Generative Adversarial Networks

An Overview

Student Name 1
ID: 2205105

Student Name 2
ID: 2205106

Student Name 3
ID: 2205108

February 27, 2026

What is a GAN?

Intuition

Generative Adversarial Networks (GANs) are generative models that learn to create new data instances resembling the training data.

- Generate realistic **images, audio, video**, etc.

Figure: GAN-generated human faces

Two Players: Generator & Discriminator

Structure

A **Generative Adversarial Network (GAN)** consists of two neural networks trained together: a **Generator** G that creates synthetic samples, and a **Discriminator** D that distinguishes real samples from fake ones.

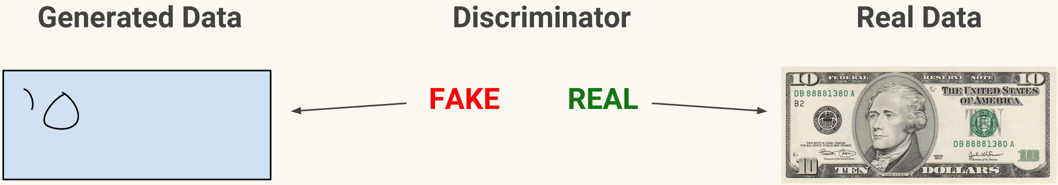
Generator G

- **Input:** random noise z
- **Output:** synthetic sample $G(z)$
- **Goal:** fool D

Discriminator D

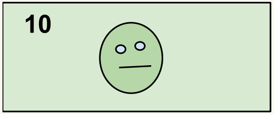
- **Input:** real x or fake $G(z)$
- **Output:** real/fake score
- **Goal:** catch fakes

Training Intuition: Early Stage



- Generator produces obvious fakes; discriminator easily detects them.

Training Intuition: Mid Stage



FAKE REAL

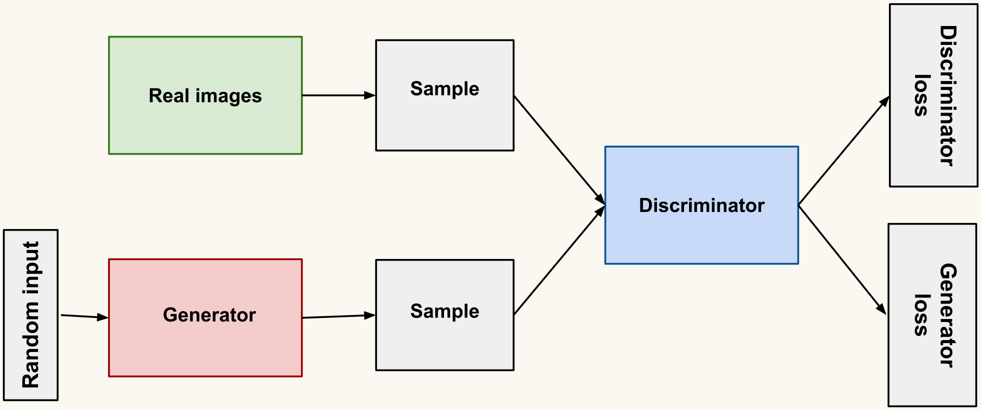
- Generator improves and begins to fool the discriminator.

Training Intuition: Late Stage

REAL **REAL**

- Discriminator struggles to distinguish real from fake; accuracy drops.

GAN Architecture: Full System Overview



The Discriminator in a GAN

What is the Discriminator?

The discriminator D is a **classifier** that learns to distinguish:

- Real data from the dataset
- Fake data produced by G
- D can use any suitable architecture (e.g., CNN for images)

Discriminator Training

Training Data & Update Process

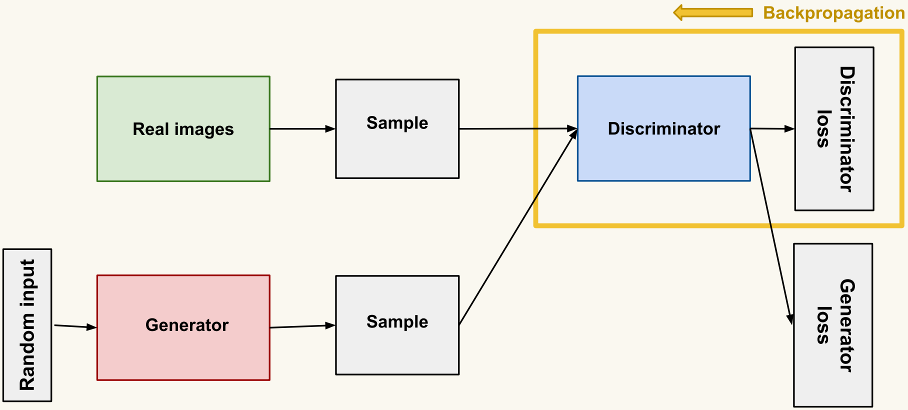
Training Data

- Real samples → positive
- $G(z) \rightarrow$ negative

Update Step

- Classify real & fake
- Compute loss
- Update **only** D

Discriminator Training: Backpropagation



The Generator in a GAN

What is the Generator?

The generator G creates **synthetic data** by transforming random noise z into an output $G(z)$.

- Learns to make fake data look real
- Tries to fool the discriminator D
- Uses noise to produce diverse outputs

Generator Training

Training & Update Process

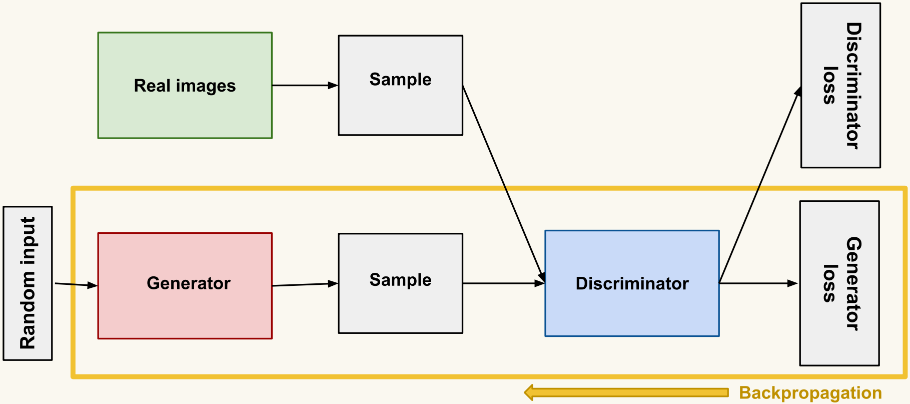
Training Steps

- Sample random noise z
- Compute fake sample $G(z)$
- Get $D(G(z))$ classification

Update Step

- Compute generator loss
- Backpropagate through $D \rightarrow G$
- Update **only** G

Generator Training: Backpropagation



Complications in GAN Training

Why is GAN training difficult?

Because a GAN contains **two separately trained networks**, its training must address two main complications:

- GANs must juggle two different training processes: **Generator** and **Discriminator**.
- GAN convergence is difficult to identify and unstable.

Core challenge

We are not optimizing one model — we are optimizing a **two-player game**.

Alternating Training

How GAN Training Works

1. Train **Discriminator** (keep G frozen)
2. Train **Generator** (keep D frozen)
3. Repeat

- Each network needs a **stable target**.
- Otherwise, training becomes a moving-target problem.

Back-and-forth training makes the adversarial game learnable.

Convergence in GANs

What happens during successful training?

- As G improves, D performance decreases.
- If G succeeds perfectly, D accuracy $\approx 50\%$.

Why convergence is fragile

- Discriminator feedback becomes less informative.
- Generator may start learning from noisy gradients.
- Training too long can cause **mode collapse**.

Loss Functions in GANs

Goal of a GAN

- Match the generated distribution p_g to the real data distribution p_{data}.
- Use a loss function that measures the **distance between two distributions**.

One Loss or Two?

- GANs use two training losses:
  - Discriminator loss
  - Generator loss
- Both come from a **single underlying objective**.
- During generator training, only the term involving fake data is optimized.

Minimax Loss Function

Original GAN Objective

$$\min_G \max_D \left(\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]\right)$$

- $\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] \rightarrow$ Reward D for correctly classifying real data.
- $\mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))] \rightarrow$ Reward D for correctly detecting fake data.
- D tries to **maximize** this objective.
- G tries to **minimize** it (fool D).

Modified Minimax Loss

Generator Objective (Modified)

$$\max_G \mathbb{E}_{z\sim p_z}[\log D(G(z))]$$

- Instead of minimizing $\log(1 - D(G(z)))$
- G maximizes $\log D(G(z))$
- This provides **stronger gradients** early in training

Common GAN Problem: Vanishing Gradients

What happens?

- If the discriminator becomes too strong,
- $D(G(z)) \to 0$
- Generator gradients become very small
- G stops learning

Attempts to Remedy

- Modified minimax loss
- Wasserstein loss

Common GAN Problem: Mode Collapse

What happens?

- Generator produces very limited variety
- Same output (or few outputs) repeated

Attempts to Remedy

- Wasserstein loss
- Unrolled GANs

Common GAN Problem: Failure to Converge

What happens?

- Training oscillates
- Generator quality collapses
- No stable equilibrium

Common Remedies

- Regularization
- Adding noise to discriminator inputs
- Penalizing discriminator weights

Notable GAN Variants

Common GAN Variations

- **Progressive GAN** — progressively increases image resolution during training for faster convergence and higher quality outputs.
- **Conditional GAN (cGAN)** — generates samples conditioned on labels; models $p(X \mid Y)$.
- **CycleGAN** — performs unpaired image-to-image translation using cycle consistency.

Dataset Overview

(Image: Input vs. Target Pair)

Each training image is a side-by-side pair:

- **Left half:** Input image
- **Right half:** Target image

Preprocessing Details:

- Resized to: 256×256
- Range: Normalized to $[-1, 1]$

Note: Pix2Pix requires pixel-level alignment between pairs.

Pix2Pix Architecture

Generator: U-Net

- **Structure:** Encoder–decoder with skip connections to preserve spatial detail.
- **Bottleneck:** Captures high-level features.
- **Activation:** `tanh` output layer for $[-1, 1]$ range.

Discriminator: PatchGAN

- **Local Focus:** Classifies $N \times N$ patches rather than the full image.
- **Output:** A probability map (Real vs. Fake).
- **Benefit:** Reduces blurring by penalizing high-frequency errors.

Objective Function

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

Training Strategy

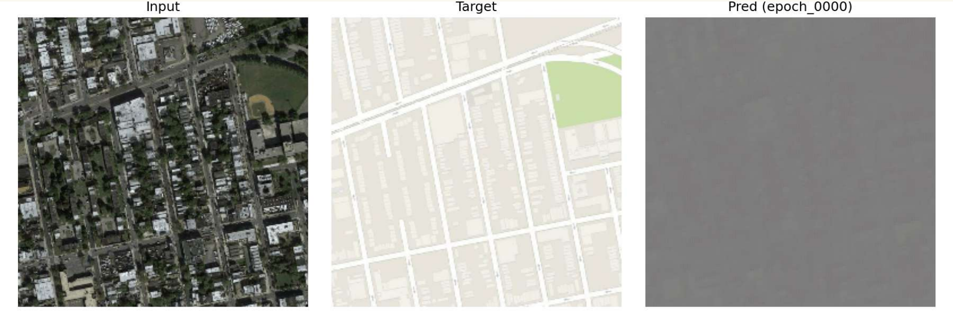
Optimizer

- Adam
- Learning rate: 2×10^{-4}
- β_1: 0.5
- **Total Epochs:** 100

Training Setup

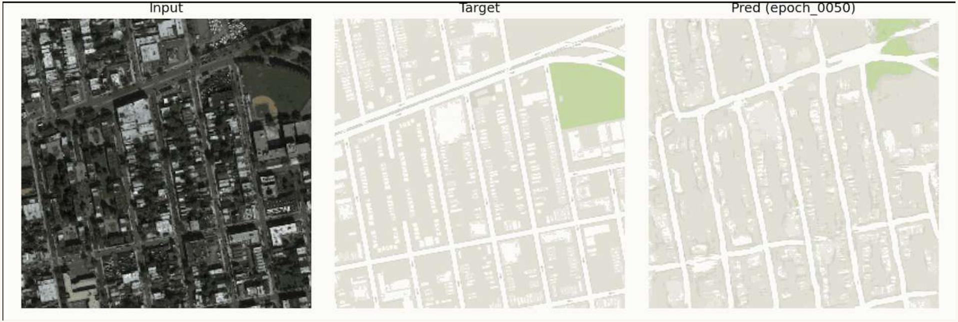
- Batch size: 1
- Alternating D and G updates
- Fixed validation image for tracking
- Snapshot saved every epoch

Generator Progress Over Epochs



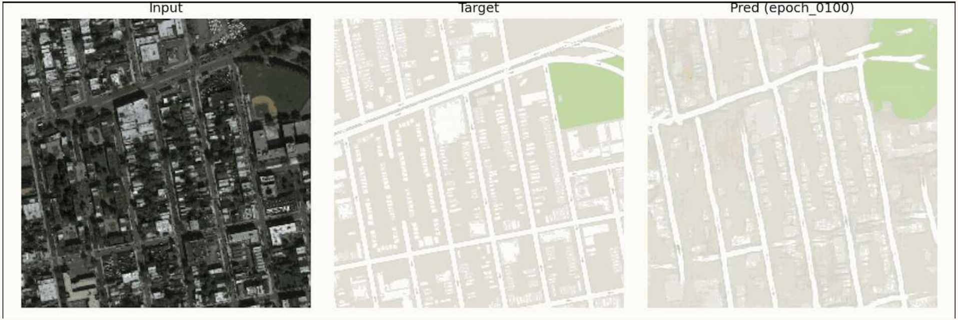
Fixed validation input → Output evolution across epochs

Generator Progress Over Epochs



Fixed validation input → Output evolution across epochs

Generator Progress Over Epochs



Fixed validation input → Output evolution across epochs

Thank you

Questions?