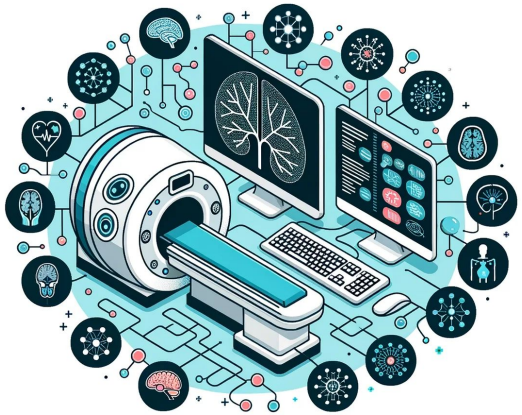
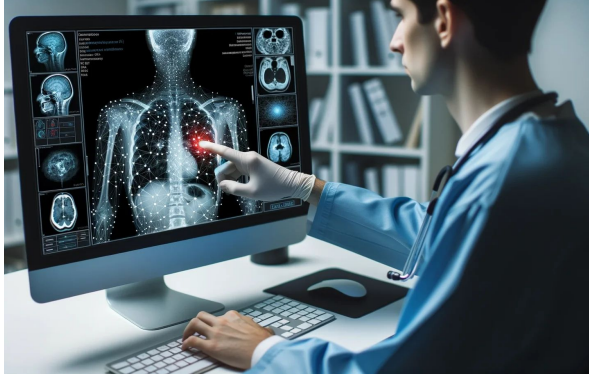
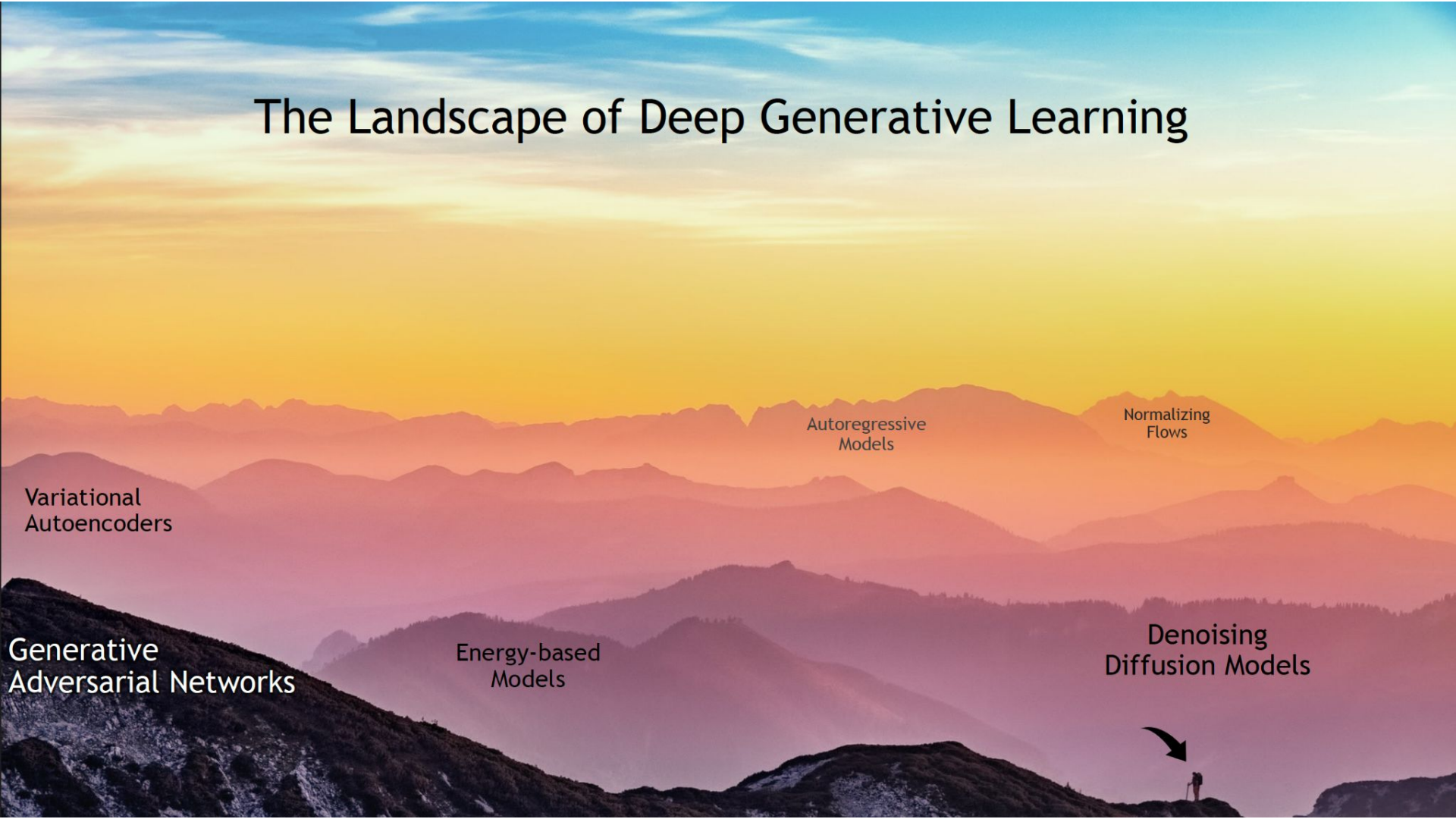


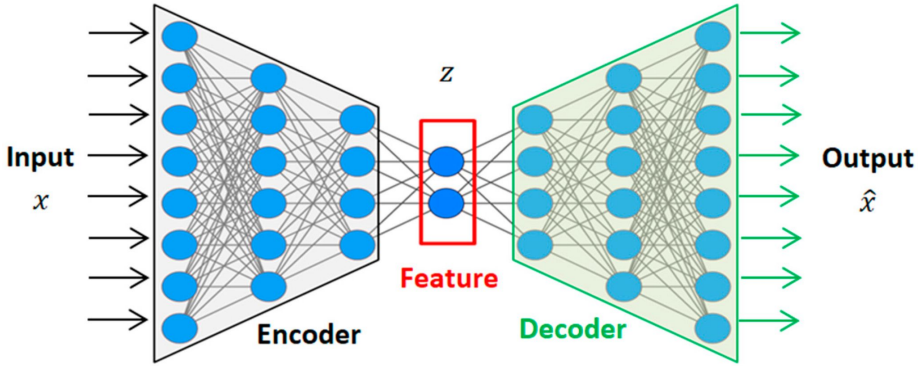


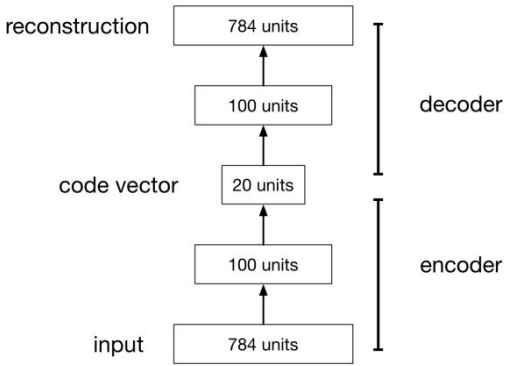
Deep Generative Learning
Session 04
Sep 27 2025


The Landscape of Deep Generative Learning
Autoregressive Models
Normalizing Flows
Variational Autoencoders
Generative Adversarial Networks
Energy-based Models
Denoising Diffusion Models

AutoEncoders

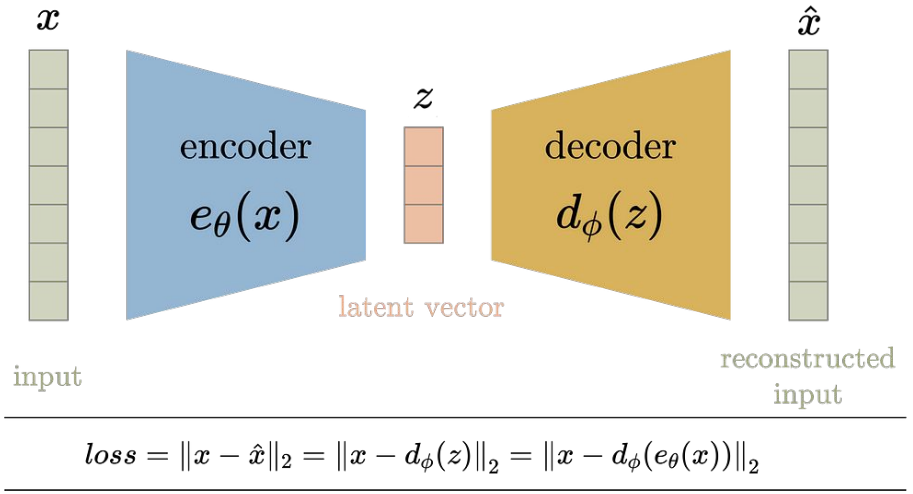
- AE is a feed-forward neural network than given input x learns to reconstruct x.
- There is a bottleneck layer whose dimensions are much smaller than the input dimensions.
- We are reducing 784 input dimensions to only 20 dimensions.
  - We are compressing, keeping only important features.
  - We are trying to reproduce X given using the extracted features.





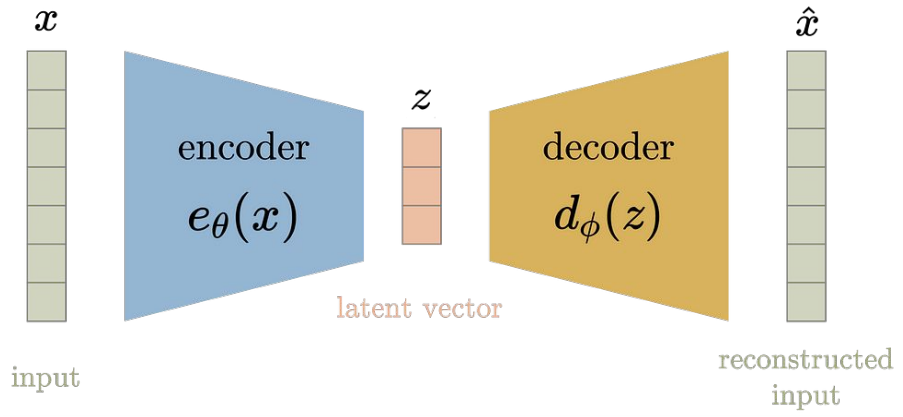
AutoEncoders

- Unsupervised neural network for learning compact embeddings of unlabeled data
- Two main components:
  - **Enoder e_theta(x):** compresses high-dimensional input x into a lower-dimensional latent vector z
  - **Decoder d_phi(z):** reconstructs x from z ensuring x retains most of the original information
- No requirement on the distribution of z it merely must allow accurate reconstruction



Encoder-Decoder Architecture

- Encoder
  - Sequence of user-defined layers (e.g. dense, convolutional, pooling)
  - Reduces input dimension from R^m to latent space R^n $(n < m)$
- Latent Space
  - Dimensionality n is a hyperparameter controlling compression level
  - Smaller n ⇒ higher compression but risk of information loss
- Decoder
  - Near-complement of encoder layers (e.g. transposed conv for conv, unpooling for pooling)
  - Mirrors encoder in reverse to rebuild input dimensions

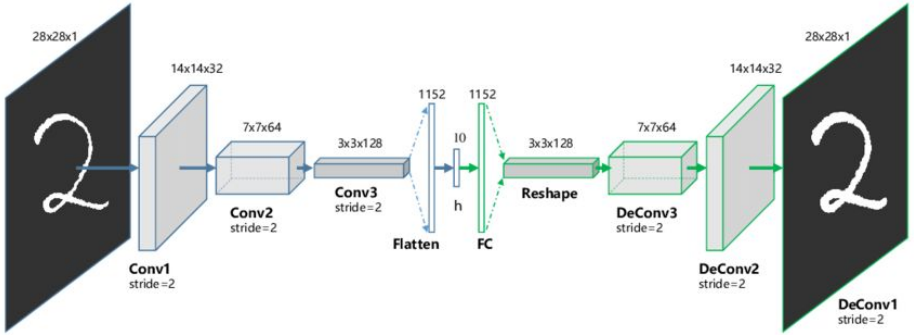


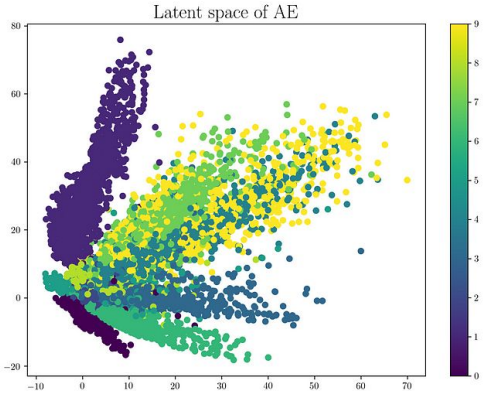
$$loss = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(e_\theta(x))\|_2$$

$$\mathcal{L}(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} \| x_i - d_\phi(e_\theta(x_i)) \|^2$$

Latent Space Visualization

- **Clustering:** Similar digits form distinct clusters in latent space
- **Unregularized Regions:** Empty areas (no training samples) yield meaningless outputs when decoded
- **Limited Generativity:** Only points within learned clusters produce valid digit variations; random samples elsewhere produce garbage
- **Decoding Reliability:** Once trained, there's no way to tell if a random latent vector will decode to a valid digit
- **Primary Use:** Autoencoders excel at compression (reducing redundancy) rather than generative modeling
- **Non-Linear Advantage:** Unlike PCA, non-linear AEs capture complex structures in lower dimensions with less information loss

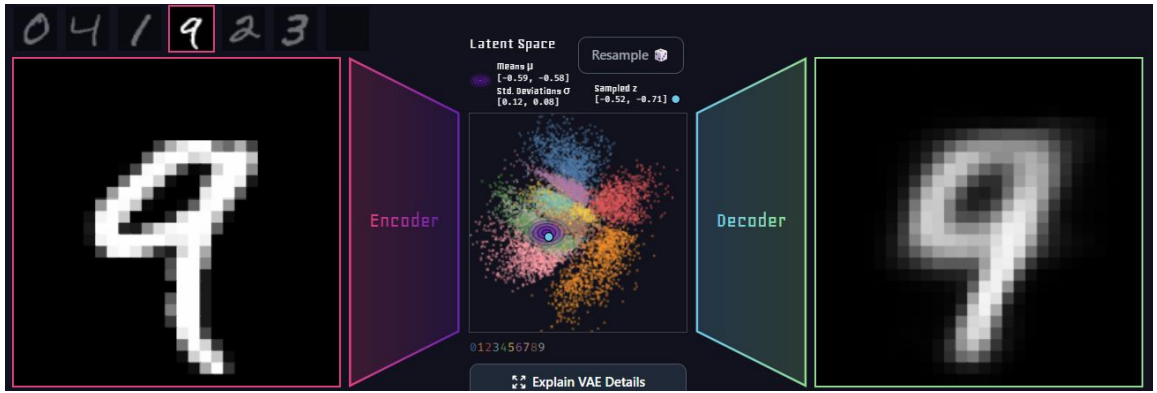




Variational AutoEncoders

- **Traditional AutoEncoders**
  - Deterministic encoding
  - Point estimates in latent space
  - No probabilistic interpretation
  - Can't generate new samples easility
- **Variational AutoEncoders**
  - Probabilistic encoding
  - Distributions in latent space
  - Principled probabilistic framework
  - Natural generation mechanism

- **VAEs** learn a probability distribution over the latent space, not just point estimates. This allows us to sample new data points.



https://xnought.github.io/vae-explainer/

Variational AutoEncoders

- Encoder Network q(Z|X)
  - Maps input X to parameters of a distribution in a latent space
    - Outputs mean \mu and standard deviation \sigma
    - Defines a Gaussian distribution N(\mu, \sigma^2) in latent space
- Latent Space Sampling
  - Sample Z from the distribution using the **reparameterization** trick
- Decoder Network: p(X|Z)
  - Maps latent code Z back to data space to reconstruct X
- VAEs model the joint probability distribution of data and latent variables
  - p(X, Z) = p(X|Z)*p(Z)
  - p(Z) is the prior: Typically N(0, 1)
  - q(Z|X) is the approximate posterior modeled by the encoder
  - p(X|Z) is the likelihood modeled by the decoder

Reparameterization Trick

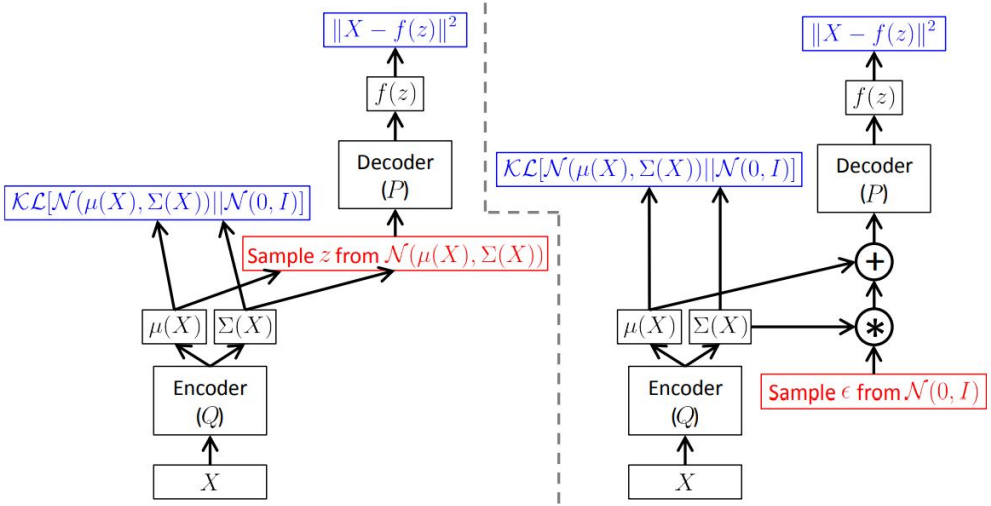
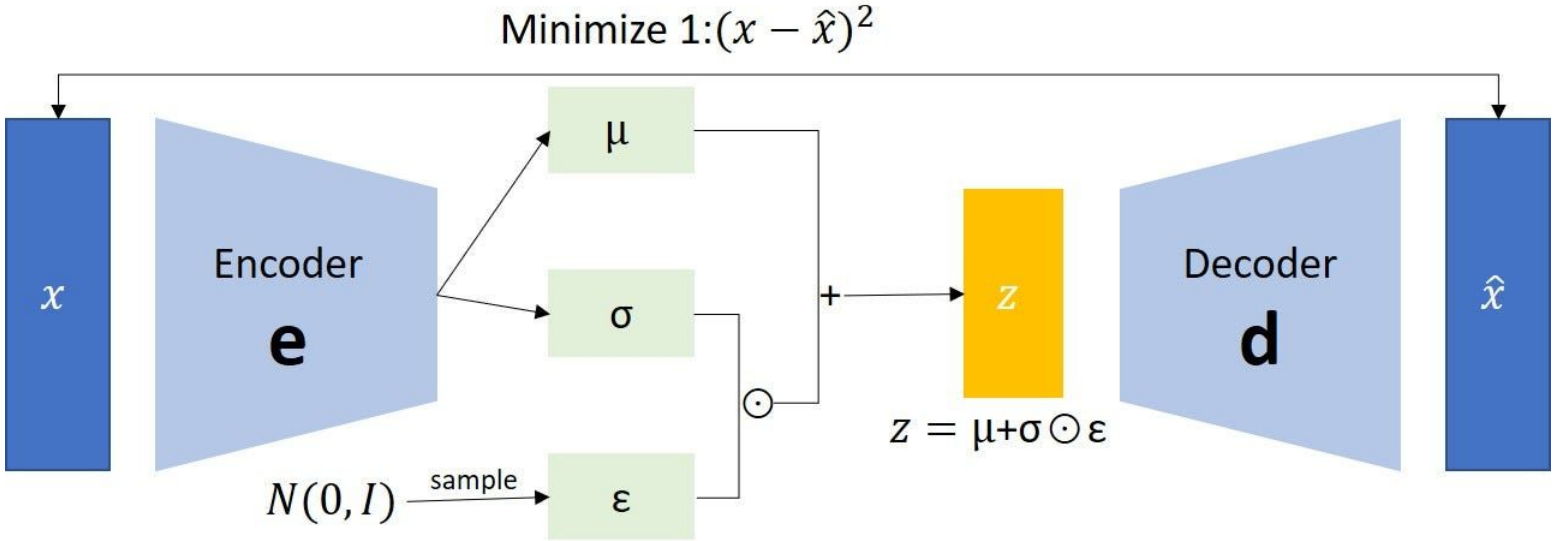


Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.

Variational AutoEncoders

- Reconstruction term
  - How can we reconstruct X from Z
  - Implemented as reconstruction loss (e.g., MSE)
- Regularization term
  - How close is our encoding to the prior
  - Keeps latent space regular and smooth
  - KL[q(Z|X) || p(Z)]
- Training algorithm
  - Pass input X through encoder, get \mu and \sigma
  - Sample \epsilon from N(0, 1) and compute Z = \mu + \sigma * \epsilon
  - Pass Z through decoder and compute losses (Reconstruction Loss + KL Divergence)
  - Backpropagate through decoder, backpropagate through reparameterization, and backpropagate through encoder to update the weights

Variational AutoEncoders



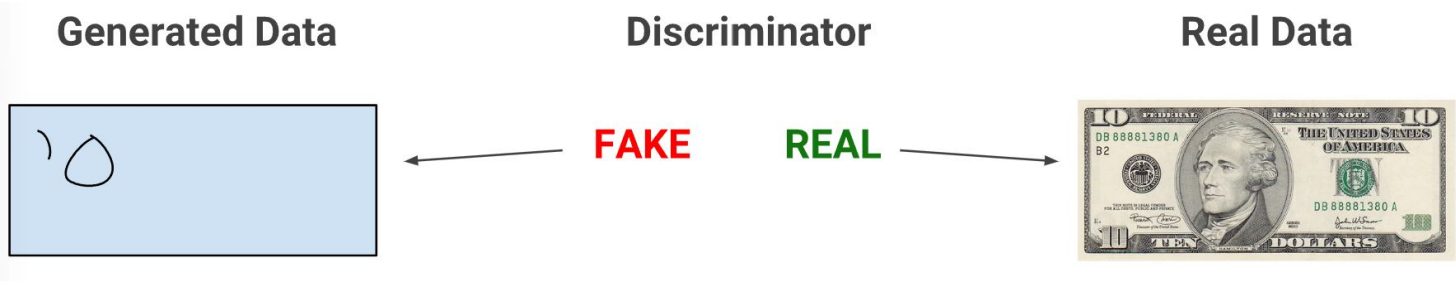
Minimize 2: $\frac{1}{2}\sum_{i=1}^{N}(\exp(\sigma_i) - (1+\sigma_i) + \mu_i{}^2)$

Variational Autoencoders



Generative Adversarial Network (GAN)

- GAN has two parts:
  - The **generator** learns to generate plausible data. The generated instances become negative training examples for the **discriminator**
  - The **discriminator** learns to distinguish the generator's fake data from real data. The **discriminator** penalizes the generator for producing implausible results.
- When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake!



Generative Adversarial Network (GAN)

- As training progresses, the generator gets closer to producing output that can fool the discriminator

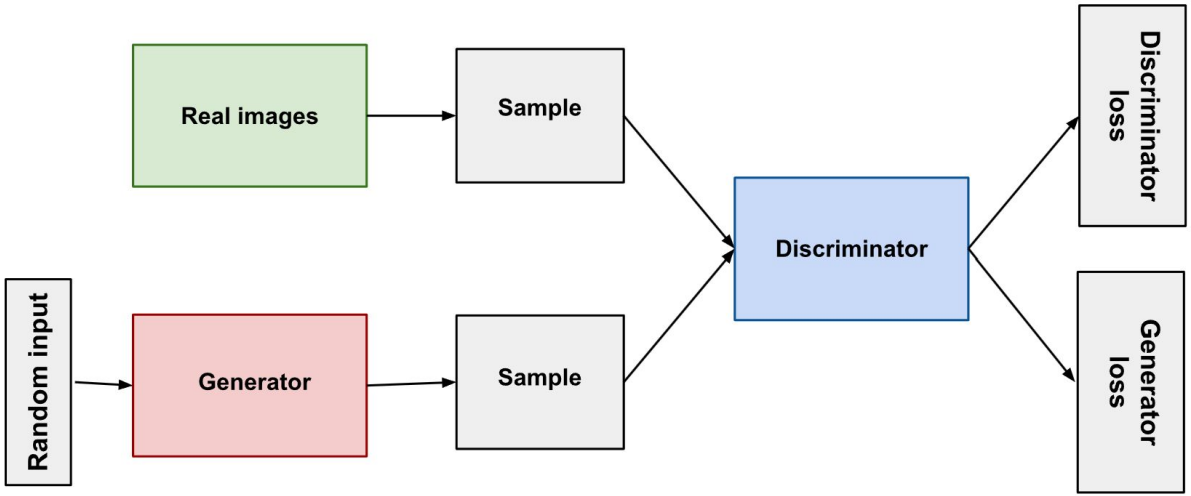


- If generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.



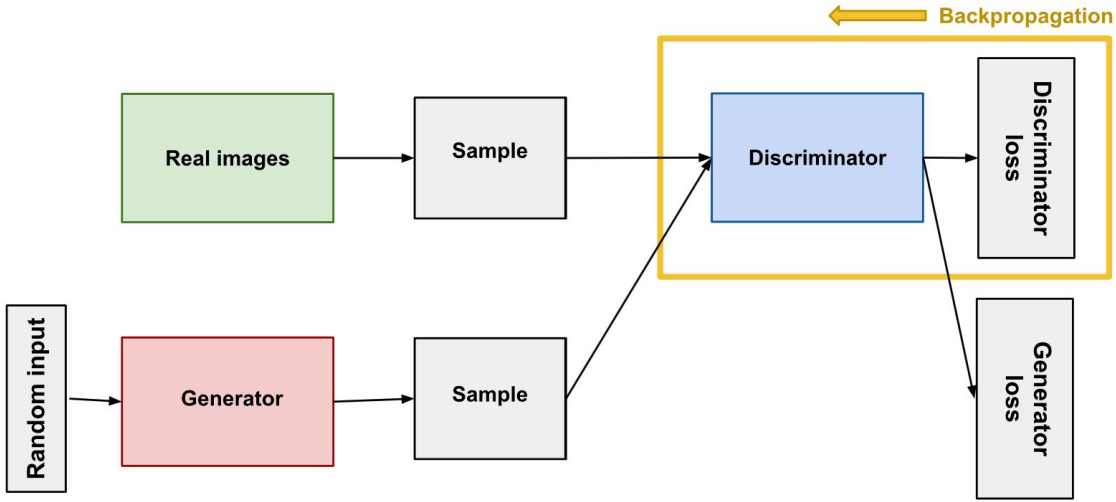
Generative Adversarial Network (GAN)

- Both generator and discriminator are neural networks.
- The generator output is connected directly to the discriminator input.
- The discriminator's classification provides a signal through backpropagation which generator uses to update its weights.



Generative Adversarial Network (GAN)

- The discriminator in a GAN is simply a classifier.
- It distinguished read data from data created by the generator.
- It could use any network architecture appropriate to the type of data it's classifying.



Generative Adversarial Network (GAN)

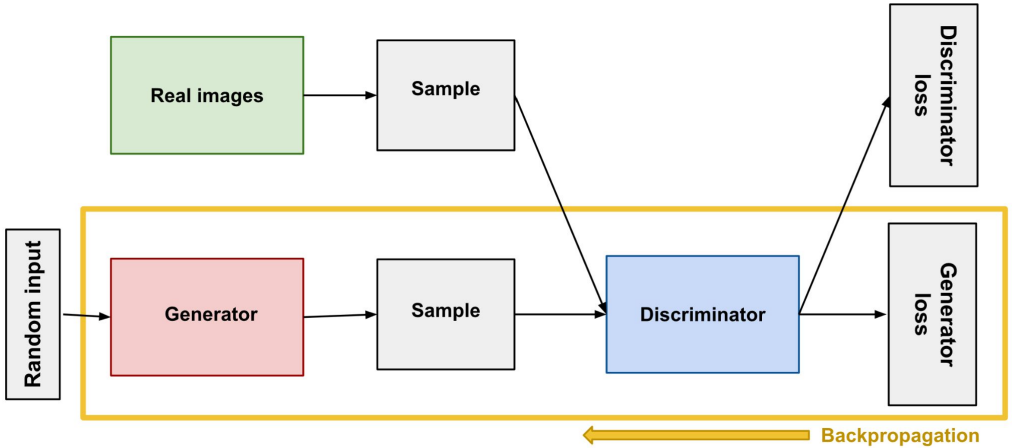
- The discriminator training data comes from two sources:
  - **Real data** instances, such as real pictures of people or real medical images. The discriminator uses these as positive examples during training.
  - **Fake data** instances created by the generator. The discriminator uses these as negative examples during training.
- The discriminator connects to two loss functions:
  - The discriminator classifies both real data and fake data from the generator
  - The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
  - The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.
  -

Generative Adversarial Network (GAN)

- The generator learns to create fake data that can fool the discriminator
- Goal: Make the discriminator classify generator output as "real"
- Random Input (Noise):
  - Usually sampled from simple distribution (e.g., uniform)
  - Lower dimensional than output space
  - Enables diverse data generation
- Generator Network
  - Transforms random noise → meaningful data instances
  - By varying input noise → samples from different parts of target distribution
- Output
  - Synthetic data matching the format of real data
  - Should be indistinguishable from real samples

Generative Adversarial Network (GAN)

- Training the generator - Adversarial Training
  - Sample random noise
  - Generate fake data from noise
  - Pass fake data through discriminator
  - Get discriminator's "Real/Fake" classification
  - Calculate generator loss (penalized when discriminator correctly identifies fakes)
  - Backpropagate through BOTH networks
  - Update ONLY generator weights



Adversarial Training Dynamics and Convergence

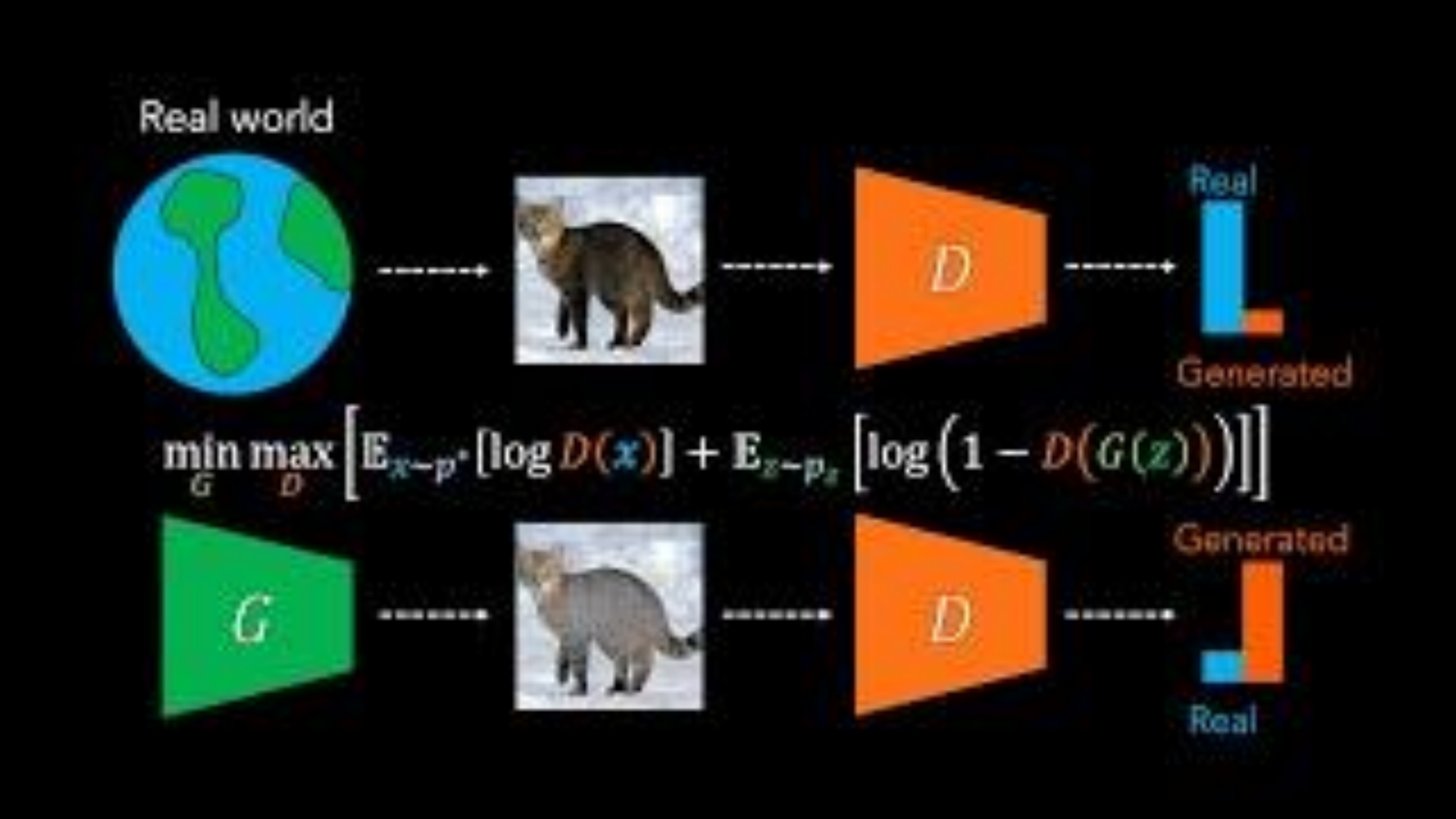
- Two players: Generator (G) vs Discriminator (D)
- Alternating updates:
  - Train D (keep G fixed) → D learns G's flaws
  - Train G (keep D fixed) → G learns to fool a stationary D
- Why alternate? Prevents G from chasing a moving target; turns a hard generation task into solvable classification steps.
- Bootstrapping: if D can't beat random on G's early (bad) samples, training can't get started.
- Convergence behavior: as G improves, D's accuracy → ~50% (can't tell real vs fake). After this point D's feedback becomes noise → risk of G quality collapse.
- In practice, GAN "convergence" is often brief/unstable; monitor and stop before feedback degrades.

GAN Losses as Distances Between Distributions

- Goal: make P_g (generated) ≈ P_data (real).
  - G and D use related but different losses derived from one distance; G only affects the "fake" term.
- Minimax (original GAN)
  - D maximizes: log D(x) + log(1 − D(G(z)))
  - G minimizes: log(1 − D(G(z)))
- Wasserstein GAN (WGAN)
  - Replace classifier with a critic (real-valued scores, not probabilities).
  - Critic maximizes: D(x) − D(G(z))
  - G maximizes: D(G(z))
  - Benefits: better gradients, fewer stalls/vanishing-gradient issues, more stable than minimax.

Common GAN Problems

- Vanishing gradients
  - What happens: D gets too good → G gets almost no learning signal.
  - Tries: Wasserstein loss (WGAN); modified minimax (maximize log D(G(z))).
- Mode collapse
  - What happens: G maps many z's to the same few outputs; G/D chase each other and cycle.
  - Tries: Wasserstein loss (train D to optimality without killing gradients); Unrolled GANs (anticipate future D steps).
- Failure to converge / instability
  - What happens: oscillations or divergence instead of settling.
  - Tries: Regularization—add noise to D inputs; penalize D weights.


Real world
D
Real
Generated
$\min_G \max_D \left[\mathbb{E}_{x \sim p^*} [\log D(x)] + \mathbb{E}_{z \sim p_z} \left[\log \left(1 - D(G(z)) \right) \right] \right]$
G
D
Generated
Real

GANs in Medical Imaging

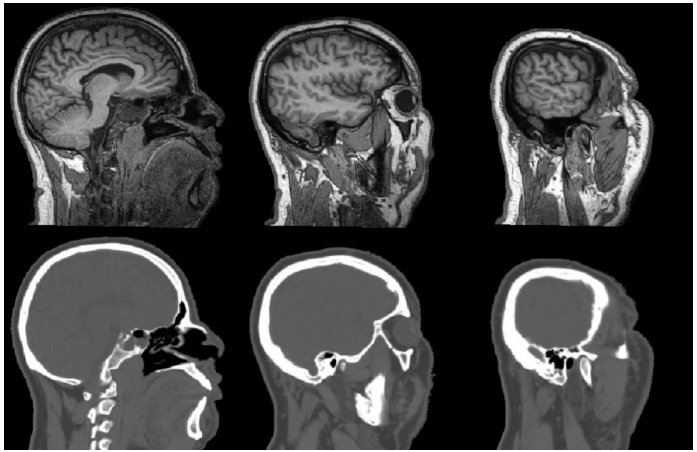
- Clinical need: CS-MRI speeds up scans by undersampling k-space, but classic sparsity methods struggle with detail, tuning, and speed. DAGAN brings a conditional GAN to de-aliasing for fast, high-quality recon.
- Key contributions: U-Net generator with skips; refinement learning for stability/speed; content loss (pixel, frequency, VGG) + adversarial; frequency-domain consistency; broad comparisons.
- Speed: ~0.2 s/CPU slice or ~5 ms/GPU slice → near real-time.

Why MR-to-CT Synthesis?

- Radiotherapy requires both MR and CT
  - MR: Tumor/organ segmentation
  - CT: Dose planning calculations
- Current workflow problems:
  - Dual scan burden (time, cost, patient comfort)
  - Registration errors between modalities
  - Spatial misalignment issues

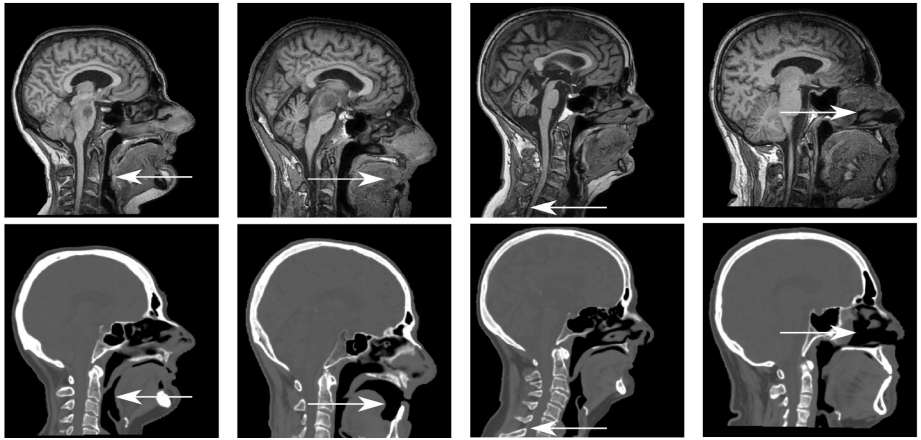
```
Traditional Workflow:
MR Scan → Registration → Treatment Planning
CT Scan ↗           ↘ Potential Errors

Proposed Solution:
MR Scan → Synthetic CT → Direct Planning
          (No registration needed)
```



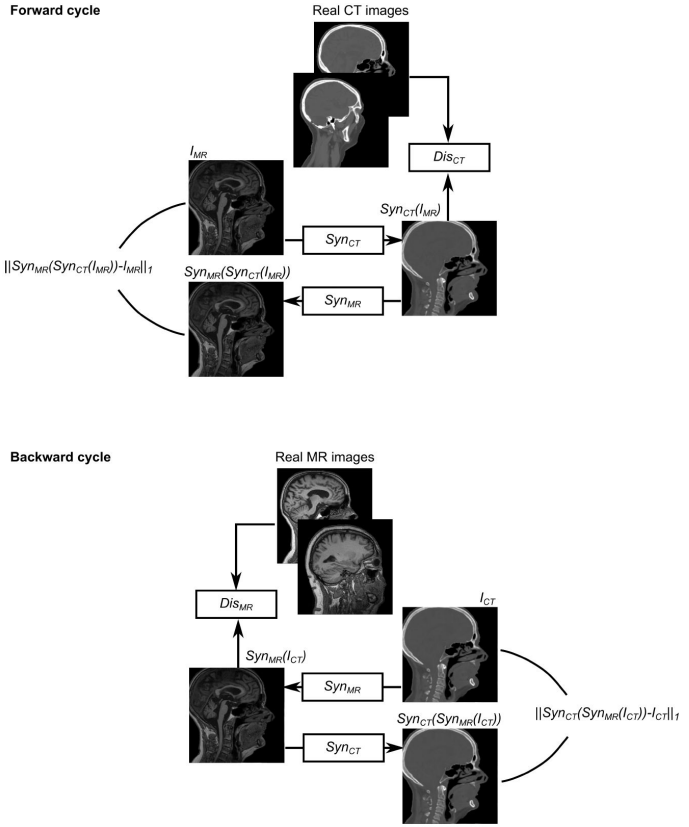
The Paired Training Problem

- Previous CNNs require paired training data
- Voxel-wise alignment assumption
- Problems with paired approaches:
  - Misalignment causes blurred synthesis
  - Limited training data availability
  - Cannot use unpaired clinical datasets



Bidirectional Translation with Cycle Consistency

- **Forward Cycle:**
  - SynCT: MR → CT synthesis
  - SynMR: CT → MR reconstruction
  - DisCT: Real vs. synthetic CT discrimination
- **Backward Cycle:**
  - SynMR: CT → MR synthesis
  - SynCT: MR → CT reconstruction
  - DisMR: Real vs. synthetic MR discrimination



CNN Components and Training

- **Synthesis Networks (SynCT, SynMR):**
  - 2 strided convolution layers
  - 9 residual blocks
  - 2 fractionally strided convolutions
  - Input/Output: 256×256 pixels
- **Discriminator Networks (DisCT, DisMR):**
  - PatchGAN architecture
  - 70×70 patch classification
  - Focus on high-frequency details

```
Training Configuration:
- 18 patients (training)
- 6 patients (testing)
- 183 sagittal slices per volume
- Adam optimizer
- 200 epochs total
  - 100 with lr = 0.0002
  - 100 with linear decay
- λ = 10 for cycle consistency
- Training time: 52 hours (Titan X)
```

Adversarial and Cycle Consistency Losses

- **Adversarial Loss (CT):**

$$L_{CT} = (1 - Dis_{CT}(I_{CT}))^2 + Dis_{CT}(Syn_{CT}(I_{MR}))^2$$

- **Adversarial Loss (MR):**

$$L_{MR} = (1 - Dis_{MR}(I_{MR}))^2 + Dis_{MR}(Syn_{MR}(I_{CT}))^2$$

- **Cycle Consistency Loss:**

$$L_{Cycle} = ||Syn_{MR}(Syn_{CT}(I_{MR})) - I_{MR}||_1 + ||Syn_{CT}(Syn_{MR}(I_{CT})) - I_{CT}||_1$$

Competing Objectives:

Generator: Fool discriminator

Discriminator: Detect fakes

Cycle Loss: Preserve content

$$\text{Total Loss} = L_{Adv} + \lambda \cdot L_{Cycle}$$

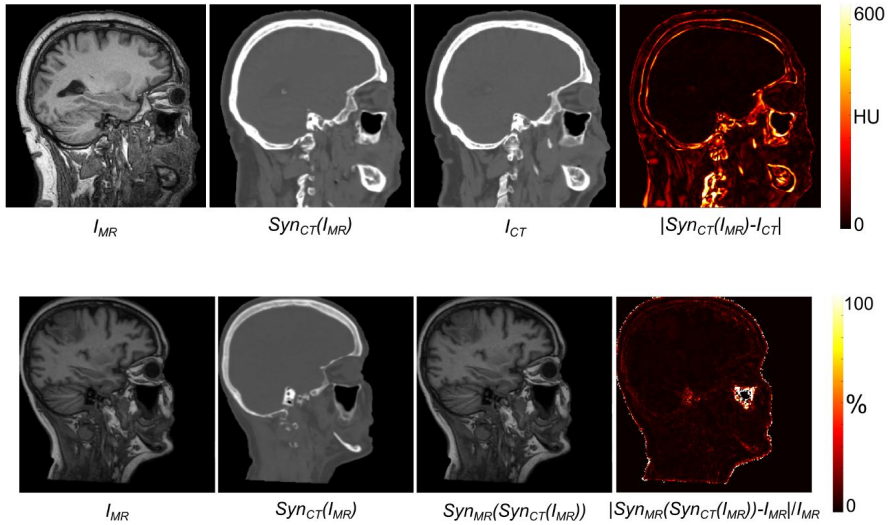
Quantitative Performance Comparison

- **Mean Absolute Error (MAE):**
  - Unpaired: 73.7 ± 2.3 HU
  - Paired: 89.4 ± 6.8 HU
  - 17.5% improvement
- **Peak Signal-to-Noise Ratio (PSNR):**
  - Unpaired: 32.3 ± 0.7
  - Paired: 30.6 ± 0.9
  - Higher is better
- **Statistical Significance:**
  - $p < 0.05$ (paired t-test)

	MAE		PSNR	
	Unpaired	Paired	Unpaired	Paired
Patient 1	70.3	86.2	31.1	29.3
Patient 2	76.2	98.8	32.1	30.1
Patient 3	75.5	96.9	32.9	30.1
Patient 4	75.2	86.0	32.9	31.7
Patient 5	72.0	81.7	32.3	31.2
Patient 6	73.0	87.0	32.5	30.9
Average ± SD	73.7 ± 2.3	89.4 ± 6.8	32.3 ± 0.7	30.6 ± 0.9

Synthesis Quality Comparison

- **Successful Differentiation:**
  - Bone vs. soft tissue
  - Ventricular fluid vs. air
  - Brain tissue structures
- **Error Distribution:**
  - Minimal in soft brain tissue
  - Higher in bone structures
  - Affected by neck misalignment
- **Unpaired vs. Paired:**
  - Less blurring with unpaired
  - Fewer artifacts
  - More realistic appearance



Accelerating MRI Acquisition

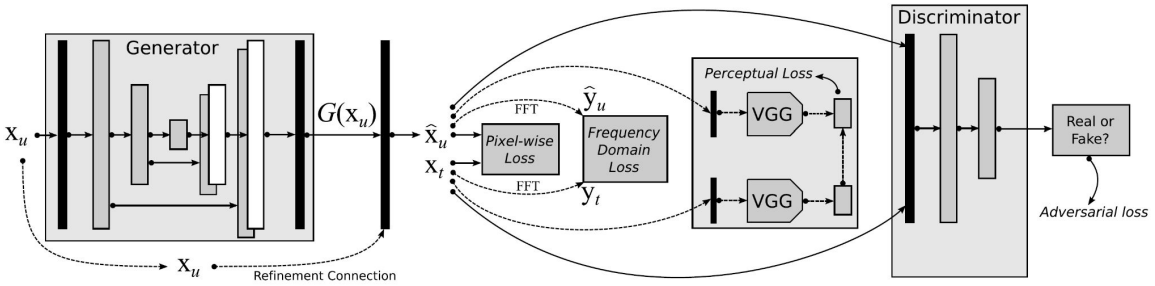
- **Problem:** MRI is inherently slow
  - Sequential k-space acquisition
  - 64-512 lines needed for quality
  - Patient discomfort, motion artifacts
  - High costs, contrast washout
- **Compressed Sensing Solution:**
  - Break Nyquist-Shannon barrier
  - Randomly undersample k-space
  - Exploit image sparsity
  - Acceleration factors: 2-6×

```
Traditional CS-MRI Pipeline:
k-space → Undersampling → Aliasing artifacts
          ↓
Iterative optimization (slow)
          ↓
Reconstructed image

DAGAN Pipeline:
k-space → Undersampling → Deep Network → 5ms
                                (trained offline)
```

Conditional GAN with Refinement Learning

- **Generator (U-Net):**
  - 8 encoder layers + 8 decoder layers
  - Skip connections preserve details
  - Refinement: G(xu) + xu (not just G(xu))
  - Stabilizes training dramatically
- **Discriminator:**
  - 11 convolutional layers
  - Distinguishes real vs. de-aliased
  - Provides adversarial feedback
- **Innovation:** Refinement learning
  - Generator learns missing information only
  - Reduces complexity, faster convergence



Multi-Component Loss Strategy

1. **Adversarial Loss (LGEN):**
   - Fool discriminator
   - Generate realistic images
2. **Pixel-wise MSE (LiMSE):**
   - Image domain fidelity
   - Preserve structural accuracy
3. **Frequency MSE (LfMSE):**
   - k-space consistency
   - Enforce data fidelity
4. **Perceptual Loss (LVGG):**
   - VGG network features
   - Preserve texture/edges
   - Prevent over-smoothing

```
LTOTAL = αLiMSE + βLfMSE + γLVGG + LGEN

Weights (balanced magnitudes):
- α = 15 (pixel loss)
- β = 0.1 (frequency loss)
- γ = 0.0025 (VGG loss)

Why this matters:
→ Adversarial alone: unstable
→ MSE alone: blurry
→ Combined: sharp, accurate
```

Performance Comparison

- **Undersampling Ratios Tested:**
  - 10%, 20%, 30%, 40%, 50% retained
  - (10×, 5×, 3.3×, 2.5×, 2× acceleration)
- **DAGAN Performance (30% retained):**
  - NMSE: 0.07±0.01
  - PSNR: 32.5±0.7 dB
  - SSIM: 0.87-0.98
- **Comparison vs. Others:**
  - TV: PSNR 28.2 dB
  - DLMRI: PSNR 28.5 dB
  - BM3D: PSNR 31.7 dB
  - **DAGAN: PSNR 32.8 dB**

