSIFIER MODELS.

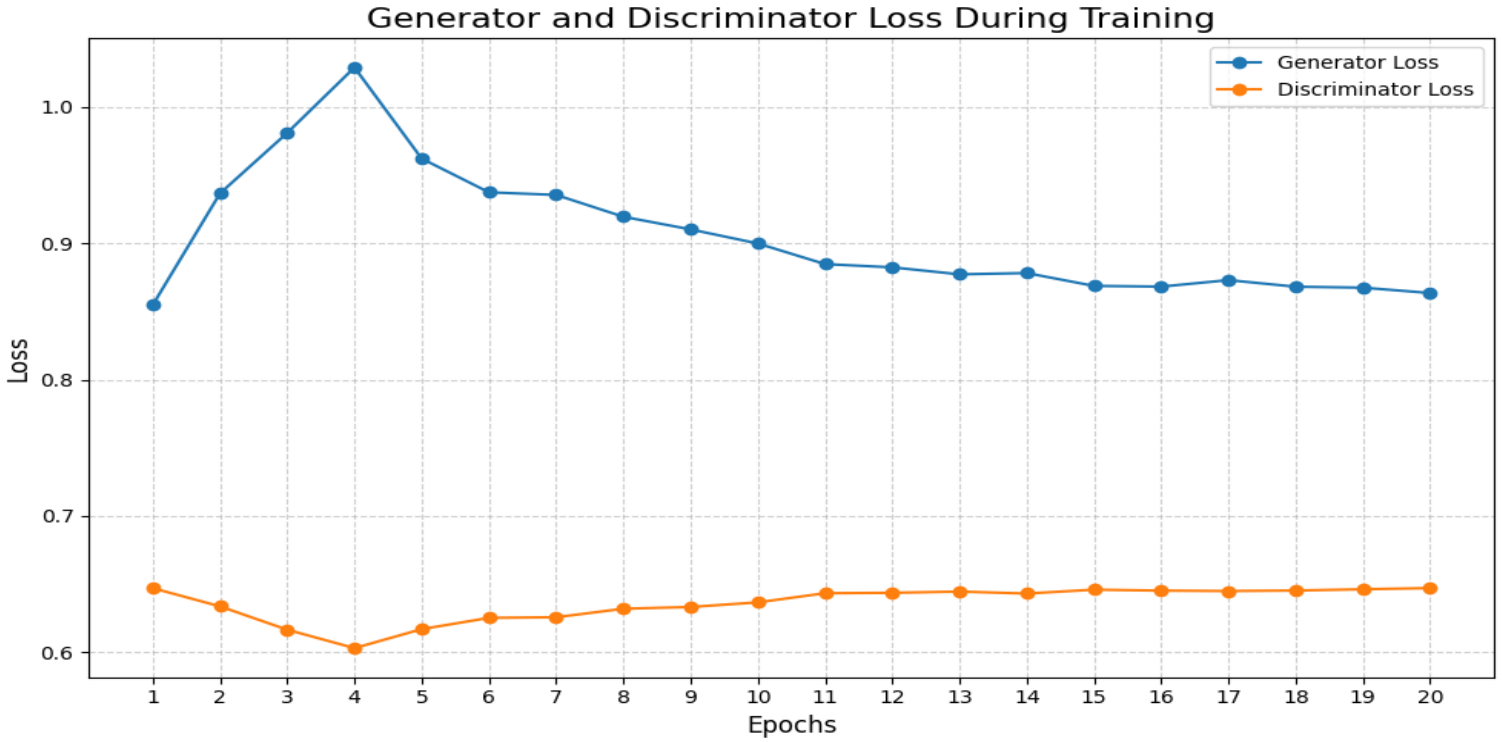
Model Architecture

Generator (The Artist): A multi-layer perceptron (MLP) that takes a 100-dimensional random noise vector as input. It progressively increases the dimensions through hidden layers and uses a Tanh activation to output a 28x28 pixel image with pixel values in the range [-1, 1].

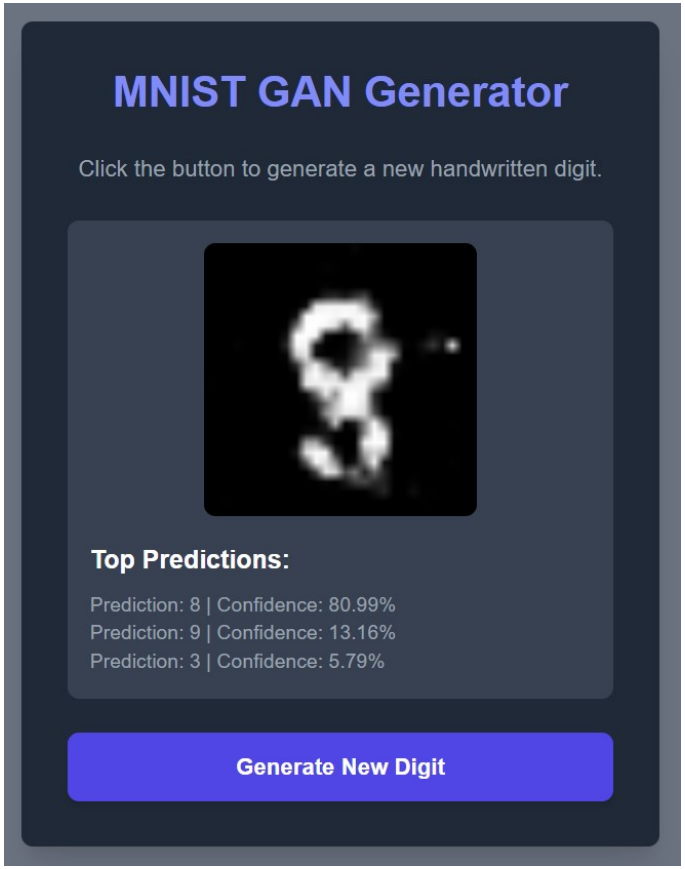
Discriminator (The Critic): An MLP that takes a flattened 784-dimensional image as input. Its purpose is to distinguish between real (MNIST) images and fake (generated) images. It outputs a single scalar value between 0 and 1 using a Sigmoid activation.

Classifier (The Judge): A separate Convolutional Neural Network (CNN). It takes the generated image as input and predicts the digit (0-9) and its confidence level. This model provides the prediction and confidence score for the web application.

Generator and Discriminator Loss During Training



Real-time Image Generation with Prediction



The app generates a new image with each click.
The Classifier model provides a prediction and confidence score for the generated digit.
This demonstrates a complete end-to-end workflow from model generation to user-facing application.

Key Challenges & Lessons Learned

Version Compatibility: Encountered a numpy version mismatch (python 3.10 vs numpy 2.3.2), which was resolved by specifying the correct version in requirements.txt. Lesson: Precise version pinning is critical for ensuring a reproducible and functional environment.

Docker Networking: A browser error (ERR_ADDRESS_INVALID) occurred when trying to connect to 0.0.0.0:8000. Lesson: The correct address for local access is http://127.0.0.1:8000, a crucial detail for deployment.

PyTorch-NumPy Interoperability: An AttributeError was encountered when trying to use a NumPy method on a PyTorch tensor. Lesson: Tensors must be explicitly converted to NumPy arrays (.numpy()) before using NumPy-specific methods.

MLOps & Reproducibility

Containerization: The Dockerfile provides a consistent environment for the application, ensuring it runs the same way on any machine, from development to production.

CI/CD: The GitHub Actions pipeline runs automated tests on the FastAPI endpoints, ensuring the application is always functional.

Test-Driven Development: The project uses pytest to verify the application's core functionality, which is a key part of a professional development workflow.

Future Work & Recommendations

Model Improvement: Train the GAN for more epochs on a larger dataset to improve the quality of the generated digits.

Deployment Automation: Finalize the CI/CD pipeline to automatically build and deploy the Docker image to a cloud service like AWS, Google Cloud, or Azure.

Improved UI: Enhance the user interface with additional features, such as the ability to save generated images or select a specific digit to generate.

Final Takeaways

The losses for both models remain relatively balanced, indicating a stable adversarial training process.

The Generator's loss and the Discriminator's loss are both decreasing, which is a good sign of a stable GAN.

The plot shows a stable trend, which is the most important indicator of a healthy GAN training run.