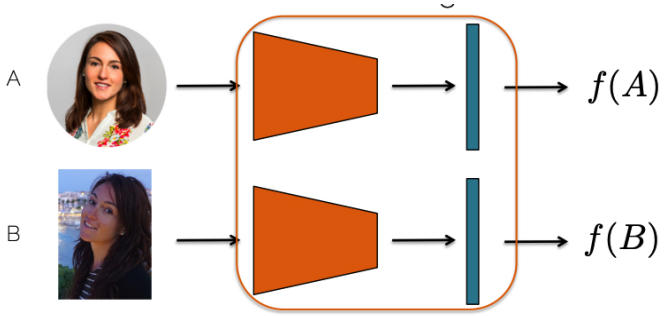
Advanced Deep Learning for Computer Vision

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

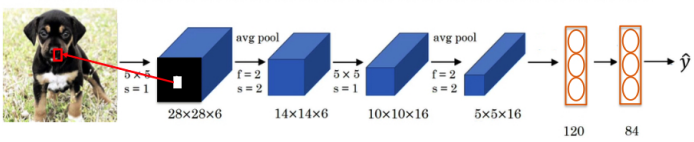
Distance function: $d(A, B) = \|f(A) - f(B)\|_2^2$

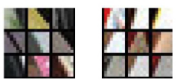


1.2 Visualization of ConvNets

Visualization in the Image Space

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



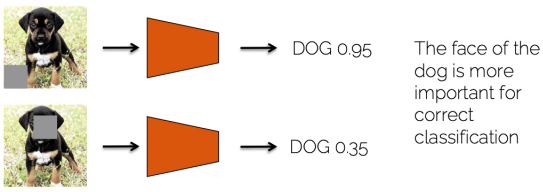


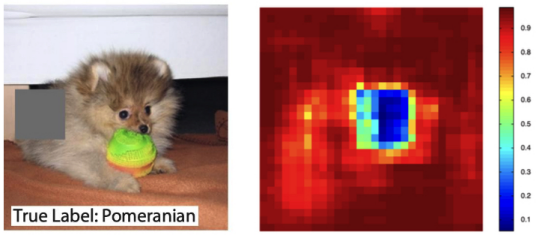
Feature map 2, layer 1, 9 image patches that provided the highest activation

Zeiler 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)

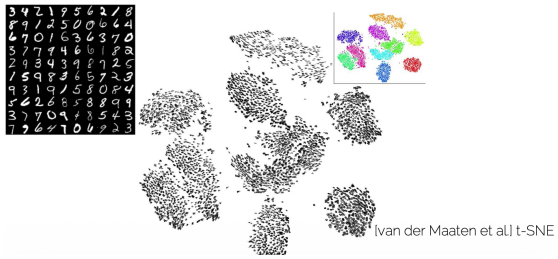




1.3 Visualization of Clusters

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 PCA or SVD capture global structure, t-SNE focuses on preserving local structure (similarity between nearby points)
- Perplexity parameter controls the balance between local and global aspects of the data
- Not useful for new data points (non-parametric)



[van der Maaten et al.] t-SNE

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)

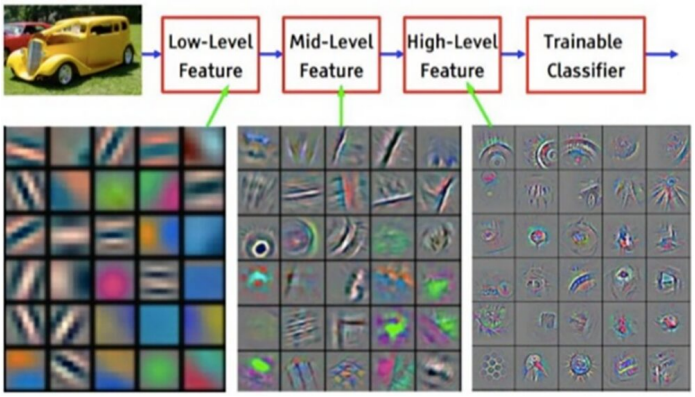
- Probabilistic extension of autoencoders
- Encoder outputs parameters of a probability distribution (mean and variance) instead of a single point in latent space
- Latent space is continuous, allowing for smooth interpolation and

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)

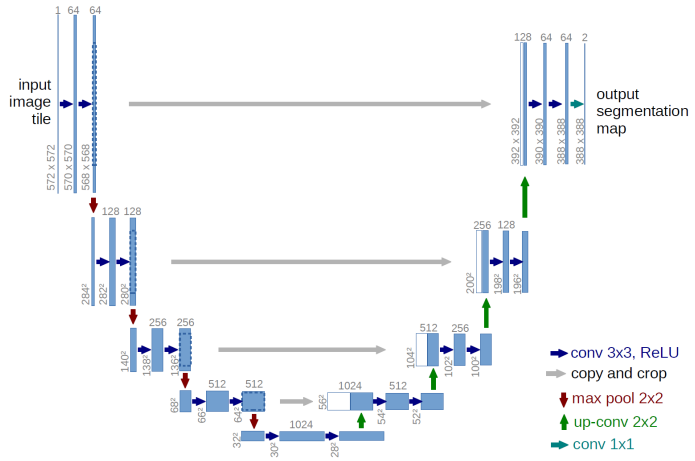
Gradient Ascent for Visualizing CNN Filters

- Initialize with a random or zero image
- Keep weights frozen and change the image via Gradient Ascent to visualize learned patterns of a CNN filter
- Goal: Adapt the input image to maximize the activation of a specific filter, revealing the features that the filter responds to.
- To visualize a specific CNN filter at a certain layer, isolate its output and treat it as objective function.



U-Net (Auto-Encoder)

- **Encoder**: Acts as the Feature Extractor
- **Decoder**: Acts as a Projector to a specific task, such as reconstruction
- **Bottleneck**: Passes high-level features through the deepest part of the network
- **Skip-connections**: Passes low-level features to maintain fine details
- **Gradient Flow**: Skip-connections solve the issue of vanishing gradients in deep NNs, leading to faster and more stable training



2 3D Data Representation

- Explicit Representations: Voxel grids, Point Clouds, Polygonal Meshes
- Implicit Representations: Signed Distance Fields (SDFs), Occupancy Fields, Neural Radiance Fields (NeRFs)

2.1 Volumetric Grids

- Discretize 3D space into regular grid of voxels
- Each voxel can store data such as occupancy, color, or distance, density

Volumetric Data Structures:

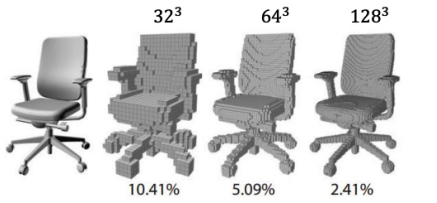
- Occupancy Fields: voxel is occupied or free (binary)
- (Signed) Distance Fields (SDFs): Store distance to nearest surface, with sign indicating inside/outside.
  Gradients lead to surface.
- Density Fields: Store density values for volume rendering

Applications:

- 3D Object Classification and Segmentation
- 3D Reconstruction from Images
- 3D Completion

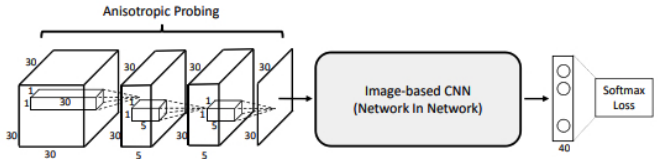
Summary:

+ 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)



Surface occupancy gets smaller with higher resolutions

Extension of 2D CNNs to 3D CNNs by using 3D convolutional kernels: Object Classification with 3DCNNs, 2016



2.2 Volumetric Hierarchies

Goal: High resolution close to surface, low resolution far away.

Octrees

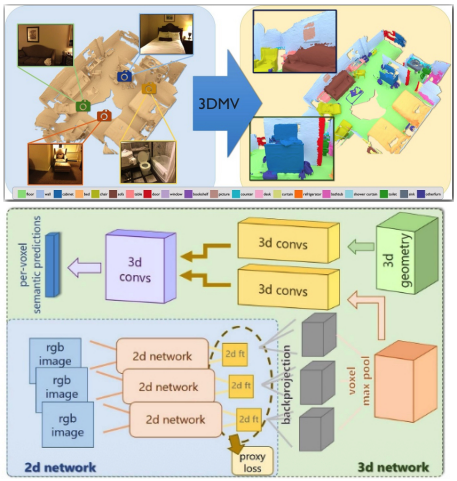
- Hierarchical data structure that recursively subdivides 3D space into eight octants (like a quadtree in 2D).
- 3D partitioning: higher resolution at surfaces, lower resolution in empty space for applying 3DCNN kernels
- Can be used for discriminative tasks (like object classification, segmentation)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



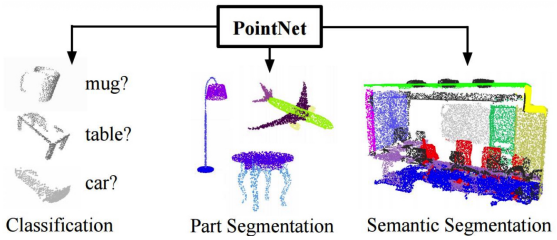
3DMV: 3D Voxel + Multi View Semantic Segmentation, 2016

Summary:

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

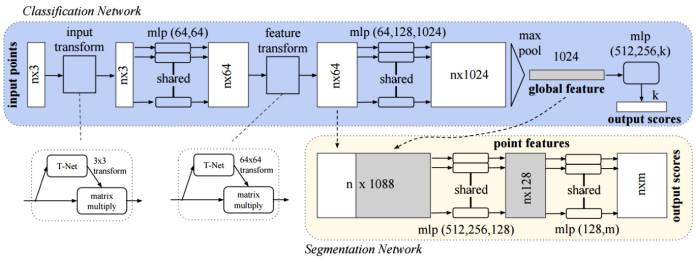
2.4 Point Clouds

- Properties: irregular, sparse, unordered
- 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
- use a T-Net to learn a spatial transformation to align the input points to a canonical space
- project aligned points to a higher-dimensional feature space
- 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
- No explicit positional encoding needed as there is no natural orders

2.5 Sparse Convolutional Networks

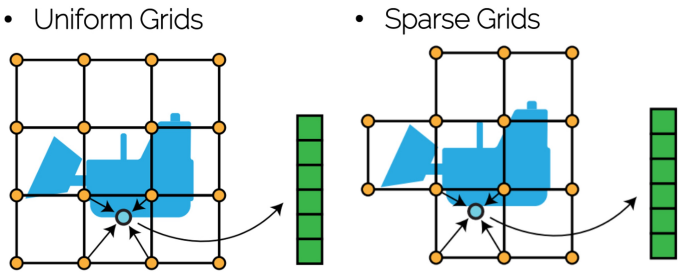
- Goal: Efficiently process sparse 3D data (e.g., point clouds, voxels with many empty spaces)
- Method:
  - Apply convolutional operations only on non-empty voxels
  - Make use of sparse hashes instead of masking conv output
  - Libraries: QSParseConv, MinkowskiEngine
- Latency on GPU for hash lookup is hidden by parallelism

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



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

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: field representation encoded 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.

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

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

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:

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

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

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

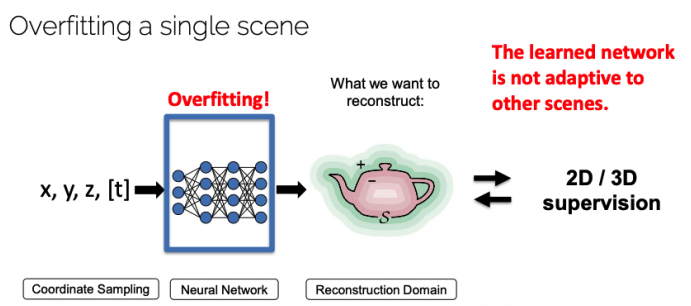
Key Difference

- **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.

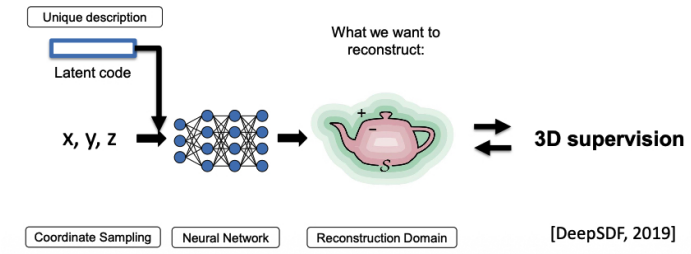
Overfitting Single Scene



- Overfitting a single scene 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

- Standard MLPs have a bias towards learning low-frequency signals (blurry).
- Also raw coordinates are biased towards absolute positions.

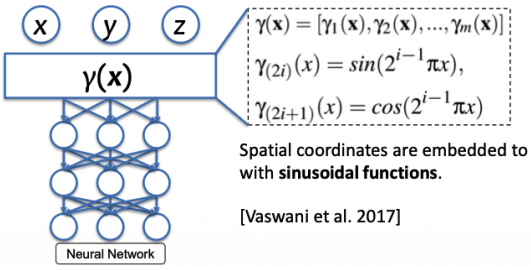
Shift invariance is important to prevent bias in activations based on absolute coordinate values.

- Positional encoding: Sinusoidal high-dimensional mapping of coordinates to keep the distance between coordinates meaningful and preserve fine details
- SIREN Activations: sinusoidal activations to represent high-frequency functions

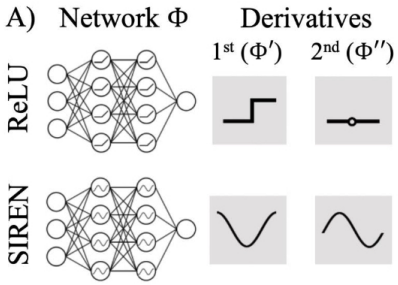
On the Spectral Bias of Neural Networks, 2018

Positional Encoding:
Sinusoidal / learned positional encodings allow:

+ smooth distance representation,
+ easy extrapolation,
+ relative comparison (difference of phases encodes distance).



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



Remark:

- Combining positional encoding and sinusoidal activations is not necessary
- It can create a complex interference pattern that can make optimization extremely noisy

4.5 Summary

1. Overfit an MLP
2. Positional Encoding: prevent inductive bias from coordinates
3. Activation Functions: non-linearities to represent high-frequency functions

---

Comparison of Network Evolution

- 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 network-based representation for 3D scenes that encodes both geometry and appearance.
- Enables photo-realistic novel view synthesis from a sparse set of 2D input images.

Neural Radiance Fields (NeRF), Mildenhall et al., ECCV 2020

Core Idea

- NeRF represents a scene as a continuous probability function:

$$F_\theta : (x, y, z, \theta, \phi) \rightarrow (\sigma, \mathbf{c})$$

- **5D Input**:
  - (x, y, z): 3D position
  - (θ, ϕ): 2D camera viewing direction
- **Outputs**:
  - $\mathbf{c}$: emitted color at sample point perceived from direction
  - σ: volume density (chance of absorption at specific location)

Method

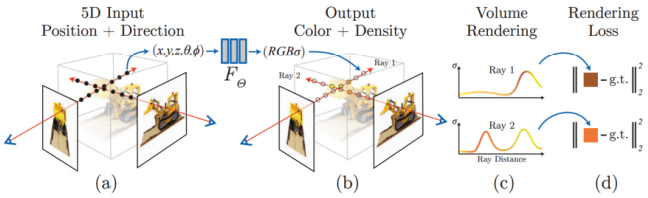
- **Ray Casting**: To render a pixel, cast a ray from the camera into the scene.
- **Neural Rendering:** Sample points along ray and query MLP for color and density at certain point.
- **Volume Rendering:** Blend colors and densities together of the sampled points to compute the final pixel color.
- All steps are end-to-end differentiable, allowing training via gradient descent.
- **Downside:** Rendering is slow due to many MLP evaluations per ray.

Inference:

- For full-HD image: $1920 \times 1080 = 2$ Mio. pixels
- $N = 128$ samples per ray $\Rightarrow$ 256 Mio. MLP evaluations per image (full MLP forward pass)

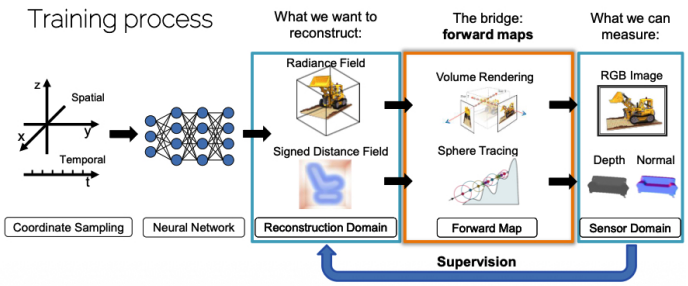
Training:

- Use positional encoding to map input coordinates [x, y, z] to a higher-dimensional space.
  Important to resolve spectral biases (smoothing of neighboring pixels) and focus on details.
- Optimization involves also image rendering + additional loss computation and backpropagation
- Pixel-wise MSE Loss: $\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left\| \hat{C}(p_i) - C(p_i) \right\|_2^2$



5.1 Radiance Fields and Depth/Normals

- Leverage NeRF for volumetric rendering of RGB image
- Also possible to extract depth and normal maps from the radiance field:
  - Depth: Compute expected depth along ray using rendering weights as a probability distribution.
  - Normals: Compute spatial gradients of the density field σ to get surface normals.

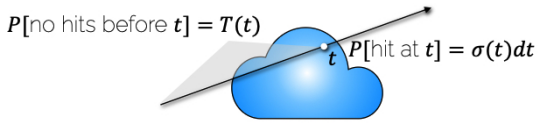


Naming: Neural Volumetric Rendering

- **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:
P[first hit at t] = P[*no hit before* t] × P[hit at t]
$= T(t)\sigma(t)dt$

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

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

Solve via Taylor Expansion:

$$T(t+dt) \approx T(t) + T'(t)\,dt = T(t) - T(t)\sigma(t)\,dt$$

$$\frac{T'(t)}{T(t)}\,dt = -\sigma(t)\,dt$$

$$\int_{t_0}^{t} \frac{T'(s)}{T(s)}\,ds = -\int_{t_0}^{t} \sigma(s)\,ds$$

$$\log T(t) = -\int_{t_0}^{t} \sigma(s)\,ds$$

$$T(t) = \exp\Big(-\int_{t_0}^{t} \sigma(s)\,ds\Big)$$

This gives P[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:

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

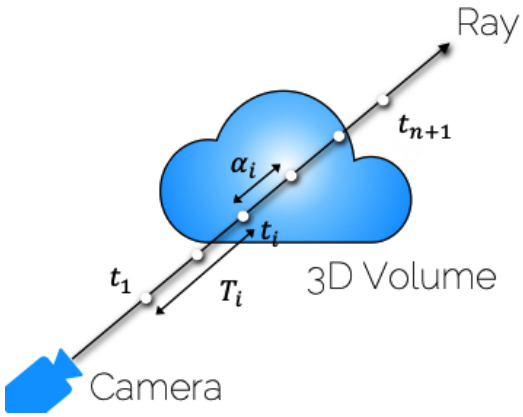
Note the nested integral, since $T(t)$ itself is an integral over density $\sigma(s)$.

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:

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

$$w_i = T_i\,\alpha_i$$

Computation of Expected Depth: The distribution of weights can be used to compute expected depth and uncertainty:

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

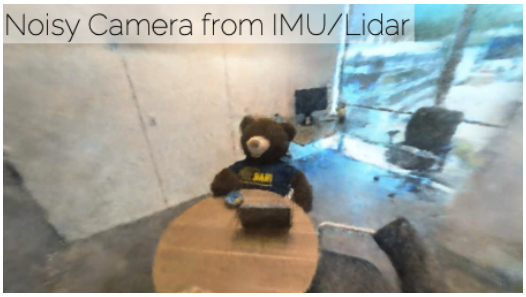
where t_i is the distance from the camera to the sample point i.

5.4 Advantages

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

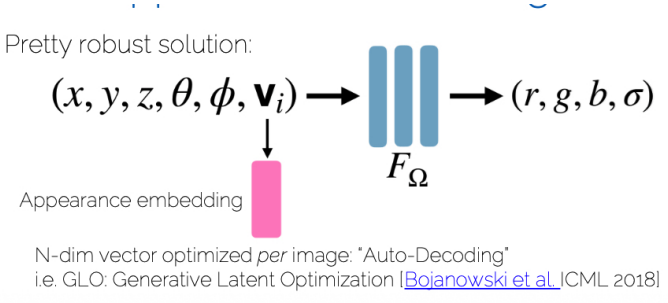
5.5 Limitations (Original NeRF)

- 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, moving pedestrains).

- Instead of forcing MLP to learn a blurry average of all appearances, learn a latent code that captures the variations in appearance across different images.
- Latent code is just a unique and random low-dimensional latent vector for each image stored in a look-up table, which is optimized during training.
- During training, optimize both the MLP parameters and the latent codes for each image, allowing the model to disentangle geometry from appearance variations.
- This allows also interpolation between latent codes to generate scenes with novel appearances



5.6 NeRFs vs 3D Meshes

NeRFs are good for **Reconstruction**:

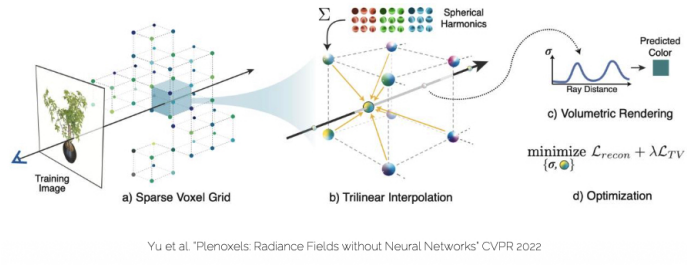
- Optimization-based
- Not just for surface representation but also for depth, normals
- photo-realistic rendering with view-dependent effects (specularities, reflections)

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

- Triangles can be rendered very efficiently on GPU (parallelization)
- Barycentric interpolation for colors
- 2D imag can also be warped around triangles
- No gradients needed

5.7 Plenoxels for NeRF-Quality without MLP

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



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

- Instead of using a MLP to represent the scene, use a sparse voxel grid where each voxel stores color and density information.
- Use spherical harmonics to represent view-dependent effects (like specular highlights) within each voxel.
- This allows for fast rendering while still achieving NeRF-level quality.
- However, a lot of memory is used by the voxel grid for storing empty space
- Gaussian Splatting took this idea to get rid of MLP but swapped rigid grid by flexible Gaussians.

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,
- 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.

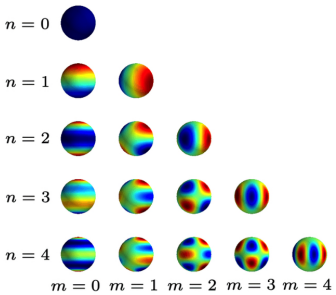
6.2 Gaussian Splatting

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

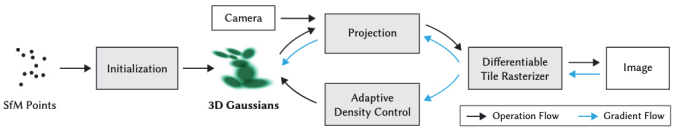
- mean (3D center),
- covariance matrix (shape, scale, and orientation),
- color (often represented by spherical harmonics),
- opacity.

Spherical Harmonics:

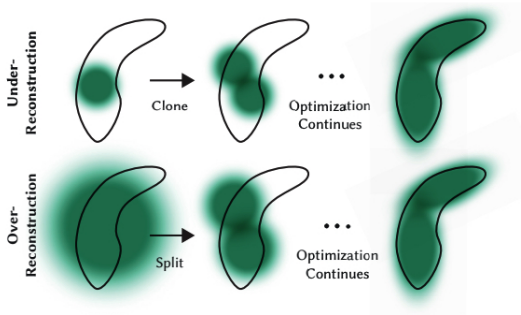
- Are a family of smooth, wave-like functions defined on the surface of a sphere.
- Used to capture view-dependent color variations.
- Instead of storing a single RGB color, each Gaussian stores coefficients for spherical harmonics.



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 (16×16 tiles)
    This GPU to process tiles in sharded-memory instead of reading all Gaussians from global VRAM.
  - 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 viewpoints
  - Backpropagate errors to update Gaussian parameters (position, shape, color, opacity)
  - To get a valid covariance matrix, optimize its decomposition $\Sigma = RSS^TR^T$
- **Adaptive Density Control**:
  - Clone Gaussians in under-reconstructed areas:
    Gaussian is small and has high gradient (error): copy gaussians and move them slightly into the gradient direction
  - Split large Gaussians in high-detail areas:
    Gaussian is large and has high gradient: split Gaussian into two smaller ones to capture details
  - Pruning: Remove Gaussians to prevent floater artifacts and reduce memory usage:
    * Remove Gaussians with opacity below a certain threshold (contribute little to final image)
    * Remove Gaussians that are too small (below a certain radius)
    * Remove Gaussians that are too large (above a certain radius)
    * Reset Step every $N = 3000$ iterations to prevent increase in number of Gaussians



6.2.1 Rendering Principle

1. **Projection:** Each 3D Gaussian is projected into the 2D image plane, resulting in an elliptical footprint.
2. **Sort Gaussians**: Globally based on depth from camera.
3. **Splat**: Compute the shapes of the Gaussian.
4. **Alpha Blending**: Perform front-to-back alpha compositing.
   For every pixel, compute the color by walking through the Gaussians from front to back.
   This is simliar to NeRF volume rendering, but instead of sampling along rays, we only consider discretized Gaussians.

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:

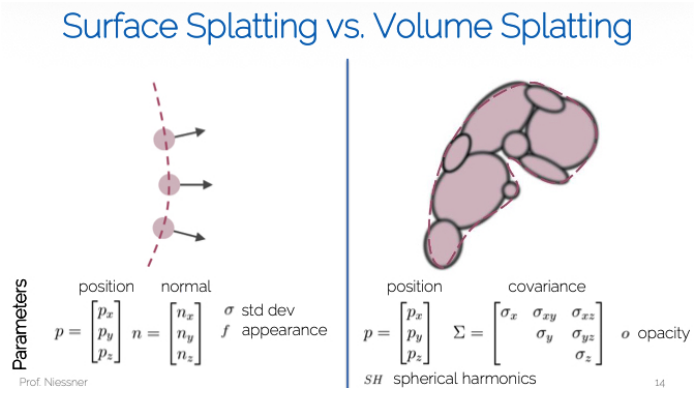
- 3D Position
- 3D Rotation

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:

- Primitives represent a solid surface without connected triangle-mesh.
- Overlapping surfaces are sorted and blended in depth order

Volume Splatting (Alpha Compositing):

- Primitives represent fuzzy cloud where light can pass through
- Overlapping clouds are blended front-to-back using their opacities (alpha compositing)

6.5 Summary

- Gaussian Splatting can be seen as a hybrid between Neural Radiance Fields and 3D Meshes
- Utilizes physics of NeRF for transmittance and blending
- Utilizes some of the efficiency of meshes for rendering by rastering, sorting and tiling

Fundamental Works

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

7 Generative Neural Networks

Goal of Generative Modeling

- Maximize the likelihood of training data $\theta^* = \arg\max_\theta p(x|\theta)$
- However, integrating over all possible hidden states (e.g. noise paths in diffusion) is intractable for some models.
- Solve this by optimizing a variational lower bound or evidence lower bound (ELBO)

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 $p(x|\theta)$.

- **Direct**: Autoregressive Models (PixelRNN, PixelCNN).
- **Approximation** : Variational Autoencoders (VAEs).

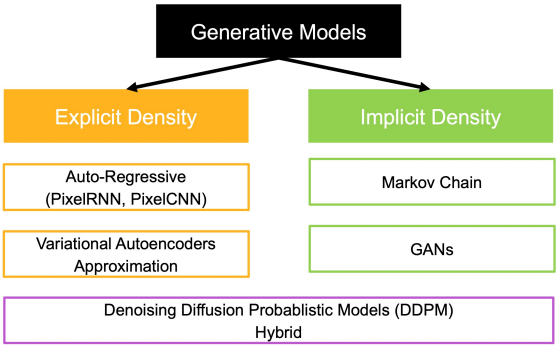
Implicit Density

Models that do not explicitly define the density function. A function that output samples that are close to the training data without modelling the underlying distribution.

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

Hybrid: DDPM

- Explicitly: optimization of a variational lower bound to approximate training data density.
- Implicity: iteratively refining noise without requiring direct calculation of single density formula.



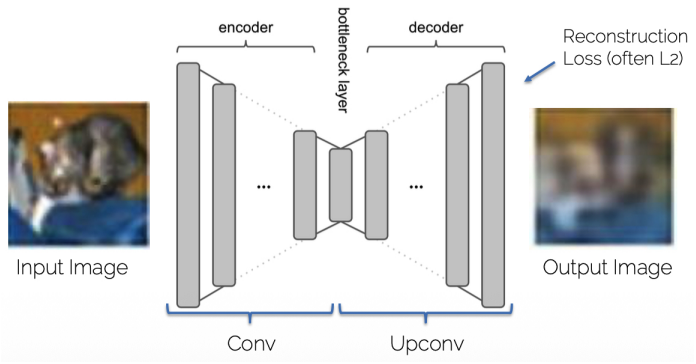
7.2 Autoencoders and Variational Autoencoders

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.

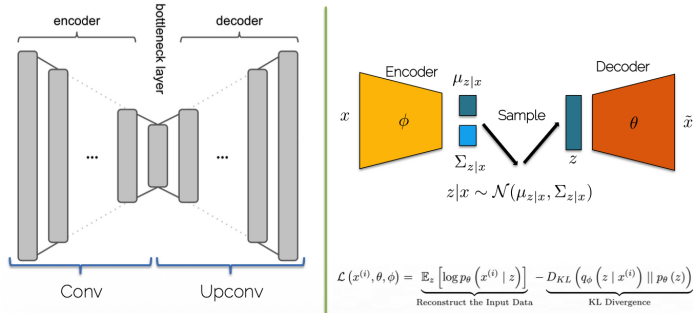


Variational Autoencoders (VAEs) A probabilistic extension where the decoder functions as a generative model by reconstructing from a random vector sampled from z.

Loss Function The VAE loss combines reconstruction quality and regularization:

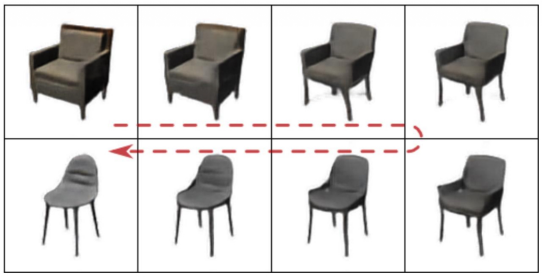
$$\mathcal{L}(x^{(i)}, \theta, \phi) = \underbrace{\mathbb{E}_z[\log p_\theta(x^{(i)}|z)]}_{\text{Reconstruct the Input Data}} - \underbrace{D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\text{KL Divergence}}$$

- **KL Divergence:** forcing the learned latent distribution $q_\phi(z|x)$ to look like a standard normal distribution $p(z)$. This ensures the latent space is smooth and continuous, rather than just "memorizing" inputs
- Note: ϕ: are the parameters of the encoder network



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.



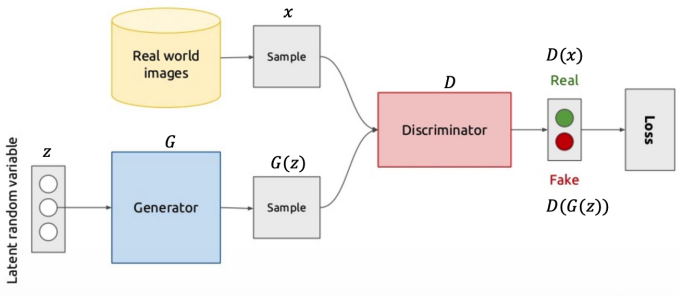
Interpolation between two chair models

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:

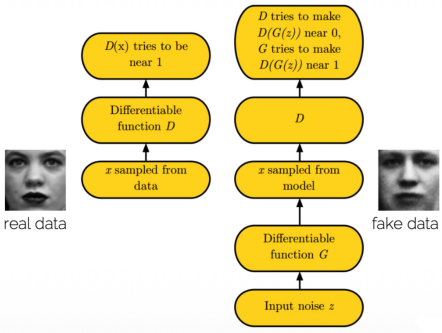
- **Generator (G)**: Generates fake data $G(z)$ from a latent random variable z.
- **Discriminator (D)**: Distinguishes real images (x) from generated images.



1. Discriminator says fake
2. Compute gradients through Discriminator (but freeze its values)
3. Backpropagate through generator to change weights

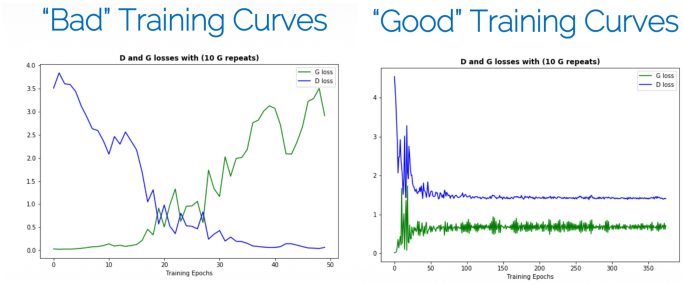
Minimax Game Training is a game where G attempts to minimize the probability that D is correct.

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



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).



7.4 Training Challenges and Stabilization

Challenges

- **Balance**: D and G must learn at similar rates
  - If D is too strong: G receives no gradient signal (vanishing gradients).
  - If G is too strong: D fails to learn, providing no useful feedback.
  - We can not use a simple classifier for D, since G would quickly learn to fool D without producing meaningful results.
- **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.
- Use Critic instead of a Discriminator, which provides a linear gradient almost everywhere, even when distributions do not overlap.
- Even when the Discriminator is optimal, the generator receives meaningful gradients to improve.

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).
- Wasserstein distance is based on the "cost" of moving probability mass.
- Therby Generator is forced to cover the entire target distribution rather than finding a single point of failure in the Discriminator.

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?

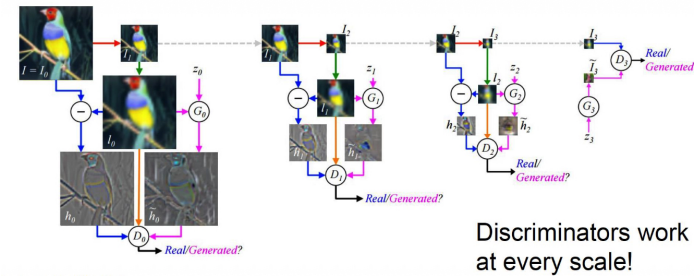
Frechet Inception Distance (FID) Calculates feature distance between real and synthetic data distributions (modeled as multivariate Gaussians).

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

7.7 Advanced Architectures

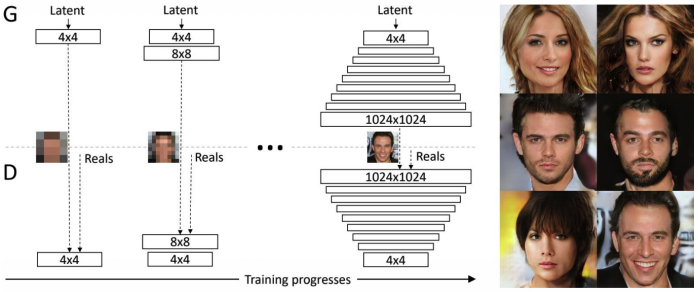
Multiscale GAN

- Generates images progressively from low to high resolution.
- Each scale has its own generator and discriminator.
- Generator is conditioned on the upsampled version of the previous lower-resolution scale



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.



Summary Besides GANs, other generative models are now more popular:

- Diffusion Models
- Autoregressive Models (like Vision Transformers or LLMs)

8 Conditional Generative Adversarial Networks

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

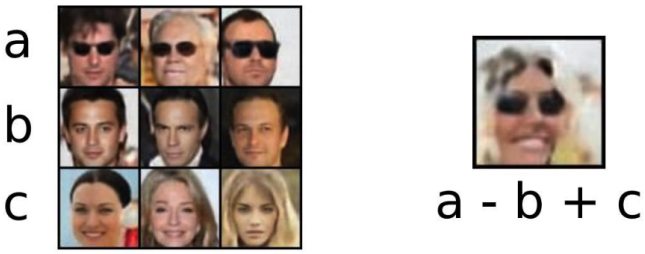
Motivation

- Have control over the output space.
- Make the latent manifold semantically meaningful.

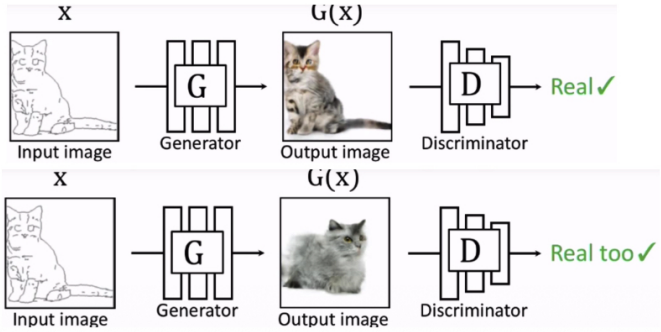
- Enable tasks such as sketch-based modeling or image-to-image translation.

8.1 Manipulating the Latent Manifold

- GAN Manifold are very smooth
- Well suited for latent space interpolation and manipulation of feature space

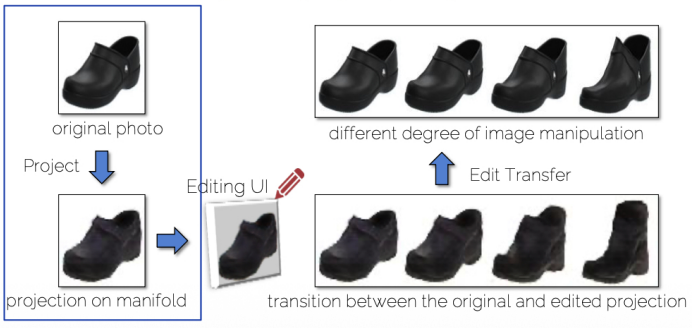


- It is not enough to just condition the generator on additional information y.
- The discriminator must also be conditioned on y to ensure that the generated output corresponds to the conditioning input.

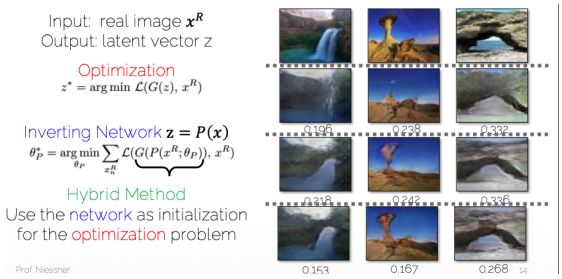


iGANs Interactive GANs aim to project real images onto a GAN's latent manifold for editing.

- 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

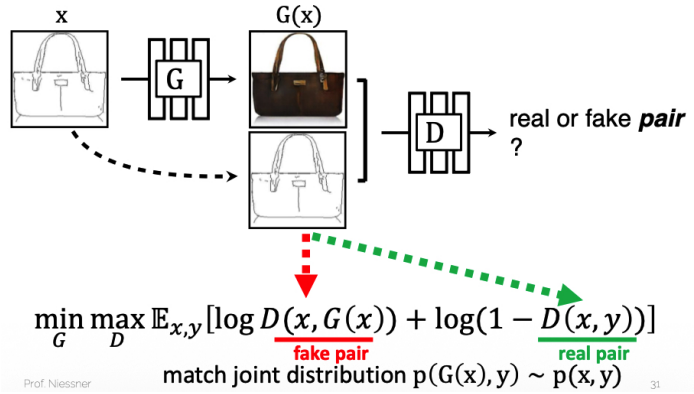
- iGAN edits v_g images by optimizing the latent vector.
- Given an initial code z_0, user edits define a guidance loss $\mathcal{L}_g$ that measures how the generated image $G(z)$ should change.
- The latent vector is updated by solving:

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

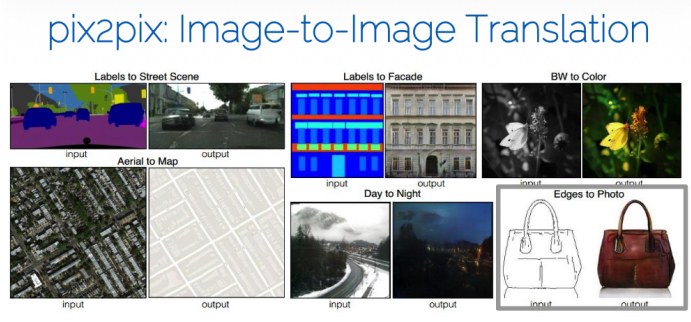
- Regularization term is used to ensure that the latent code does not deviate too much from the original code z_0.

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 $x \to y$.



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

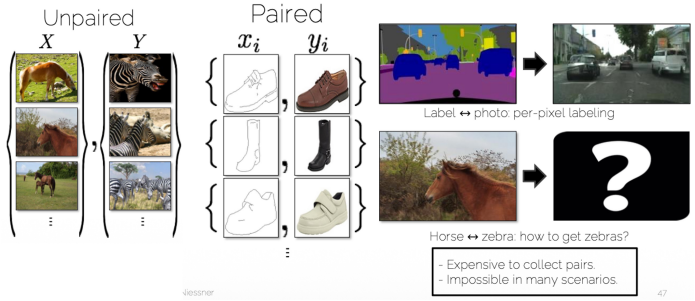
Key Ideas

- 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

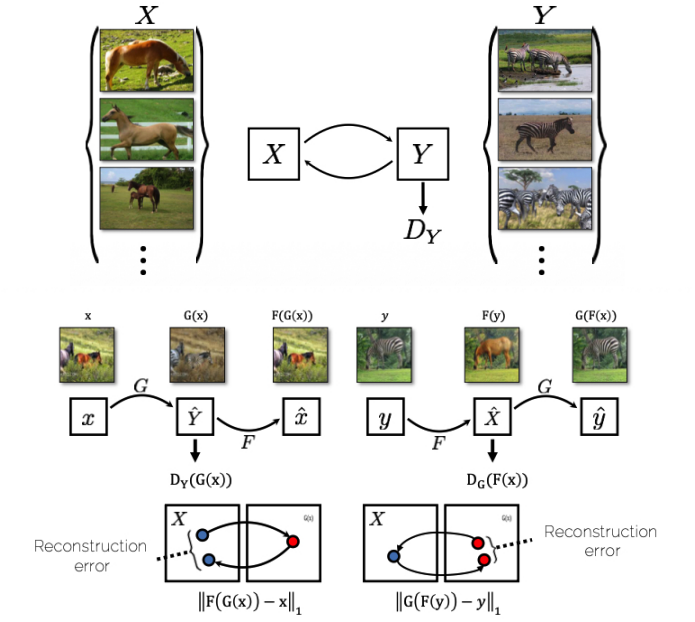
- Paired datasets are often unavailable or expensive to collect.
- CycleGAN enables image-to-image translation without paired data by enforcing cycle consistency.



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.



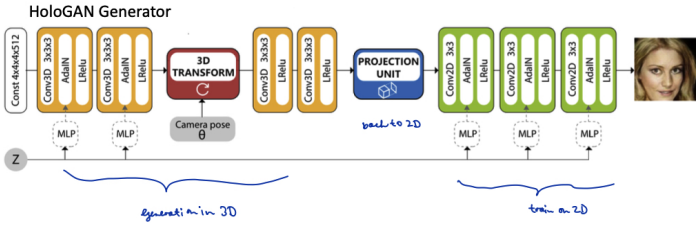
Architecture

- Two generators $G : X \to Y$ and $F : Y \to X$.
- 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.

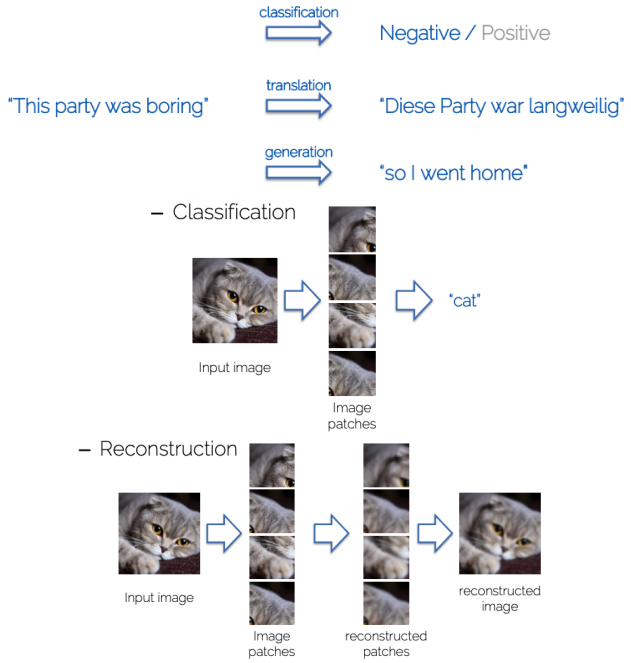
8.7 Summary

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)

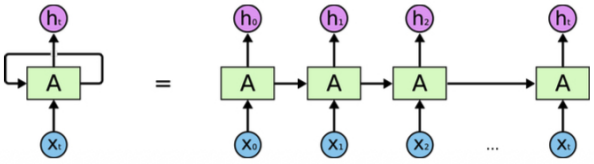
9 Sequence Modelling

- **Text as Sequences:** Sequences are natural representations for text data, utilized for classification (sentiment analysis), translation, and generation.
- **Images as Sequences:** Images can also be treated as sequences by breaking them into patches for classification or reconstruction tasks.



9.1 Recurrent Neural Networks (RNNs)

- RNNs process data sequentially and maintain a hidden state h_t that acts as memory.
- They can be "unrolled" in time, sharing weights A across time steps.
- **Limitation:** While good at short sequences, RNNs suffer from the **long-term dependency issue** (vanishing/exploding gradients), making it difficult to relate distinct information in long sequences.
- When training RNN over multiple time steps, the gradient must be back propagated through each time step.
-



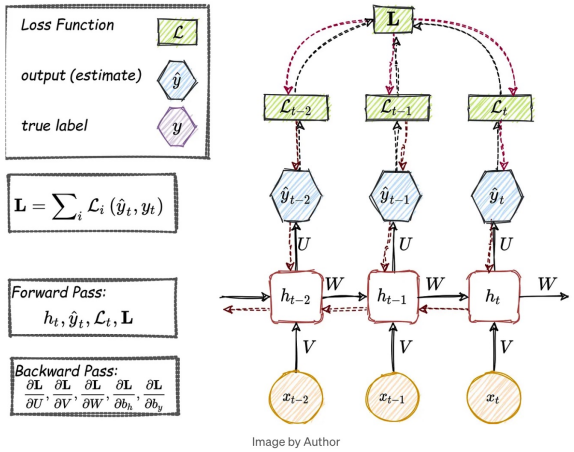


Image by Author

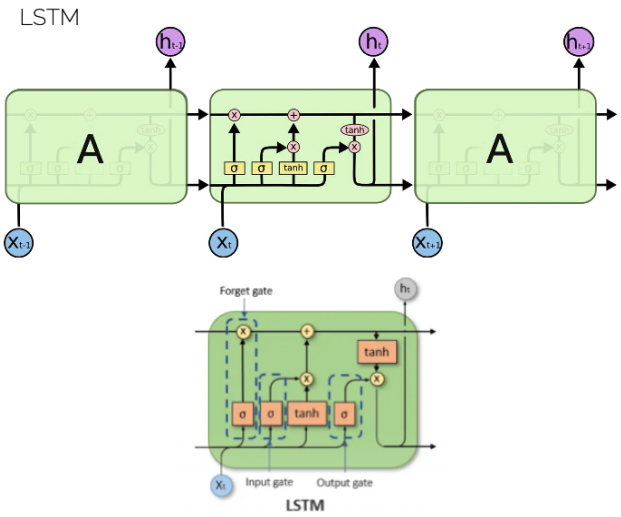
Back Propagation Through Time $\frac{\partial \mathbf{L}}{\partial W} \propto \sum_{i=0}^{T} \left(\prod_{i=k+1}^{y} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$

Vanishing Gradient: $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$

Exploding Gradient: $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$

9.2 Long Short-Term Memory (LSTMs)

- Designed to alleviate the long-term dependency issue.
- Utilizes gating mechanisms (sigmoid σ and tanh activations) to regulate information flow (forgetting, updating, and outputting).
- However, for extremely long sequences (e.g., full documents), performance still degrades.



9.3 Attention Mechanism

Motivation

- "Not all words are born equal." In a sequence, certain words are more important for predicting specific outputs (e.g., "France" is crucial for predicting "French").
- Attention allows the model to focus on relevant parts of the input signal regardless of distance.

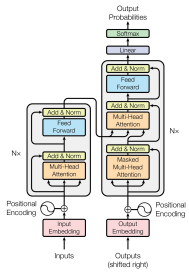
Types of Attention

- **Soft Attention:** Deterministic and differentiable; attends to the entire input with weights summing to 1.
  Standard attention used in almost all modern "Transformer" models (GPT, BERT, ViT, etc.).
- **Hard Attention:** Stochastic and non-differentiable; Model must select a single part of the input (requires Monte Carlo estimation).
  Used when the model is forced to should just focus on one part (e.g., image captioning).

- **Self-Attention:** The signal attends to itself to learn internal dependencies.
- **Cross-Attention:** Attends to another signal (e.g., an image) as side information.

9.4 The Transformer

- Introduced in "Attention is All You Need" (Vaswani et al., 2017).
- Replaces recurrence with self-attention mechanisms, allowing for parallel processing of sequences.
- Consists of an encoder-decoder architecture, where both the encoder and decoder are stacks of identical layers.
- The output are probabilities over the vocabulary for each position in the output sequence.



9.5 Scaled Dot-Product Attention

The core mechanism acts as a differentiable database retrieval system involving **Queries (Q)**, **Keys (K)**, and **Values (V)**.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3}$$

Input Projection (for a single token vector x)

$$q = x \cdot W_Q \tag{4}$$
$$k = x \cdot W_K \tag{5}$$
$$v = x \cdot W_V \tag{6}$$

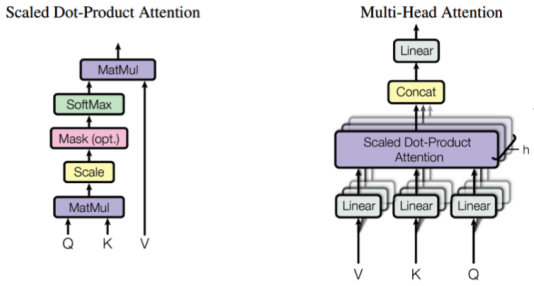
Input Projection (Matrix form for sequence length L)

$$Q = X \cdot W_Q \tag{7}$$
$$K = X \cdot W_K \tag{8}$$
$$V = X \cdot W_V \tag{9}$$

- **Dot Product QK^T:** Computes similarity between queries and keys.
- **Scaling $\frac{1}{\sqrt{d_k}}$:** Prevents the dot products from growing too large, which would push the softmax into regions with small gradients.
- **Softmax:** Induces a probability distribution (soft indexing) over values.
- $x \in \mathbb{R}^{1 \times d_{model}}$: This is the embedding vector for one token. Its size (d_{model}) is a hyperparameter (e.g., 512 in the original Transformer)
- $W_Q, W_K \in \mathbb{R}^{d_{model} \times d_k}$: These are the learnable weight matrices for Queries and Keys.
- $W_V \in \mathbb{R}^{d_{model} \times d_v}$: This is the weight matrix for Values.
- **Context window:** Q, K, V all inherit that same L dimension (e.g., $Q \in \mathbb{R}^{L \times d_k}$)
  Attention is $O(L^2)$ since the final matrix has a size of $L \times L$ when multiplying Q ($L \times d_k$) by K^T ($d_k \times L$)

9.6 Multi-Head Attention

- Instead of a single attention function, the model projects queries, keys, and values h times with different linear projections.
- Allows the model to attend to different parts of the input signal simultaneously (e.g., capturing both syntactic and semantic relationships).
- The outputs are concatenated and linearly projected.



9.7 Architecture Components

- **Encoder-Decoder Structure:** The original architecture consists of an encoder stack and a decoder stack.
- **Positional Encoding:** Since there is no recurrence, fixed trigonometric embeddings (sine/cosine) are added to input embeddings to provide sequence order information.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{10}$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{11}$$

- **Masking:** Used in the decoder (self-attention) to prevent attending to future tokens (causal modeling).

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

9.8 BERT

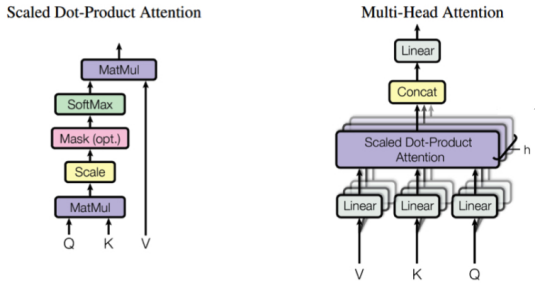
Bidirectional Encoder Representations from Transformers

- A large transformer encoder designed for representation learning.
- **Input Representation:** Sum of Token Embeddings, Segment Embeddings (sentence A vs B), and Position Embeddings.
- **Pre-training Objectives:** Self-supervised learning using two tasks:
  1. Masked Language Modelling (MLM): Randomly mask words

(e.g., 15%) and predict them using bidirectional context. No human annotation required.

2. Next Sentence Prediction (NSP): Predict if sentence B follows sentence A (binary classification).

- Position Embedding: Indicate position of each token in the sequence.
- Segment Embedding: Differentiate between sentences in tasks involving sentence pairs.
- Token Embedding: Standard word embeddings (e.g., WordPiece).



9.9 GPT (Generative Pre-trained Transformer)

Important Elements:

- Tokenization/dictionary
- Sequence Model (transformer)
- Training Scheme (multimodal, cross entropy loss, context window)

Dictionary

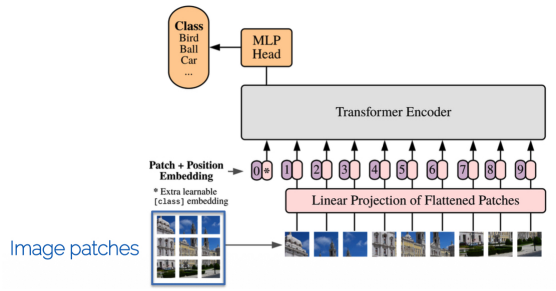
- Two different extreme cases for tokenization:
  - Character-level: small vocabulary (e.g. 26 tokens or ASCII = 256 tokens) $\Rightarrow$ Context problem of attention.
  - Word-level: vocabulary explodes but sentences consist of few tokens $\Rightarrow$ Out-of-vocabulary problem.
- Uses Byte Pair Encoding (BPE) to create a subword vocabulary.
- Balances vocabulary size.

Context Window

- Models long-range dependencies using a fixed-size context window (e.g., 1024 tokens).
- Attention is computed only within this window.
- Longer sequences are processed by sliding the window.

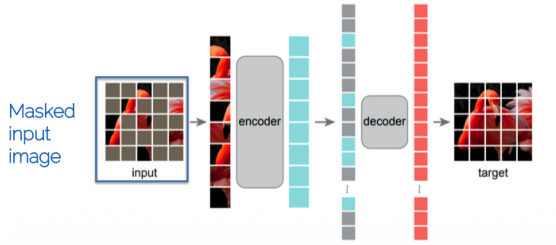
9.10 Vision Transformers (ViT)

- Replaces CNNs with a pure transformer architecture for image classification.
- **Patching:** Images are split into fixed-size patches (e.g., 16×16), flattened, and linearly projected to form a sequence of vectors.
- **Positional Embeddings:** Added to patch embeddings to retain spatial information.
- **Class Token:** A learnable '[class]' token is prepended; its output state serves as the image representation for classification (similar to BERT's '[CLS]').
- **Interpretability:** Attention maps visualize which image parts are relevant for prediction.



9.11 Masked Auto-Encoder (MAE)

- Self-supervised learner acting as an auto-encoder for ViTs.
- **Asymmetric Design:**
  - **Encoder:** Processes *only* the visible (unmasked) patches. Since a large ratio (e.g., 75%) is masked, this is computationally efficient.
  - **Decoder:** Reconstructs the full image from latent representation and mask tokens.
- **Loss:** MSE (L_2) loss computed only on masked patches.



9.12 Vision-Language Transformers (ViLT)

- Simplified architecture without region supervision or deep convolutional encoders.
- **Input:** Concatenation of image patches and text embeddings.
- **Objectives:** Image Text Matching (ITM) and Masked Language Modeling (MLM).

9.13 Detection Transformer (DETR)

- End-to-end object detection treating detection as a set prediction problem.
- **Backbone + Transformer:** CNN extracts features, flattened and fed into Transformer Encoder-Decoder.
- **Object Queries:** Learnable positional embeddings that the decoder transforms into bounding box predictions.
- **Bipartite Matching:** Uses the Hungarian algorithm to assign predictions to ground truth one-to-one, eliminating the need for Non-Maximum Suppression (NMS).

9.14 Summary

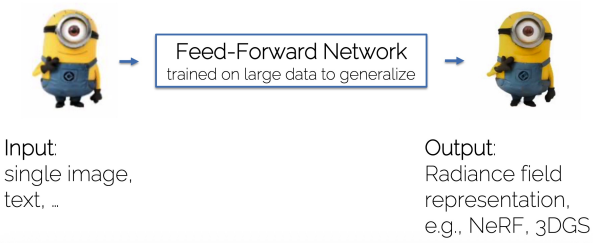
Transformers have revolutionized sequential modeling, moving from RNNs/LSTMs to Attention-based architectures.

- **NLP:** Dominance of BERT (Encoder) and GPT (Decoder) models.
- **Vision:** ViTs treat images as patch sequences, performing comparably to CNNs with sufficient data/pre-training.
- **Multimodal:** Unified architectures (ViLT) and new paradigms for detection (DETR) streamline cross-modal learning and structural prediction.

10 Large Reconstruction Models (LRM)

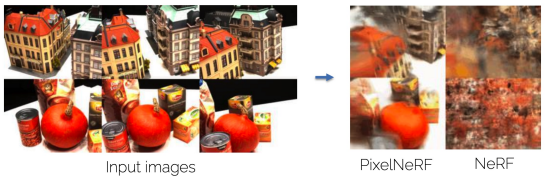
Reconstruction vs. Generation

- **Reconstruction:** Focuses on creating accurate 3D representations from specific inputs
  - Image-to-3D
  - Text-to-3D
  - Video-to-3D
  - Multi-view-to-3D
- **Generation:** Involves creating new 3D content from learned distributions
- **Shift to Feed-Forward Networks:** Unlike traditional reconstruction which often requires per-scene optimization, Large Reconstruction Models (LRMs) use Feed-Forward Networks trained on large datasets to generalize to new inputs.
- **Input and Output:** These models accept inputs like single images, text, or sparse views and output 3D representations such as Neural Radiance Fields (NeRF) or 3D Gaussian Splats.



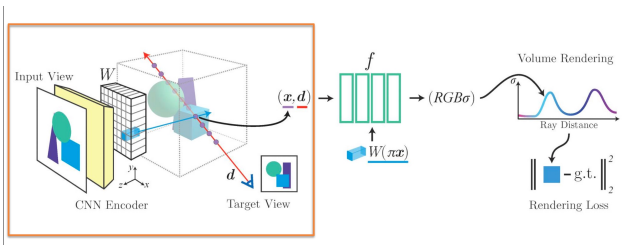
10.1 Generalizable Implicit Fields (pixelNeRF)

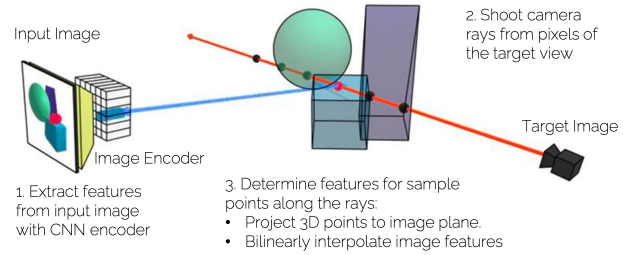
Generalizes across scenes without per-scene optimization by learning a scene-prior directly from images.

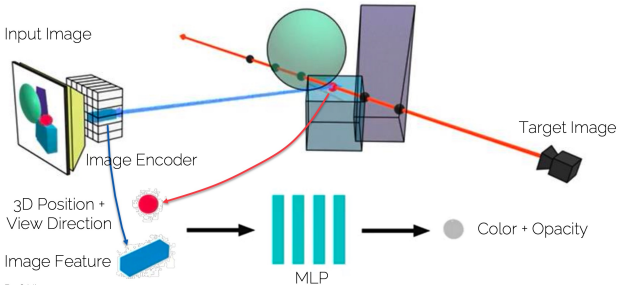


Mechanism:

- **Feature Extraction**: Uses a CNN encoder to extract features from one or few input images.
- **Feature Mapping**: Extracts a feature vector for any 3D point by projecting it onto the input image planes.
- **Interpolation**: Image features are bilinearly interpolated to determine the precise features at the projected coordinates.







Supervision:

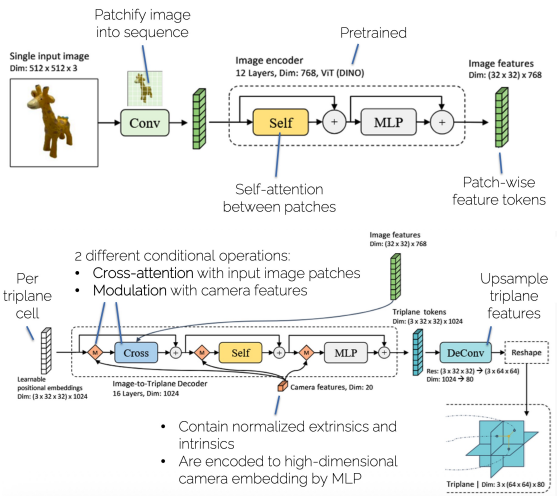
1. Shoot camera rays from pixels of the target view.
2. Sample points along these rays to determine spatial features.
3. Predict color and opacity (RGB, σ) using an MLP based on the 3D position, view direction, and extracted image features.
4. Trained using standard volume rendering and rendering loss against ground-truth images.

10.2 Scaling with Transformers (LRM)

A high-capacity architecture designed for fast 3D reconstruction from a single image using transformer-based components.

Architecture:

- **Image Encoder**: Utilizes a pretrained DINO ViT (Vision Transformer) with 12 layers and a 768-dimension hidden state.
- **Patchification**: The input image is "patchified" into a sequence of feature tokens with self-attention between patches.
- **Decoder**: A 16-layer Image-to-Triplane Decoder with a 1024-dimension processing space.



Conditioning Operations:

- **Cross-Attention**: With input image patches to control fine-grained geometric and color information.
- **Modulation**: With camera features (normalized extrinsics/intrinsics) to control the orientation and distortion of the

reconstructed shape.

Output Representation:

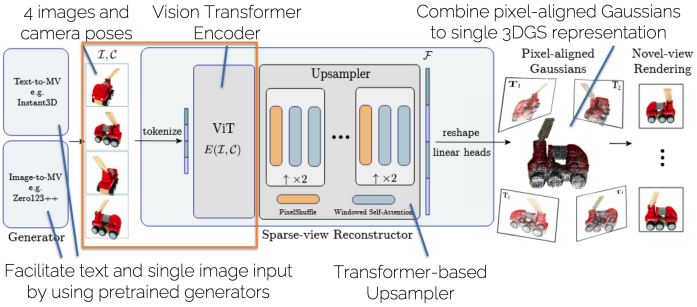
- **Triplane NeRF**: Outputs three axis-aligned feature planes with dimensions $3 \times (64 \times 64) \times 80$.
- **Point Querying**: To extract features for a 3D point, the point is projected onto each plane, and features are trilinearly interpolated.
- **MLP Mapping**: A 10-layer MLP maps interpolated point features to color and density for volumetric rendering.

10.3 Real-Time Gaussian Reconstructors (GRM)

Introduces efficiency by replacing volumetric querying with pixel-aligned 3D Gaussian Splatting (3DGS).

Primitive:

- Predicts one 3D Gaussian per pixel, creating an attribute map.
- Attributes include depth (1), rotation (4), scale (3), opacity (1), and RGB (3) for a total of 12 dimensions.



Mechanism:

- **Encoder**: Processes input images and camera poses (encoded as Plücker rays) using a ViT encoder.
- **Upsampler**: A Transformer-based upsampler progressively reaches the original $H \times W$ resolution.
- **PixelShuffle**: Spatial dimensions are doubled while quadrupling channels to increase resolution efficiently.
- **Computation**: Employs Windowed Self-Attention to balance global information aggregation with feasible computation costs.

Performance:

- Capable of reconstruction from input images in approximately 0.1 seconds.
- 3D Gaussians allow for high-fidelity, real-time novel-view rendering.

10.4 Long-Sequence Scene Reconstruction

Scales the input capability to handle wide-coverage scenes by processing long sequences of input views.

Hybrid Architecture:

- **Hybrid Blocks**: Combines Mamba2 blocks for efficient linear scaling and Transformer blocks for global attention.
- **Scaling**: Mamba2 provides $O(L)$ scalability to denser views compared to $O(L^2)$ for standard self-attention.
- **Bi-directional Scanning**: Implements bi-directional scans (forward and backward) over token sequences to handle image data effectively.

Capacity and Optimization:

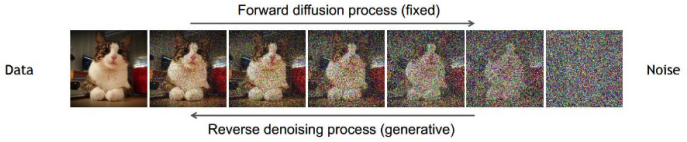
- **Sequence Length**: Processes up to 32 images with Plücker rays in 1.35 seconds.
- **Token Merge**: An optional module reduces sequence length to $1/4$ using 2D convolution with stride 2 to manage long sequences.
- **Gaussian Pruning**: Prunes Gaussians by opacity at test-time to maintain efficiency during high-resolution scene reconstruction.
- **Training Objective**: Combines image loss (MSE + Perceptual), an opacity regularizer to reduce Gaussian count, and a depth regularizer to eliminate "floater" artifacts.

11 Diffusion Models

11.1 Denoising Diffusion Probabilistic Models (DDPMs)

Core Idea

- Diffusion models learn to generate data by **iterative denoising**.
- Two complementary processes:
  - **Forward diffusion:** Gradually corrupt data with Gaussian noise.
  - **Reverse diffusion:** Learn to remove noise step-by-step to recover data.
- Grounded in nonequilibrium thermodynamics and probabilistic modeling.



Relevant Works:
[Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015]
[Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020]
[Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021]

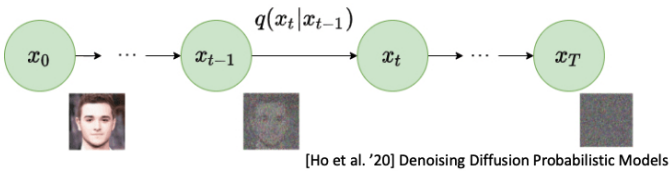
11.1.1 Forward Diffusion Process

Markovian Noise Injection

- Data sample $x_0 \sim q(x)$ is progressively noised over T steps.
- Each step depends only on the previous one:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \mu_t = \sqrt{1-\beta_t}x_{t-1}, \Sigma_t = \beta_t I)$$

- β_t is a variance schedule (e.g., linear from 10^{-4} to 0.02 over $T = 1000$ steps).



Closed-Form Sampling Inefficient to add noise step-wise.
One-step prediction of a noise at a specific timestep by applying Parametrization Trick for Gaussian distributions.

- Define $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.
- Allows direct sampling from x_0:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

- Enables efficient training by sampling arbitrary timesteps.

11.1.2 Reverse Diffusion Process

Generative Denoising

- As $t \to T = \infty$, x_T approaches pure Gaussian noise.
- Generation starts from sampling $x_T \sim \mathcal{N}(0, I)$.
- Reverse process:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_t)$$

- Since the true reverse posterior $p_\theta(x_{t-1} \mid x_t)$ is intractable, it is approximated with a parameterized model
- A neural network is trained to predict the parameters of this distribution, effectively learning to denoise x_t step-by-step.
- Repeating this process from $t = T$ to $t = 0$ produces a novel sample drawn from the original data distribution.

11.1.3 Training Objective

Optimzie negative log-likelihood of training data:

$$\mathcal{L} = \mathbb{E}_{x_0,\epsilon,t}\left[-\log p_\theta(x_{t-1} \mid x_t)\right]$$

Variational Lower Bound (VLB)

- However, direct likelihood maximization is intractable.
- Optimize a variational bound decomposed into timestep-wise KL terms.
- Most terms are either constant or simplified.

Simplified Noise Prediction Loss

- Instead of predicting μ_θ (image at specific timestep), predict noise ϵ_θ:

$$\mathcal{L}_{simple} = \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

- Fixed variance schedule Σ_t simplifies training.
- Leads to stable optimization and strong empirical performance.

11.2 Denoising Diffusion Implicit Models (DDIM)

Motivation

- DDPMs require many sequential denoising steps ($T = 1000$) for high-quality generation.
- DDIM accelerates sampling while maintaining quality by using a non-Markovian diffusion process.
- Enables deterministic generation with fewer steps.

[Denoising Diffusion Implicit Models, Song et al., ICLR 2021]

Core Idea

- DDPM sampling is inherently stochastic due to added noise at each step.
- DDIM proposes a non-Markovian reverse process that is deterministic.
- DDIM allows to skip steps during sampling, leading to less overall steps and faster generation.

Key Advantages

- **Accelerated Sampling:** Reduce steps from 1000 to 10-50 with minimal quality loss.

- **Determinism:** Set $\sigma_t = 0$ for deterministic generation (same latent z always produces same output).
- **No Retraining:** Uses existing DDPM models directly.
- **Semantically Meaningful Latent:** Interpolation in latent space produces smooth transitions.

Trade-offs

- Fewer steps may reduce sample diversity if too aggressive.
- Stochasticity-determinism trade-off: deterministic samples have less diversity but more consistency.

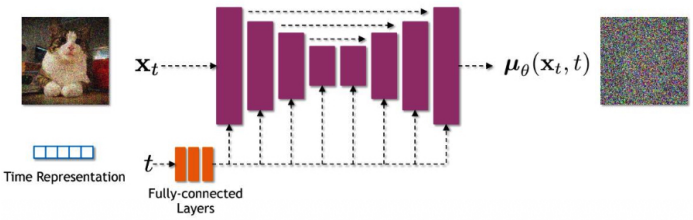
11.3 Diffusion Model Architectures

General Properties

- Input and output dimensions must match.
- Highly flexible architecture choices (U-Net, Transformer backbones)
- Conditioning information (time, text, camera, etc.) injected via embeddings.

U-Net Backbone

- Most common architecture for image diffusion.
- Encoder-decoder with skip connections.
- Time-step embeddings modulate intermediate layers.



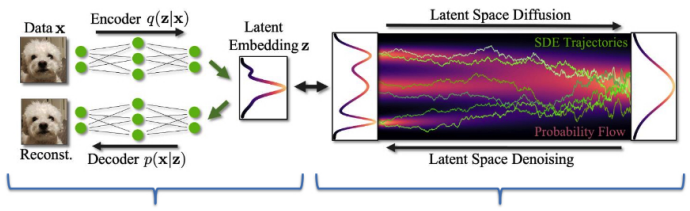
Transformer-Based Diffusion

- Replace U-Net with a Vision Transformer backbone.
- Patchify images and treat diffusion as sequence modeling.
- Strong scaling behavior with large datasets and models.

11.4 Latent Diffusion Models (U-Net)

Motivation

- Pixel-space diffusion is computationally expensive.
- Latent diffusion operates in a compressed representation.



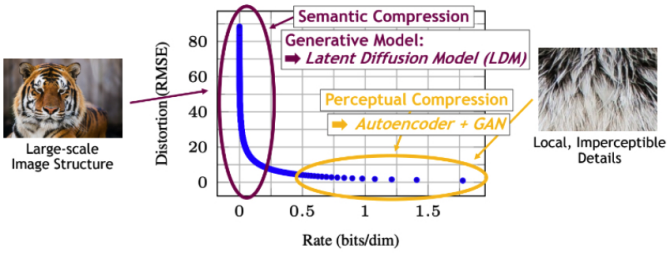
Two-Stage Training

1. Train an autoencoder (AE / VAE / VQ-VAE / VQ-GAN).
2. Perform diffusion in the learned latent space.

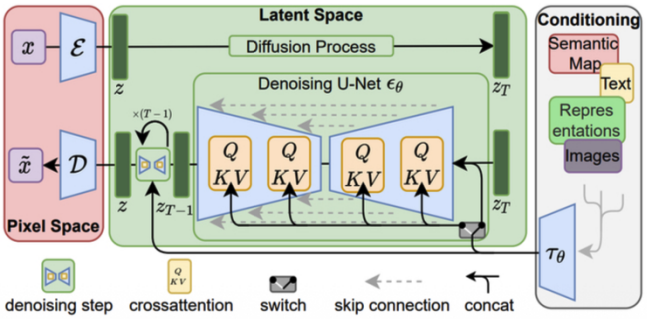
- Reduces memory and compute cost.
- Enables high-resolution synthesis (e.g., Stable Diffusion).

Findings:

- Latent diffusion for large-scale structure in images
- Autoencoder/GAN for fine details and textures

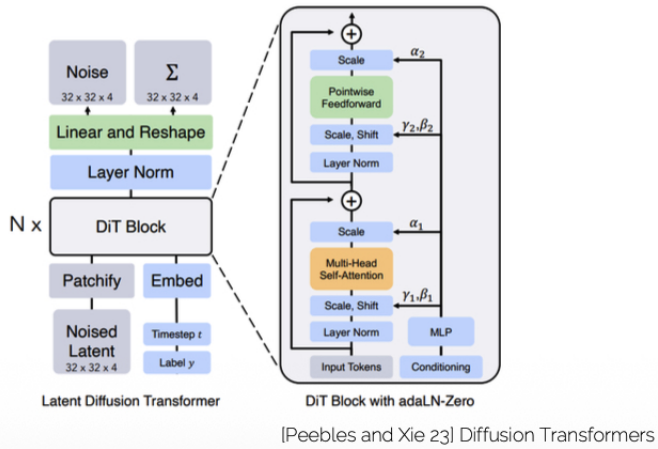


- Condition latent diffusion model on text embeddings, class labels, or other modalities.
- Enables powerful conditional generation (e.g., text-to-image).
- More guidance improves training signal and sample quality.



11.5 Diffusion Transformers (DiT)

- Train on image patches and treat diffusion as sequence modeling.
- Use transformer blocks with self-attention and cross-attention.
- Scales well with model size and dataset size.
- Achieves state-of-the-art image generation quality.



[Peebles and Xie 23] Diffusion Transformers

Scalable Diffusion Models with Transformers, DiT, CVPR 2023

11.6 Conditional Diffusion

Conditional Generation

- Model reverse process as $p_\theta(x_{t-1} \mid x_t, y)$.
- Supports text-to-image, class-to-image, and other modalities.

Classifier-Free Guidance

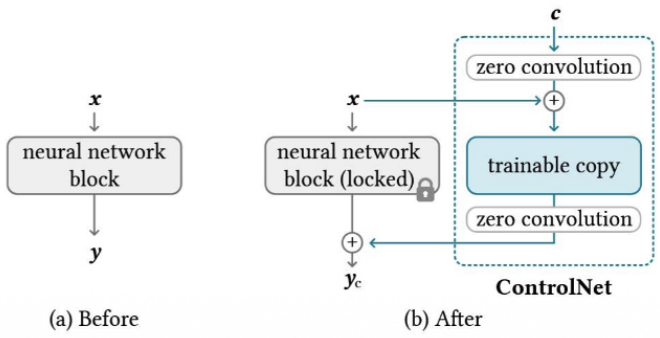
- Randomly drop condition during training.
- Intuition: takes steps towards fewer, higher-quality modes
- At inference, combine conditional and unconditional predictions:

$$\hat{\epsilon} = (1+\omega)\epsilon_\theta(x_t, t, y) - \omega\epsilon_\theta(x_t, t)$$

- Trades diversity for higher fidelity.
- Most prominent: ControlNet

11.7 ControlNet

- Learn to control a pretrained model with an additional input condition c.
- Copy weights from pretrained diffusion model and add trainable "zero-conv" blocks.
- Zero-initialized convolutions ensure initial behavior matches pretrained model.
- Train only the control branch on paired data (x, c).



[Adding Conditional Control to Text-to-Image Diffusion Models, ControlNet, CVPR 2023]

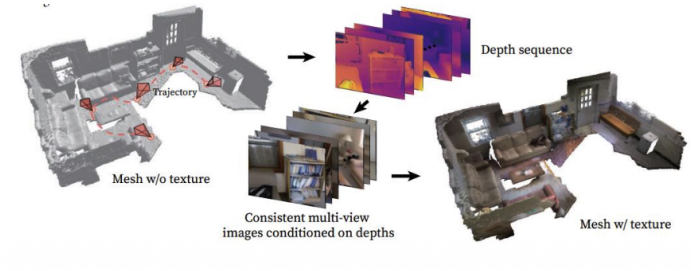
Controlled Generation

- Adds spatial or structural conditions (edges, depth, poses).
- Uses:
  - **Frozen pretrained diffusion model**
  - **Trainable control branch**
- Zero-initialized convolutions prevent early distortion.
- Enables training with limited additional data.

11.8 Multi-View Diffusion Models (MVDiffusion)

View-Consistent Image Generation

- Generate multiple views consistent with 3D geometry.
- Conditioning via camera parameters (R, T) or ray-based encodings.



Representative Methods

- **Zero-1-to-3:** Finetunes image diffusion on view pairs.
- **SyncDreamer:** Joint denoising of multiple views.
- **CAT3D:** Camera conditioning using raymaps.

11.9 Video Diffusion Models

Temporal Generation

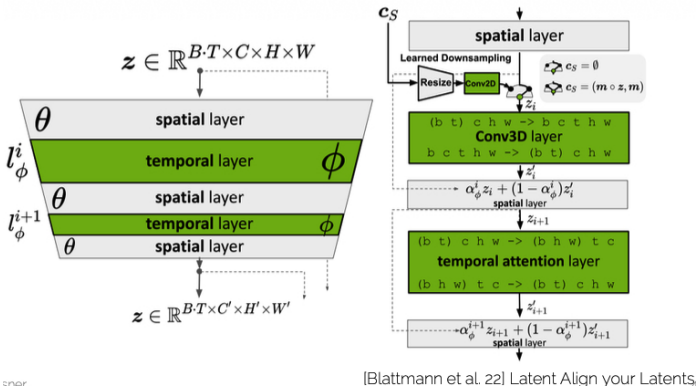
- Extend diffusion to spatio-temporal domains.
- Key challenge: temporal consistency.

Latent Video Diffusion

- Operate in compressed video latent space.
- Combine image and video pretraining.
- Examples: Align Your Latents, MovieGen, Sora.

Align your Latents

- Inject temporal information into latent diffusion.
- Use temporal attention layers in U-Net.
- Achieves high-quality video generation from text prompts.



[Blattmann et al. 22] Latent Align your Latents

11.10 General Latent Design

Either use pre-trained image-based latend diffusion models or train directly on video latents.

1. Pre-trained image-based LDM:
   - Can leverage massive amounts of pre-training
   - Issues in low-level temporal consistency
2. Train directly on video latents:
   - Better temporal consistency
   - Requires large-scale video datasets for training
   - However, no pre-trained models can be used (expensive)

11.11 Autoregressive Video Generation

Directly trained on video latents. **NOVA**

- Generates videos frame-by-frame in latent space.
- Uses optical flow for motion guidance.
- More efficient and less error accumulation than diffusion-only approaches.

11.12 3D-Aware Diffusion

Explicit 3D Supervision

- Train diffusion models with 3D ground truth.
- Example: DiffRF predicts radiance fields via diffusion.

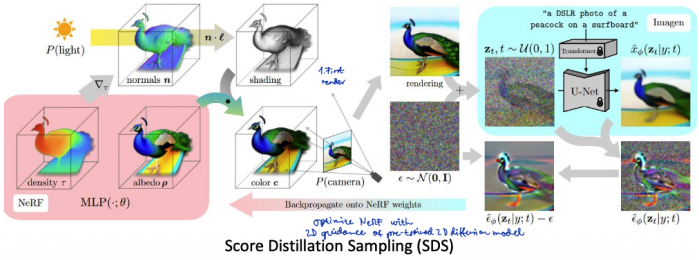
Latent 3D Gaussian Diffusion (L3DG)

- Diffusion in sparse latent space of 3D Gaussians.
- Enables object- and scene-level 3D generation.
- Scales to room-sized environments.

11.13 Score Distillation Sampling (SDS)

Text-to-3D via Diffusion Priors

- Use pretrained 2D diffusion models as supervision.
- Optimize 3D representation by matching diffusion scores.
- Introduced in DreamFusion.
- Slow: for each gradient step we need to perform a forward pass through the NeRF rendering.



Follow-Up Work

- ProlificDreamer (variational SDS).
- DreamGaussian (Gaussian-based 3D generation).

11.14 World and Scene Generation

Motivation

- Manual creation of scenes is labor-intensive and requires time and expertise.
- Challenges
  - scale
  - consistency
  - detail
  - context window: 3D scene diverge from previous views after a few steps (3D caching)
- Idea: leverage powerful 2D diffusion priors to generate 3D content.

Leveraging 2D Priors for 3D Worlds

- Iteratively lift generated images into 3D.
- Render-refine-repeat pipelines.

Representative Systems

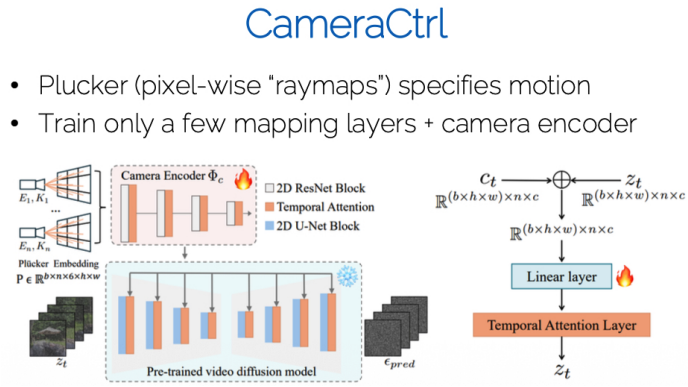
- Text2Room
- ControlRoom3D
- WonderWorld
- WorldExplorer

Text2Room

11.15 Camera-Controlled Video-to-3D

Explicit Camera Conditioning

- Pixel-wise raymaps or Plücker coordinates.
- Enables navigation and consistent viewpoint synthesis.



Key Methods

- CameraCtrl
- ViewCrafter
- GEN3C

11.16 Discussion: Diffusion vs GANs

- Diffusion models require same input-output dimensionality.
- Advantage on GANs: discrimminate on images where no 3D representation is available.
- GANs allow partial or indirect supervision.
- Diffusion models can not be trained on partial data, only optimized (e.g. use 2D Loss to push 3D diffusion: Distillation of pre-trained model)
- For both: Performance ultimately limited by training data quality and diversity.

12 HyperNetworks and Hyperdiffusion

- Use a network (HyperNetwork) to generate the weights of another network (Target Network).
- Enables parameter-efficient learning, multi-task learning, and dynamic weight generation.

Idea

- Instead of directly learning the parameters θ of a model, learn a function H that maps some input (e.g., task embedding) to θ: $\theta = H(z)$.
- The HyperNetwork H can be a simple MLP or a more complex architecture depending on the application.
- Instead of storing a massive set of fixed weights for a large network, you store a much smaller "controller" network that generates weights on the fly.
- This allows a model to have a high "effective" number of parameters while maintaining a small memory footprint.

Bridge to HyperDiffsuion

- Generate weights that represent the implicit neural occupancy field of a 3D shape.
- Use a diffusion model to learn the distribution of these weights across a dataset of shapes, enabling the generation of new shapes by sampling in weight space.

12.1 Hyperdiffusion

12.1.1 Methodology

1. **MLP Overfitting**: Each object (3D or 4D) in the dataset is represented by its own dedicated MLP.
2. **Training**: The MLP is trained to perfectly reproduce that specific object's occupancy field.
3. **Weight-Space Diffusion**: A diffusion model is trained directly on these MLP weights, which are flattened into 1D vectors.
4. **Transformer Backbone**: A transformer architecture is utilized to manage the high-dimensional weight vectors during the denoising process.
5. **Inference**: During generation, random noise in the weight space is gradually denoised to produce new MLP parameters corresponding to novel shapes.

HyperDiffusion: Generating Implicit Neural Fields with Convolutional Hypernetworks, Erkoç et al., ICCV 2023

12.1.2 Architecture

Architecture of Overfitted MLPs:

- **Hidden Layers**: 3 layers.
- **Hidden Width**: 128 neurons per layer.
- **Inputs**: Input dimension depends on data dimensionality (e.g., (x, y, z) or (x, y, z, t)).

12.1.3 Architecture of Weight-Space Diffusion

- **Embedding Size**: 2880
- **Heads**: 16 Heads
- **Layers**: 12 layers
- **Split Policy**: Layer-by-layer

Datasets:

- **3D**: ShapeNet categories including airplanes, cars, and chairs.
- **4D**: DeformingThings4D dataset containing 16-frame animated animals.

Metrics:

- **MMD** $\downarrow$ (Minimum Matching Distance): Measures Fidelity (Quality of the images). Calculates the average distance between every point in the reference (GT) set and its nearest neighbor in the generated set.
- **COV** $\uparrow$ (Coverage): Measures Diversity (Variety of the generated samples). Calculates the percentage of points in the reference set that are within a certain distance of any point in the generated set.
- **1-NNA** 50% (1-Nearest-Neighbor Accuracy): Overall distribution similarity. Taking a mix of real and generated shapes, for each shape we find the closest neighbor. If accuracy is 50% this means the real and generated shapes are perfectly mixed and classifier can not differentiate them.
- **FPD** $\downarrow$ (Frechet PointNet++ Distance): Perceptual Quality. Pass points through a pre-trained PointNet++ network to extract high-level features and calculates the distance between distribution of these features.

Results:

- Outperforms baselines like Point-Voxel Diffusion in perceptual quality (FPD).
- Produces smoother, temporally consistent 4D animations compared to voxel-based models.
- Demonstrates the ability to generate novel, non-memorized shapes.

Limitations:

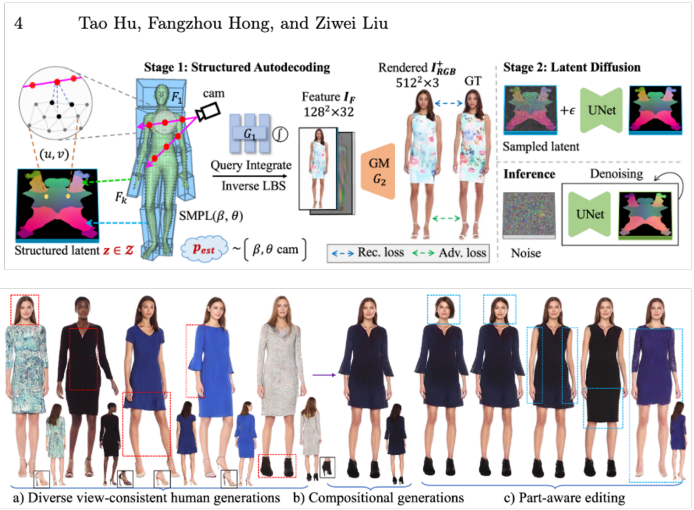
- **Geometry Blindness**: Diffusion operates on weights without direct awareness of the resulting 3D/4D geometry.
- **Scalability**: Does not yet handle large-scale scene-level representations.
- **Future Work**: Integration of reconstruction-aware diffusion could significantly improve geometric accuracy.

12.2 Semantic HyperDiffusion

12.2.1 Related Work

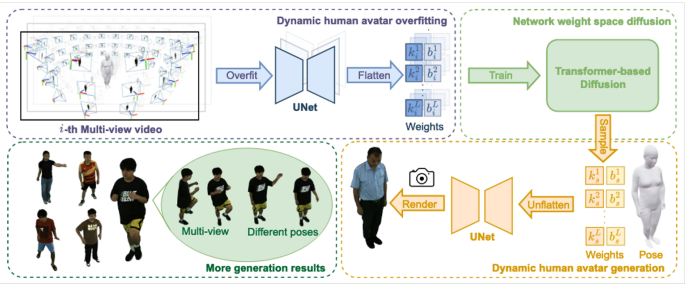
StructLDM (ICCV, 2024)

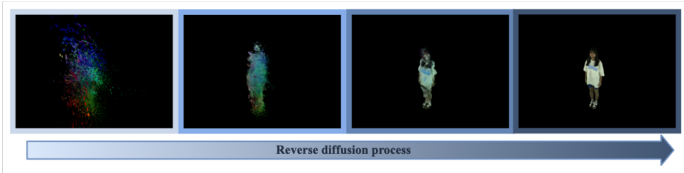
- Diffusion over structured space representing distinct parts of a 3D human body
- Part-aware encoder learned to optimize structured latent for each subject
- Increase semantic capture and controllability of output



Hyper Diffusion Avatars (CVPR, 2025)

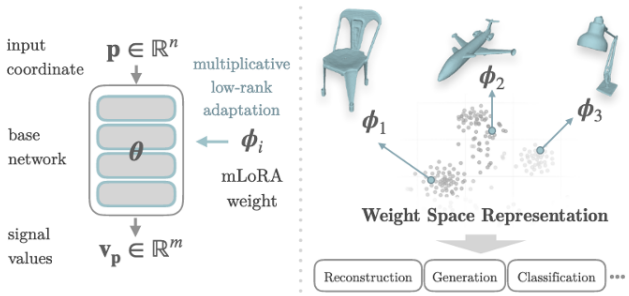
- Optimize a set of person-specific U-Nets, representing a dynamic human avatar
- Train a diffusion model to sample novel dynamic human avatars that look photorealistic
- Diffusion transformer learns the correlation between early layers (geometry) and later layers (appearance/details)





Weight Space Representation Learning with Neural Fields (CVPR, 2025)

- Base neural field from an MLP is adapted via multiplicative LoRA weights
- LoRA weights form a structured representation in weight space



12.2.2 Architecture

Architecture of Overfitted MLPs:

- **Hidden Layers**: 3 layers.
- **Hidden Width**: 60 neurons per layer.
- **Num Experts**: 4 experts
- **Inputs**: 3D Input coordinates (x, y, z).
- **Outputs**: Occupancy

Training:

- **Loss**: Binary Cross-Entropy Loss.
- **Batch Size/Epochs**: 2048 points, 1000 epochs, 7 Minutes per MLP.
- **Optimizer**: Adam with learning rate 10^{-5}.

12.2.3 Architecture of Weight-Space Diffusion

- Diffusion and transformer architecture modified versions of OpenAI and miniGPT implementations.
- **Embedding Size**: 768
- **Heads**: 16 Heads
- **Layers**: 12 layers
- **Split Policy**: Part-by-part and then Layer-by-layer