

Advanced Deep Learning for Computer Vision

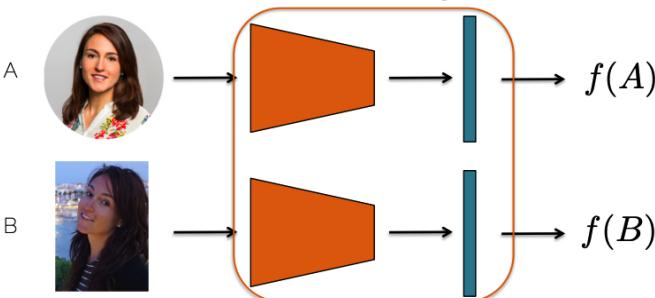


1 Recap: Introduction to Deep Learning

1.1 Siamese Network

- Idea: Train a network to learn a similarity function between two inputs
- Architecture: Two identical subnetworks sharing weights, process the two inputs separately
- Output: A distance metric (e.g., Euclidean distance) between the two feature vectors produced by the subnetworks
- Loss Function: Contrastive Loss or Triplet Loss to encourage similar inputs to have smaller distances and dissimilar inputs to have larger distances
- Applications: Face verification, signature verification

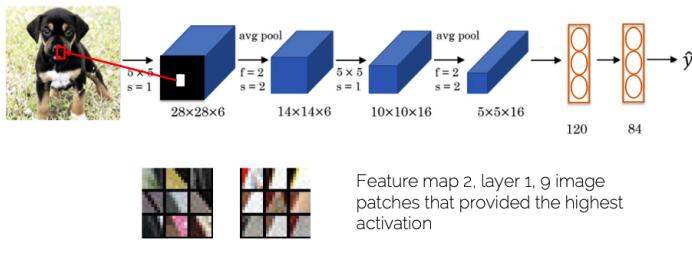
Distance function: $d(A, B) = \|f(A) - f(B)\|_2^2$



1.2 Visualization of ConvNets

Visualization in the Image Space

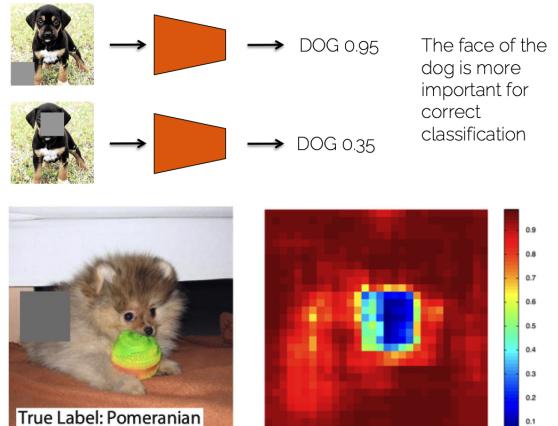
- Layer 1: basic geometric shapes
- Deeper Layers: more complex patterns and object parts
- Deeper layers have a much higher resolution since they display the receptive field



Zeller and Fergus, "Visualizing and understanding convolutional neural networks", ECCV 2014

Visualize Importance: Occlusion Experiment

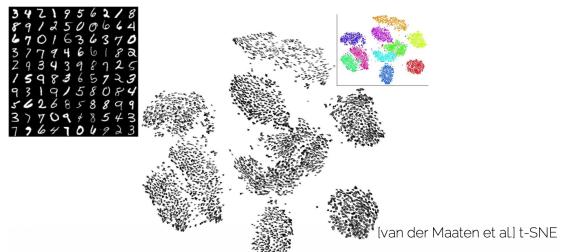
- block different parts in the image and see how classification score changes
- heatmap shows which parts of the image are most important for classification (lower score = more important)



1.3 Visualization of Clusters

t-SNE: t-Distributed Stochastic Neighbor Embedding

- Non-linear dimensionality reduction technique
- Maps high-dimensional data to a lower-dimensional space (typically 2D or 3D) while preserving pairwise distance of local points
- Perplexity parameter controls the balance between local and global aspects of the data
- Not useful for new data points (non-parametric)



1.4 Autoencoders and VAE

Autoencoders

- Unsupervised approach for learning lower-dimensional feature representations
- Consist of an encoder (maps input to latent space) and a decoder (reconstructs input from latent space)
- Trained to minimize reconstruction error (e.g., Mean Squared Error)
- Can be used for dimensionality reduction, de-noising, and generative modeling
- Bottleneck layer: dim (latent space) << dim(input space)

Variational Autoencoders (VAE)

- Probabilistic extension of autoencoders
- Encoder outputs parameters of a probability distribution (mean and variance) instead of a single point in latent space
- Latent space is continuous, allowing for smooth interpolation and generation of new samples
- Loss function includes reconstruction error and a regularization

term (Kullback-Leibler divergence) to ensure latent space follows a prior distribution (usually Gaussian)

2 3D Data Representation

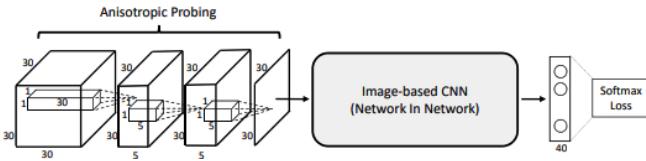
- Explicit Representations: Voxel grids, Point Clouds, Polygonal Meshes
- Implicit Representations: Signed Distance Functions (SDFs), Occupancy Networks

2.1 Volumetric Grids

Volumetric Data Structures

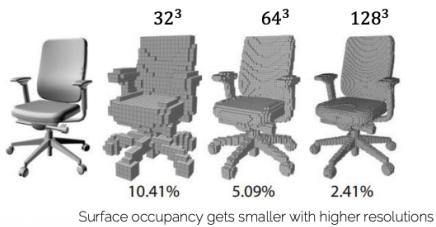
- Occupancy grids
- Ternary grids (2-D triangular coordinate system, e.g., Barycentric coordinate)
- (Signed) Distance fields

Extension of AlexNet/2D CNNs to 3D CNNs by using 3D convolutional kernels: [Object Classification with 3DCNNs, 2016](#)



Summary:

- + Simple data structure
- + Encode free space and distance fields
- + Naturally extend 2D CNNs and other concepts
- + Faster training due to an additional dimension per sample (but often lack 3D data)
- High memory consumption (cubic growth), a lot is used for empty space the higher the resolution
- Require a lot of processing time (use sliding window or fully-convolutions)



2.2 Volumetric Hierarchies

Octrees

- Hierarchical data structure that recursively subdivides 3D space into eight octants (like a quadtree in 2D).
- 3D partitioning: higher resolution at surfaces, lower resolution in empty space for applying 3DCNN kernels
- Can be used for discriminative tasks (like object classification, segmentation) or generative tasks (like 3D reconstruction).

Summary:

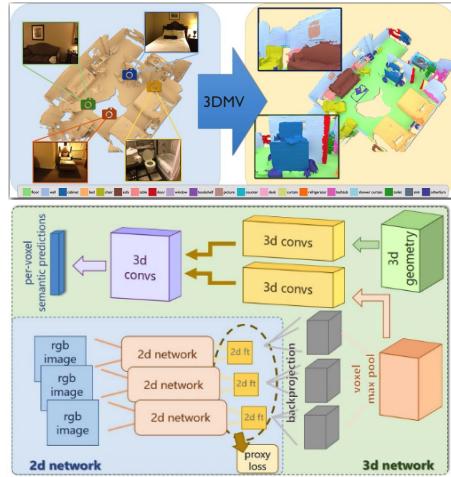
- + Great for reducing memory and runtime
- + Increases strongly performance of 3D CNNs

- Easier for discriminative tasks when structure is known, more complex for generative tasks (split voxel that are partially occupied)

2.3 Hybrid: Volumetric + Multi-View

- Combine volumetric representation with multi-view images (colors) to improve segmentation
- Separate 2D (image) and 3D (voxel) feature extraction + back-projection of 2D features into 3D space

[3DMV: 3D Voxel + Multi View Semantic Segmentation, 2016](#)



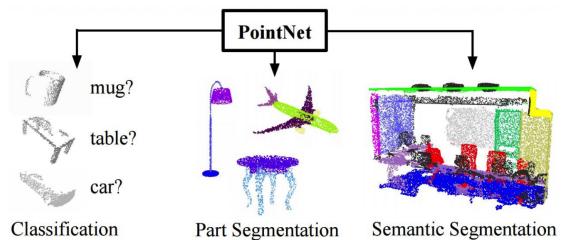
Summary:

- + Nice way to combine color and geometry
- + Good performance
- End-to-End training helps less than expected
- Could be faster

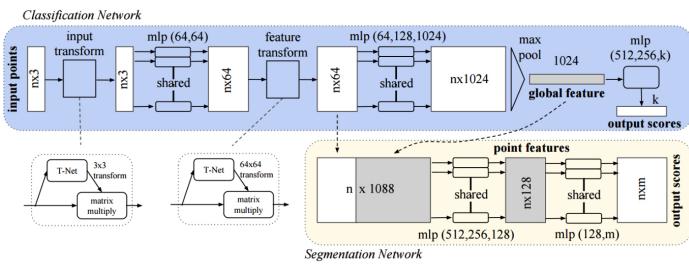
2.4 Point Clouds

- Unordered set of 3D points (often from LiDAR or depth sensors)
- Each point can have additional features (e.g., color, intensity, normals)
- Direct processing of point clouds using specialized neural networks (e.g., PointNet, PointNet++)

[PointNet: 3D Classification and Segmentation for Point Clouds, 2017](#)



- take n points as input
- apply feature transformations and aggregate features by max pooling
- classification net: output scores for k classes
- segmentation net: concatenate global and local features for per-point classification



PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space, 2017

- learn hierarchical representation of point clouds
- apply multiple PointNets at different locations and scales
- multiscale grouping (MSG) and multi-resolution grouping (MRG) for local features

Point Convolutions

- Transform points to continuous representation using radial basis functions (RBFs) or kernel density estimation (KDE)
- Apply convolutional operations on the continuous representation

Point Transformer

- Use self-attention mechanisms to capture relationships between points
- Learn point features by attending to neighboring points

2.5 Sparse Convolutional Networks

- Efficiently process sparse 3D data (e.g., point clouds, voxels with many empty spaces)
- Use sparse convolutional operations that only compute on non-empty voxels
- To improve efficiency, use sparse hashes (e.g. QSParseConv, MinkowskiEngine) instead of masking conv output
- Latency on GPU for hash lookup is hidden by parallelism

Summary:

- + Efficient representation, only store surface points
- + Fast training and testing
- + Cover large space in one shot
- Can not represent free space
- Perform worse than volumetric methods in some tasks (a lot ongoing research)

2.6 Polygonal Meshes

- Represent 3D surfaces using vertices, edges, and faces (typically triangles or quadrilaterals)
- Widely used in computer graphics and 3D modeling
- Process via graph neural networks
 - Message Passing
 - Graph Convolutions
 - Transformers
- Process via specialized mesh convolutional networks
 - MeshCNN
 - Geodesic CNNs
 - Spectral CNNs

2.7 Signed Distance Functions (SDFs)

2.8 Occupancy Networks

3 3D Datasets

3.1 3D Shapes

ShapeNet

- Main Dataset, synthetic 3D models
- 51.3k shapes, 55 classes
- mostly chairs, mediocre textures

Objaverse-XL

- 10mio 3D shapes
- heterogeneous distributed

3.2 3D Scenes

3D-FRONT

- furnished rooms with layouts and semantics
- 18k rooms, 7k furniture objects

ScanNet

- Kinect-style reconstructions of indoor scenes
- 1.5k scenes, 2.5 million views
- Semantic and instance annotations
- from TUM, with standardized evaluation benchmark

4 Neural Fields

- Field: quantity defined for all spatial or temporal coordinates
- Neural Field: represent a field represented by a neural network

4.1 MLP as Data Structure

- MLP can represent complex signals (images, 3D shapes, etc)
- Sparse encoding of signal, shifts capacity where it needs it (based on optimization)
- Can be used for compression, denoising, super-resolution, etc.
- Often also referred to as Implicit Neural Representation (INR) or Coordinate-based MLP

4.2 Implicit vs Explicit

Implicit Neural Representation: Misleading since only the data structure is implicit not the function itself.

$$\text{Explicit function: } f(x) = y \quad (1)$$

$$\text{Implicit function: } x^2 + y^2 - 1 = 0 \quad (2)$$

Signed Distance Field (SDF) A signed distance field assigns to each point $x \in \mathbb{R}^n$ a real value that represents the distance to the closest surface, with the sign indicating whether the point lies inside or outside the surface:

$$\text{SDF}(x) = \begin{cases} < 0, & \text{inside surface,} \\ = 0, & \text{on surface,} \\ > 0, & \text{outside surface.} \end{cases}$$

Occupancy Field An occupancy field assigns to each point $x \in \mathbb{R}^n$ a binary indication of whether the point lies inside the sur-

face:

$$\text{Occ}(x) = \begin{cases} 1, & \text{if } \text{SDF}(x) \leq 0, \\ 0, & \text{if } \text{SDF}(x) > 0. \end{cases}$$

Key Difference

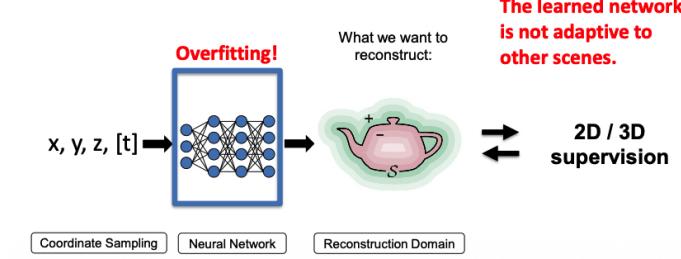
- **SDF**: continuous-valued; encodes signed distance from the surface.
- **Occupancy**: binary-valued; encodes only inside/outside.

4.3 Overfitting vs. Generalization

- Overfitting as an artifact of neural fields: MLPs can memorize training data without generalizing.
- Overfitting as a debugging tool: helps to verify that the network architecture and training procedure are functioning correctly.
- Overfitting as a goal: to represent a specific signal or shape accurately.

Overfitting Single Scene

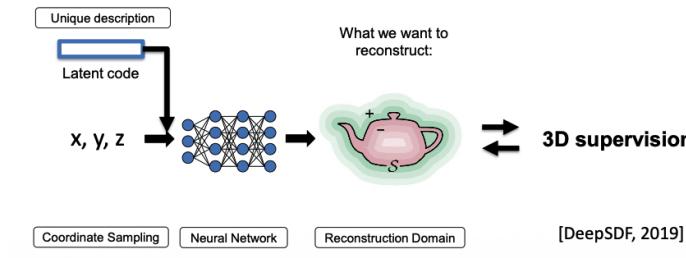
Overfitting a single scene



- Overfitting a single scene from multiple views
- Input set of images and their camera parameters

Overfitting Multiple Scenes

- Each scene should have a unique description
- To prevent latent code bias in activations, either a positional encoding is required or a learned latent code (optimized during training)
- learned latent code: has semantic meaning: like parametrization of Radial Basis Functions (RBFs)
- Changing latent code allows interpolation between data



4.4 Inductive Bias

Shift invariance is very important to account for bias in activation based on position/coordinate values

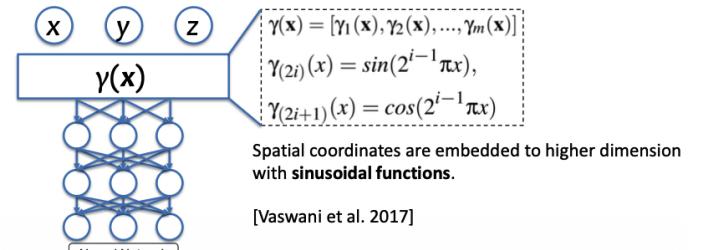
- Positional encoding (Sinusoidal high-dimensional mapping of coordinates)

- SIREN Activations (sinusoidal activations to represent high-frequency functions)

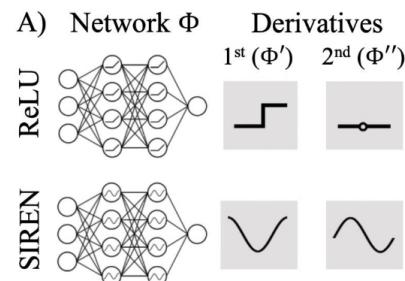
[On the Spectral Bias of Neural Networks, 2018](#)

Positional Encoding Sinusoidal / learned positional encodings allow:

- smooth distance representation,
- easy extrapolation,
- relative comparison (difference of phases encodes distance).
- This provides a useful inductive bias: tokens that are close in sequence have similar positional embeddings.
- Raw integer positions do *not* provide this similarity.



Activation Functions SIREN uses sinusoidal activations to fit to high-frequency functions.



4.5 Hybrid Representations

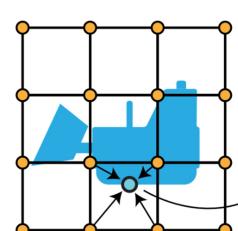
• Uniform grid

- Dense, regularly-sampled grid over the whole domain
- Easy indexing and interpolation
- Memory and computation scale poorly with resolution
- Wastes resources in empty or simple regions

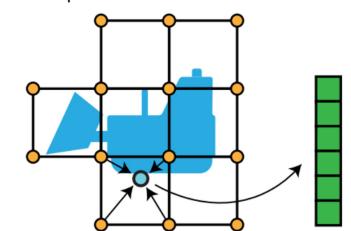
• Sparse grid

- Only stores cells where signal / geometry is complex
- Greatly reduces memory and compute, $O(\log(n))$ access time
- GPU compatible data structure
- Established operations like sparse 3D convs
- Requires hierarchical / hashed data structures
- More complex to implement and query

• Uniform Grids



• Sparse Grids



4.6 Summary

- 1. Overfit an MLP
- 2. Positional Encoding: prevent inductive bias from coordinates
- 3. Activation Functions: non-linearities to represent high-frequency functions
- MLP: fully connected layers, connect everything with everything
- CNN: local connections, weight sharing, translation equivariance
- Transformer: learn which connections are important

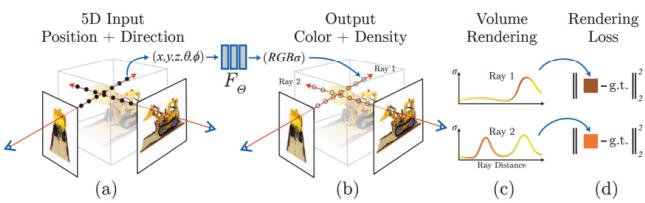
5 Neural Radiance Fields (NeRF)

- Neural Radiance Field (NeRF) is a neural network-based representation for 3D scenes that encodes both geometry and appearance.
- Enables photo-realistic novel view synthesis from a sparse set of input images.
- Now-standard approach to neural volumetric rendering.

[Neural Radiance Fields \(NeRF\), Mildenhall et al., ECCV 2020](#)

5.1 Core Idea

- NeRF represents a scene as a continuous 5D function:
$$F_\theta : (x, y, z, \theta, \phi) \rightarrow (\sigma, \mathbf{c})$$
- 5D Input:
 - (x, y, z) : 3D position
 - (θ, ϕ) : viewing direction
- Outputs (MLP):
 - \mathbf{c} : emitted color at sample point perceived from direction
 - σ : volume density (chance of absorption at specific location)
- Volumetric rendering allows to reconstruct the final image using sampled points along camera rays.
- All steps are end-to-end differentiable, allowing training via gradient descent.

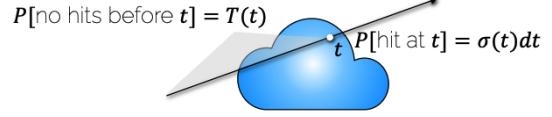


- Neural: Use a Neural Network as scene representation instead of explicit data structures.
- Volumetric: continuous, differentiable rendering model without concrete surface representation (similar to a cloud of particles).
- Rendering: compute color along rays through 3D space.



5.2 Probabilistic Interpretation

- Ray traveling through scene hits a particle at distance t , where color $c(t)$ returned.
- Probability of hitting a particle at small interval dt at distance t : $\sigma(t)dt$
- Probability of not hitting any particle before distance t (Transmittance): $T(t)$



The product of these probabilities tells us how much you see the particles at t :

$$\begin{aligned} P[\text{first hit at } t] &= P[\text{no hit before } t] \times P[\text{hit at } t] \\ &= T(t)\sigma(t)dt \end{aligned}$$

- $P[\text{no hit before } t + dt] = P[\text{no hit before } t] * P[\text{no hit at } t]$
- $T(t + dt) = T(t) * (1 - \sigma(t)dt)$

$$T(t + dt) = T(t)(1 - \sigma(t)dt)$$

Solve via Taylor Expansion:

$$\begin{aligned} T(t + dt) &\approx T(t) + T'(t) dt = T(t) - T(t)\sigma(t) dt \\ \frac{T'(t)}{T(t)} dt &= -\sigma(t) dt \\ \int_{t_0}^t \frac{T'(s)}{T(s)} ds &= - \int_{t_0}^t \sigma(s) ds \\ \log T(t) &= - \int_{t_0}^t \sigma(s) ds \\ T(t) &= \exp\left(- \int_{t_0}^t \sigma(s) ds\right) \end{aligned}$$

This gives $P[\text{first hit at } t]$:

$$p(t) = T(t) * \sigma(t)dt$$

To get the expected color returned by the ray, we need to integrate over all sampling points t_i along the ray:

$$C = \int_{t_0}^{t_n} T(t) \sigma(t) c(t) dt$$

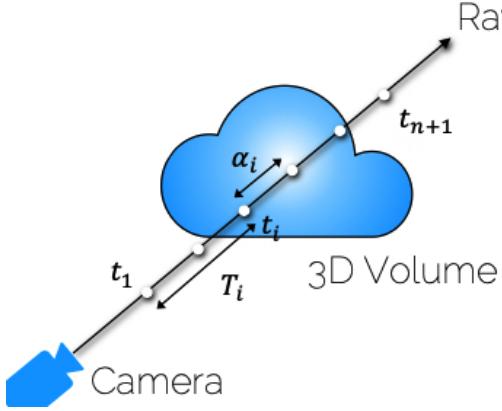
Note the nested integral, since $T(t)$ itself is an integral over density $\sigma(s)$.

5.3 Discrete Integration (Approximation)

Sampling at discrete points t_i along the ray:

$$C = \int_{t_0}^{t_n} T(t) \sigma(t) c(t) dt \approx \sum_{i=1}^N T_i \alpha_i c_i$$

This is differentiable w.r.t. c, σ . Where: α_i are the length of the integral at sample point t_i :



How much light is blocked earlier along the ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

How much light is contributed by ray segment i :

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$

Since the rendering weights T_i and α_i are dependent on the points AND viewing direction:

$$C \approx \sum_{i=1}^N T_i \alpha_i c_i$$

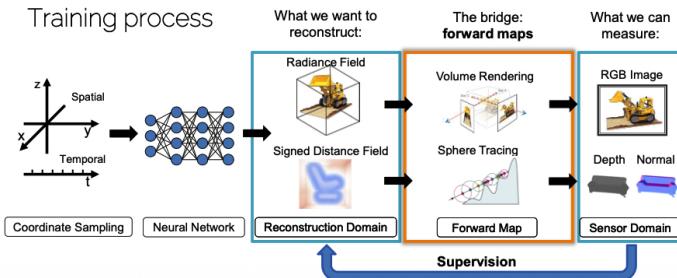
$$w_i = T_i \alpha_i$$

Rendering Weight PDF: The distribution of weights can be used to compute expected depth and uncertainty:

$$\bar{t} = \sum_{i=1}^N w_i t_i$$

where t_i is just the from the camera to the sample point i .

5.4 General Architecture



However, MLP are not required. There are works that use spares voxel grids and spherical harmonics.

5.5 Advantages

- Continuous, high-resolution scene representation.
- High-quality novel-view synthesis with realistic lighting and specularities.
- Compact representation: scene stored in MLP weights.

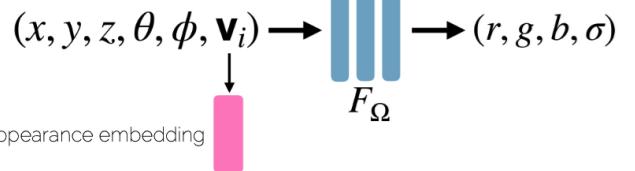
5.6 Limitations (Original NeRF)

- Slow training and rendering due to MLP evaluations on every ray sample.
- Challenges with unknown or inaccurate camera poses
- Dynamic scenes and dynamic lighting are complicated to model.



Appearance Embedding Serves as a robust solution to learn a latent representation of varying appearance conditions (lighting, weather, etc.)

Pretty robust solution:



N-dim vector optimized per image: "Auto-Decoding"
i.e. GLO: Generative Latent Optimization [Bojanowski et al.] ICML 2018]

5.7 NeRFs vs 3D Meshes

NeRFs are good for **Reconstruction**:

- Optimization-based
- Need good gradients
- Not just for surface representation but also for depth, normals

3D Meshes are good for **Rendering**:

- Fast rendering of surfaces (only)
- Efficient
- No gradients needed

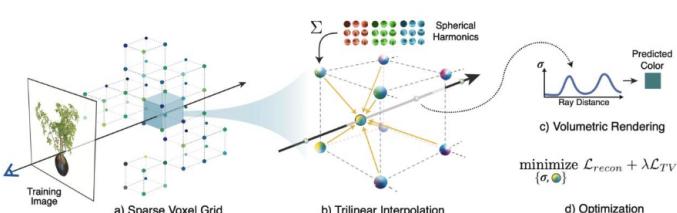
6 Gaussian Splatting

6.1 Point-Based Splatting Surfels

Belong to traditional point-based graphics methods.

Definition Surfels (surface elements) are 3D points that approximate local surface patches.

- position (x, y, z) ,
- a surface normal,
- a radius for disc size,



Yu et al. 'Plenoxels: Radiance Fields without Neural Networks' CVPR 2022

- color and optional confidence.

Rendering Principle:

1. Project each surfel center into the camera.
 2. Render its elliptical footprint.
 3. Blend footprints in depth order (front to back): Gaussian kernel.
 4. Moving the camera closer scales the discs accordingly.

Advantages:

- No mesh connectivity required.
 - Robust to noisy or incomplete geometry.
 - Projection step of each point into image can be parallelized.
 - (Almost) real-time rendering feasible.

Limitations:

- Limited handling of transparency.
 - Not volumetric; represents only surfaces.
 - Blending requires heuristics and is not fully differentiable.

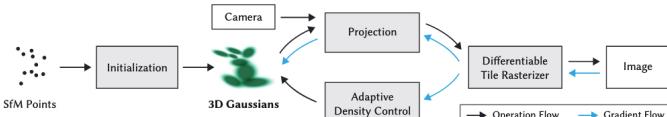
6.2 Gaussian Splatting

3D Gaussian Splatting represents a scene with anisotropic 3D Gaussian ellipsoids. Each Gaussian is defined by:

- mean (3D center),
 - covariance matrix (shape, scale, and orientation),
 - color (often spherical harmonics),
 - opacity.

Spherical Harmonics: Are a family of smooth, wave-like functions defined on the surface of a sphere.

Overview



- Initialization: sparse structure from motion points are converted into 3D Gaussians
 - Rendering Loop:
 - The camera defines the viewpoint for rendering
 - 3D Gaussians are projected into 2D image space
 - Differentiable Tile Rasterer:
 - Sort Gaussians by depth
 - Split screen into tiles for efficient rendering
 - Alpha-blending: Blend Gaussians front-to-back using their opacities
 - Fully differentiable: allows gradient-based optimization
 - Optimization:
 - There is no neural network involved, optimization is performed directly on Gaussian parameters.
 - Compare rendered image to ground truth: Photometric Loss
 - Multi-view constraint: use multiple images from different view points
 - Backpropagate errors to update Gaussian parameters (position, shape, color, opacity)

- To get a valid covariance matrix, optimize its decomposition
 $\Sigma = RSS^T R^T$
 - Adaptive Density Control:
 - Add new Gaussians in high-error regions
 - Remove Gaussians in low-importance areas
 - Split large Gaussians in high-variance regions
 - Clone Gaussians in underfilled regions

6.2.1 Rendering Principle

1. **Sort Gaussians:** Globally based on depth from camera.
 2. **Splat:** Compute the shapes of the Gaussian after projection into 2D.
 3. **Blend:** Perform front-to-back alpha composite.

Advantages:

- NeRF-level quality with real-time rendering.
 - Fully differentiable projection and blending.
 - Supports volumetric effects (soft edges, transparency).
 - Can be optimized end-to-end from images.

Limitations:

- No explicit surface mesh.
 - Memory usage can grow with dense scenes.
 - Topology changes are harder to capture.

6.3 Dynamic Scene Representation

Make Gaussians move over time: **Fixed Parameters:**

- Scale
 - Color
 - Opacity

These get optimized on the first frame and after that stay constant.

Time-Varying Parameters:

- 3D Position
 - 3D Rotation

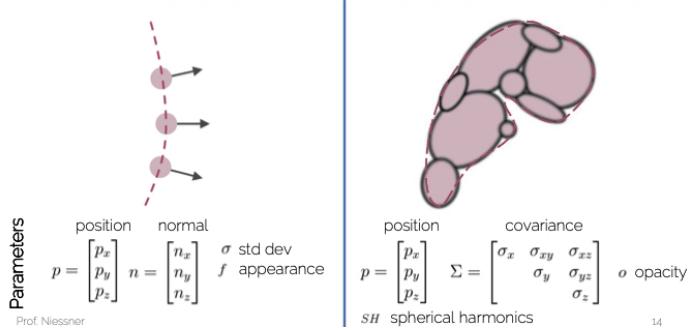
Optimize for each timestep relative to the previous.

6.4 Surfels vs Gaussians

Surfels: represent surface discs; **Gaussians**: represent smooth volumetric blobs.

- Surfels are 2D oriented patches.
 - Gaussians are 3D ellipsoids that naturally project to 2D.
 - Surfels require heuristic blending; Gaussians use volumetric accumulation.
 - Gaussians allow end-to-end differentiable optimization.

Surface Splatting vs. Volume Splatting



Surface Splatting:

$$c = \frac{\sum_{i \in \mathcal{N}} c_i w_i}{\sum_{i \in \mathcal{N}} w_i}$$

Weighted average of colors c_i with weights w_i . Simple normalization, no opacity.

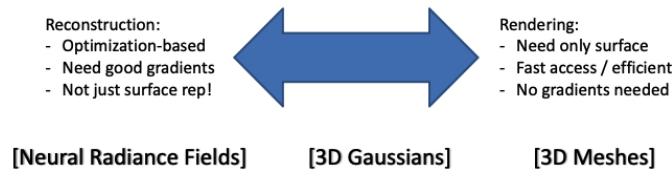
Volume Splatting (Alpha Compositing):

$$c = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j w_j)$$

- Front-to-back blending where α_i is opacity of splat i .
 - The product term $\prod_{j=1}^{i-1} (1 - \alpha_j w_j)$ is the transmittance - how much light passes through all previous splats.
- Similar to NeRF volume rendering equation.
- Each splat contributes its color c_i scaled by its opacity α_i and the accumulated transparency from splats in front.

6.5 Summary

Gaussian Splatting can be seen as a hybrid between Neural Radiance Fields and 3D Meshes



Fundamental Works

Surface Splatting, Zwicker, 2001
3D Gaussian Splatting for Real-Time Radiance Field Rendering, Kerbl, 2023

7 Generative Neural Networks

7.1 Taxonomy of Generative Models

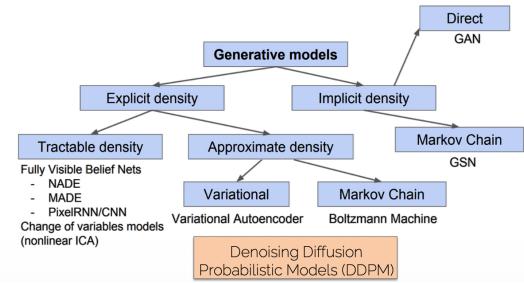
Definition Generative models are broadly categorized based on how they represent the probability density function.

Explicit Density Models that explicitly define the probability density function.

- **Tractable density:** Fully Visible Belief Nets (NADE, MADE, PixelRNN/CNN) and Change of variables models (nonlinear ICA).
- **Approximate density:** Includes Variational models (VAE) and Markov Chain models (Boltzmann Machine).

Implicit Density Models that do not explicitly define the density function.

- **Markov Chain:** Generative Stochastic Network (GSN) learns transition operator.
- **Direct:** Generative Adversarial Networks (GAN).



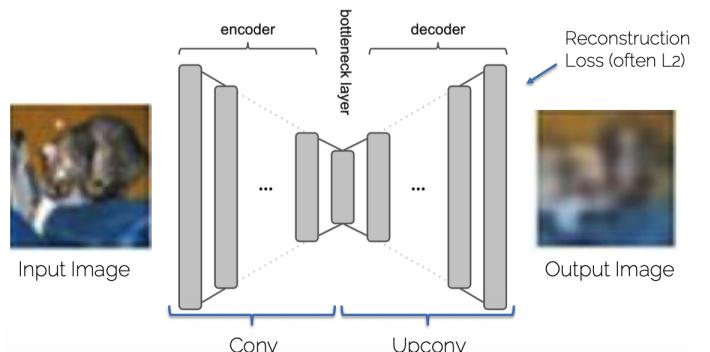
7.2 Autoencoders and Variational Autoencoders

Autoencoders Consist of an encoder, a bottleneck layer, and a decoder.

- Encoder maps input x into latent space z , where $\dim(z) < \dim(x)$.
- Decoder reconstructs output from the latent vector.

Reconstruction Loss Typically minimizes L2 (sum of squared distances).

- **Limitation:** L2 distributes error equally, leading to an optimal mean.
- **Result:** Often produces a **blurry output**.



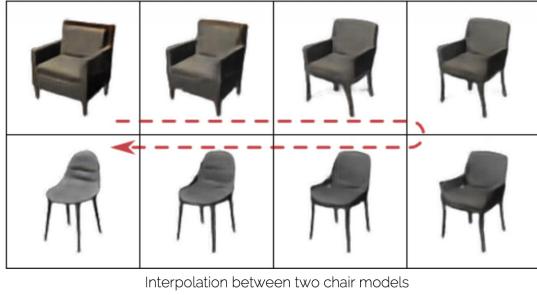
Variational Autoencoders (VAEs) A probabilistic extension where the decoder functions as a generative model by reconstructing from a random vector sampled from z .

Loss Function The VAE loss combines reconstruction quality and regularization:

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \underbrace{\mathbb{E}_z[\log p_\theta(x^{(i)}|z)]}_{\text{Reconstruct the Input Data}} - \underbrace{D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\text{KL Divergence}}$$

Decoder as Generative Model

- At test time: reconstruct a new output by sampling z from random prior $p_\theta(z)$
- This also allows interpolation in latent space between two encoded inputs.

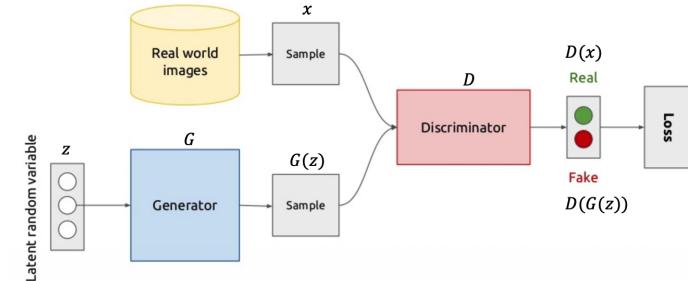


7.3 Generative Adversarial Networks (GANs)

- Instead of applying L2 loss, GANs use a learned loss function via a discriminator network.
- At test time: sample a latent random variable z and obtain a generated sample $G(z)$.

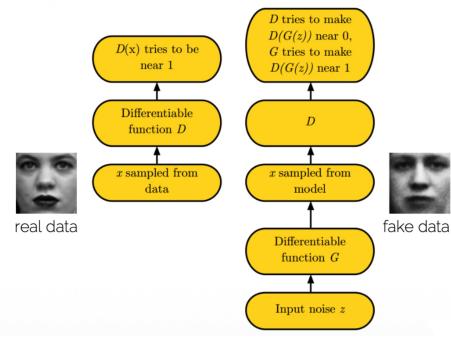
Overview GANs rely on two competing networks:

- Generator (G):** Generates fake data $G(z)$ from a latent random variable z .
- Discriminator (D):** Distinguishes real images (x) from generated images.



Minimax Game Training is a game where G attempts to minimize the probability that D is correct.

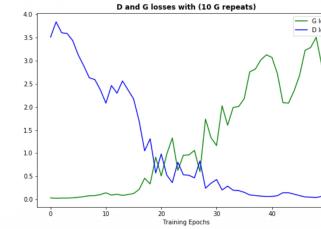
$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$



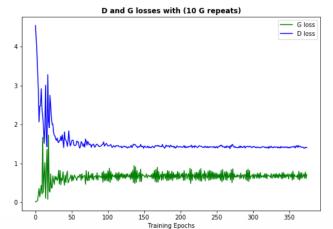
Training Principle

- Alternating Updates:** Fix G while training D; fix D while training G.
- Heuristic Method:** Instead of standard minimization, G maximizes $\log D(G(z))$.
- This prevents vanishing gradients when D is very strong (rejecting all samples).

"Bad" Training Curves



"Good" Training Curves



7.4 Training Challenges and Stabilization

Challenges

- Balance:** D and G must learn at similar rates; if D is too weak, G gets poor gradients. If D is too strong, D will always be right and G is unable to learn.
- Mode Collapse:** Generator produces only a small subset of the training distribution.
- Structure:** Generated images often fail global structure checks (e.g., counting limbs).

Stabilization Techniques (Hacks)

- Normalization:** Inputs between -1 and 1; Tanh on generator output.
- Sampling:** Sample z from Gaussian; interpolate via **great circle**.
- Optimizers:** ADAM for Generator, SGD for Discriminator.
- Label Smoothing:** Use one-sided smoothing (target < 1) to reduce D's confidence.
- Sparse Gradients:** Avoid by using LeakyReLU in both G and D.
- Weight Averaging:** Exponential averaging of weights for D.

7.5 Wasserstein GAN (WGAN)

Definition Reformulates GAN loss using the **Earth Mover Distance (EMD)**, or Wasserstein Distance.

1-Lipschitz Constraint The critic function f must change slowly and satisfy 1-Lipschitz continuity:

$$f(x_1) - f(x_2) \leq |x_1 - x_2|$$

- WGAN attempts to enforce this by restricting maximum weight values (weight clipping).

Advantages:

- Mitigates mode collapse.
- Generator learns even when the critic is optimal.
- Provides a meaningful convergence metric.

Limitations:

- Enforcing Lipschitz is difficult.
- Weight clipping can lead to slow training or vanishing gradients.

7.6 Evaluation of Generative Models

Inception Score (IS) Measures two components using a pre-trained classifier:

1. **Saliency**: Can images be classified with high confidence?
2. **Diversity**: Are samples obtained across all classes?

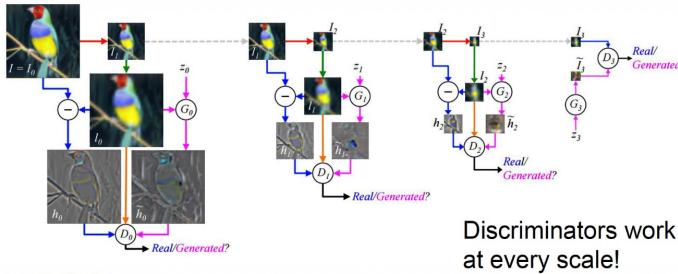
Frechet Inception Distance (FID) Calculates feature distance between real and synthetic data distributions (modeled as multivariate Gaussians).

- More robust to noise than IS.
- Does not require class labels.

7.7 Advanced Architectures

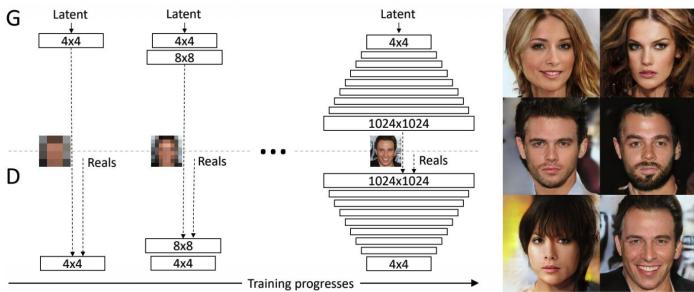
Multiscale GAN

- Generates images progressively from low to high resolution.
- Each scale has its own generator and discriminator.



Progressive Growing GAN

- Starts training at low resolution and progressively adds layers to increase resolution.
- Stabilizes training and improves quality.
- This architecture is also used in StyleGAN.



8 Conditional Generative Adversarial Networks

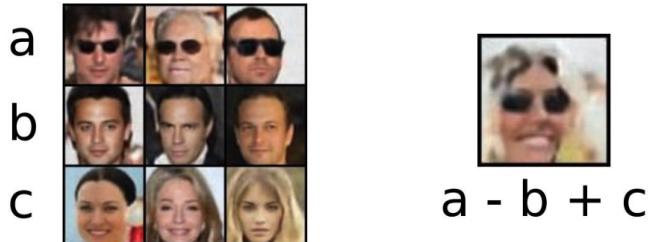
- Extend GANs by conditioning both the generator and discriminator on additional information y .
- Instead of learning a mapping $z \rightarrow x$, cGANs learn a mapping $(x_{\text{cond}}, z) \rightarrow y$, which allows controlled generation, semantic manipulation, and domain transfer.

Motivation

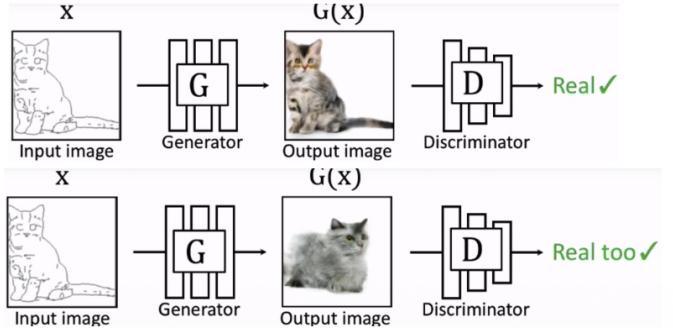
- Have control over the output space.
- Make the latent manifold semantically meaningful.
- Enable tasks such as sketch-based modeling or image-to-image translation.

8.1 Manipulating the Latent Manifold

- GAN Manifolds are very smoothing
- Well suited for latent space interpolation and manipulation of feature space

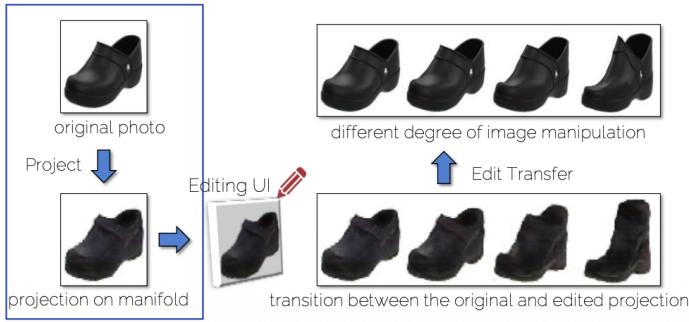


- It is not enough to just condition the generator on additional information y .
- The discriminator must also be conditioned on y to ensure that the generated output corresponds to the conditioning input.

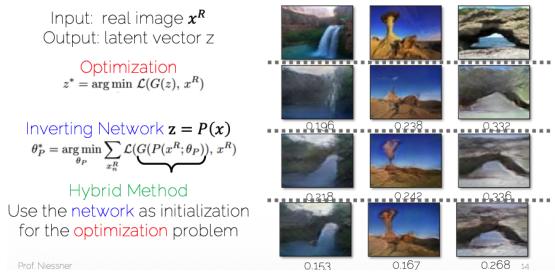


iGANs Interactive GANs aim to project real images onto a GAN's latent manifold for editing.

- Projection: optimize z so that $G(z)$ reconstructs a real image.
- Editing: modify the latent vector using user guidance constraints.
- Transfer: interpolate between latent codes and transfer geometric/color edits.



- Projecting a real image onto a GAN's latent space is a highly non-convex optimization problem
- Without a good starting point the optimization will almost always get stuck in a poor local minimum.
- Use an Inverting Network P that learns to map images to latent codes.
- $G(P(x^R, \theta_P))$ is like an Autoencoder with a fixed decoder G
- This ensures faster convergence and better reconstruction quality.



iGAN Editing

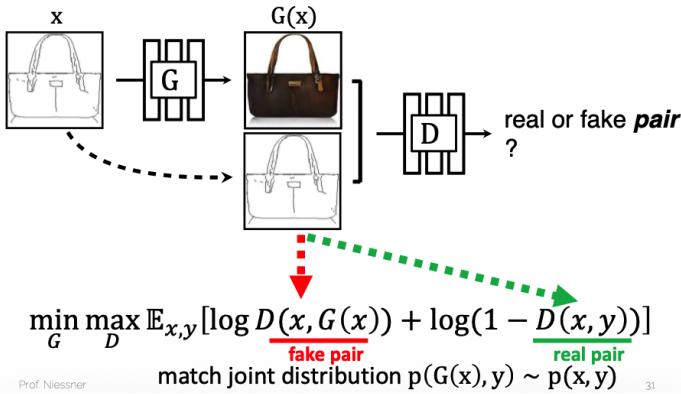
- iGAN edits v_g images by optimizing the latent vector.
- Given an initial code z_0 , user edits define a guidance loss \mathcal{L}_g that measures how the generated image $G(z)$ should change.
- The latent vector is updated by solving:

$$z^* = \arg \min_z \mathcal{L}_g(G(z)) + \lambda \|z - z_0\|^2$$

- Regularization term is used to ensure that the latent code does not deviate too much from the original code z_0 .

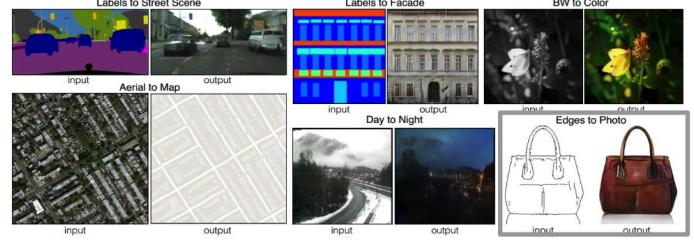
8.2 Paired Conditional GANs: Pix2Pix

- Generation becomes easier since more guidance is provided to G .
- Discriminating becomes harder since D has to learn a joint distribution of input and output images.



Paired Supervision Pix2Pix uses aligned image pairs (x, y) and trains a cGAN to model $x \rightarrow y$.

pix2pix: Image-to-Image Translation



Objective

$$\mathcal{L} = \mathcal{L}_{\text{GAN}} + \lambda \mathcal{L}_1$$

- GAN loss enforces realism and sharp local structure.
- ℓ_1 loss preserves global content and low frequencies.

Architecture

- **UNet generator** with skip connections for structural preservation.
- **PatchGAN discriminator**: operates on local patches for high-frequency detail.
- Noise injected only through dropout; network often ignores explicit z .

8.3 High-Resolution cGANs: Pix2PixHD

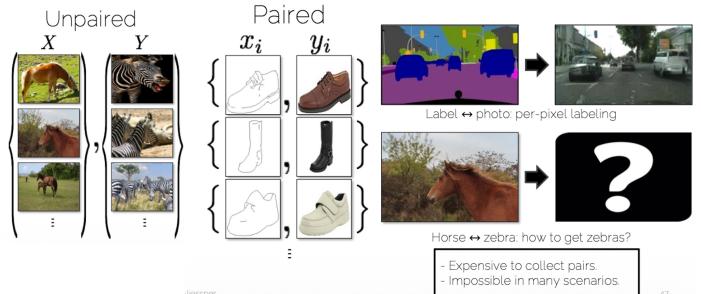
Key Ideas

- Multi-scale discriminators for strong supervision across resolutions.
- Coarse-to-fine generators to increase realism.
- Improved stability and output resolution.

8.4 Unpaired Conditional GANs: CycleGAN

Motivation

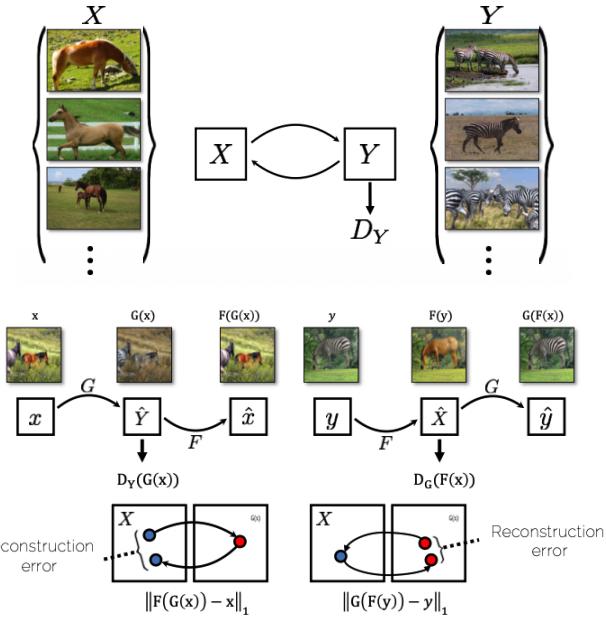
- Paired datasets are often unavailable or expensive to collect.
- CycleGAN enables image-to-image translation without paired data by enforcing cycle consistency.



Cycle Consistency CycleGAN introduces a cycle loss:

$$F(G(x)) \approx x, \quad G(F(y)) \approx y$$

- Ensures that input and output correspond meaningfully.
- Prevents mode collapse in unpaired translation.



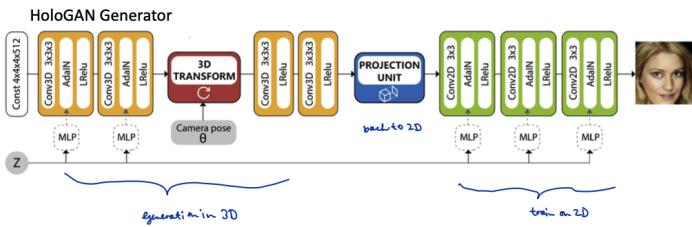
Architecture

- Two generators $G : X \rightarrow Y$ and $F : Y \rightarrow X$.
- Two discriminators operating in both domains.

8.5 3D-Aware GAN Extensions

HoloGAN

- Introduces 3D representation inside the generator.
- Enforces view consistency.
- Camera parameters act on the latent code before rendering.
- Projection unit transforms 3D features to 2D image space.
- Inside projection unit is a Volumetric Renderer (NeRF-like).
- This makes training very slow, every forward pass requires rendering the full 3D volume.



8.6 GRAF, Pi-GAN, EG3D

Neural radiance fields and volumetric rendering integrate with GAN frameworks:

- GRAF: radiance-field generator with patch-based discriminator.
- Pi-GAN: progressive training for higher resolution NeRF-based synthesis.
- EG3D: efficient 3D-aware GAN with tri-plane representation for fast rendering.
- GGHead: replaces NeRF with 3D Gaussian Splatting for real-time 3D-aware generation.

8.7 Summary

cGANs provide controllability and structure for generation tasks, enabling:

- Latent-space manipulation (iGANs)
- Paired image translation (Pix2Pix)
- Unpaired translation (CycleGAN)
- 3D-aware generation.
- Evolution started with
 - learned 3D features and projection units (HoloGAN, 2019)
 - followed by NeRF-style radiance fields (GRAF and Pi-GAN, 2021),
 - then using more efficient data structures like tri-plane representations (EG3D, 2022)
 - finally faster rendering methods like 3D Gaussian Splatting (GGHead, 2024)