

Flexible Architectures for Image Synthesis

Ajay Jain



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-115

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-115.html>

May 14, 2021

Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Flexible Architectures for Image Synthesis

Ajay Jain

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for
the degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee

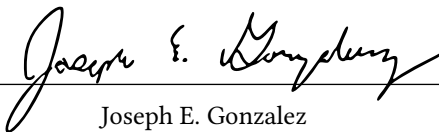


Pieter Abbeel
Research Advisor

5/14/2021

(Date)

★ ★ ★ ★ ★ ★ ★



Joseph E. Gonzalez
Second Reader

5/13/2021

(Date)

Copyright © 2021, by the author(s).

All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgements

I would like to thank all of the individuals I've worked with over the past two years at UC Berkeley. I have been fortunate to collaborate on projects with Pieter Abbeel, Paras Jain, Kurt Keutzer, Amir Gholami, Joseph E. Gonzalez, Jonathan Ho, Aniruddha Nrusimha, Deepak Pathak, Ion Stoica, Matthew Tancik, and Tianjun Zhang (alphabetical order), including the work presented in this report. I am especially grateful to my brother and co-author, Paras Jain, who introduced me to UC Berkeley's PhD program and who has been a constant friend and discussion partner throughout the program. I also especially thank my advisor, Pieter Abbeel, and Joseph Gonzalez, who mentored me when I first came to Berkeley. During the program, I collaborated with Kumar Krishna Agrawal, Sean O'Brien, Aravind Srinivas, and Jeffrey Tao on ongoing or unpublished work. I'm grateful for the fun memories made with friends, exciting discussions with labmates, and wonderful teaching from TAs and professors. Last, I dedicate this thesis to my family, who have supported me in pursuing education and research, and made me who I am. Additional acknowledgements are included in each chapter to individuals who provided invaluable feedback and to funding agencies.

Flexible Architectures for Image Synthesis

Ajay Jain

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 8 |
| 2 | Locally Masked Convolution for Autoregressive Models | 10 |
| 2.1 | Introduction | 10 |
| 2.2 | Background | 12 |
| 2.3 | Image Completion with Maximum Receptive Field | 14 |
| 2.4 | Local Masking | 16 |
| 2.5 | Architecture | 19 |
| 2.6 | Experiments | 20 |
| 2.6.1 | Whole-image Density Estimation | 21 |
| 2.6.2 | Generalization to Novel Orders | 23 |
| 2.6.3 | Image Completion | 24 |
| 2.6.4 | Qualitative Results | 24 |
| 2.7 | Related Work | 25 |
| 2.8 | Conclusion | 26 |
| 3 | Denoising Diffusion Probabilistic Models | 28 |
| 3.1 | Introduction | 28 |
| 3.2 | Background | 30 |
| 3.3 | Diffusion models and denoising autoencoders | 31 |
| 3.3.1 | Forward process | 32 |
| 3.3.2 | Reverse process | 32 |
| 3.3.3 | Data scaling and reverse process decoder | 33 |
| 3.3.4 | Simplified training objective | 34 |
| 3.4 | Experiments | 35 |
| 3.4.1 | Sample quality | 36 |
| 3.4.2 | Reverse process parameterization and training objective ablation | 36 |
| 3.4.3 | Progressive coding | 37 |

| | | |
|----------|--|-----------|
| 3.4.4 | Interpolation | 39 |
| 3.5 | Related Work | 40 |
| 3.6 | Conclusion | 41 |
| 4 | Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis | 42 |
| 4.1 | Introduction | 42 |
| 4.2 | Background on Neural Radiance Fields | 45 |
| 4.3 | NeRF Struggles at Few-Shot View Synthesis | 46 |
| 4.4 | Semantically Consistent Radiance Fields | 48 |
| 4.4.1 | Semantic consistency loss | 48 |
| 4.4.2 | Interpreting representations across views | 49 |
| 4.4.3 | Pose sampling distribution | 50 |
| 4.4.4 | Improving efficiency and quality | 50 |
| 4.5 | Experiments | 51 |
| 4.5.1 | Realistic Synthetic scenes from scratch | 52 |
| 4.5.2 | Single-view synthesis by fine-tuning | 53 |
| 4.5.3 | Reconstructing unobserved regions | 54 |
| 4.6 | Ablations | 56 |
| 4.7 | Related work | 57 |
| 4.8 | Conclusions | 58 |
| | Bibliography | 60 |
| 5 | Appendix: Locally Masked Convolution for Autoregressive Models | 73 |
| 5.1 | Order visualization | 73 |
| 5.2 | Mask conditioning | 73 |
| 5.3 | Experimental setup | 74 |
| 5.4 | Additional samples | 75 |
| 5.5 | Implementation | 76 |
| 6 | Appendix: Denoising Diffusion Probabilistic Models | 79 |
| 6.1 | Extended derivations | 80 |
| 6.2 | Experimental details | 81 |
| 6.3 | Discussion on related work | 83 |
| 6.4 | Samples | 83 |

| | | |
|----------|--|-----------|
| 7 | Appendix: Putting NeRF on a Diet | 95 |
| 7.1 | Experimental details | 95 |
| 7.2 | Per-scene metrics | 98 |
| 7.3 | Qualitative results and ground-truth | 99 |
| 7.4 | Adversarial approaches | 99 |

1 Introduction

Estimating high-dimensional distributions from true samples is a long-standing challenge problem in machine learning and statistics. Such a distribution estimate requires a model to capture interdependencies between a collection of variables, such as the dimensions of a random vector. Access to a parametric distribution estimate enables almost magical effects when applied to real-world data. When the distributions describe images, these applications include unconditional image generation *e.g.* synthesizing infinite artificial data, image generation conditioned on some known properties, photo editing, enhancements such as superresolution or inpainting, domain translation and more. Deep generative models also drive progress in other data modalities including speech synthesis, music generation and natural language generation.

Much of the research in deep generative models focuses on the estimation of an unconditional parametric distribution $p_\theta(\mathbf{x})$, measuring progress by task-independent sample quality and likelihood metrics. Still, the appeal of generative modeling lies in the flexibility of the prior p_θ to transfer to downstream tasks, where we usually have access to some conditioning information like a class label y or corrupted observation $\tilde{\mathbf{x}}$. In these settings, it is crucial to be able to access desired *conditional* distributions with low computational cost, such as $p_\theta(\mathbf{x}|\tilde{\mathbf{x}})$. General-purpose inference algorithms can sample from the desired conditionals in some cases given only the prior, but we would ideally like a family of generative models that readily exposes controllable generation knobs and gives the practitioner flexibility to adapt to various downstream tasks cheaply.

Our overall goal in this work is to show how generative image models can be made more flexible and adaptable, first by removing architectural limitations, then by easing data requirements for estimation via prior knowledge. First, in Chapter 2, we propose a variant of the PixelCNN autoregressive model architecture that supports image completion applications with arbitrary conditional distributions over data dimensions. Our modified architecture, the Locally Masked PixelCNN, allows parameter sharing across an ensemble that improves density estimation. Still, autoregressive models are powerful density estimators, but suffer from poor sample quality at small scale, are slow to sample, and are fairly inflexible for conditional generation tasks. In particular, autoregressive models like PixelCNNs sample only one data dimension at a time, typically with a full neural network forward pass that is wasteful.

In Chapter 3, we study diffusion probabilistic models, a family of generative models based on a parametric Markov chain, and propose a variant called DDPM that achieves high sample quality. We show that diffusion probabilistic models generalize autoregressive models but can sample multiple data dimensions in parallel, giving them more flexibility. This allows DDPM to scale up to high resolution images, with faster sampling than a corresponding autoregressive model. We develop a network architecture that significantly improves sample quality. We also show that diffusion models are connected to the estimation of energy-based models via denoising score matching, motivating a simplified reweighted variational objective that achieves better results.

Finally, in Chapter 4, we explore a challenging application of image synthesis: the novel view synthesis problem. In novel view synthesis, our goal is to interpolate sparse views of a scene from new camera poses. Given sparsely sampled observed views, existing approaches based on neural radiance fields estimate the parameters of a neural network that encodes a specific scene’s geometry and appearance. Then, volumetric rendering is used to generate novel views. In our work, we propose an auxiliary loss in feature space that allows prior knowledge from large image encoders to be transferred to the view synthesis problem. This gives neural radiance fields the ability to extrapolate to unseen regions—an important capability for generative models. Using the auxiliary loss to constrain the scene representation also improves the quality of view synthesis with as few as 1 to 8 observed images. Transferring prior knowledge from self-supervised models or classifiers is a promising approach to improve the data efficiency, flexibility and controllability of generative models.

2 Locally Masked Convolution for Autoregressive Models

Work by Ajay Jain, Pieter Abbeel, and Deepak Pathak

High-dimensional generative models have many applications including image compression, multimedia generation, anomaly detection and data completion. State-of-the-art estimators for natural images are autoregressive, decomposing the joint distribution over pixels into a product of conditionals parameterized by a deep neural network, *e.g.* a convolutional neural network such as the PixelCNN. However, PixelCNNs only model a single decomposition of the joint, and only a single generation order is efficient. For tasks such as image completion, these models are unable to use much of the observed context. To generate data in arbitrary orders, we introduce LMCONV: a simple modification to the standard 2D convolution that allows arbitrary masks to be applied to the weights at each location in the image. Using LMCONV, we learn an ensemble of distribution estimators that share parameters but differ in generation order, achieving improved performance on whole-image density estimation (2.89 bpd on unconditional CIFAR10), as well as globally coherent image completions. Our code is available at <https://ajayjain.github.io/lmconv>.

2.1 Introduction

Learning generative models of high-dimensional data such as images is a holy grail of machine learning with pervasive applications. Significant progress on this problem would naturally lead to a wide range of applications, including multimedia generation, compression, probabilistic time series forecasting, representation learning, and missing data completion. Many generative modeling frameworks have been proposed. Current state-of-the-art models for high-dimensional image data include (a) autoregressive models [4, 28], (b) normalizing flow density estimators [114], (c) generative adversarial networks (GANs) [36], (d) latent variable models such as the VAE [75,

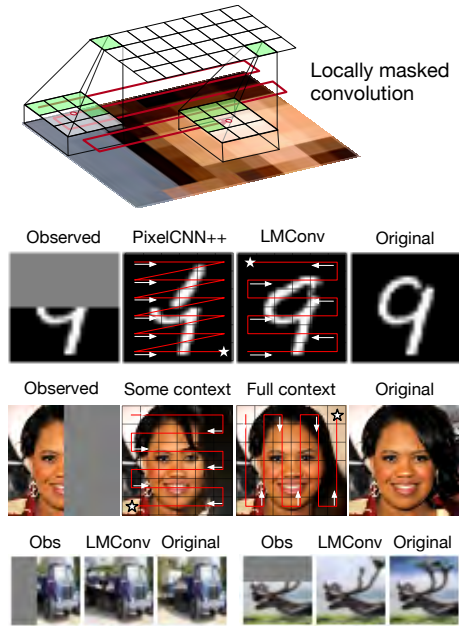


Figure 2.1: The ideal autoregressive joint distribution decomposition and sampling order are task-dependent. We learn to generate images under multiple orderings with the same parameters via *locally masked convolutions* (top), enabling global coherence for image completion (bottom).

112] and (e) energy-based models *e.g.* [26, 51, 80, 132]. While GANs, VAEs and EBMs have had great success in high-dimensional image generation, exact likelihoods are generally intractable. Likelihood estimation is key for many practical applications from uncertainty estimation, robustness, reliability and safety perspectives. In contrast, autoregressive and flow models estimate exact likelihoods and can be used for uncertainty estimation, though still have room for improved generation quality. In this work, our focus is on autoregressive models.

Given n variables, one can generate $n!$ autoregressive decompositions of the joint likelihood, each corresponding to a forward sampling order, and more if we assume conditional independence. Early autoregressive texture synthesis [28, 106] work could support multiple orders. However, recent CNN-based autoregressive models for images [97, 98, 122] capture only one of these orders (typically left-to-right raster scan, Fig. 2.2) for practical computational efficiency. Training and testing with a single order will not support all scenarios. Consider the image completion task in first row of Figure 2.1. If the top half of the image is missing, a raster scan generation order from left-to-right and top-to-bottom does not allow the model to condition on the context given in the observed bottom half of the image as the required conditionals are not estimated by the model.

In this work, we propose a scalable, yet simple modification to convolutional autoregressive models to estimate more accurate likelihoods with a minor change in computation during training. Our goal is to support arbitrary orders in a scalable manner, allowing more precise likelihoods by averaging over several graphical models corresponding to orders (a form of Bayesian model averaging). Some past works have supported arbitrary orders in autoregressive models by learning separate parameters for each model [30], or by masking the input image to hide successor variables [78]. A more efficient approach is to estimate densities in parallel across dimensions by masking network weights [34] differently for each order. However, all these methods are still computationally inefficient and difficult to scale beyond fully-connected networks to convolutional architectures.

In this work, we perform order-agnostic distribution estimation for natural images with state-of-the-art convolutional architectures. We propose to support arbitrary orderings by introducing masking at the level of features, rather than on inputs or weights. We show how an autoregressive CNN can support and learn multiple orders, with a single set of weights, via *locally masked convolutions* that efficiently apply location-specific masks to patches of each feature map. These local convolutions can be efficiently implemented purely via matrix multiplication by incorporating masking at the level of the im2col and col2im separation of convolution [61].

Arbitrary orders allow us to customize the traversal based on the needs of the task, which we evaluate in experiments. For instance, consider the examples shown in Fig. 2.1. The flexibility allows us to select the sampling order that exposes the maximum possible context for image completion, choose orderings that eliminate blind-spots (unobservable pixels) in image generation, and ensemble across multiple orderings using the same network weights. Note that such a model is able to support these image completions without training on any inpainting masks.

In experiments, we show that our approach can be efficiently implemented and is flexible without sacrificing the overall distribution estimation performance. By introducing order-agnostic training via LMCONV, we significantly outperform PixelCNN++ on the unconditional CIFAR10 dataset, achieving code lengths of 2.89 bits per dimension. We show that the model can generalize to some novel orders. Finally, we significantly outperform raster-scan baselines on conditional likelihoods relevant to image completion by customizing the generation order.

2.2 Background

Deep autoregressive models estimate high-dimensional data distributions using samples from the joint distribution over D -dimensions $p_{\text{data}}(\mathbf{x}_1, \dots, \mathbf{x}_D)$. In this setting, we wish to approximate

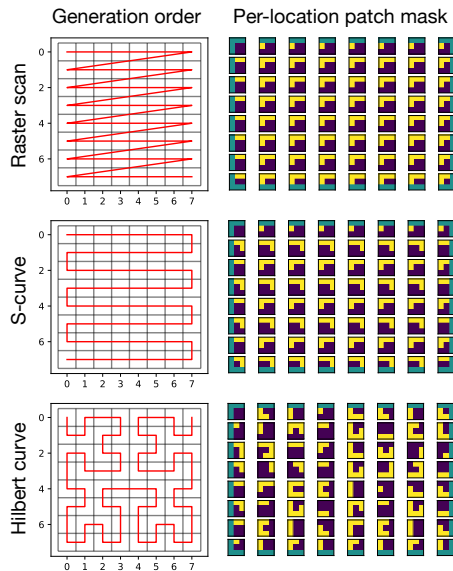


Figure 2.2: The three pixel generation orders and corresponding local masks that we consider in this work.

the joint with a parametric model $p_\theta(\mathbf{x}_1, \dots, \mathbf{x}_D)$ by minimizing KL-divergence $D_{KL}(p_{\text{data}}||p_\theta)$, or equivalently by maximizing the log-likelihood of the samples. As a general modeling principle, we can divide high-dimensional variables into many low-dimensional parts such as single dimensions, and capture dependencies between dimensions with a directed graphical model. Following the notation of [76], these autoregressive (AR) models represent the joint distribution as a product of conditionals,

$$\begin{aligned}
 p_\theta(\mathbf{x}) &= p_\theta(x_1, \dots, x_D) \\
 &= p_\theta(x_{\pi(1)}) \prod_{i=2}^D p_\theta(x_{\pi(i)} | Pa(\mathbf{x}_{\pi(i)}))
 \end{aligned} \tag{2.1}$$

where $\pi : [D] \rightarrow [D]$ is a permutation defining an order over the dimensions, $Pa(\mathbf{x}_{\pi(i)}) = \mathbf{x}_{\pi(1)}, \dots, \mathbf{x}_{\pi(i-1)}$ defines the parents of $x_{\pi(i)}$ in the graphical model, and θ is a parameter vector. As any joint can be decomposed in this manner according to the product rule, this factorization provides the foundation for many models including ours. The primary challenge in autoregressive models is defining a sufficiently expressive family for the conditionals where parameter estimation is efficient. Deep autoregressive models parameterize the conditionals with a neural network that is provided the context $Pa(\mathbf{x}_{\pi(i)})$.

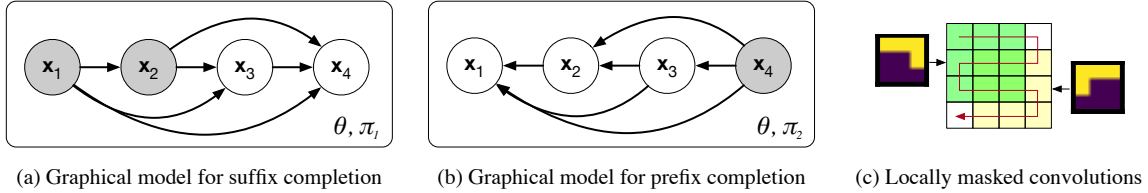


Figure 2.3: (a) A graphical model where the final, unobserved variables x_3, x_4 can be efficiently completed via forward sampling conditioned on the observed variables x_1, x_2 . (b) When x_4 is observed, we sample x_1, x_2 , and x_3 in the second graphical model using the same parameters. (c) LMCONV defines the model with masks at each filter location.

Decomposition (2.1) converts the joint modeling problem into a sequence modeling problem. Forward (ancestral) sampling draws root variable $x_{\pi(1)}$ first, then samples the remaining dimensions in order $x_{\pi(2)}, \dots, x_{\pi(D)}$ from their respective conditionals. Given a particular autoregressive decomposition of the joint, forward sampling supports a single data generation order. The joint model density for an observed variable can be computed exactly by evaluating each conditional, allowing density estimation and maximum likelihood parameter estimation,

$$\mathcal{L}(\theta) = \mathbb{E}_{x \sim p_{\text{data}}} \sum_{i=1}^D \log p_{\theta}(x_{\pi(i)} \mid x_{\pi(1)}, \dots, x_{\pi(i-1)})$$

$$\theta^* = \arg_{\theta} \max \mathcal{L}(\theta) \quad (2.2)$$

With some choices of network architecture, the conditionals can be computed in parallel by masking weights [34, 98]. In the PixelCNN model family, masked convolutions are *causal*: the features output by a masked convolution can only depend on features earlier in the order.

While the choice of order is arbitrary, temporal and sequential data modalities have a natural ordering from the first dimension in the sequence to the last. For spatial data such as images, a natural ordering is not clear. For computational reasons, a *raster scan* order is generally used where the top left pixel is modeled unconditionally and generation proceeds in row-major fashion across each row from left to right, depicted in Figure 2.1, second column.

2.3 Image Completion with Maximum Receptive Field

For estimating the distribution of 2D images, a raster scan ordering is perhaps as good of an order as any other choice. That said, the raster scan order has necessitated architectural innovations to allow the neural network to access information far back in the sequence such as two-dimensional

PixelRNNs [98], two-stream shift-based convolutional architectures [97], and self-attention combined with convolution [17]. These structures significantly improve test-set likelihoods and sample quality, but marry network architectures to the raster scan order.

Fixing a particular order is limiting for missing data completion tasks. Letting $\pi(i) = i$ denote the raster scan order, PixelRNN and PixelCNN architectures can complete only the bottom part of the image via forward sampling: given observations x_1, \dots, x_d , raster scan autoregressive models sequentially sample,

$$\hat{x}_i \sim p_\theta(x_i \mid x_1, \dots, x_d, \hat{x}_{d+1}, \dots, \hat{x}_{i-1}). \quad (2.3)$$

If all dimensions other than x_i are observed, ideally we would sample \hat{x}_i using maximum conditioning context,

$$\hat{x}_i \sim p_\theta(x_i \mid x_{<i}, x_{>i}). \quad (2.4)$$

Unfortunately, the raster scan model only predicts distributions of the form $p_\theta(x_i \mid x_{<i})$, and ignores observations $x_{>i}$ during completion. In the worst case, a model with a raster scan generation order *cannot observe any of the context* for an inpainting task where the top half of the image is unknown (Figure 2.1, PixelCNN++). This leads to image completions that do not respect global structure. Small numbers of dimensions could be sampled by computing the posterior, e.g. for $i = 1$,

$$p_\theta(\hat{x}_1 \mid x_{>1}) = \frac{p_\theta(\hat{x}_1, x_{>1})}{\sum_{x'_1} p_\theta(x'_1, x_{>1})}, \quad (2.5)$$

but this is expensive as each summand requires neural network evaluation, and becomes intractable when several dimensions are unknown. Instead of approximating the posterior, we estimate parameters θ that achieve high likelihood with multiple autoregressive decompositions,

$$\begin{aligned} \mathcal{L}_{\text{OA}}(\theta) &= \mathbb{E}_{x \sim p_{\text{data}}} \mathbb{E}_{\pi \sim p_\pi} \log p_\theta(x_1, \dots, x_D; \pi) \\ \theta^* &= \arg_\theta \max \mathcal{L}_{\text{OA}}(\theta) \end{aligned} \quad (2.6)$$

with p_π denoting a uniform distribution over several orderings. The joint distribution under π factorizes according to (2.1). The resulting conditionals are all parameterized by the same neural network. By choosing order prior p_π that supports a π such that $\pi(D) = i$, we can use the network with such an ordering to query (2.4) directly.

During optimization with stochastic gradient descent, we make single-sample estimates of the inner expectation in (2.6) according to order-agnostic training [34, 140]. A single order is used within each batch.

For a test-time task where $\{x_i : i \in T_{\text{obs}}\}$ are observed, we select a π that the model was trained with such that

$$\{\pi(1), \dots, \pi(|T_{\text{obs}}|)\} = T_{\text{obs}},$$

i.e. the first $|T_{\text{obs}}|$ dimensions in the generation order are the observed dimensions, then sample according to the rest of the order so that the model posterior over each unknown dimension is conditioned either on observed or previously sampled dimensions.

2.4 Local Masking

In this section, we develop *locally masked convolutions* (LMCONV): a modification to the standard convolution operator that allows control over generation order and parallel computation of conditionals for evaluating likelihood. In the first convolutional layer of a neural network, C_{out} filters of size $k \times k$ are applied to the input image with spatial invariance: the same parameters are used at all locations in a sliding window. Each filter has $k^2 * C_{\text{in}}$ parameters. For images with discretized intensities, convolutional autoregressive networks transform a spatial $H \times W$, multi-channel image into a tensor of log-probabilities that define the conditional distributions of (2.1). These log-probabilities take the form of an $H \times W$ image, with channel count equal to the number of color channels times the number of bins per color channel. The output log-probabilities at coordinate i, j in the output define the distribution $p_{\theta}(x_{i,j} | Pa(p(x_{i,j})))$. Critically, this distribution must not depend on observations of successors in the Bayesian network, or the product of conditionals will not define a valid distribution due to cyclicity.

NADE [78] circumvents the problem by masking the input image, though requires independent forward passes to compute each factor of the autoregressive decomposition (2.1). Instead, the PixelCNN model family controls information flow through the network by setting certain weights of the convolution filters to zero, similar to how MADE [34] masks the weight matrices in fully-connected layers. We depict masked convolutions for the first convolutional layer in Figure 2.4. As a single mask is applied to the $C_{\text{in}} \times k \times k$ parameter tensor defining each convolutional filter, the same masking pattern is in effect applied at all locations in the image. The architectural constraint that the masking pattern is shared limits the possible orders supported by the PixelCNN model family, and leads to blind spots which the output distribution is unable to condition upon.

In practice, convolutions are implemented through general matrix multiplication (GEMM) due to widely available, heavily optimized and parallelized implementations of the operation on GPU and CPU. To use matrix multiplication, the input to a layer is rearranged in memory via the im2col algorithm, which extracts $C_{\text{in}} \times k \times k$ patches from the $C_{\text{in}} \times H \times W$ input at each location that a

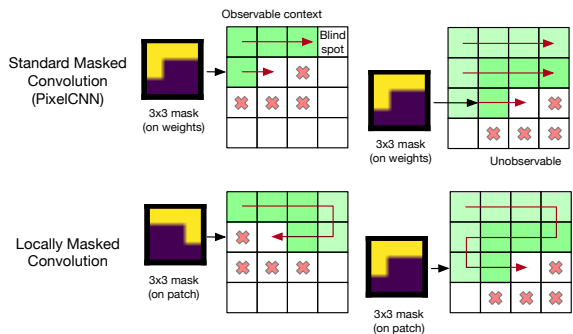


Figure 2.4: A comparison of standard weight masked convolutions and the proposed locally masked convolution.

convolutional filter will be applied. Assuming padding and a stride of 1 is used, the rearrangement yields matrix X with $C_{in} * k^2$ rows and $H * W$ columns. To perform convolution, the framework left-multiplies weight matrix \mathcal{W} , storing $Y = \mathcal{W}X$, adds a bias, and finally rearranges Y into a spatial format via the colzim algorithm.

We exploit this data rearrangement to arbitrarily mask the input to the convolutional filter at each location it is applied. The inputs to the convolution at each location, *i.e.* the input patches, form columns of X . For a given generation order, we construct mask matrix \mathcal{M} of the same dimensions as X and set $X = \mathcal{M} \odot X$ prior to matrix multiplication. In particular, our locally masked convolution masks *patches of the input to each layer*, rather than masking weights and rather than masking the initial input to the network. LMCONV combines the flexibility of NADE and the parallelizability of MADE and PixelCNN. The LMCONV algorithm is summarized in Algorithm 1, and mask construction is detailed in Algorithm 2.

We implement two versions of the layer with the PyTorch machine learning framework [104]. The first is an implementation that uses autodifferentiation to compute gradients. As only the forward pass is defined by the user, the implementation is under 20 lines of Python.

However, reverse-mode autodifferentiation incurs significant memory overheads during back-propagation as the output of nearly every operation during the forward pass must be stored until gradient computation [43, 58]. Data rearrangement with im2col is memory intensive as features patches overlap and are duplicated. We implement a custom, memory efficient backward pass that only stores the input, the mask and the output of the layer during the forward pass and recomputes the im2col operation during the backward pass. Recomputing the im2col operation achieves 2.7× memory savings at a 1.3× slowdown.

Algorithm 1 LMCONV: Locally masked 2D convolution

- 1: **Input:** image x , weights \mathcal{W} , bias b , generation order π . x is $B \times C_{\text{in}} \times H \times W$ dimensional and \mathcal{W} is $C_{\text{out}} \times C_{\text{in}} * k_1 * k_2$ dimensional
 - 2: Create mask matrix \mathcal{M} with Algorithm 2
 - 3: Extract patches: $X = \text{im2col}(\text{pad}(x), k_1, k_2)$
 - 4: Mask patches: $X = \mathcal{M} \odot X$
 - 5: Perform convolution via batch MM: $Y = \mathcal{W}X + b$
 - 6: Assemble patches: $y = \text{col2im}(Y)$
 - 7: **return** y
-

Algorithm 2 Create input mask matrix

- 1: **Input:** Generation order $\pi(\cdot)$, constants C_{in}, k_1, k_2 , dilation d , is this the first layer?
 - 2: Start with an empty set of generated coordinates
 - 3: Initialize \mathcal{M} as $k_1 * k_2 \times H * W$ zero matrix
 - 4: **for** i from 1 to $H * W$ **do**
 - 5: Let (r, c) be coordinates of dimension $\pi(i)$
 - 6: **for** offsets Δ_r, Δ_c in $k_1 \times k_2$ kernel **do**
 - 7: **if** $(r + d\Delta_r, c + d\Delta_c)$ has been generated **then**
 - 8: Allow output location (r, c) to access features at $(r + d\Delta_r, c + d\Delta_c)$ in previous layer: set $\mathcal{M}_{k_2\Delta_r + \Delta_c, Wr+c} = 1$
 - 9: **end if**
 - 10: **end for**
 - 11: Add (r, c) to generated coordinates
 - 12: **end for**
 - 13: **if** not the first layer **then**
 - 14: Allow previous layer features to be observed at all locations: set center row $\lfloor \frac{k_1 * k_2}{2} \rfloor$ of \mathcal{M} to 1
 - 15: **end if**
 - 16: Repeat rows of \mathcal{M} , C_{in} times
 - 17: **return** binary mask matrix \mathcal{M}
-

Using locally masked convolutions, we can experiment with many different image generation orders. In this work, we consider three classes of orderings: raster scan, implemented in baseline PixelCNNs, an S-curve order that traverses rows in alternating directions, and a Hilbert space-filling curve order that generates nearby pixels in the image consecutively. Alternate orderings provide several benefits. Nearby pixels in an image are highly correlated. By generating these pixels close in a Hilbert curve order, we might expect information to propagate from the most important, nearby observations for each dimension and reduce the vanishing gradient problem.

If the image is considered a graph with nodes for each pixel and edges connecting adjacent pixels, a convolutional autoregressive model using an order defined by a Hamiltonian path over the image graph will also suffer no blind spot in a D layer network. To see this, note that the features corresponding to dimension $x_{\pi(i)}$ in the Hamiltonian path order will always be able to

observe the previous layer’s features corresponding to $x_{\pi(i-1)}$. After at least D layers of depth, the features for $x_{\pi(i)}$ will incorporate information from all $i - 1$ previous dimensions. In practice, information propagates with fewer required layers in these architectures as multiple neighbors are observed in each layer. Finally, we select multiple orderings at inference and average the resulting joint distributions to compute better likelihood estimates.

2.5 Architecture

We use a network architecture similar to PixelCNN++ [122], the best-in-class density estimator in the fully convolutional autoregressive PixelCNN model family. Convolution operations are masked according to Algorithm 1. While our locally masked convolutions can benefit from self-attention mechanisms used in later work, we choose a fully convolutional architecture for simplicity and to study the benefit of local masking in isolation of other architectural innovations. We make three modifications to the PixelCNN++ architecture that simplify it and allow for arbitrary generation orders. Gated PixelCNN uses a two-stream architecture composed of two network stacks with $\lfloor \frac{k}{2} \rfloor \times 1$ and $\lfloor \frac{k}{2} \rfloor \times k$ convolutions to enforce the raster scan order. In the horizontal stream, Gated PixelCNN applies non-square convolutions and feature map shifts or pads to extract information within the same row, to the left of the current dimension. In the vertical stream, Gated PixelCNN extracts information from above. Skip connections between streams allow information to propagate. PixelCNN++ uses a similar architecture based on a U-Net [116] with approximately 54M parameters. We replace the two streams with a simple, single stream with the same depth, using LMCONV to maintain the autoregressive property. Masks for these convolutions are computed and cached at the beginning of training. Due to the regularizing effect of order-agnostic training, we do not use dropout.

Second, we use dilated convolutions [158] at regular intervals in the model rather than down-sampling the feature map. Downsampling precludes many orders, as the operation aggregates information from contiguous squares of pixels together without a mask. Dilated convolutions expand the receptive field without limiting the order, as local masks can be customized to hide or reveal specific features accessed by the filter.

Finally, we normalize the feature map across the channel dimension by applying positional normalization [82]. Normalization allows masks to have varying numbers of ones at each spatial location by rescaling features to the same scale.

As in PixelCNN++, our model represents each conditional with a mixture of 10 discretized logistic distributions that imposes a distribution over binned pixel intensities. For the binarized

Table 2.1: Average negative log likelihood of binarized and grayscale MNIST digits under baselines and our model. Lower is better.

| BINARIZED MNIST, 28x28 | NLL (nats) |
|---------------------------------|-------------------|
| DARN (Intractable) [42] | ≈ 84.13 |
| NADE [140] | 88.33 |
| EoNADE 2hl (128 orders) [140] | 85.10 |
| EoNADE-5 2hl (128 orders) [109] | 84.68 |
| MADE 2hl (32 orders) [34] | 86.64 |
| PixelCNN [98] | 81.30 |
| PixelRNN [98] | 79.20 |
| Ours, S-curve (1 order) | 78.47 |
| Ours, S-curve (8 orders) | 77.58 |

| GRAYSCALE MNIST, 28x28 | NLL (bpd) |
|-------------------------------|------------------|
| Spatial PixelCNN [1] | 0.88 |
| PixelCNN++ (1 stream) | 0.77 |
| Ours, S-curve (1 order) | 0.68 |
| Ours, S-curve (8 orders) | 0.65 |

MNIST dataset [119], we instead use a softmax over two logits. We train with 8 variants of an S-curve (zig-zag) order that traverses each row of the image in alternating directions so that consecutively generated pixels are adjacent, and so that locally masked CNNs with sufficient depth can achieve the maximum allowed receptive field.

Across all quantitative experiments, we use a model with approximately 46M parameters, trained with the Adam optimizer with a learning rate of $2 * 10^{-4}$ decayed by a factor of $1 - 5 * 10^{-6}$ per iteration with clipped gradients. For CelebA-HQ qualitative results, we increase filter count and train a model with 184M parameters. More details are provided in the appendix.

2.6 Experiments

To evaluate the benefits of our approach, we study three scientific questions: (1) *do locally masked autoregressive ensembles estimate more accurate likelihoods on image datasets than single-order models?* (2) *can the model generalize to novel orders?* and (3) *how important is order selection for image completion?*

Table 2.2: Average negative log likelihood of CIFAR10 images under our model. Lower is better.

| CIFAR10, 32x32 | NLL (bpd) |
|------------------------------------|------------------|
| Uniform Distribution | 8.00 |
| Multivariate Gaussian [98] | 4.70 |
| Attention-based | |
| Image Transformer [103] | 2.90 |
| PixelSNAIL [17] | 2.85 |
| Sparse Transformer [19] | 2.80 |
| Convolutional | |
| PixelCNN (1 stream) [98] | 3.14 |
| Gated PixelCNN (2 stream) [97] | 3.03 |
| PixelCNN++ (1 stream) | 2.99 |
| PixelCNN++ (2 stream) [122] | 2.92 |
| Ours, S-curve (1 stream, 1 order) | 2.91 |
| Ours, S-curve (1 stream, 8 orders) | 2.89 |

We estimate the distribution of three image datasets: 28×28 grayscale and binary [119] MNIST digits, 32×32 8-bit color CIFAR10 natural images, and high-resolution CelebA-HQ 5-bit color face photographs [66]. Unlike classification, density estimation remains challenging on these datasets. We train the CelebA-HQ models at 256×256 resolution to compare with prior density estimation work, and at a bilinearly downsampled 64×64 resolution.

Our locally masked model achieves better likelihoods than PixelCNN++ by using multiple generation orders. We then show that the model can generalize to generation orders that it has not been trained with. Finally, for image completion, we achieve the best results over strong baselines by using orders that expose all observed pixels.

2.6.1 Whole-image Density Estimation

Tractable generative models are generally evaluated via the average negative log likelihood (NLL) of test data. For interpretability, many papers normalize base 2 NLL by the number of dimensions. By normalizing, we can measure bits per dimension (bpd), or a lower-bound for the expected number of bits needed per pixel to losslessly compress images using a Huffman code with $p(\mathbf{x})$ estimated by our model. Better estimates of the distribution should result in higher compression rates. Tables 2.1 and 2.2 show likelihoods for our model and prior models.

Table 2.3: Average conditional negative log likelihood for **Top**, **Left** and **Bottom** half image completion.

| BINARIZED MNIST 28x28 (nats) | T | L | B |
|---------------------------------------|--------------|--------------|--------------|
| Ours (adversarial order) | 41.76 | 39.83 | 43.35 |
| Ours (1 max context order) | 34.99 | 32.47 | 36.57 |
| Ours (2 max context orders) | 34.82 | 32.25 | 36.36 |
| CIFAR10 32x32 (bpd) | T | L | B |
| PixelCNN++, 1 stream | 3.07 | 3.10 | 3.05 |
| PixelCNN++, 2 stream | 2.97 | 2.98 | 2.93 |
| Ours (1 stream, adversarial order) | 2.93 | 2.98 | 3.05 |
| Ours (1 stream, 1 max context order) | 2.77 | 2.83 | 2.89 |
| Ours (1 stream, 2 max context orders) | 2.76 | 2.82 | 2.88 |

On binarized MNIST (Table 2.1), our locally masked PixelCNN achieves significantly higher likelihoods (lower NLL) than baselines, including neural autoregressive models NADE, EoNADE, and MADE that average across large numbers of orderings. This is due to architectural advantages of our CNN and increased model capacity. Our model also outperforms the standard PixelCNN, which suffers from a blind spot problem due to sharing the same mask at all locations. Likelihood is further improved by using ensemble averaging across 8 orders that share parameters. These results are also observed on grayscale MNIST where each pixel has one of 256 intensity levels.

On CIFAR10, we achieve 2.89 bpd test set likelihood when averaging the joint probability of 8 graphical models, each defined by an S-curve generation order. Our results outperform the state-of-the-art convolutional autoregressive model, PixelCNN++. We significantly outperform a 1 stream architectural variant of PixelCNN++ that has the same number of parameters as our model and uses a similar architecture, differing only in that it uses a single raster scan order. By introducing order-agnostic ensemble averaging to convolutional autoregressive models, we combined the best of fully-connected density estimators that average over orders, and the inductive biases of CNNs. These results could further improve with self-attention mechanisms and additional capacity, which have been observed to improve the performance of single-order estimation, marking an opportunity for future research.

Our model is also scalable to high resolution distribution estimation. On the CelebA-HQ 256x256 dataset at 5-bit color depth, our model achieves 0.74 bpd with a single S-curve order, outperforming



Figure 2.5: CIFAR10 image completions using our locally-masked convolutions with a specialized ordering.

Glow [73], an exact likelihood normalizing flow. In comparison, the state-of-the-art model, SPN [86], achieves 0.61 bpd by using self-attention and a specialized architecture for high resolutions.

2.6.2 Generalization to Novel Orders

Ideally, an order-agnostic model would be able to generate images in orders that it has not been trained with. To understand generalization to novel orders, we evaluate the test-set likelihood of a CIFAR10 model that achieves 2.93 bpd with a single S-curve order and 2.91 bpd with 8 S-curve orders under a raster scan decomposition. The model achieves 3.75 bpd with 1 raster scan order (28% increase) and 3.67 bpd with 8 raster scan orders (26% increase). While the novel order degrades compression rate, the model was trained with 8 fixed orders of the same S-curve type, which are fairly different from a raster scan.

To study generalization to more similar orders, we trained a model on Binarized MNIST with 7 S-curves for 120 epochs. On the test set, the model has 0.144 bpd using each train order. Testing with the held out (8th) S-curve, the model achieves 0.151 bpd, only 5% higher.

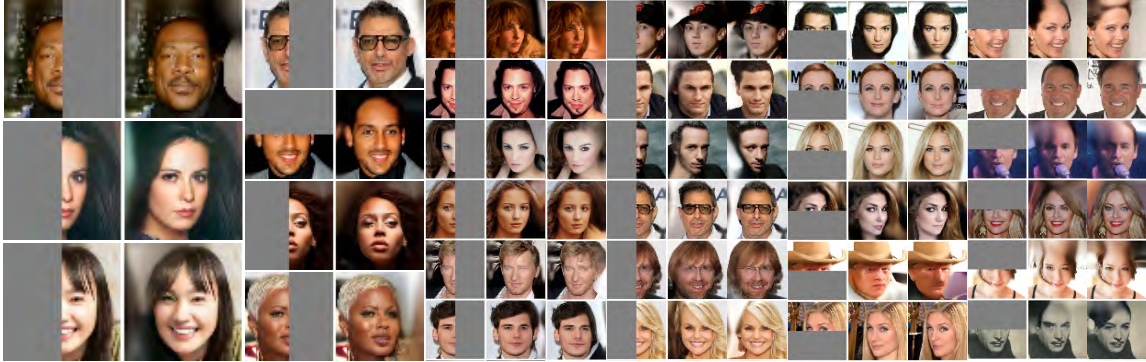


Figure 2.6: Completions of 64×64 px CelebA-HQ images at 5-bit color depth. Up to 2 samples are shown to the right of each half-observed face provided to the model. Missing pixels are generated along an S-curve that first traverses the observed region. Additional samples and ground truth completions are provided in the appendix.

2.6.3 Image Completion

To quantitatively assess whether control over generation order improves image completions, we measure the average conditional negative log likelihood of hidden regions of held-out test images on the MNIST and CIFAR10 datasets, measured in bits per dimension. We compute the NLL of the top half, left half, and bottom half of the image conditioned on the remainder of the image. The hidden region is set to zero in the model input, as well as hidden via masks used in each model.

Table 2.3 shows average NLL on binary MNIST and CIFAR10. Top half inpainting is challenging for PixelCNN baselines that use a raster scan order, as model conditional $p_\theta(x_i|x_{<i})$ does not condition on observed pixels that lie below x_i in the image. Similarly, our architecture under an adversarial order, a single S-shaped curve from the top left to bottom left of the image, achieves 2.93 bpd on CIFAR in the **T** setting. In contrast, using the same parameters, when we decomposes the joint favorably for maximum context with an S-curve generation order from the bottom left to the top left of the image, we achieve 2.77 bpd. Averaging over two maximum context orders further improves log likelihood to 2.76 bpd. A similar trend is observed for the other completion tasks, **L** and **B**.

2.6.4 Qualitative Results

Figure 2.1 shows completions of MNIST and CelebA-HQ 64×64 images. PixelCNN++ produces MNIST digits that are inconsistent with the observed context. With a poor choice of order, our model only respects some attributes of the input image, but not overall facial structure. The model

distributions over each missing pixel should condition on the entire observed region. This is accomplished when the missing region is generated last via a maximum context order. With this order, completions by our model are consistent with the given context.

Figures 2.5 and 2.6 show completions of held-out CIFAR10 32×32 and CelebA-HQ 64×64 images for four different missing regions. The masked input to the model (Obs), our sampled completion (Ours) and the ground truth image (GT) are shown. Missing image regions are generated in a maximum context order. While samples have some artifacts such as blurring due to long sequence lengths, images are globally coherent, with matching colors and object structure (CIFAR10) or facial structure (CelebA-HQ). Across datasets and image masks, our model effectively uses available context to generate coherent samples.

2.7 Related Work

Autoregressive models are a popular choice to estimate the joint distribution of high-dimensional, multivariate data in deep learning. [30] proposes logistic autoregressive Bayesian networks where each conditional is learned through logistic regression, capturing first-order dependencies between variables. While different orders had similar performance, averaging densities from 10 differently ordered models achieved small improvements in likelihood. [4] extend this idea, using artificial neural networks to capture conditionals with some parameter sharing. [78] propose the neural autoregressive distribution estimator (NADE) for binary and discrete data, reducing the complexity of density estimation from quadratic in the number of dimensions to linear. [141] extend NADE to real-valued vectors (RNADE), expressing conditionals as mixture density networks. The autoregressive approach is desirable due to the lack of conditional independence assumptions, easy training via maximum likelihood, tractable density, and tractable, though sequential, forward sampling directly from the conditionals.

These works all use a single, arbitrary order per estimated model. However, it is possible to use the same parameters to define a family of differently ordered autoregressive Bayesian networks. [140] propose EoNADE, an ensemble of input-masked NADE models trained with an order-agnostic training procedure that achieve higher likelihoods when averaged and allows forward sampling of arbitrary regions. Each iteration, EoNADE chooses a random prefix of an ordering $\pi(1), \dots, \pi(d)$, sample a training example x and maximize the likelihood of x_d under their model. ConvNADE [139] adapts EoNADE with a convolutional architecture and conditions the model on the input mask defining the order. Still, NADE, EoNADE and ConvNADE are serial: only a single conditional is trained at a time, and density estimation requires D passes. [34] propose

an order-agnostic MADE that masks the weights of a fully connected autoencoder to estimate densities with a single forward pass by computing conditionals in parallel. While MADE supports multiple orders, it is limited by a fully-connected architecture. Our Locally Masked PixelCNN can be seen as a generalization of MADE that supports convolutional inductive bias.

Other deep autoregressive models use recurrent, convolutional or self-attention architectures. In language modeling, autoregressive recurrent neural networks (RNNs) predict a distribution over the next token in a sequence conditioned on a recurrently updated representation of the previous words [87]. [98] extend this idea to images, proposing a multi-dimensional, sequential PixelRNN for image generation and discrete distribution estimation, and a parallelizable PixelCNN. Subsequent works capture correlations between pixels in an image with convolutional architectures inspired by the PixelCNN [86, 97, 111, 122], often improving the ability of the network to capture long-range dependencies. The PixelCNN family can generate entire high-fidelity images and, until recently, achieved state-of-the-art test set likelihood among tractable, likelihood-based generative models. PixelCNNs have also been used as a prior for latent variables [101], and can be sampled in parallel using fixed-point methods [134, 148]. While convolutions process information locally in an image, self-attention mechanisms have been used to gain global receptive field [17, 19, 103] for improved statistical performance.

Normalizing flows [114] are parametric density estimators that give exact expressions for likelihood using the change-of-variables formula by transforming samples from a simple prior with learned, invertible functions. If tractable densities are not required, other families are possible. Implicit generative models such as GANs [36] have been applied to high resolution image generation [66] and inpainting [105]. Nonparametric approaches have also been successful for inpainting [3, 28, 46]. Partial convolutions [83] improve CNN inpainting quality by rescaling filter responses that access missing pixels, but are not causal unlike LMCONV. Latent-variable models like the VAE [75, 112] jointly learn a generative model for data x given latent z and an approximation for the posterior over z . Other latent-variable models are based on Markov chains [6, 93, 130].

2.8 Conclusion

In this work, we proposed an efficient, scalable and easy to implement approach for supporting arbitrary autoregressive orderings within convolutional networks. To do so, we propose *locally masked convolutions* that allow arbitrary orderings by masking features at each layer while simultaneously sharing filter weights. This formulation can be efficiently implemented purely via

matrix multiplication. Our work is a synthesis of prior lines of inquiry in autoregressive models. Locally Masked PixelCNNs support parallel estimation, convolutional inductive biases, and control over order, all with one simple layer. Foundational work in this area each supported some of these, but with incompatible architectures. As an additional benefit, arbitrary orderings allow image completion with diverse regions. We achieve globally coherent image completions by choosing a favorable order at test time, without specifically training the model to inpaint.

Acknowledgements

We thank Paras Jain, Nilesch Tripuraneni, Joseph Gonzalez and Jonathan Ho for helpful discussions, and reviewers for helpful suggestions. This research is supported in part by the NSF GRFP under grant number DGE-1752814, Berkeley Deep Drive and the Open Philanthropy Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

3 Denoising Diffusion Probabilistic Models

Work by Jonathan Ho, Ajay Jain, and Pieter Abbeel

We present high quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics. Our best results are obtained by training on a weighted variational bound designed according to a novel connection between diffusion probabilistic models and denoising score matching with Langevin dynamics, and our models naturally admit a progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding. On the unconditional CIFAR10 dataset, we obtain an Inception score of 9.46 and a state-of-the-art FID score of 3.17. On 256x256 LSUN, we obtain sample quality similar to ProgressiveGAN. Our implementation is available at <https://github.com/hojonathanho/diffusion>.

3.1 Introduction

Deep generative models of all kinds have recently exhibited high quality samples in a wide variety of data modalities. Generative adversarial networks (GANs), autoregressive models, flows, and variational autoencoders (VAEs) have synthesized striking image and audio samples [9, 24, 36, 63, 64, 66, 73, 75, 86, 96, 98, 107, 110], and there have been remarkable advances in energy-based modeling and score matching that have produced images comparable to those of GANs [27, 132].

This paper presents progress in diffusion probabilistic models [130]. A diffusion probabilistic model (which we will call a “diffusion model” for brevity) is a parameterized Markov chain trained using variational inference to produce samples matching the data after finite time. Transitions of this chain are learned to reverse a diffusion process, which is a Markov chain that gradually adds noise to the data in the opposite direction of sampling until signal is destroyed. When the diffusion consists of small amounts of Gaussian noise, it is sufficient to set the sampling chain transitions to conditional Gaussians too, allowing for a particularly simple neural network parameterization.

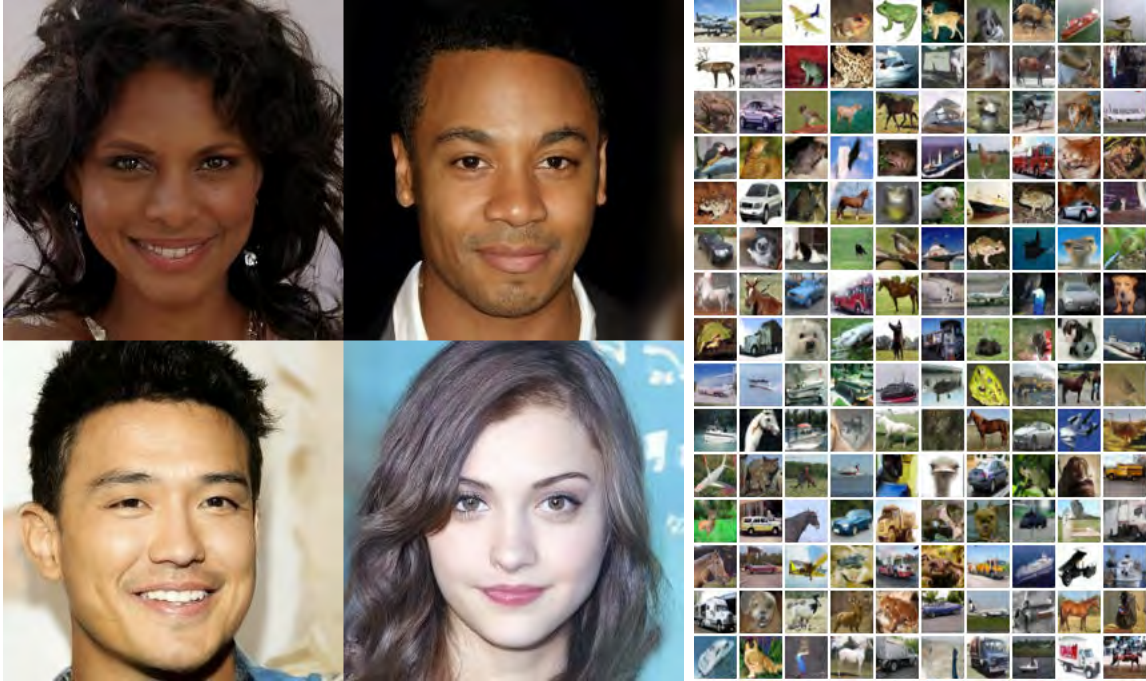


Figure 3.1: Generated samples on CelebA-HQ 256×256 (left) and unconditional CIFAR10 (right)

Diffusion models are straightforward to define and efficient to train, but to the best of our knowledge, there has been no demonstration that they are capable of generating high quality samples. We show that diffusion models actually are capable of generating high quality samples, sometimes better than the published results on other types of generative models (Section 3.4). In addition, we show that a certain parameterization of diffusion models reveals an equivalence with denoising score matching over multiple noise levels during training and with annealed Langevin dynamics during sampling (Section 3.3.2) [132, 143]. We obtained our best sample quality results using this parameterization (Section 3.4.2), so we consider this equivalence to be one of our primary contributions.

Despite their sample quality, our models do not have competitive log likelihoods compared to other likelihood-based models (our models do, however, have log likelihoods better than the large estimates annealed importance sampling has been reported to produce for energy based models and score matching [27, 132]). We find that the majority of our models' lossless codelengths are consumed to describe imperceptible image details (Section 3.4.3). We present a more refined analysis of this phenomenon in the language of lossy compression, and we show that the sampling procedure of diffusion models is a type of progressive decoding that resembles autoregressive

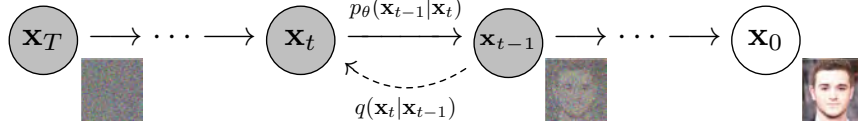


Figure 3.2: The directed graphical model considered in this work.

decoding along a bit ordering that vastly generalizes what is normally possible with autoregressive models.

3.2 Background

Diffusion models [130] are latent variable models of the form $p_\theta(\mathbf{x}_0) := \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$, where $\mathbf{x}_1, \dots, \mathbf{x}_T$ are latents of the same dimensionality as the data $\mathbf{x}_0 \sim q(\mathbf{x}_0)$. The joint distribution $p_\theta(\mathbf{x}_{0:T})$ is called the *reverse process*, and it is defined as a Markov chain with learned Gaussian transitions starting at $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{o}, \mathbf{I})$:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \quad (3.1)$$

What distinguishes diffusion models from other types of latent variable models is that the approximate posterior $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$, called the *forward process* or *diffusion process*, is fixed to a Markov chain that gradually adds Gaussian noise to the data according to a variance schedule β_1, \dots, β_T :

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1}), \quad q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad (3.2)$$

Training is performed by optimizing the usual variational bound on negative log likelihood:

$$\mathbb{E}[-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] = \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] =: L \quad (3.3)$$

The forward process variances β_t can be learned by reparameterization [75] or held constant as hyperparameters, and expressiveness of the reverse process is ensured in part by the choice of Gaussian conditionals in $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, because both processes have the same functional form when

β_t are small [130]. A notable property of the forward process is that it admits sampling \mathbf{x}_t at an arbitrary timestep t in closed form: using the notation $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^t \alpha_s$, we have

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}) \quad (3.4)$$

Efficient training is therefore possible by optimizing random terms of L with stochastic gradient descent. Further improvements come from variance reduction by rewriting L (3.3) as:

$$\mathbb{E}_q \left[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T | \mathbf{x}_0) \parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}_{L_0} \right] \quad (3.5)$$

(See Section 6.1 for details. The labels on the terms are used in Section 3.3.) Equation (3.5) uses KL divergence to directly compare $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$ against forward process posteriors, which are tractable when conditioned on \mathbf{x}_0 :

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\boldsymbol{\beta}}_t \mathbf{I}), \quad (3.6)$$

$$\text{where } \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1 - \bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{\bar{\alpha}_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t \quad \text{and} \quad \tilde{\boldsymbol{\beta}}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t \quad (3.7)$$

Consequently, all KL divergences in Eq. (3.5) are comparisons between Gaussians, so they can be calculated in a Rao-Blackwellized fashion with closed form expressions instead of high variance Monte Carlo estimates.

3.3 Diffusion models and denoising autoencoders

Diffusion models might appear to be a restricted class of latent variable models, but they allow a large number of degrees of freedom in implementation. One must choose the variances β_t of the forward process and the model architecture and Gaussian distribution parameterization of the reverse process. To guide our choices, we establish a new explicit connection between diffusion models and denoising score matching (Section 3.3.2) that leads to a simplified, weighted variational bound objective for diffusion models (Section 3.3.4). Ultimately, our model design is justified by simplicity and empirical results (Section 3.4). Our discussion is categorized by the terms of Eq. (3.5).

3.3.1 Forward process and L_T

We ignore the fact that the forward process variances β_t are learnable by reparameterization and instead fix them to constants (see ?? for details). Thus, in our implementation, the approximate posterior q has no learnable parameters, so L_T is a constant during training and can be ignored.

3.3.2 Reverse process and $L_{1:T-1}$

Now we discuss our choices in $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$ for $1 < t \leq T$. First, we set $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$ to untrained time dependent constants. Experimentally, both $\sigma_t^2 = \beta_t$ and $\sigma_t^2 = \tilde{\beta}_t = \frac{1-\bar{\alpha}_t}{1-\alpha_t} \beta_t$ had similar results. The first choice is optimal for $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{o}, \mathbf{I})$, and the second is optimal for \mathbf{x}_0 deterministically set to one point. These are the two extreme choices corresponding to upper and lower bounds on reverse process entropy for data with coordinatewise unit variance [130].

Second, to represent the mean $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$, we propose a specific parameterization motivated by the following analysis of L_t . With $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})$, we can write:

$$L_{t-1} = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2 \right] + C \quad (3.8)$$

where C is a constant that does not depend on θ . So, we see that the most straightforward parameterization of $\boldsymbol{\mu}_\theta$ is a model that predicts $\tilde{\boldsymbol{\mu}}_t$, the forward process posterior mean. However, we can expand Eq. (3.8) further by reparameterizing Eq. (3.4) as $\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}$ for $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{o}, \mathbf{I})$ and applying the forward process posterior formula (3.7):

$$L_{t-1} - C = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\boldsymbol{\mu}}_t \left(\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}), \frac{1}{\sqrt{\bar{\alpha}_t}} (\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) - \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}) \right) - \boldsymbol{\mu}_\theta(\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}), t) \right\|^2 \right] \quad (3.9)$$

$$= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{1}{2\sigma_t^2} \left\| \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon} \right) - \boldsymbol{\mu}_\theta(\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}), t) \right\|^2 \right] \quad (3.10)$$

Equation (3.10) reveals that $\boldsymbol{\mu}_\theta$ must predict $\frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon} \right)$ given \mathbf{x}_t . Since \mathbf{x}_t is available as input to the model, we may choose the parameterization

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \tilde{\boldsymbol{\mu}}_t \left(\mathbf{x}_t, \frac{1}{\sqrt{\bar{\alpha}_t}} (\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t)) \right) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) \quad (3.11)$$

where $\boldsymbol{\epsilon}_\theta$ is a function approximator intended to predict $\boldsymbol{\epsilon}$ from \mathbf{x}_t . To sample $\mathbf{x}_{t-1} \sim p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is to compute $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(\mathbf{o}, \mathbf{I})$. The complete sampling

Algorithm 3 Training

```

1: repeat
2:    $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ 
3:    $t \sim \text{Uniform}(\{1, \dots, T\})$ 
4:    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
5:   Take gradient descent step on
      $\nabla_{\theta} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t)\|^2$ 
6: until converged

```

Algorithm 4 Sampling

```

1:  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
2: for  $t = T, \dots, 1$  do
3:    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  if  $t > 1$ , else  $\mathbf{z} = \mathbf{0}$ 
4:    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}} \left( \mathbf{x}_t - \frac{1 - \bar{\alpha}_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ 
5: end for
6: return  $\mathbf{x}_0$ 

```

procedure, Algorithm 4, resembles Langevin dynamics with $\boldsymbol{\epsilon}_{\theta}$ as a learned gradient of the data density. Furthermore, with the parameterization (3.11), Eq. (3.10) simplifies to:

$$\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t)\|^2 \right] \quad (3.12)$$

which resembles denoising score matching over multiple noise scales indexed by t [132]. As Eq. (3.12) is equal to (one term of) the variational bound for the Langevin-like reverse process (3.11), we see that optimizing an objective resembling denoising score matching is equivalent to using variational inference to fit the finite-time marginal of a sampling chain resembling Langevin dynamics.

To summarize, we can train the reverse process mean function approximator $\boldsymbol{\mu}_{\theta}$ to predict $\tilde{\boldsymbol{\mu}}_t$, or by modifying its parameterization, we can train it to predict $\boldsymbol{\epsilon}$. (There is also the possibility of predicting \mathbf{x}_0 , but we found this to lead to worse sample quality early in our experiments.) We have shown that the $\boldsymbol{\epsilon}$ -prediction parameterization both resembles Langevin dynamics and simplifies the diffusion model’s variational bound to an objective that resembles denoising score matching. Nonetheless, it is just another parameterization of $p_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t)$, so we verify its effectiveness in Section 3.4 in an ablation where we compare predicting $\boldsymbol{\epsilon}$ against predicting $\tilde{\boldsymbol{\mu}}_t$.

3.3.3 Data scaling, reverse process decoder, and L_0

We assume that image data consists of integers in $\{0, 1, \dots, 255\}$ scaled linearly to $[-1, 1]$. This ensures that the neural network reverse process operates on consistently scaled inputs starting

from the standard normal prior $p(\mathbf{x}_T)$. To obtain discrete log likelihoods, we set the last term of the reverse process to an independent discrete decoder derived from the Gaussian $\mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \sigma_1^2 \mathbf{I})$:

$$p_\theta(\mathbf{x}_0|\mathbf{x}_1) = \prod_{i=1}^D \int_{\delta_-(x_0^i)}^{\delta_+(x_0^i)} \mathcal{N}(x; \mu_\theta^i(\mathbf{x}_1, 1), \sigma_1^2) dx$$

$$\delta_+(x) = \begin{cases} \infty & \text{if } x = 1 \\ x + \frac{1}{255} & \text{if } x < 1 \end{cases} \quad \delta_-(x) = \begin{cases} -\infty & \text{if } x = -1 \\ x - \frac{1}{255} & \text{if } x > -1 \end{cases} \quad (3.13)$$

where D is the data dimensionality and the i superscript indicates extraction of one coordinate. (It would be straightforward to instead incorporate a more powerful decoder like a conditional autoregressive model, but we leave that to future work.) Similar to the discretized continuous distributions used in VAE decoders and autoregressive models [74, 121], our choice here ensures that the variational bound is a lossless codelength of discrete data, without need of adding noise to the data or incorporating the Jacobian of the scaling operation into the log likelihood. At the end of sampling, we display $\boldsymbol{\mu}_\theta(\mathbf{x}_1, 1)$ noiselessly.

3.3.4 Simplified training objective

With the reverse process and decoder defined above, the variational bound, consisting of terms derived from Eqs. (3.12) and (3.13), is clearly differentiable with respect to θ and is ready to be employed for training. However, we found it beneficial to sample quality (and simpler to implement) to train on the following variant of the variational bound:

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[\left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t) \right\|^2 \right] \quad (3.14)$$

where t is uniform between 1 and T . The $t = 1$ case corresponds to L_0 with the integral in the discrete decoder definition (3.13) approximated by the Gaussian probability density function times the bin width, ignoring σ_1^2 and edge effects. The $t > 1$ cases correspond to an unweighted version of Eq. (3.12), analogous to the loss weighting used by the NCSN denoising score matching model [132]. (L_T does not appear because the forward process variances β_t are fixed.) Algorithm 3 displays the complete training procedure with this simplified objective.

Since our simplified objective (3.14) discards the weighting in Eq. (3.12), it is a weighted variational bound that emphasizes different aspects of reconstruction compared to the standard variational bound [41, 50]. In particular, our diffusion process setup in Section 3.4 causes the simplified objective to down-weight loss terms corresponding to small t . These terms train the

Table 3.1: CIFAR10 results. NLL measured in bits/dim.

| Model | IS | FID | NLL Test (Train) |
|--|--------------------|-------------|--------------------|
| Conditional | | | |
| EBM [27] | 8.30 | 37.9 | |
| JEM [40] | 8.76 | 38.4 | |
| BigGAN [9] | 9.22 | 14.73 | |
| StyleGAN2 + ADA (v1) [67] | 10.06 | 2.67 | |
| Unconditional | | | |
| Diffusion (original) [130] | | | ≤ 5.40 |
| Gated PixelCNN [99] | 4.60 | 65.93 | 3.03 (2.90) |
| Sparse Transformer [19] | | | 2.80 |
| PixelIQN [102] | 5.29 | 49.46 | |
| EBM [27] | 6.78 | 38.2 | |
| NCSNv2 [133] | | 31.75 | |
| NCSN [132] | 8.87±0.12 | 25.32 | |
| SNGAN [90] | 8.22±0.05 | 21.7 | |
| SNGAN-DDLS [12] | 9.09±0.10 | 15.42 | |
| StyleGAN2 + ADA (v1) [67] | 9.74 ± 0.05 | 3.26 | |
| Ours (L , fixed isotropic Σ) | 7.67±0.13 | 13.51 | ≤ 3.70 (3.69) |
| Ours (L_{simple}) | 9.46±0.11 | 3.17 | ≤ 3.75 (3.72) |

Table 3.2: Unconditional CIFAR10 reverse process parameterization and training objective ablation. Blank entries were unstable to train and generated poor samples with out-of-range scores.

| Objective | IS | FID |
|--|------------------|-------------|
| $\tilde{\mu}$ prediction (baseline) | | |
| L , learned diagonal Σ | 7.28±0.10 | 23.69 |
| L , fixed isotropic Σ | 8.06±0.09 | 13.22 |
| $\ \tilde{\mu} - \tilde{\mu}_\theta\ ^2$ | - | - |
| ϵ prediction (ours) | | |
| L , learned diagonal Σ | - | - |
| L , fixed isotropic Σ | 7.67±0.13 | 13.51 |
| $\ \tilde{\epsilon} - \epsilon_\theta\ ^2$ (L_{simple}) | 9.46±0.11 | 3.17 |

network to denoise data with very small amounts of noise, so it is beneficial to down-weight them so that the network can focus on more difficult denoising tasks at larger t terms. We will see in our experiments that this reweighting leads to better sample quality.

3.4 Experiments

We set $T = 1000$ for all experiments so that the number of neural network evaluations needed during sampling matches previous work [130, 132]. We set the forward process variances to constants increasing linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$. These constants were chosen to be small relative to data scaled to $[-1, 1]$, ensuring that reverse and forward processes have approximately the same functional form while keeping the signal-to-noise ratio at \mathbf{x}_T as small as possible ($L_T = D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})) \approx 10^{-5}$ bits per dimension in our experiments).

To represent the reverse process, we use a U-Net backbone similar to an unmasked PixelCNN++ [117, 121] with group normalization throughout [151]. Parameters are shared across time, which is specified to the network using the Transformer sinusoidal position embedding [142]. We use self-attention at the 16×16 feature map resolution [142, 147]. Details are in Section 6.2.

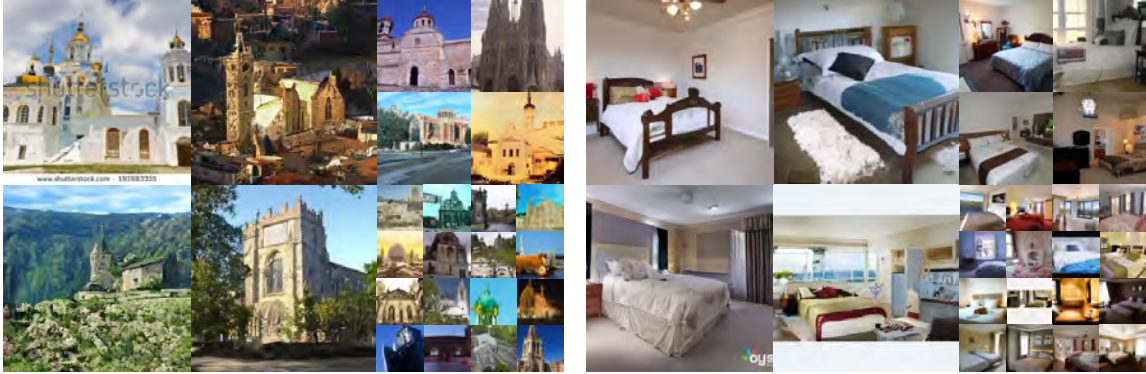


Figure 3.3: LSUN Church samples. FID=7.89 Figure 3.4: LSUN Bedroom samples. FID=4.90

3.4.1 Sample quality

Table 3.1 shows Inception scores, FID scores, and negative log likelihoods (lossless codelengths) on CIFAR10. With our FID score of 3.17, our unconditional model achieves better sample quality than most models in the literature, including class conditional models. Our FID score is computed with respect to the training set, as is standard practice; when we compute it with respect to the test set, the score is 5.24, which is still better than many of the training set FID scores in the literature.

We find that training our models on the true variational bound yields better codelengths than training on the simplified objective, as expected, but the latter yields the best sample quality. See Fig. 3.1 for CIFAR10 and CelebA-HQ 256×256 samples, Fig. 3.3 and Fig. 3.4 for LSUN 256×256 samples [159], and Section 6.4 for more.

3.4.2 Reverse process parameterization and training objective ablation

In Table 3.2, we show the sample quality effects of reverse process parameterizations and training objectives (Section 3.3.2). We find that the baseline option of predicting $\tilde{\mu}$ works well only when trained on the true variational bound instead of unweighted mean squared error, a simplified objective akin to Eq. (3.14). We also see that learning reverse process variances (by incorporating a parameterized diagonal $\Sigma_{\theta}(\mathbf{x}_t)$ into the variational bound) leads to unstable training and poorer sample quality compared to fixed variances. Predicting ϵ , as we proposed, performs approximately as well as predicting $\tilde{\mu}$ when trained on the variational bound with fixed variances, but much better when trained with our simplified objective.

| Algorithm 5 Sending \mathbf{x}_0 | Algorithm 6 Receiving |
|---|--|
| 1: Send $\mathbf{x}_T \sim q(\mathbf{x}_T \mathbf{x}_0)$ using $p(\mathbf{x}_T)$ 2: for $t = T - 1, \dots, 2, 1$ do 3: Send $\mathbf{x}_t \sim q(\mathbf{x}_t \mathbf{x}_{t+1}, \mathbf{x}_0)$ using $p_\theta(\mathbf{x}_t \mathbf{x}_{t+1})$ 4: end for 5: Send \mathbf{x}_0 using $p_\theta(\mathbf{x}_0 \mathbf{x}_1)$ | 1: Receive \mathbf{x}_T using $p(\mathbf{x}_T)$ 2: for $t = T - 1, \dots, 1, 0$ do 3: Receive \mathbf{x}_t using $p_\theta(\mathbf{x}_t \mathbf{x}_{t+1})$ 4: end for 5: return \mathbf{x}_0 |

3.4.3 Progressive coding

Table 3.1 also shows the codelengths of our CIFAR10 models. The gap between train and test is at most 0.03 bits per dimension, which is comparable to the gaps reported with other likelihood-based models and indicates that our diffusion model is not overfitting (see Section 6.4 for nearest neighbor visualizations). Still, while our lossless codelengths are better than the large estimates reported for energy based models and score matching using annealed importance sampling [27], they are not competitive with other types of likelihood-based generative models [19].

Since our samples are of high quality, we conclude that diffusion models have an inductive bias that makes them excellent lossy compressors. Treating the variational bound terms $L_1 + \dots + L_T$ as rate and L_0 as distortion, our CIFAR10 model with the highest quality samples has a rate of **1.78** bits/dim and a distortion of **1.97** bits/dim, which amounts to a root mean squared error of 0.95 on a scale from 0 to 255. More than half of the lossless codelength describes imperceptible distortions.

Progressive lossy compression We can probe further into the rate-distortion behavior of our model by introducing a progressive lossy code that mirrors the form of Eq. (3.5): see Algorithms 5 and 6, which assume access to a procedure, such as minimal random coding [44, 45], that can transmit a sample $\mathbf{x} \sim q(\mathbf{x})$ using approximately $D_{\text{KL}}(q(\mathbf{x}) \parallel p(\mathbf{x}))$ bits on average for any distributions p and q , for which only p is available to the receiver beforehand.

When applied to $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, Algorithms 5 and 6 transmit $\mathbf{x}_T, \dots, \mathbf{x}_0$ in sequence using a total expected codelength equal to Eq. (3.5). The receiver, at any time t , has the partial information \mathbf{x}_t fully available and can progressively estimate:

$$\mathbf{x}_0 \approx \hat{\mathbf{x}}_0 = \left(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \epsilon_\theta(\mathbf{x}_t) \right) / \sqrt{\bar{\alpha}_t} \quad (3.15)$$

due to Eq. (3.4). (A stochastic reconstruction $\mathbf{x}_0 \sim p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ is also valid, but we do not consider it here because distortion is more difficult to evaluate.) Figure 3.5 shows the resulting rate-distortion plot on the CIFAR10 test set. At each time t , the distortion is calculated as the root mean squared

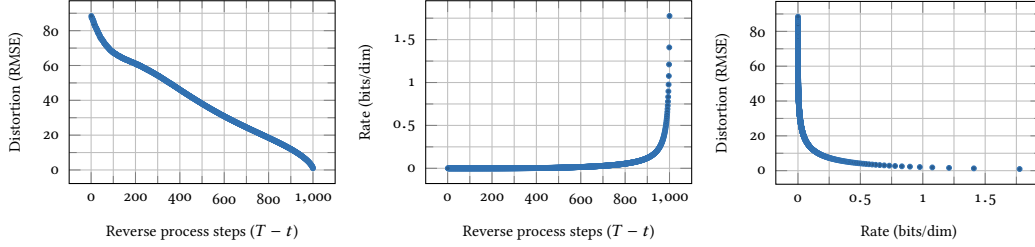


Figure 3.5: Unconditional CIFAR10 test set rate-distortion vs. time. Distortion is measured in root mean squared error on a $[0, 255]$ scale. See Table 6.2 for details.

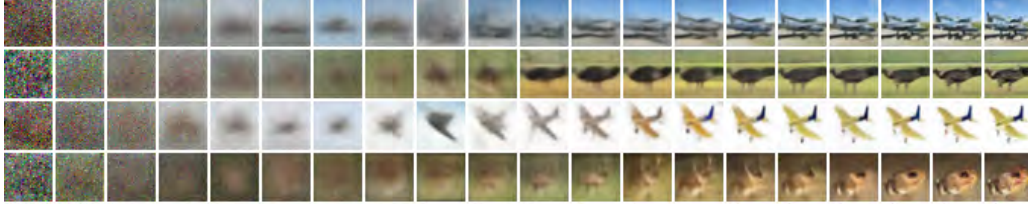


Figure 3.6: Unconditional CIFAR10 progressive generation ($\hat{\mathbf{x}}_0$ over time, from left to right). Extended samples and sample quality metrics over time in the appendix (Figs. 6.2 and 6.6).

error $\sqrt{\|\mathbf{x}_0 - \hat{\mathbf{x}}_0\|^2/D}$, and the rate is calculated as the cumulative number of bits received so far at time t . The distortion decreases steeply in the low-rate region of the rate-distortion plot, indicating that the majority of the bits are indeed allocated to imperceptible distortions.

Progressive generation We also run a progressive unconditional generation process given by progressive decompression from random bits. In other words, we predict the result of the reverse process, $\hat{\mathbf{x}}_0$, while sampling from the reverse process using Algorithm 4. Figures 3.6 and 6.2 show the resulting sample quality of $\hat{\mathbf{x}}_0$ over the course of the reverse process. Large scale image features appear first and details appear last. Figure 3.7 shows stochastic predictions $\mathbf{x}_0 \sim p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ with \mathbf{x}_t frozen for various t . When t is small, all but fine details are preserved, and when t is large, only large scale features are preserved. Perhaps these are hints of conceptual compression [41].

Connection to autoregressive decoding Note that the variational bound (3.5) can be rewritten as follows:

$$L = D_{\text{KL}}(q(\mathbf{x}_T) \parallel p(\mathbf{x}_T)) + \mathbb{E}_q \left[\sum_{t \geq 1} D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \right] + H(\mathbf{x}_0) \quad (3.16)$$



Figure 3.7: When conditioned on the same latent, CelebA-HQ 256×256 samples share high-level attributes. Bottom-right quadrants are \mathbf{x}_t , and other quadrants are samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$.

(See Section 6.1 for a derivation.) Now consider setting the diffusion process length T to the dimensionality of the data, defining the forward process so that $q(\mathbf{x}_t|\mathbf{x}_0)$ places all probability mass on \mathbf{x}_0 with the first t coordinates masked out (i.e. $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ masks out the t^{th} coordinate), setting $p(\mathbf{x}_T)$ to place all mass on a blank image, and, for the sake of argument, taking $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to be a fully expressive conditional distribution. With these choices, $D_{\text{KL}}(q(\mathbf{x}_T) \parallel p(\mathbf{x}_T)) = 0$, and minimizing $D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$ trains p_θ to copy coordinates $t + 1, \dots, T$ unchanged and to predict the t^{th} coordinate given $t + 1, \dots, T$. Thus, training p_θ with this particular diffusion is training an autoregressive model.

We can therefore interpret the Gaussian diffusion model (3.2) as a kind of autoregressive model with a generalized bit ordering that cannot be expressed by reordering data coordinates. Prior work has shown that such reorderings introduce inductive biases that have an impact on sample quality [86], so we speculate that the Gaussian diffusion serves a similar purpose, perhaps to greater effect since Gaussian noise might be more natural to add to images compared to masking noise. Moreover, the Gaussian diffusion length is not restricted to equal the data dimension; for instance, we use $T = 1000$, which is less than the dimension of the $32 \times 32 \times 3$ or $256 \times 256 \times 3$ images in our experiments. Gaussian diffusions can be made shorter for fast sampling or longer for model expressiveness.

3.4.4 Interpolation

We can interpolate source images $\mathbf{x}_0, \mathbf{x}'_0 \sim q(\mathbf{x}_0)$ in latent space using q as a stochastic encoder, $\mathbf{x}_t, \mathbf{x}'_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$, then decoding the linearly interpolated latent $\bar{\mathbf{x}}_t = (1 - \lambda)\mathbf{x}_0 + \lambda\mathbf{x}'_0$ into image space by the reverse process, $\bar{\mathbf{x}}_0 \sim p(\mathbf{x}_0|\bar{\mathbf{x}}_t)$. In effect, we use the reverse process to remove artifacts from linearly interpolating corrupted versions of the source images, as depicted in Fig. 3.8 (left). We fixed the noise for different values of λ so \mathbf{x}_t and \mathbf{x}'_t remain the same. Fig. 3.8 (right) shows



Figure 3.8: Interpolations of CelebA-HQ 256x256 images with 500 timesteps of diffusion.

interpolations and reconstructions of original CelebA-HQ 256×256 images ($t = 500$). The reverse process produces high-quality reconstructions, and plausible interpolations that smoothly vary attributes such as pose, skin tone, hairstyle, expression and background, but not eyewear. Larger t results in coarser and more varied interpolations, with novel samples at $t = 1000$ (Fig. 6.1).

3.5 Related Work

While diffusion models might resemble flows [14, 23, 24, 39, 52, 73, 114] and VAEs [75, 85, 112], diffusion models are designed so that q has no parameters and the top-level latent \mathbf{x}_T has nearly zero mutual information with the data \mathbf{x}_0 . Our ϵ -prediction reverse process parameterization establishes a connection between diffusion models and denoising score matching over multiple noise levels with annealed Langevin dynamics for sampling [132, 133]. Diffusion models, however, admit straightforward log likelihood evaluation, and the training procedure explicitly trains the Langevin dynamics sampler using variational inference (see Section 6.3 for details). The connection also has the reverse implication that a certain weighted form of denoising score matching is the same as variational inference to train a Langevin-like sampler. Other methods for learning transition operators of Markov chains include infusion training [8], variational walkback [38], generative stochastic networks [2], and others [79, 81, 93, 123, 131, 150].

By the known connection between score matching and energy-based modeling, our work could have implications for other recent work on energy-based models [21, 27, 32, 33, 40, 92, 152, 153, 154, 155]. Our rate-distortion curves are computed over time in one evaluation of the variational bound, reminiscent of how rate-distortion curves can be computed over distortion penalties in one run of annealed importance sampling [54]. Our progressive decoding argument can be seen in convolutional DRAW and related models [41, 91] and may also lead to more general designs for subscale orderings or sampling strategies for autoregressive models [86, 148].

3.6 Conclusion

We have presented high quality image samples using diffusion models, and we have found connections among diffusion models and variational inference for training Markov chains, denoising score matching and annealed Langevin dynamics (and energy-based models by extension), autoregressive models, and progressive lossy compression. Since diffusion models seem to have excellent inductive biases for image data, we look forward to investigating their utility in other data modalities and as components in other generative models and machine learning systems.

Broader Impact

Our work on diffusion models takes on a similar scope as existing work on other types of deep generative models, such as efforts to improve the sample quality of GANs, flows, autoregressive models, and so forth. Our paper represents progress in making diffusion models a generally useful tool in this family of techniques, so it may serve to amplify any impacts that generative models have had (and will have) on the broader world.

Unfortunately, there are numerous well-known malicious uses of generative models. Sample generation techniques can be employed to produce fake images and videos of high profile figures for political purposes. While fake images were manually created long before software tools were available, generative models such as ours make the process easier. Fortunately, CNN-generated images currently have subtle flaws that allow detection [146], but improvements in generative models may make this more difficult. Generative models also reflect the biases in the datasets on which they are trained. As many large datasets are collected from the internet by automated systems, it can be difficult to remove these biases, especially when the images are unlabeled. If samples from generative models trained on these datasets proliferate throughout the internet, then these biases will only be reinforced further.

On the other hand, diffusion models may be useful for data compression, which, as data becomes higher resolution and as global internet traffic increases, might be crucial to ensure accessibility of the internet to wide audiences. Our work might contribute to representation learning on unlabeled raw data for a large range of downstream tasks, from image classification to reinforcement learning, and diffusion models might also become viable for creative uses in art, photography, and music.

Acknowledgements

This work was supported by ONR PECASE and the NSF Graduate Research Fellowship under grant number DGE-1752814. Google’s TensorFlow Research Cloud (TFRC) provided Cloud TPUs.

4 Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis

Work by Ajay Jain, Matthew Tancik, and Pieter Abbeel

We present DietNeRF, a 3D neural scene representation estimated from a few images. Neural Radiance Fields (NeRF) learn a continuous volumetric representation of a scene through multi-view consistency, and can be rendered from novel viewpoints by ray casting. While NeRF has an impressive ability to reconstruct geometry and fine details given many images, up to 100 for challenging 360° scenes, it often finds a degenerate solution to its image reconstruction objective when only a few input views are available. To improve few-shot quality, we propose DietNeRF. We introduce an auxiliary semantic consistency loss that encourages realistic renderings at novel poses. DietNeRF is trained on individual scenes to (1) correctly render given input views from the same pose, and (2) match high-level semantic attributes across different, random poses. Our semantic loss allows us to supervise DietNeRF from arbitrary poses. We extract these semantics using a pre-trained visual encoder such as CLIP, a Vision Transformer trained on hundreds of millions of diverse single-view, 2D photographs mined from the web with natural language supervision. In experiments, DietNeRF improves the perceptual quality of few-shot view synthesis when learned from scratch, can render novel views with as few as one observed image when pre-trained on a multi-view dataset, and produces plausible completions of completely unobserved regions. Our project website is available at <https://www.ajayj.com/dietnerf>.

4.1 Introduction

In the novel view synthesis problem, we seek to rerender a scene from arbitrary viewpoint given a set of sparsely sampled viewpoints. View synthesis is a challenging problem that requires some degree of 3D reconstruction in addition to high-frequency texture synthesis. Recently, great

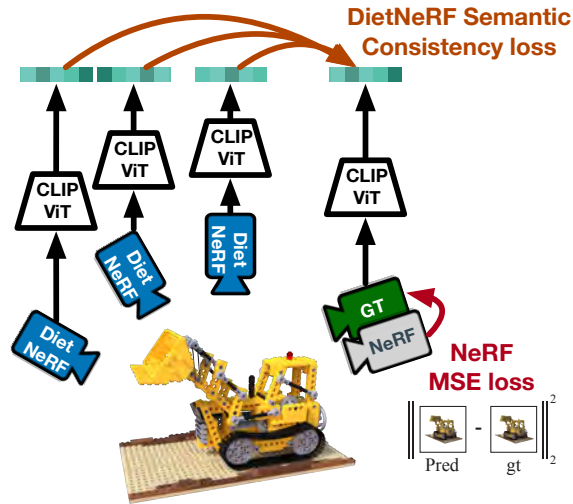


Figure 4.1: Neural Radiance Fields are trained to represent a scene by supervising renderings from the *same pose* as ground-truth observations (**MSE loss**). However, when only a few views are available, the problem is underconstrained. NeRF often finds degenerate solutions unless heavily regularized. Based on the principle that “**a bulldozer is a bulldozer from any perspective**”, our proposed DietNeRF supervises the radiance field from arbitrary poses (**DietNeRF cameras**). This is possible because we compute a **semantic consistency loss** in a feature space capturing high-level scene attributes, not in pixel space. We extract semantic representations of renderings using the CLIP Vision Transformer [108], then maximize similarity with representations of ground-truth views. In effect, we use prior knowledge about scene semantics learned by *single-view* 2D image encoders to constrain a 3D representation.

progress has been made on high-quality view synthesis when many observations are available. A popular approach is to use Neural Radiance Fields (NeRF) [89] to estimate a continuous neural scene representation from image observations. During training on a particular scene, the representation is rendered from observed viewpoints using volumetric ray casting to compute a reconstruction loss. At test time, NeRF can be rendered from novel viewpoints by the same procedure. While conceptually very simple, NeRF can learn high-frequency view-dependent scene appearances and accurate geometries that allow for high-quality rendering.

Still, NeRF is estimated per-scene, and cannot benefit from prior knowledge acquired from other images and objects. Because of the lack of prior knowledge, NeRF requires a large number of input views to reconstruct a given scene at high-quality. Given 8 views, Figure 4.2B shows that novel views rendered with the full NeRF model contain many artifacts because the optimization finds a degenerate solution that is only accurate at observed poses. We find that the core issue is that

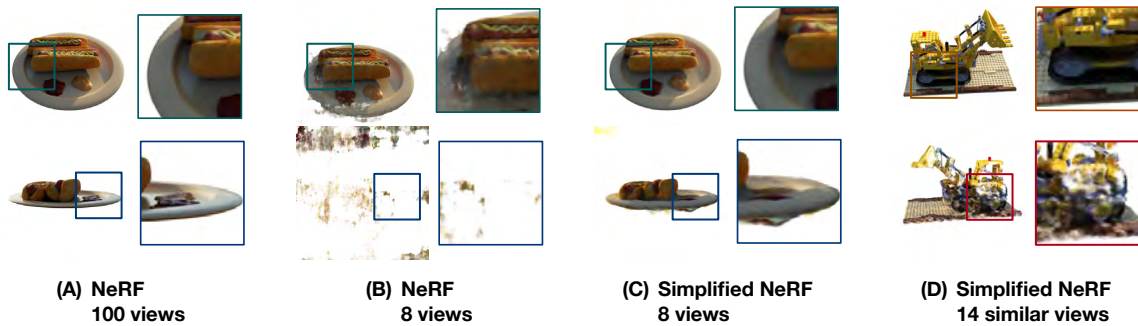


Figure 4.2: **Few-shot view synthesis is a challenging problem for Neural Radiance Fields.** (A) When we have 100 observations of an object from uniformly sampled poses, NeRF estimates a detailed and accurate representation that allows for high-quality view synthesis purely from multi-view consistency. (B) However, with only 8 views, the same NeRF overfits by placing the object in the near-field of the training cameras, leading to misplaced objects at poses near training cameras and degeneracies at novel poses. (C) We find that NeRF can converge when regularized, simplified, tuned and manually reinitialized, but no longer captures fine details. (D) Finally, without prior knowledge about similar objects, single-scene view synthesis cannot plausibly complete unobserved regions, such as the left side of an object seen from the right. In this work, we find that **these failures occur because NeRF is only supervised from the sparse training poses.**

prior 3D reconstruction systems based on rendering losses are *only supervised at known poses*, so they overfit when few poses are observed. Regularizing NeRF by simplifying the architecture avoids the worst artifacts, but comes at the cost of fine-grained detail.

Further, prior knowledge is needed when the scene reconstruction problem is underdetermined. 3D reconstruction systems struggle when regions of an object are never observed. This is particularly problematic when rendering an object at significantly different poses. When rendering a scene with an extreme baseline change, unobserved regions during training become visible. A view synthesis system should generate plausible missing details to fill in the gaps. Even a regularized NeRF learns poor extrapolations to unseen regions due to its lack of prior knowledge (Figure 4.2D).

Recent work trained NeRF on *multi-view* datasets of similar scenes [126, 136, 137, 145, 157] to bias reconstructions of novel scenes. Unfortunately, these models often produce blurry images due to uncertainty, or are restricted to a single object category such as ShapeNet classes as it is challenging to capture large, diverse, multi-view data.

In this work, we exploit the consistency principle that “*a bulldozer is a bulldozer from any perspective*”: objects share high-level semantic properties between their views. Image recognition

models learn to extract many such high-level semantic features including object identity. We transfer prior knowledge from pre-trained image encoders learned on highly diverse 2D *single-view* image data to the view synthesis problem. In the single-view setting, such encoders are frequently trained on millions of realistic images like ImageNet [20]. CLIP is a recent multi-modal encoder that is trained to match images with captions in a massive web scrape containing 400M images [108]. Due to the diversity of its data, CLIP showed promising zero- and few-shot transfer performance to image recognition tasks. We find that CLIP and ImageNet models also contain prior knowledge useful for novel view synthesis.

We propose DietNeRF, a neural scene representation based on NeRF that can be estimated from only a few photos, and can generate views with unobserved regions. In addition to minimizing NeRF’s mean squared error losses at known poses in pixel-space, DietNeRF penalizes a *semantic consistency* loss. This loss matches the final activations of CLIP’s Vision Transformer [25] between ground-truth images and rendered images at *different* poses, allowing us to supervise the radiance field from arbitrary poses. In experiments, we show that DietNeRF learns realistic reconstructions of objects with as few as 8 views without simplifying the underlying volumetric representation, and can even produce reasonable reconstructions of completely occluded regions. To generate novel views with as few as 1 observation, we fine-tune pixelNeRF [157], a generalizable scene representation, and improve perceptual quality.

4.2 Background on Neural Radiance Fields

A plenoptic function, or light field, is a five-dimensional function that describes the light radiating from every point in every direction in a volume such as a bounded scene. While explicitly storing or estimating the plenoptic function at high resolution is impractical due to the dimensionality of the input, Neural Radiance Fields [89] parameterize the function with a continuous neural network such as a multi-layer perceptron (MLP). A Neural Radiance Field (NeRF) model is a five-dimensional function $f_{\theta}(\mathbf{x}, \mathbf{d}) = (\mathbf{c}, \sigma)$ of spatial position $\mathbf{x} = (x, y, z)$ and viewing direction (θ, ϕ) , expressed as a 3D unit vector \mathbf{d} . NeRF predicts the RGB color \mathbf{c} and differential volume density σ from these inputs. To encourage view-consistency, the volume density only depends on \mathbf{x} , while the color also depends on viewing direction \mathbf{d} to capture viewpoint dependent effects like specular reflections. Images are rendered from a virtual camera at any position by integrating color along rays cast from the observer according to volume rendering [62]:

$$\mathbf{C}(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \quad (4.1)$$

where the ray originating at the camera origin \mathbf{o} follows path $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, and the transmittance $T(t) = \exp\left(-\int_{t_n}^{t_f} \sigma(\mathbf{r}(s))ds\right)$ weights the radiance by the probability that the ray travels from the image plane at t_n to t unobstructed. To approximate the integral, NeRF employs a hierarchical sampling algorithm to select function evaluation points near object surfaces along each ray. NeRF separately estimates two MLPs, a coarse network and a fine network, and uses the coarse network to guide sampling along the ray for more accurately estimating (4.1). The networks are trained from scratch on *each scene* given tens to hundreds of photos from various perspectives. Given observed multi-view training images $\{I_i\}$ of a scene, NeRF uses COLMAP SfM [125] to estimate camera extrinsics (rotations and origins) $\{\mathbf{p}_i\}$, creating a posed dataset $\mathcal{D} = \{(I_i, \mathbf{p}_i)\}$.

4.3 NeRF Struggles at Few-Shot View Synthesis

View synthesis is a challenging problem when a scene is only sparsely observed. Systems like NeRF that train on individual scenes especially struggle without prior knowledge acquired from similar scenes. We find that NeRF fails at few-shot novel view synthesis in several settings.

NeRF overfits to training views Conceptually, NeRF is trained by mimicking the image-formation process at observed poses. The radiance field can be estimated repeatedly sampling a training image and pose (I, \mathbf{p}_i) , rendering an image $\hat{I}_{\mathbf{p}_i}$ from the **same pose** by volume integration (4.1), then minimizing the mean-squared error (MSE) between the images, which should align pixel-wise:

$$\mathcal{L}_{\text{full}}(I, \hat{I}_{\mathbf{p}_i}) = \frac{1}{HW} \|I - \hat{I}_{\mathbf{p}_i}\|_2^2 \quad (4.2)$$

In practice, NeRF samples a smaller batch of rays across all training images to avoid the computational expense of rendering full images during training. Given subsampled rays \mathcal{R} cast from the training cameras, NeRF minimizes:

$$\mathcal{L}_{\text{MSE}}(\mathcal{R}) = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} \|\mathbf{C}(\mathbf{r}) - \hat{\mathbf{C}}(\mathbf{r})\|_2^2 \quad (4.3)$$

With many training views, \mathcal{L}_{MSE} provides training signal to f_θ densely in the volume and does not overfit to individual training views. Instead, the MLP recovers accurate textures and occupancy that allow interpolations to new views (Figure 4.2A). Radiance fields with sinusoidal positional embeddings are quite effective at learning high-frequency functions [136], which helps the MLP represent fine details.

Unfortunately, this high-frequency representational capacity allows NeRF to overfit to each input view when only a few are available. \mathcal{L}_{MSE} can be minimized by packing the reconstruction $\hat{I}_{\mathbf{p}}$ of training view (I, \mathbf{p}) close to the camera. Fundamentally, the plenoptic function representation suffers from a near-field ambiguity [161] where distant cameras each observe significant regions of space that no other camera observes. In this case, the optimal scene representation is underdetermined. Degenerate solutions can also exploit the view-dependence of the radiance field. Figure 4.2B shows novel views from the same NeRF trained on 8 views. While a rendered view from a pose near a training image has reasonable textures, it is skewed incorrectly and has cloudy artifacts from incorrect geometry. As the geometry is not estimated correctly, a distant view contains almost none of the correct information. High-opacity regions block the camera. Without supervision from any nearby camera, opacity is sensitive to random initialization.

Regularization fixes geometry, but hurts fine-detail High-frequency artifacts such as spurious opacity and rapidly varying colors can be avoided in some cases by regularizing NeRF. We simplify the NeRF architecture by removing hierarchical sampling and learning only a single MLP, and reducing the maximum frequency positional embedding in the input layer. This biases NeRF toward lower frequency solutions, such as placing content in the center of the scene farther from the training cameras. We also can address some few-shot optimization challenges by lowering the learning rate to improve initial convergence, and manually restarting training if renderings are degenerate. Figure 4.2C shows that these regularizers successfully allow NeRF to recover plausible object geometry. However, high-frequency, fine details are lost compared to 4.2A.

No prior knowledge, no generalization to unseen views As NeRF is estimated from scratch per-scene, it has no prior knowledge about natural objects such as common symmetries and object parts. In Figure 4.2D, we show that NeRF trained with 14 views of the right half of a Lego vehicle generalizes poorly to its left side. We regularized NeRF to remove high-opacity regions that originally blocked the left side entirely. Even so, the essential challenge is that NeRF receives no supervisory signal from \mathcal{L}_{MSE} to the unobserved regions, and instead relies on the inductive bias of the MLP for any inpainting. We would like to introduce prior knowledge that allows NeRF to exploit bilateral symmetry for plausible completions.

4.4 Semantically Consistent Radiance Fields

Motivated by these challenges, we introduce the DietNeRF scene representation. DietNeRF uses prior knowledge from a pre-trained image encoder to guide the NeRF optimization process in the few-shot setting.

4.4.1 Semantic consistency loss

DietNeRF supervises f_θ at arbitrary camera poses during training with a semantic loss. While pixel-wise comparison between ground-truth observed images and rendered images with \mathcal{L}_{MSE} is only useful when the rendered image is aligned with the observed pose, humans are easily able to detect whether two images are views of the same object from semantic cues. We can in general compare a *representation* of images captured from different viewpoints:

$$\mathcal{L}_{\text{SC}, \ell_2}(I, \hat{I}) = \frac{\lambda}{2} \|\phi(I) - \phi(\hat{I})\|_2^2 \quad (4.4)$$

If $\phi(x) = x$, Eq. (4.4) reduces to $\mathcal{L}_{\text{full}}$ up to a scaling factor. However, the identity mapping is view-dependent. We need a representation that is similar across views of the same object and captures important high-level semantic properties like object class. We evaluate the utility of two sources of supervision for representation learning. First, we experiment with the recent CLIP model pre-trained for multi-modal language and vision reasoning with contrastive learning [108]. We then evaluate visual classifiers pre-trained on labeled ImageNet images [25]. In both cases, we use similar Vision Transformer (ViT) architectures.

A Vision Transformer is appealing because its performance scales very well to large amounts of 2D data. Training on a large variety of images allows the network to encounter multiple views of an object class over the course of training without explicit multi-view data capture. It also allows us to transfer the visual encoder to diverse objects of interest in graphics applications, unlike prior class-specific reconstruction work that relies on homogeneous datasets [10, 65]. ViT extracts features from non-overlapping image patches in its first layer, then aggregates increasingly abstract representations with Transformer blocks based on global self-attention [142] to produce a single, global embedding vector. ViT outperformed CNN encoders in our early experiments.

In practice, CLIP produces normalized image embeddings. When $\phi(\cdot)$ is a unit vector, Eq. (4.4) simplifies to cosine similarity up to a constant and a scaling factor that can be absorbed into the loss weight λ :

$$\mathcal{L}_{\text{SC}}(I, \hat{I}) = \lambda \phi(I)^T \phi(\hat{I}) \quad (4.5)$$

Algorithm 7 Training DietNeRF on a single scene

```
1: Input: Observed views  $\mathcal{D} = \{(I, \mathbf{p})\}$ , semantic embedding function  $\phi(\cdot)$ , pose distribution  $\pi$ , consistency interval  $K$ , weight  $\lambda$ , rendering size, batch size  $|\mathcal{R}|$ , lr  $\eta_{it}$ 
2: Result: Trained Neural Radiance Field  $f_{\theta}(\cdot, \cdot)$ 
3: Initialize NeRF  $f_{\theta}(\cdot, \cdot)$ 
4: Pre-compute target embeddings  $\{\phi(I) : I \in \mathcal{D}\}$ 
5: for it from 1 to num_iters do
6:   Sample ray batch  $\mathcal{R}$ , ground-truth colors  $C(\cdot)$ 
7:   Render rays  $\hat{C}(\cdot)$  by (4.1)
8:    $\mathcal{L} \leftarrow \mathcal{L}_{\text{MSE}}(\mathcal{R}, C, \hat{C})$ 
9:   if it %  $K = 0$  then
10:     Sample target image, pose  $(I, \mathbf{p}) \sim \mathcal{D}$ 
11:     Sample source pose  $\hat{\mathbf{p}} \sim \pi$ 
12:     Render image  $\hat{I}$  from pose  $\hat{\mathbf{p}}$ 
13:      $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{\text{SC}}(I, \hat{I})$ 
14:   end if
15:   Update parameters:  $\theta \leftarrow \text{Adam}(\theta, \eta_{it}, \nabla_{\theta} \mathcal{L})$ 
16: end for
```

We refer to \mathcal{L}_{SC} (4.5) as a *semantic consistency* loss because it measures the similarity of high-level semantic features between observed and rendered views. In principle, semantic consistency is a very general loss that can be applied to any 3D reconstruction system based on differentiable rendering.

4.4.2 Interpreting representations across views

The pre-trained CLIP model that we use is trained on hundreds of millions of images with captions of varying detail. Image captions provide rich supervision for image representations. On one hand, short captions express semantically sparse learning signal as a flexible way to express labels [22]. For example, the caption “A photo of hotdogs” describes Fig. 4.2A. Language also provides semantically dense learning signal by describing object properties, relationships and appearances [22] such as the caption “Two hotdogs on a plate with ketchup and mustard”. To be predictive of such captions, an image representation must capture some high-level semantics that are stable across viewpoints. Concurrently, [35] found that CLIP representations capture visual attributes of images like art style and colors, as well as high-level semantic attributes including object tags and categories, facial expressions, typography, geography and brands.

In Figure 4.3, we measure the pairwise cosine similarity between CLIP representations of views circling an object. We find that pairs of views have highly similar CLIP representations, even for

diametrically opposing cameras. This suggests that large, diverse single-view datasets can induce useful representations for multi-view applications.

4.4.3 Pose sampling distribution

We augment the NeRF training loop with \mathcal{L}_{SC} minimization. Each iteration, we compute \mathcal{L}_{SC} between a random training image sampled from the observation dataset $I \sim \mathcal{D}$ and rendered image \hat{I}_p from random pose $\mathbf{p} \sim \pi$. For bounded scenes like NeRF’s Realistic Synthetic scenes where we are interested in 360° view synthesis, we define the pose sampling distribution π to be a uniform distribution over the upper hemisphere, with radius sampled uniformly in a bounded range. For unbounded forward-facing scenes or scenes where a pose sampling distribution is difficult to define, we interpolate between three randomly sampled known poses $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \sim \mathcal{D}$ with pairwise interpolation weights $\alpha_1, \alpha_2 \sim \mathcal{U}(0, 1)$.

4.4.4 Improving efficiency and quality

Volume rendering is computationally intensive. Computing a pixel’s color evaluates NeRF’s MLP f_θ at many points along a ray. To improve the efficiency of DietNeRF during training, we render images for semantic consistency at low resolution, requiring only 15-20% of the rays as a full resolution training image. Rays are sampled on a strided grid across the full extent of the image plane, ensuring that objects are mostly visible in each rendering. We found that sampling poses from a continuous distribution was helpful to avoid aliasing artifacts when training at a low resolution.

In experiments, we found that \mathcal{L}_{SC} converges faster than \mathcal{L}_{MSE} for many scenes. We hypothesize that the semantic consistency loss encourages DietNeRF to recover plausible scene geometry early in training, but is less helpful for reconstructing fine-grained details due to the relatively low dimensionality of the ViT representation $\phi(\cdot)$. We exploit the rapid convergence of \mathcal{L}_{SC} by only minimizing \mathcal{L}_{SC} every k iterations. DietNeRF is robust to the choice of k , but a value between 10 and 16 worked well in our experiments. StyleGAN2 [69] used a similar strategy for efficiency, referring to periodic application of a loss as *lazy regularization*.

As backpropagation through rendering is memory intensive with reverse-mode automatic differentiation, we render images for \mathcal{L}_{SC} with mixed precision computation and evaluate $\phi(\cdot)$ at half-precision. We delete intermediate MLP activations during rendering and rematerialize them during the backward pass [15, 57]. All experiments use a single 16 GB NVIDIA V100 or 11 GB 2080 Ti GPU.

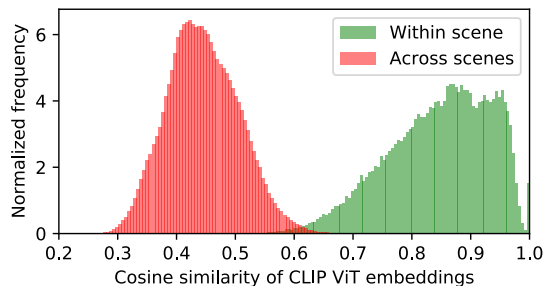


Figure 4.3: CLIP’s Vision Transformer learns low-dimensional image representations through language supervision. We find that these representations transfer well to multi-view 3D settings. We sample pairs of ground-truth views of the same scene and of different scenes from NeRF’s Realistic Synthetic object dataset, then compute a histogram of representation cosine similarity. Even though camera poses vary dramatically (views are sampled from the upper hemisphere), views within a scene have similar representations (**green**). Across scenes, representations have low similarity (**red**)

Since \mathcal{L}_{SC} converges before \mathcal{L}_{MSE} , we found it helpful to fine-tune DietNeRF with \mathcal{L}_{MSE} alone for 20-70k iterations to refine details. Alg. 7 details our overall training process.

4.5 Experiments

In experiments, we evaluate the quality of novel views synthesized by DietNeRF and baselines for both synthetically rendered objects and real photos of multi-object scenes. (1) We evaluate training *from scratch* on a specific scene with 8 views §4.5.1. (2) We show that DietNeRF improves perceptual quality of view synthesis from *only a single real photo* §4.5.2. (3) We find that DietNeRF can reconstruct regions that are never observed §4.5.3, and finally (4) run ablations §4.6.

Datasets The Realistic Synthetic benchmark of [88] includes detailed multi-view renderings of 8 realistic objects with view-dependent light transport effects. We also benchmark on the DTU multi-view stereo (MVS) dataset [59] used by pixelNeRF [157]. DTU is a challenging dataset that includes sparsely sampled real photos of physical objects.

Low-level full reference metrics Past work evaluates novel view quality with respect to ground-truth from the same pose with Peak Signal-to-Noise Ratio (**PSNR**) and Structural Similarity Index Measure (**SSIM**) [129]. PSNR expresses mean-squared error in log space. However, SSIM often disagrees with human judgements of similarity [162].

Table 4.1: Quality metrics for novel view synthesis on subsampled splits of the Realistic Synthetic dataset [89]. We randomly sample 8 views from the available 100 ground truth training views to evaluate how DietNeRF performs with limited observations.

| Method | PSNR \uparrow | SSIM \uparrow | LPIPS \downarrow | FID \downarrow | KID \downarrow |
|---|-----------------|-----------------|--------------------|------------------|------------------|
| NeRF | 14.934 | 0.687 | 0.318 | 228.1 | 0.076 |
| NV | 17.859 | 0.741 | 0.245 | 239.5 | 0.117 |
| Simplified NeRF | 20.092 | 0.822 | 0.179 | 189.2 | 0.047 |
| DietNeRF (ours) | 23.147 | 0.866 | 0.109 | 74.9 | 0.005 |
| DietNeRF, \mathcal{L}_{MSE} ft | 23.591 | 0.874 | 0.097 | 72.0 | 0.004 |
| NeRF, 100 views | 31.153 | 0.954 | 0.046 | 50.5 | 0.001 |

Perceptual metrics Deep CNN activations mirror aspects of human perception. NeRF measures perceptual image quality using **LPIPS** [162], which computes MSE between normalized features from all layers of a pre-trained VGG encoder [127]. Generative models also measure sample quality with feature space distances. The Fréchet Inception Distance (**FID**) [48] computes the Fréchet distance between Gaussian estimates of penultimate Inception v3 [135] features for real and fake images. However, FID is a biased metric at low sample sizes. We adopt the conceptually similar Kernel Inception Distance (**KID**), which measures the MMD between Inception features and has an unbiased estimator [7, 94]. All metrics use a different architecture and data than our CLIP ViT encoder.

4.5.1 Realistic Synthetic scenes from scratch

NeRF’s Realistic Synthetic dataset includes 8 detailed synthetic objects with 100 renderings from virtual cameras arranged randomly on a hemisphere pointed inward. To test few-shot performance, we *randomly* sample a training subset of 8 images from each scene. Table 4.1 shows results. The original NeRF model achieves much poorer quantitative quality with 8 images than with the full 100 image dataset. Neural Volumes [84] performs better as it tightly constrains the size of the scene’s bounding box and explicitly regularizes its scene representation using a penalty on spatial gradients of voxel opacity and a Beta prior on image opacity. This avoids the worst artifacts, but reconstructions are still low-quality. Simplifying NeRF and tuning it for each individual scene also regularizes the representation and helps convergence (+5.1 PSNR over the full NeRF). The best performance is achieved by regularizing with DietNeRF’s \mathcal{L}_{SC} loss. Additionally, fine-tuning with \mathcal{L}_{MSE} even further improves quality, for a total improvement of +8.5 PSNR, -0.2 LPIPS, and -156

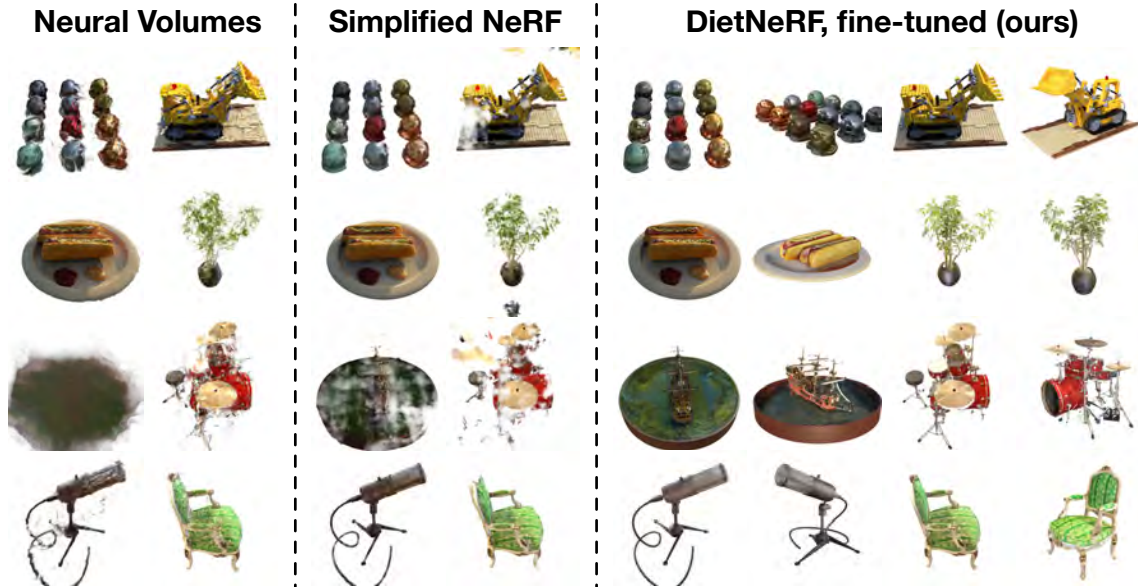


Figure 4.4: Novel views synthesized from eight observations of scenes in the Realistic Synthetic dataset.

FID over NeRF. This shows that semantic consistency is a valuable prior for high-quality few-shot view synthesis. Figure 4.4 visualizes results.

4.5.2 Single-view synthesis by fine-tuning

NeRF only uses observations during training, not inference, and uses no auxiliary data. Accurate 3D reconstruction from a single view is not possible purely from \mathcal{L}_{MSE} , so NeRF performs poorly in the single-view setting (Table 4.2).

To perform single- or few-shot view synthesis, pixelNeRF [157] learns a ResNet-34 encoder and a feature-conditioned neural radiance field on a multi-view dataset of similar scenes. The encoder learns priors that generalize to new single-view scenes. Table 4.2 shows that pixelNeRF significantly outperforms NeRF given a single photo of a held-out scene. However, novel views are blurry and unrealistic (Figure 4.5). We propose to fine-tune pixelNeRF on a single scene using \mathcal{L}_{MSE} alone or using both \mathcal{L}_{MSE} and \mathcal{L}_{SC} . Fine-tuning per-scene with MSE improves local image quality metrics, but only slightly helps perceptual metrics. Figure 4.6 shows that pixel-space MSE fine-tuning from one view mostly only improves quality for that view.

We refer to fine-tuning with both losses for a short period as DietPixelNeRF. Qualitatively, DietPixelNeRF has significantly sharper novel views (Fig. 4.5, 4.6). DietPixelNeRF outperforms



Figure 4.5: **Novel views synthesized from a *single input image*** from the DTU object dataset. Even with 3 input views, NeRF [89] fails to learn accurate geometry or textures (reprinted from [157]). While pixelNeRF [157] has mostly consistent object geometry as the camera pose is varied, renderings are blurry and contain artifacts like inaccurate placement of density along the observed camera’s z-axis. In contrast, fine-tuning with DietNeRF (DietPixelNeRF) learns realistic textures visually consistent with the input image, though some geometric defects are present due to the ambiguous nature of the view synthesis problem.

baselines on perceptual LPIPS, FID, and KID metrics (Tab. 4.2). For the very challenging single-view setting, ground-truth novel views will contain content that is completely occluded in the input. Because of uncertainty, blurry renderings will outperform sharp but incorrect renderings on average error metrics like MSE and PSNR. Arguably, perceptual quality and sharpness are better metrics than pixel error for graphics applications like photo editing and virtual reality as plausibility is emphasized.

4.5.3 Reconstructing unobserved regions

We evaluate whether DietNeRF produces plausible completions when the reconstruction problem is underdetermined. For training, we sample 14 nearby views of the right side of the Realistic Synthetic Lego scene (Fig. 4.7, right). Narrow baseline multi-view capture rigs are less costly than 360° captures, and support unbounded scenes. However, narrow-baseline observations suffer from occlusions: the left side of the Lego bulldozer is unobserved. NeRF fails to reconstruct this side of the scene, while our Simplified NeRF learns unrealistic deformations and incorrect colors (Fig. 4.7, left). Remarkably, DietNeRF learns quantitatively (Tab. 4.3) and qualitatively more accurate colors in the missing regions, suggesting the value of semantic image priors for sparse reconstruction

Table 4.2: **Single-view novel view synthesis on the DTU dataset.** NeRF and pixelNeRF PSNR, SSIM and LPIPS results are from [157]. Finetuning pixelNeRF with DietNeRF’s semantic consistency loss (DietPixelNeRF) improves perceptual quality measured by the deep perceptual LPIPS, FID and KID evaluation metrics, but can degrade PSNR and SSIM which are local pixel-aligned metrics due to geometric defects.

| Method | PSNR | SSIM | LPIPS | FID | KID |
|--|---------------|--------------|--------------|--------------|--------------|
| NeRF | 8.000 | 0.286 | 0.703 | — | — |
| pixelNeRF | 15.550 | 0.537 | 0.535 | 266.1 | 0.166 |
| pixelNeRF, \mathcal{L}_{MSE} ft | 16.048 | 0.564 | 0.515 | 265.2 | 0.159 |
| DietPixelNeRF | 14.242 | 0.481 | 0.487 | 190.7 | 0.066 |

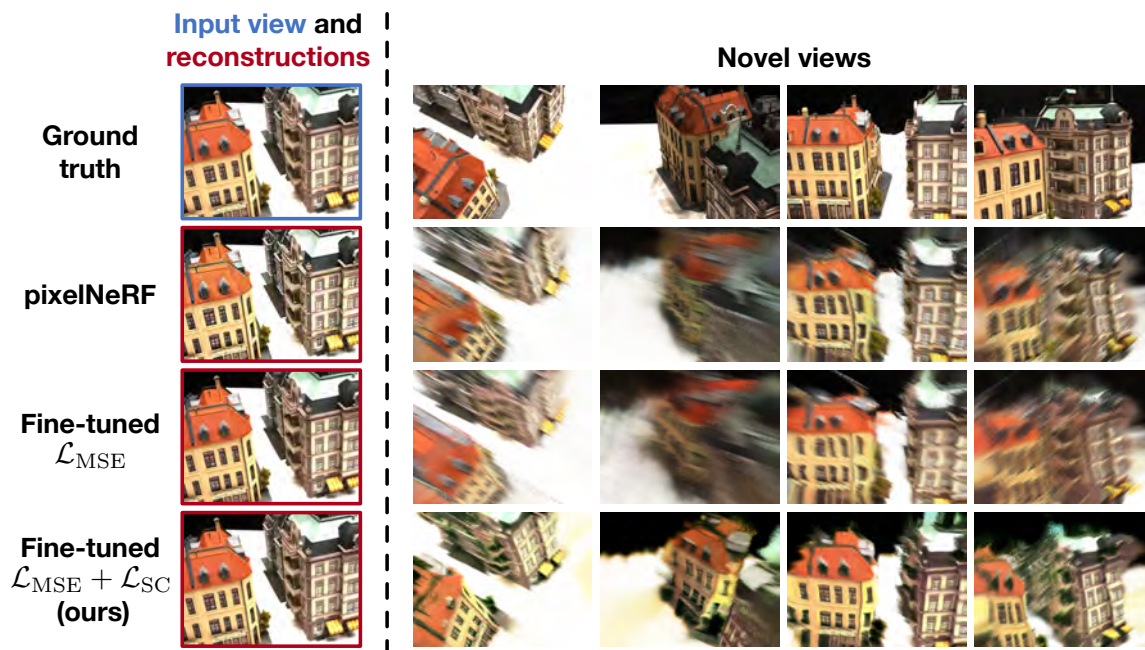


Figure 4.6: **Semantic consistency improves perceptual quality.** Fine-tuning pixelNeRF with \mathcal{L}_{MSE} slightly improves a rendering of the input view, but does not remove most perceptual flaws like blurriness in *novel views*. Fine-tuning with both \mathcal{L}_{MSE} and \mathcal{L}_{SC} (DietPixelNeRF, bottom) improves sharpness of all views.

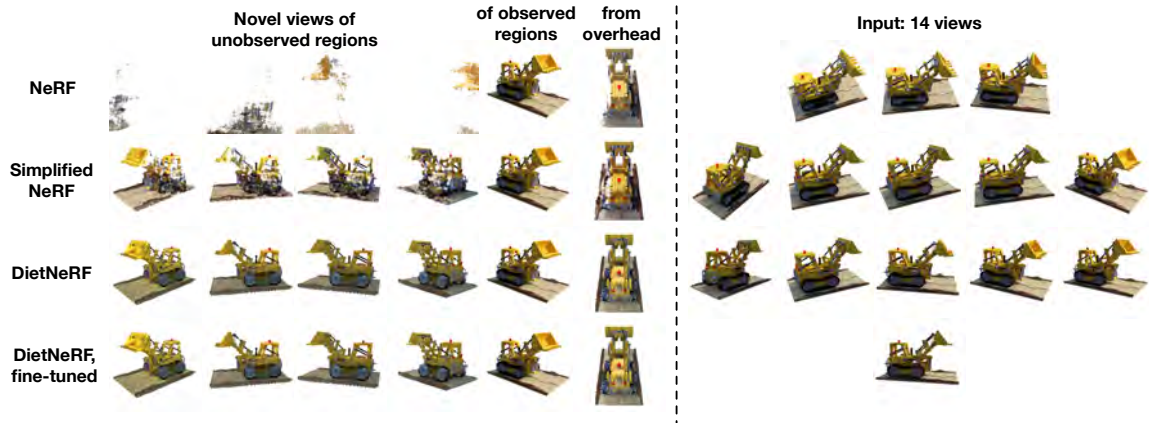


Figure 4.7: **Renderings of occluded regions during training.** 14 images of the right half of the Realistic Synthetic lego scene are used to estimate radiance fields. NeRF either learns high-opacity occlusions blocking the left of the object, or fails to generalize properly to the unseen left side. In contrast, DietNeRF fills in details for a reconstruction that is mostly consistent with the observed half.

Table 4.3: **Extrapolation metrics.** Novel view synthesis with observations of **only one side** of the Realistic Synthetic Lego scene.

| Views | Method | PSNR \uparrow | SSIM \uparrow | LPIPS \downarrow |
|-------|-----------------------------------|-----------------|-----------------|--------------------|
| 14 | NeRF | 19.662 | 0.799 | 0.202 |
| 14 | Simplified NeRF | 21.553 | 0.818 | 0.160 |
| 14 | DietNeRF (ours) | 20.753 | 0.810 | 0.157 |
| 14 | DietNeRF + \mathcal{L}_{MSE} ft | 22.211 | 0.824 | 0.143 |
| 100 | NeRF [89] | 31.618 | 0.965 | 0.033 |

problems. We exclude FID and KID since a single scene has too few samples for an accurate estimate.

4.6 Ablations

Choosing an image encoder Table 4.4 shows quality metrics with different semantic encoder architectures and pre-training datasets. We evaluate on the Lego scene with 8 views. Large ViT models (ViT L) do not improve results over the base ViT B. Fixing the architecture, CLIP offers a +1.8 PSNR improvement over an ImageNet model, suggesting that data diversity and language

Table 4.4: **Ablating supervision and architectural parameters for the ViT image encoder $\phi(\cdot)$ used to compare image features.** Metrics are measured on the Realistic Synthetic Lego scene.

| Semantic image encoder | PSNR \uparrow | SSIM \uparrow | LPIPS \downarrow |
|----------------------------|-----------------|-----------------|--------------------|
| ImageNet ViT L/16, 384^2 | 21.501 | 0.809 | 0.167 |
| ImageNet ViT L/32, 384^2 | 20.498 | 0.801 | 0.174 |
| ImageNet ViT B/32, 224^2 | 22.059 | 0.836 | 0.131 |
| CLIP ViT B/32, 224^2 | 23.896 | 0.863 | 0.110 |

Table 4.5: **Varying the number of iterations that DietNeRF is fine-tuned with \mathcal{L}_{MSE} on Realistic Synthetic scenes.** All models are initially trained for 200k iterations with \mathcal{L}_{MSE} and \mathcal{L}_{SC} . Further minimizing \mathcal{L}_{MSE} is helpful, but the model can overfit.

| Method | PSNR \uparrow | SSIM \uparrow | LPIPS \downarrow |
|--|-----------------|-----------------|--------------------|
| DietNeRF, no fine-tuning | 23.147 | 0.866 | 0.109 |
| DietNeRF, \mathcal{L}_{MSE} ft 10k iters | 23.524 | 0.872 | 0.101 |
| DietNeRF, \mathcal{L}_{MSE} ft 50k iters | 23.591 | 0.874 | 0.097 |
| DietNeRF, \mathcal{L}_{MSE} ft 100k iters | 23.521 | 0.874 | 0.097 |
| DietNeRF, \mathcal{L}_{MSE} ft 200k iters | 23.443 | 0.872 | 0.098 |

supervision is helpful for 3D tasks. Still, both induce useful representations that transfer to view synthesis.

Varying \mathcal{L}_{MSE} fine-tuning duration Fine-tuning DietNeRF with \mathcal{L}_{MSE} can improve quality by better reconstructing fine-details. In Table 4.5, we vary the number of iterations of fine-tuning for the Realistic Synthetic scenes with 8 views. Fine-tuning for up to 50k iterations is helpful, but reduces performance with longer optimization. It is possible that the model starts overfitting to the 8 input views.

4.7 Related work

Few-shot radiance fields Several works condition NeRF on latent codes describing scene geometry or appearance rather than estimating NeRF per scene [126, 137, 157]. An image encoder and radiance field decoder are learned on a multi-view dataset of similar objects or scenes ahead of time. At test time, on a new scene, novel viewpoints are rendered using the decoder conditioned on encodings of a few observed images. GRAF renders patches of the scene every iteration to

supervise the network with a discriminator [126]. Concurrent to our work, IBRNet [145] also fine-tunes a latent-conditioned radiance field on a specific scene using NeRF’s reconstruction loss, but needed at least 50 views. Rather than generalizing between scenes through a shared encoder and decoder, [31, 136] meta-learn radiance field weights that can be adapted to a specific scene in a few gradient steps. Meta-learning improves performance in the few-view setting. Similarly, a signed distance field can be meta-learned for shape representation problems [128]. Much literature studies single-view reconstruction with other, explicit 3D representations. Notable recent examples include voxel [138], mesh [53] and point-cloud [149] approaches.

Novel view synthesis, image-based rendering Neural Volumes [84] proposes a VAE [75, 113] encoder-decoder architecture to predict a volumetric representation of a scene from posed image observations. NV uses priors as auxiliary objectives like DietNeRF, but penalizes opacity based on geometric intuitions rather than RGB image semantics. TBNs [95] learn an autoencoder with a 3-dimensional latent that can be rotated to render new perspectives for a single-category. SRNs [129] fit a continuous representation to a scene and also generalize to novel single-category objects if trained on a large multi-view dataset. It can be extended to predict per-point semantic segmentation maps [77]. Local Light Field Fusion [88] estimates and blends multiple MPI representations for each scene. Free View Synthesis [115] uses geometric approaches to improve view synthesis in unbounded in-the-wild scenes. NeRF++ [161] also improves unbounded scenes using multiple NeRF models and changing NeRF’s parameterization.

Semantic representation learning Representation learning with deep supervised and unsupervised approaches has a long history [5]. Without labels, generative models can learn useful representations for recognition [13], but self-supervised models like CPC [47, 100] tend to be more parameter efficient. Contrastive methods including CLIP learn visual representations by matching similar pairs of items, such as captions and images [60, 108], augmented variants of an image [16], or video patches across frames [56].

4.8 Conclusions

Our results suggest that single-view 2D representations transfer effectively to challenging, underconstrained 3D reconstruction problems such as volumetric novel view synthesis. While pre-trained image encoder representations have certainly been transferred to 3D vision applications in the past by fine-tuning, the recent emergence of visual models trained on enormous 100M+ image datasets like CLIP have enabled surprisingly effective few-shot transfer. We exploited this

transferrable prior knowledge to solve optimization issues as well as to cope with partial observability in the NeRF family of scene representations, offering notable improvements in perceptual quality. In the future, we believe “diet-friendly” few-shot transfer will play a greater role in a wide range of 3D applications.

Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under grant number DGE-1752814 and by Berkeley Deep Drive. We would like to thank Paras Jain, Aditi Jain, Alexei Efros, Angjoo Kanazawa, Aravind Srinivas, Deepak Pathak and Alex Yu for feedback and discussions.

Bibliography

1. N. Akoury and A. Nguyen. “Spatial PixelCNN: Generating Images from Patches”. In: *arXiv:1712.00714*, 2017.
2. G. Alain, Y. Bengio, L. Yao, J. Yosinski, E. Thibodeau-Laufer, S. Zhang, and P. Vincent. “GSNs: generative stochastic networks”. In: *Information and Inference: A Journal of the IMA* 5:2, 2016, pp. 210–249.
3. C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. “PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing”. In: *ACM TOG* 28:3, 2009.
4. Y. Bengio and S. Bengio. “Modeling high-dimensional discrete data with multi-layer neural networks”. In: *NIPS*. 2000.
5. Y. Bengio, A. Courville, and P. Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35:8, 2013, pp. 1798–1828.
6. Y. Bengio, E. Laufer, G. Alain, and J. Yosinski. “Deep generative stochastic networks trainable by backprop”. In: *ICML*. 2014.
7. M. Bińkowski, D.J. Sutherland, M. Arbel, and A. Gretton. “Demystifying MMD GANs”. In: *International Conference on Learning Representations*. 2018. URL: <https://openreview.net/forum?id=r11UOzWCW>.
8. F. Bordes, S. Honari, and P. Vincent. “Learning to Generate Samples from Noise through Infusion Training”. In: *International Conference on Learning Representations*. 2017.
9. A. Brock, J. Donahue, and K. Simonyan. “Large Scale GAN Training for High Fidelity Natural Image Synthesis”. In: *International Conference on Learning Representations*. 2019.
10. T. J. Cashman and A. W. Fitzgibbon. “What Shape Are Dolphins? Building 3D Morphable Models from 2D Images.” In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35:1, 2013, pp. 232–244. URL: <http://dblp.uni-trier.de/db/journals/pami/pami35.html#CashmanF13>.
11. J. Červený. *Generalized Hilbert space-filling curve*. 2019.

12. T. Che, R. Zhang, J. Sohl-Dickstein, H. Larochelle, L. Paull, Y. Cao, and Y. Bengio. “Your GAN is Secretly an Energy-based Model and You Should use Discriminator Driven Latent Sampling”. In: *arXiv preprint arXiv:2003.06060*, 2020.
13. M. Chen, A. Radford, R. Child, J. Wu, H. Jun, P. Dhariwal, D. Luan, and I. Sutskever. “Generative Pretraining from Pixels”. In: 2020.
14. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. “Neural ordinary differential equations”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 6571–6583.
15. T. Chen, B. Xu, C. Zhang, and C. Guestrin. *Training Deep Nets with Sublinear Memory Cost*. 2016. arXiv: 1604.06174 [cs.LG].
16. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. “A Simple Framework for Contrastive Learning of Visual Representations”. In: *arXiv preprint arXiv:2002.05709*, 2020.
17. X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel. “PixelSNAIL: An Improved Autoregressive Generative Model”. In: 2018.
18. X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel. “PixelSNAIL: An Improved Autoregressive Generative Model”. In: *International Conference on Machine Learning*. 2018, pp. 863–871.
19. R. Child, S. Gray, A. Radford, and I. Sutskever. “Generating long sequences with sparse transformers”. In: *arXiv*, 2019.
20. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
21. Y. Deng, A. Bakhtin, M. Ott, A. Szlam, and M. Ranzato. “Residual energy-based models for text generation”. In: *arXiv preprint arXiv:2004.11714*, 2020.
22. K. Desai and J. Johnson. “VirTex: Learning Visual Representations from Textual Annotations”. In: *CVPR*. 2021.
23. L. Dinh, D. Krueger, and Y. Bengio. “NICE: Non-linear independent components estimation”. In: *arXiv preprint arXiv:1410.8516*, 2014.
24. L. Dinh, J. Sohl-Dickstein, and S. Bengio. “Density estimation using Real NVP”. In: *ICLR*. 2016.

25. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2021. URL: <https://openreview.net/forum?id=YicbFdNTTy>.
26. Y. Du and I. Mordatch. “Implicit Generation and Modeling with Energy Based Models”. In: *NeurIPS*, 2019.
27. Y. Du and I. Mordatch. “Implicit Generation and Modeling with Energy Based Models”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 3603–3613.
28. A. A. Efros and T. K. Leung. “Texture synthesis by non-parametric sampling”. In: *ICCV*. 1999.
29. P. Esser, R. Rombach, and B. Ommer. *Taming Transformers for High-Resolution Image Synthesis*. 2020. arXiv: 2012.09841 [cs.CV].
30. B. J. Frey. *Graphical models for machine learning and digital communication*. MIT Press, 1998.
31. C. Gao, Y. Shih, W.-S. Lai, C.-K. Liang, and J.-B. Huang. “Portrait Neural Radiance Fields from a Single Image”. In: *arXiv preprint arXiv:2012.05903*, 2020.
32. R. Gao, Y. Lu, J. Zhou, S.-C. Zhu, and Y. Nian Wu. “Learning generative ConvNets via multi-grid modeling and sampling”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9155–9164.
33. R. Gao, E. Nijkamp, D. P. Kingma, Z. Xu, A. M. Dai, and Y. N. Wu. “Flow contrastive estimation of energy-based models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 7518–7528.
34. M. Germain, K. Gregor, I. Murray, and H. Larochelle. “MADE: Masked autoencoder for distribution estimation”. In: *ICML*. 2015, pp. 881–889.
35. G. Goh, N. Cammarata, C. Voss, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah. “Multimodal Neurons in Artificial Neural Networks”. In: *Distill*, 2021. <https://distill.pub/2021/multimodal-neurons>. DOI: 10.23915/distill.00030.
36. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative Adversarial Nets”. In: *NIPS*. 2014.
37. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014. URL: <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>.

38. A. Goyal, N. R. Ke, S. Ganguli, and Y. Bengio. “Variational walkback: Learning a transition operator as a stochastic recurrent net”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4392–4402.
39. W. Grathwohl, R. T. Q. Chen, J. Bettencourt, and D. Duvenaud. “FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models”. In: *International Conference on Learning Representations*. 2019.
40. W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, M. Norouzi, and K. Swersky. “Your classifier is secretly an energy based model and you should treat it like one”. In: *International Conference on Learning Representations*. 2020.
41. K. Gregor, F. Besse, D. J. Rezende, I. Danihelka, and D. Wierstra. “Towards conceptual compression”. In: *Advances In Neural Information Processing Systems*. 2016, pp. 3549–3557.
42. K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra. “Deep AutoRegressive Networks”. In: *ICML*. Vol. 32. 2014.
43. A. Griewank and A. Walther. “Algorithm 799: revolve”. In: *ACM TOMS*, 2000.
44. P. Harsha, R. Jain, D. McAllester, and J. Radhakrishnan. “The communication complexity of correlation”. In: *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC’07)*. IEEE. 2007, pp. 10–23.
45. M. Havasi, R. Peharz, and J. M. Hernández-Lobato. “Minimal Random Code Learning: Getting Bits Back from Compressed Model Parameters”. In: *International Conference on Learning Representations*. 2019.
46. J. Hays and A. A. Efros. “Scene Completion Using Millions of Photographs”. In: *ACM TOG (SIGGRAPH)* 26:3, 2007.
47. O. J. Henaff, A. Srinivas, J. D. Fauw, A. Razavi, C. Doersch, S. M. A. Eslami, and A. van den Oord. *Data-Efficient Image Recognition with Contrastive Predictive Coding*. 2020. URL: <https://openreview.net/forum?id=rJerHlrYwH>.
48. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Curran Associates Inc., Long Beach, California, USA, 2017, pp. 6629–6640. ISBN: 9781510860964.
49. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. “GANs trained by a two time-scale update rule converge to a local Nash equilibrium”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6626–6637.

50. I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations*. 2017.
51. G. E. Hinton. "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14:8, 2002.
52. J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design". In: *International Conference on Machine Learning*. 2019.
53. R. Hu and D. Pathak. "Worldsheet: Wrapping the World in a 3D Sheet for View Synthesis from a Single Image". In: *arXiv preprint arXiv:2012.09854*, 2020.
54. S. Huang, A. Makhzani, Y. Cao, and R. Grosse. "Evaluating Lossy Compression Rates of Deep Generative Models". In: *International Conference on Machine Learning*. 2020.
55. P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks". In: *CVPR*, 2017.
56. A. Jabri, A. Owens, and A. A. Efros. "Space-Time Correspondence as a Contrastive Random Walk". In: *Advances in Neural Information Processing Systems*, 2020.
57. P. Jain, A. Jain, A. Nrusimha, A. Gholami, P. Abbeel, J. Gonzalez, K. Keutzer, and I. Stoica. "Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization". In: *Proceedings of Machine Learning and Systems*. Ed. by I. Dhillon, D. Papailiopoulos, and V. Sze. Vol. 2. 2020, pp. 497–511. URL: <https://proceedings.mlsys.org/paper/2020/file/084b6fbb10729ed4da8c3d3f5a3ae7c9-Paper.pdf>.
58. P. Jain, A. Jain, A. Nrusimha, A. Gholami, P. Abbeel, K. Keutzer, I. Stoica, and J. E. Gonzalez. "Checkmate: Breaking the memory wall with optimal tensor rematerialization". In: 2020.
59. R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanæs. "Large Scale Multi-view Stereopsis Evaluation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 406–413. DOI: 10.1109/CVPR.2014.59.
60. C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. V. Le, Y. Sung, Z. Li, and T. Duerig. "Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision". In: *arXiv preprint arXiv:2102.05918*, 2021.
61. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional architecture for fast feature embedding". In: *ACM MM*. 2014.

62. J. T. Kajiya and B. P. Von Herzen. “Ray Tracing Volume Densities”. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’84. Association for Computing Machinery, New York, NY, USA, 1984, pp. 165–174. ISBN: 0897911385. DOI: 10.1145/800031.808594. URL: <https://doi.org/10.1145/800031.808594>.
63. N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. v. d. Oord, S. Dieleman, and K. Kavukcuoglu. “Efficient Neural Audio Synthesis”. In: *International Conference on Machine Learning*. 2018, pp. 2410–2419.
64. N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. “Video Pixel Networks”. In: *International Conference on Machine Learning*. 2017, pp. 1771–1779.
65. A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik. “Learning Category-Specific Mesh Reconstruction from Image Collections”. In: *ECCV*. 2018.
66. T. Karras, T. Aila, S. Laine, and J. Lehtinen. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: *ICLR*. 2018.
67. T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila. “Training Generative Adversarial Networks with Limited Data”. In: *arXiv preprint arXiv:2006.06676v1*, 2020.
68. T. Karras, S. Laine, and T. Aila. “A style-based generator architecture for generative adversarial networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4401–4410.
69. T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. “Analyzing and Improving the Image Quality of StyleGAN”. In: *Proc. CVPR*. 2020.
70. T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. “Analyzing and improving the image quality of StyleGAN”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8110–8119.
71. D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: <http://arxiv.org/abs/1412.6980>.
72. D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations*. 2015.
73. D. P. Kingma and P. Dhariwal. “Glow: Generative flow with invertible 1x1 convolutions”. In: *NeurIPS*. 2018.

74. D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. “Improved variational inference with inverse autoregressive flow”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4743–4751.
75. D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. In: *ICLR*. Vol. 1. 2014.
76. D. P. Kingma, M. Welling, et al. “An introduction to variational autoencoders”. In: *Foundations and Trends in Machine Learning* 12:4, 2019, pp. 307–392.
77. A. Kohli, V. Sitzmann, and G. Wetzstein. *Semantic Implicit Neural Scene Representations With Semi-Supervised Training*. 2021. arXiv: 2003.12673 [cs.CV].
78. H. Larochelle and I. Murray. “The neural autoregressive distribution estimator”. In: *AISTATS*. 2011, pp. 29–37.
79. J. Lawson, G. Tucker, B. Dai, and R. Ranganath. “Energy-Inspired Models: Learning with Sampler-Induced Distributions”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 8501–8513.
80. Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. “A tutorial on energy-based learning”. In: 2006.
81. D. Levy, M. D. Hoffman, and J. Sohl-Dickstein. “Generalizing Hamiltonian Monte Carlo with Neural Networks”. In: *International Conference on Learning Representations*. 2018.
82. B. Li, F. Wu, K. Q. Weinberger, and S. Belongie. “Positional normalization”. In: *NeurIPS*. 2019.
83. G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. “Image Inpainting for Irregular Holes Using Partial Convolutions”. In: *ECCV*. 2018.
84. S. Lombardi, T. Simon, J. Saragih, G. Schwartz, A. Lehrmann, and Y. Sheikh. “Neural Volumes: Learning Dynamic Renderable Volumes from Images”. In: *ACM Trans. Graph.* 38:4, 2019, 65:1–65:14.
85. L. Maaløe, M. Fraccaro, V. Liévin, and O. Winther. “BIVA: A very deep hierarchy of latent variables for generative modeling”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 6548–6558.
86. J. Menick and N. Kalchbrenner. “Generating high fidelity images with subscale pixel networks and multidimensional upscaling”. In: *ICLR*. 2019.
87. T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur. “Recurrent neural network based language model”. In: *INTERSPEECH*. 2010.

88. B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar. "Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines". In: *ACM Transactions on Graphics (TOG)*, 2019.
89. B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. arXiv: 2003.08934 [cs.CV].
90. T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. "Spectral Normalization for Generative Adversarial Networks". In: *International Conference on Learning Representations*. 2018.
91. A. Nichol. "VQ-DRAW: A Sequential Discrete VAE". In: *arXiv preprint arXiv:2003.01599*, 2020.
92. E. Nijkamp, M. Hill, T. Han, S.-C. Zhu, and Y. N. Wu. "On the anatomy of MCMC-based maximum likelihood learning of energy-based models". In: *arXiv preprint arXiv:1903.12370*, 2019.
93. E. Nijkamp, M. Hill, S.-C. Zhu, and Y. N. Wu. "Learning non-convergent non-persistent short-run MCMC toward energy-based model". In: *NeurIPS*. 2019, pp. 5232–5242.
94. A. Obukhov, M. Seitzer, P.-W. Wu, S. Zhydenko, J. Kyl, and E. Y.-J. Lin. *High-fidelity performance metrics for generative models in PyTorch*. Version v0.2.0. Version: 0.2.0, DOI: 10.5281/zenodo.3786540. 2020. DOI: 10.5281/zenodo.3786540. URL: <https://github.com/toshas/torch-fidelity>.
95. K. Olszewski, S. Tulyakov, O. Woodford, H. Li, and L. Luo. "Transformable Bottleneck Networks". In: *The IEEE International Conference on Computer Vision (ICCV)*, 2019.
96. A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. "Wavenet: A generative model for raw audio". In: *arXiv:1609.03499*, 2016.
97. A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. "Conditional image generation with PixelCNN decoders". In: *NeurIPS*. 2016.
98. A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. "Pixel Recurrent Neural Networks". In: *ICML*. 2016.
99. A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. "Conditional image generation with PixelCNN decoders". In: *Advances in Neural Information Processing Systems*. 2016, pp. 4790–4798.

100. A. van den Oord, Y. Li, and O. Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG].
101. A. van den Oord, O. Vinyals, and K. Kavukcuoglu. “Neural discrete representation learning”. In: *NIPS*. 2017.
102. G. Ostrovski, W. Dabney, and R. Munos. “Autoregressive Quantile Networks for Generative Modeling”. In: *International Conference on Machine Learning*. 2018, pp. 3936–3945.
103. N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. “Image Transformer”. In: *ICML*. 2018.
104. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. “PyTorch: An imperative style, high-performance deep learning library”. In: *NeurIPS*. 2019, pp. 8024–8035.
105. D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. “Context encoders: Feature learning by Inpainting”. In: *CVPR*. 2016.
106. K. Popat and R. W. Picard. “Novel cluster-based probability model for texture synthesis, classification, and compression”. In: *VCIP*. Vol. 2094. 1993, pp. 756–768.
107. R. Prenger, R. Valle, and B. Catanzaro. “WaveGlow: A flow-based generative network for speech synthesis”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 3617–3621.
108. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].
109. T. Raiko, Y. Li, K. Cho, and Y. Bengio. “Iterative neural autoregressive distribution estimator NADE-k”. In: *NIPS*. 2014.
110. A. Razavi, A. van den Oord, and O. Vinyals. “Generating diverse high-fidelity images with VQ-VAE-2”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 14837–14847.
111. S. Reed, A. van den Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. de Freitas. “Parallel Multiscale Autoregressive Density Estimation”. In: *ICML*. Sydney, NSW, Australia, 2017.
112. D. J. Rezende, S. Mohamed, and D. Wierstra. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”. In: *ICML*. 2014.

113. D.J. Rezende, S. Mohamed, and D. Wierstra. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by E. P. Xing and T. Jebara. Vol. 32. Proceedings of Machine Learning Research 2. PMLR, Beijing, China, 2014, pp. 1278–1286. URL: <http://proceedings.mlr.press/v32/rezende14.html>.
114. D. Rezende and S. Mohamed. “Variational Inference with Normalizing Flows”. In: *ICML*. 2015.
115. G. Riegler and V. Koltun. “Free View Synthesis”. In: *Computer Vision – ECCV 2020*. Ed. by A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm. Springer International Publishing, Cham, 2020, pp. 623–640. ISBN: 978-3-030-58529-7.
116. O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional networks for biomedical image segmentation”. In: 2015.
117. O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241.
118. T. Rott Shaham, T. Dekel, and T. Michaeli. “SinGAN: Learning a Generative Model from a Single Natural Image”. In: *Computer Vision (ICCV), IEEE International Conference on*. 2019.
119. R. Salakhutdinov and I. Murray. “On the quantitative analysis of deep belief networks”. In: *ICML*. 2008.
120. T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. “Improved techniques for training gans”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.
121. T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. “PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications”. In: *International Conference on Learning Representations*. 2017.
122. T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. “PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications”. In: *ICLR*. 2017.
123. T. Salimans, D. Kingma, and M. Welling. “Markov Chain Monte Carlo and variational inference: Bridging the gap”. In: *International Conference on Machine Learning*. 2015, pp. 1218–1226.

124. T. Salimans and D.P. Kingma. “Weight normalization: A simple reparameterization to accelerate training of deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 901–909.
125. J.L. Schonberger and J.-M. Frahm. “Structure-From-Motion Revisited”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
126. K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger. *GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis*. 2020. arXiv: 2007.02442 [cs.CV].
127. K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations*. 2015.
128. V. Sitzmann, E. R. Chan, R. Tucker, N. Snavely, and G. Wetzstein. *MetaSDF: Meta-learning Signed Distance Functions*. 2020. arXiv: 2006.09662 [cs.CV].
129. V. Sitzmann, M. Zollhöfer, and G. Wetzstein. “Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations”. In: *Advances in Neural Information Processing Systems*. 2019.
130. J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. In: *ICML*. 2015, pp. 2256–2265.
131. J. Song, S. Zhao, and S. Ermon. “A-NICE-MC: Adversarial training for MCMC”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5140–5150.
132. Y. Song and S. Ermon. “Generative modeling by estimating gradients of the data distribution”. In: *NeurIPS*. 2019.
133. Y. Song and S. Ermon. “Improved Techniques for Training Score-Based Generative Models”. In: *arXiv preprint arXiv:2006.09011*, 2020.
134. Y. Song, C. Meng, R. Liao, and S. Ermon. *Nonlinear Equation Solving: A Faster Alternative to Feedforward Computation*. 2020. arXiv: 2002.03629 [cs.LG].
135. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. “Rethinking the Inception Architecture for Computer Vision”. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016. URL: <http://arxiv.org/abs/1512.00567>.
136. M. Tancik, B. Mildenhall, T. Wang, D. Schmidt, P.P. Srinivasan, J. T. Barron, and R. Ng. *Learned Initializations for Optimizing Coordinate-Based Neural Representations*. 2020. arXiv: 2012.02189 [cs.CV].
137. A. Trevithick and B. Yang. “GRF: Learning a General Radiance Field for 3D Scene Representation and Rendering”. In: *arXiv:2010.04595*. 2020.

138. S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik. “Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2017.
139. B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. “Neural Autoregressive Distribution Estimation”. In: *JMLR*, 2016.
140. B. Uria, I. Murray, and H. Larochelle. “A deep and tractable density estimator”. In: *ICML*. 2014.
141. B. Uria, I. Murray, and H. Larochelle. “RNADE: The real-valued neural autoregressive density-estimator”. In: *NIPS*. 2013.
142. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.
143. P. Vincent. “A connection between score matching and denoising autoencoders”. In: *Neural Computation* 23:7, 2011, pp. 1661–1674.
144. S. van der Walt, J.L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J.D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. “scikit-image: image processing in Python”. In: *PeerJ* 2, 2014, e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: <https://doi.org/10.7717/peerj.453>.
145. Q. Wang, Z. Wang, K. Genova, P. Srinivasan, H. Zhou, J.T. Barron, R. Martin-Brualla, N. Snavely, and T. Funkhouser. “IBRNet: Learning Multi-View Image-Based Rendering”. In: *CVPR*, 2021.
146. S.-Y. Wang, O. Wang, R. Zhang, A. Owens, and A. A. Efros. “CNN-generated images are surprisingly easy to spot...for now”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020.
147. X. Wang, R. Girshick, A. Gupta, and K. He. “Non-local neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7794–7803.
148. A.J. Wiggers and E. Hoogeboom. *Predictive Sampling with Forecasting Autoregressive Models*. 2020. arXiv: 2002.09928 [cs.LG].
149. O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson. “SynSin: End-to-end View Synthesis from a Single Image”. In: *CVPR*. 2020.
150. H. Wu, J. Köhler, and F. Noé. “Stochastic Normalizing Flows”. In: *arXiv preprint arXiv:2002.06707*, 2020.

151. Y. Wu and K. He. “Group normalization”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 3–19.
152. J. Xie, Y. Lu, S.-C. Zhu, and Y. Wu. “A theory of generative convnet”. In: *International Conference on Machine Learning*. 2016, pp. 2635–2644.
153. J. Xie, Z. Zheng, R. Gao, W. Wang, S.-C. Zhu, and Y. Nian Wu. “Learning descriptor networks for 3D shape synthesis and analysis”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8629–8638.
154. J. Xie, S.-C. Zhu, and Y. Nian Wu. “Synthesizing dynamic patterns by spatial-temporal generative convnet”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7093–7101.
155. J. Xie, S.-C. Zhu, and Y. N. Wu. “Learning energy-based spatial-temporal generative convnets for dynamic patterns”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.
156. L. Yen-Chen. *PyTorchNeRF: a PyTorch implementation of NeRF*. 2020. URL: <https://github.com/yenchenlin/nerf-pytorch/>.
157. A. Yu, V. Ye, M. Tancik, and A. Kanazawa. *pixelNeRF: Neural Radiance Fields from One or Few Images*. 2020.
158. F. Yu and V. Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *ICLR*. 2015.
159. F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao. “LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop”. In: *arXiv preprint arXiv:1506.03365*, 2015.
160. S. Zagoruyko and N. Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146*, 2016.
161. K. Zhang, G. Riegler, N. Snavely, and V. Koltun. “NeRF++: Analyzing and Improving Neural Radiance Fields”. In: *arXiv:2010.07492*, 2020.
162. R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric”. In: *CVPR*. 2018.
163. J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.

5 Appendix: Locally Masked Convolution for Autoregressive Models

5.1 Order visualization

Figure 2.2 shows three image generation orders and corresponding local masks used by the first DietNeRF layer in the autoregressive generator. On the left, we show the raster scan, S-curve and Hilbert curve orders over the pixels of a small 8×8 image. On the right, we show the corresponding, local 3×3 binary masks applied to image patches in the first layer. Masks applied to zero-pad pixels are colored green as their value is arbitrary. The center pixel in each image patch is masked out (set to 0), so that the network cannot include ground truth information in the representation of its context. The raster scan masks are the same for all image patches, so weights can be masked rather than image patches. However, other orders require diverse masks to respect the autoregressive property of the model. Figure 5.1 shows the 8 variants of the S-curve generation order used for order-agnostic training.

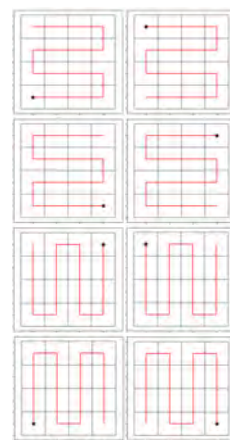


Figure 5.1: Eight variants of the S-curve generation order.

5.2 Mask conditioning

ConvNADE [139] is a convolutional neural autoregressive distribution estimator that can be trained with different masks on the input image. ConvNADE concatenates the mask with the image, allowing the model to distinguish between a zero-valued pixel and a zero-valued mask. Locally masked convolutions can also condition upon the mask in each layer. Algorithm 8 is an adaptation of Algorithm 1 that supports mask conditioning, with modifications shown in green. Algorithm 8 applies a learned weight matrix \mathcal{W}_M to the first C_{in} rows of the mask matrix as the

Algorithm 8 LMCONV with mask conditioning

- 1: **Input:** image x , weights \mathcal{W}_X , \mathcal{W}_M , bias b , generation order π . x is $B \times C_{\text{in}} \times H \times W$, \mathcal{W}_X is $C_{\text{out}} \times C_{\text{in}} * k_1 * k_2$, and \mathcal{W}_M is $C_{\text{out}} \times k_1 * k_2$.
 - 2: Create mask matrix \mathcal{M} with Algorithm 2
 - 3: Extract patches: $X = \text{im2col}(\text{pad}(x), k_1, k_2)$
 - 4: Mask patches: $X = \mathcal{M} \odot X$
 - 5: Perform convolution: $Y = \mathcal{W}_X X + \mathcal{W}_M \mathcal{M}_{1:C_{\text{in}}} + b$
 - 6: Assemble patches: $y = \text{col2im}(Y)$
 - 7: **return** y
-

mask is repeated $k_1 * k_2$ times by Algorithm 2. Equivalently, the mask $\mathcal{M}_{1:C_{\text{in}}}$ can be concatenated with X after masking.

We evaluate mask conditioning on the Binarized MNIST dataset with 8 S-curve orders. After training for 60 epochs (not converged for the purposes of comparison), the model without mask conditioning achieves a test NLL of 77.85 nats, while the mask conditioned model achieves a comparable test NLL of 77.94 nats. However, mask conditioning could improve generalization to novel orders.

5.3 Experimental setup

We tune hyperparameters such as the learning rate and batch size as well as the network architecture (Section 2.5) on the Grayscale MNIST dataset, and train models with the exact same architecture and hyperparameters on Binarized MNIST, CIFAR and CelebA-HQ. We used a batch size of 32 images, learning rate 2×10^{-4} , and gradient clipping to norm 2×10^6 . The exception is that we use batch size 5 on CelebA-HQ to save memory and 2-way softmax output instead of logistics for binary data. CelebA-HQ [66] contains 30,000 256×256 8-bit color celebrity photos. For experiments, we use the same CelebA-HQ data splits as Glow [73], with 27,000 training images and 3,000 validation images at reduced 5-bit color depth.

We trained the 1 stream baseline and our model for about the same number of epochs. Longer training improves performance, perhaps because order-agnostic training and dropout regularize, so epoch count was determined by time limitations. Most models are trained with 4 V100 or Quadro RTX 6000 GPUs. We train our CIFAR10 model for 2.6M steps (1644 epochs) with order-agnostic training over 8 precomputed S-curve variants, then average model parameters from the last 45 epochs of training. Early in our experimental process, we compared Hilbert curve generation



Figure 5.2: Unconditionally generating MNIST digits with two Hilbert curve orders, starting at the top or bottom left.

orders against the S-curve, visualized for small images in Figure 2.2, but did not see improved results.

For qualitative results, we train the 184M parameter 64×64 CelebA-HQ model for 375K iterations at batch size 32. Inspired by Progressive GAN [66], we train the model at a reduced 32×32 resolution for the first 242K iterations. As the architecture is fully convolutional, it is straightforward to increase image resolution during training.

5.4 Additional samples

Figure 5.2 shows intermediate states of the forward sampling process for unconditional generation of grayscale MNIST digits. We sample pixels along a Hilbert space-filling curve. As Hilbert curves are defined recursively for power-of-two sized grids, we use a generalization of the Hilbert curve [11] for 28×28 image generation. Our Locally Masked PixelCNN is optimized via order-agnostic training with eight variants of the order. Two variants are used for sampling digits in Fig. 5.2. The top two digits are sampled beginning at the top left of the image, and the bottom two digits are sampled beginning at the bottom left of the image. Images are shown at intervals of roughly 156 sampling steps. With the same parameters, the model is able to unconditionally generate plausible digits in multiple orders.

Figure 5.3 shows uncurated image completions using the large CelebA-HQ model. Initial network input is shown to the left of two image completions sampled from our Locally Masked PixelCNN with an S-curve variant that generates missing pixels last. The input images are taken from the validation set. The rightmost column contains the original image, *i.e.* the ground truth image

completion. Two samples with the same context vary due to the stochasticity of the decoding process, *e.g.* varying in terms of hairstyle, facial hair, attire and expression.

5.5 Implementation

Locally Masked Convolutions are simple to implement using the basic linear algebra subprograms exposed in machine learning frameworks, including matrix multiplication. It also requires an implementation of the `imzcol` operation. We provide an abbreviated Python code sample implementing DietNeRF using the PyTorch library in Figure 5.4. The full source including gradient computation, parameter initialization and mask conditioning is available at <https://ajayjain.github.io/lmconv>.



Figure 5.3: Uncurated CelebA-HQ 64x64 completions.

```

import math

import torch
import torch.nn as nn
import torch.nn.functional as F

class _locally_masked_conv2d(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, mask, weight, bias=None, dilation=1, padding=1):
        # Save values for backward pass
        ctx.save_for_backward(x, mask, weight)
        ctx.dilation, ctx.padding = dilation, padding
        ctx.H, ctx.W = x.size(2), x.size(3)
        ctx.output_shape = (x.shape[2], x.shape[3])
        out_channels, in_channels, k1, k2 = weight.shape

        # Step 1: Unfold (im2col)
        x = F.unfold(x, (k1, k2), dilation=dilation, padding=padding)
        # Step 2: Mask x. Avoid repeating mask in_channels times by reshaping x
        x_channels_batched = x.view(x.size(0) * in_channels,
                                   x.size(1) // in_channels, x.size(2))
        x = torch.mul(x_channels_batched, mask).view(x.shape)
        # Step 3: Perform convolution via matrix multiplication and addition
        weight_matrix = weight.view(out_channels, -1)
        x = weight_matrix.matmul(x)
        if bias is not None:
            x = x + bias.unsqueeze(0).unsqueeze(2)
        # Step 4: Restore shape
        return x.view(x.size(0), x.size(1), *ctx.output_shape)

    @staticmethod
    def backward(ctx, grad_output):
        x, mask, weight, mask_weight = ctx.saved_tensors
        ...
        if ctx.needs_input_grad[2]:
            # Recompute unfold and masking to save memory
            x_ = F.unfold(x, (k1, k2), dilation=ctx.dilation, padding=ctx.padding)
            ...
        ...

class locally_masked_conv2d(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, dilation, bias):
        super(locally_masked_conv2d, self).__init__()
        ...
        self.weight = nn.Parameter(torch.Tensor(out_channels, in_channels, *kernel_size))
        self.bias = nn.Parameter(torch.Tensor(out_channels)) if bias else None
        self.reset_parameters()

    def reset_parameters(self):
        ...

    def forward(self, x, mask):
        return _locally_masked_conv2d.apply(x, mask, self.weight,
                                            self.bias, self.dilation, self.padding)

```

Figure 5.4: A memory-efficient PyTorch v1.5.1 implementation of DietNeRF. Gradient calculation is omitted for brevity. See <https://ajayjain.github.io/lmconv> for full code.

6 Appendix: Denoising Diffusion Probabilistic Models

Extra information

LSUN FID scores for LSUN datasets are included in Table 6.1. Scores marked with * are reported by StyleGAN2 as baselines, and other scores are reported by their respective authors.

Table 6.1: FID scores for LSUN 256×256 datasets

| Model | LSUN Bedroom | LSUN Church | LSUN Cat |
|-------------------------------------|--------------|-------------|-------------|
| ProgressiveGAN [66] | 8.34 | 6.42 | 37.52 |
| StyleGAN [68] | 2.65 | 4.21* | 8.53* |
| StyleGAN2 [70] | - | 3.86 | 6.93 |
| Ours (L_{simple}) | 6.36 | 7.89 | 19.75 |
| Ours ($L_{\text{simple, large}}$) | 4.90 | - | - |

Progressive compression Our lossy compression argument in Section 3.4.3 is only a proof of concept, because Algorithms 5 and 6 depend on a procedure such as minimal random coding [45], which is not tractable for high dimensional data. These algorithms serve as a compression interpretation of the variational bound (3.5) of [130], not yet as a practical compression system.

Table 6.2: Unconditional CIFAR10 test set rate-distortion values (accompanies Fig. 3.5)

| Reverse process time ($T - t + 1$) | Rate (bits/dim) | Distortion (RMSE [0, 255]) |
|--------------------------------------|-----------------|----------------------------|
| 1000 | 1.77581 | 0.95136 |
| 900 | 0.11994 | 12.02277 |
| 800 | 0.05415 | 18.47482 |
| 700 | 0.02866 | 24.43656 |
| 600 | 0.01507 | 30.80948 |
| 500 | 0.00716 | 38.03236 |
| 400 | 0.00282 | 46.12765 |
| 300 | 0.00081 | 54.18826 |
| 200 | 0.00013 | 60.97170 |
| 100 | 0.00000 | 67.60125 |

6.1 Extended derivations

Below is a derivation of Eq. (3.5), the reduced variance variational bound for diffusion models. This material is from [130]; we include it here only for completeness.

$$L = \mathbb{E}_q \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \quad (6.1)$$

$$= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \quad (6.2)$$

$$= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t > 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} \right] \quad (6.3)$$

$$= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t > 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} \right] \quad (6.4)$$

$$= \mathbb{E}_q \left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)} - \sum_{t > 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)} - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1) \right] \quad (6.5)$$

$$= \mathbb{E}_q \left[D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T)) + \sum_{t > 1} D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1) \right] \quad (6.6)$$

The following is an alternate version of L . It is not tractable to estimate, but it is useful for our discussion in Section 3.4.3.

$$L = \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \quad (6.7)$$

$$= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t)} \cdot \frac{q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t)} \right] \quad (6.8)$$

$$= \mathbb{E}_q \left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T)} - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t)} - \log q(\mathbf{x}_0) \right] \quad (6.9)$$

$$= D_{\text{KL}}(q(\mathbf{x}_T) \parallel p(\mathbf{x}_T)) + \mathbb{E}_q \left[\sum_{t \geq 1} D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \right] + H(\mathbf{x}_0) \quad (6.10)$$

6.2 Experimental details

Our neural network architecture follows the backbone of PixelCNN++ [121], which is a U-Net [117] based on a Wide ResNet [160]. We replaced weight normalization [124] with group normalization [151] to make the implementation simpler. Our 32×32 models use four feature map resolutions (32×32 to 4×4), and our 256×256 models use six. All models have two convolutional residual blocks per resolution level and self-attention blocks at the 16×16 resolution between the convolutional blocks [18]. Diffusion time t is specified by adding the Transformer sinusoidal position embedding [142] into each residual block. Our CIFAR10 model has 35.7 million parameters, and our LSUN and CelebA-HQ models have 114 million parameters. We also trained a larger variant of the LSUN Bedroom model with approximately 256 million parameters by increasing filter count.

We used TPU v3-8 (similar to 8 V100 GPUs) for all experiments. Our CIFAR model trains at 21 steps per second at batch size 128 (10.6 hours to train to completion at 800k steps), and sampling a batch of 256 images takes 17 seconds. Our CelebA-HQ/LSUN (256^2) models train at 2.2 steps per second at batch size 64, and sampling a batch of 128 images takes 300 seconds. We trained on CelebA-HQ for 0.5M steps, LSUN Bedroom for 2.4M steps, LSUN Cat for 1.8M steps, and LSUN Church for 1.2M steps. The larger LSUN Bedroom model was trained for 1.15M steps.

Apart from an initial choice of hyperparameters early on to make network size fit within memory constraints, we performed the majority of our hyperparameter search to optimize for CIFAR10 sample quality, then transferred the resulting settings over to the other datasets:

- We chose the β_t schedule from a set of constant, linear, and quadratic schedules, all constrained so that $L_T \approx 0$. We set $T = 1000$ without a sweep, and we chose a linear schedule from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.
- We set the dropout rate on CIFAR10 to 0.1 by sweeping over the values $\{0.1, 0.2, 0.3, 0.4\}$. Without dropout on CIFAR10, we obtained poorer samples reminiscent of the overfitting artifacts in an unregularized PixelCNN++ [121]. We set dropout rate on the other datasets to zero without sweeping.
- We used random horizontal flips during training for CIFAR10; we tried training both with and without flips, and found flips to improve sample quality slightly. We also used random horizontal flips for all other datasets except LSUN Bedroom.
- We tried Adam [72] and RMSProp early on in our experimentation process and chose the former. We left the hyperparameters to their standard values. We set the learning rate to 2×10^{-4} without any sweeping, and we lowered it to 2×10^{-5} for the 256×256 images, which seemed unstable to train with the larger learning rate.
- We set the batch size to 128 for CIFAR10 and 64 for larger images. We did not sweep over these values.
- We used EMA on model parameters with a decay factor of 0.9999. We did not sweep over this value.

Final experiments were trained once and evaluated throughout training for sample quality. Sample quality scores and log likelihood are reported on the minimum FID value over the course of training. On CIFAR10, we calculated Inception and FID scores on 50000 samples using the original code from the OpenAI [120] and TTUR [49] repositories, respectively. On LSUN, we calculated FID scores on 50000 samples using code from the StyleGAN2 [70] repository. CIFAR10 and CelebA-HQ were loaded as provided by TensorFlow Datasets (<https://www.tensorflow.org/datasets>), and LSUN was prepared using code from StyleGAN. Dataset splits (or lack thereof) are standard from the papers that introduced their usage in a generative modeling context. All details can be found in the source code release.

6.3 Discussion on related work

Our model architecture, forward process definition, and prior differ from NCSN [132, 133] in subtle but important ways that improve sample quality, and, notably, we directly train our sampler as a latent variable model rather than adding it after training post-hoc. In greater detail:

1. We use a U-Net with self-attention; NCSN uses a RefineNet with dilated convolutions. We condition all layers on t by adding in the Transformer sinusoidal position embedding, rather than only in normalization layers (NCSNv1) or only at the output (v2).
2. Diffusion models scale down the data with each forward process step (by a $\sqrt{1 - \beta_t}$ factor) so that variance does not grow when adding noise, thus providing consistently scaled inputs to the neural net reverse process. NCSN omits this scaling factor.
3. Unlike NCSN, our forward process destroys signal ($D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel \mathcal{N}(\mathbf{o}, \mathbf{I})) \approx 0$), ensuring a close match between the prior and aggregate posterior of \mathbf{x}_T . Also unlike NCSN, our β_t are very small, which ensures that the forward process is reversible by a Markov chain with conditional Gaussians. Both of these factors prevent distribution shift when sampling.
4. Our Langevin-like sampler has coefficients (learning rate, noise scale, etc.) derived rigorously from β_t in the forward process. Thus, our training procedure directly trains our sampler to match the data distribution after T steps: it trains the sampler as a latent variable model using variational inference. In contrast, NCSN’s sampler coefficients are set by hand post-hoc, and their training procedure is not guaranteed to directly optimize a quality metric of their sampler.

6.4 Samples

Additional samples Fig. 6.3, 6.5, 6.8, 6.9, 6.10, and 6.11 show uncurated samples from the diffusion models trained on CelebA-HQ, CIFAR10 and LSUN datasets.

Latent structure and reverse process stochasticity During sampling, both the prior $\mathbf{x}_T \sim \mathcal{N}(\mathbf{o}, \mathbf{I})$ and Langevin dynamics are stochastic. To understand the significance of the second source of noise, we sampled multiple images conditioned on the same intermediate latent for the CelebA 256×256 dataset. Fig. 3.7 shows multiple draws from the reverse process $\mathbf{x}_0 \sim p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ that share the latent \mathbf{x}_t for $t \in \{1000, 750, 500, 250\}$. To accomplish this, we run a single reverse chain from an initial draw from the prior. At the intermediate timesteps, the chain is split to sample multiple

images. When the chain is split after the prior draw at $x_{T=1000}$, the samples differ significantly. However, when the chain is split after more steps, samples share high-level attributes like gender, hair color, eyewear, saturation, pose and facial expression. This indicates that intermediate latents like x_{750} encode these attributes, despite their imperceptibility.

Coarse-to-fine interpolation Fig. 6.1 shows interpolations between a pair of source CelebA 256×256 images as we vary the number of diffusion steps prior to latent space interpolation. Increasing the number of diffusion steps destroys more structure in the source images, which the model completes during the reverse process. This allows us to interpolate at both fine granularities and coarse granularities. In the limiting case of 0 diffusion steps, the interpolation mixes source images in pixel space. On the other hand, after 1000 diffusion steps, source information is lost and interpolations are novel samples.

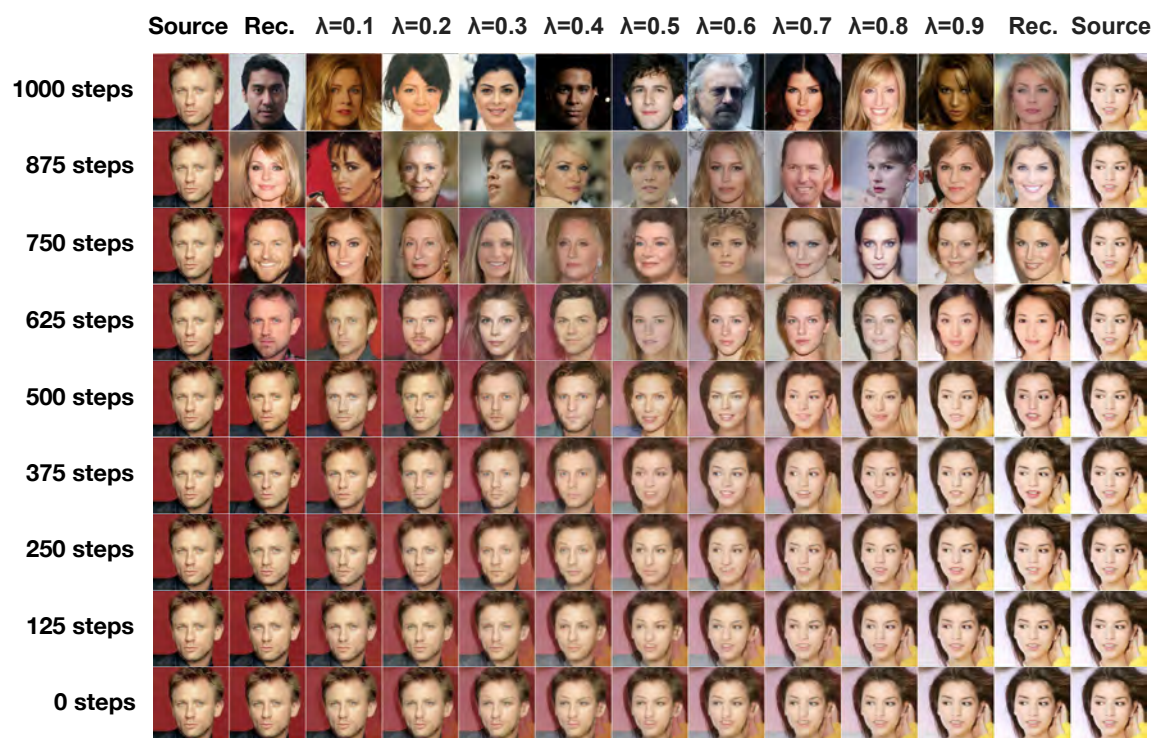


Figure 6.1: Coarse-to-fine interpolations that vary the number of diffusion steps prior to latent mixing.

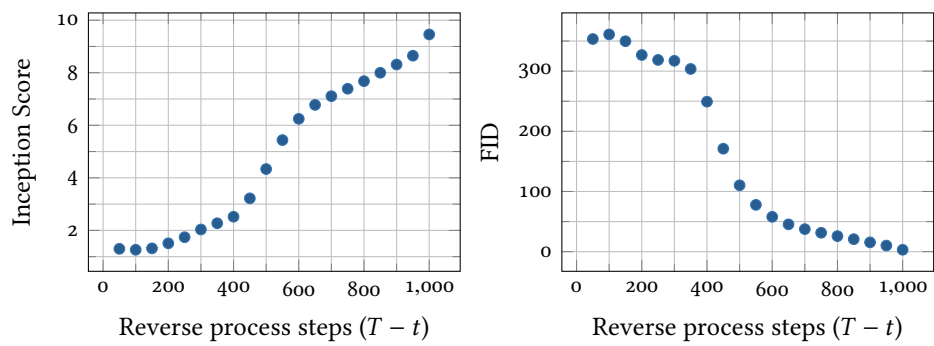
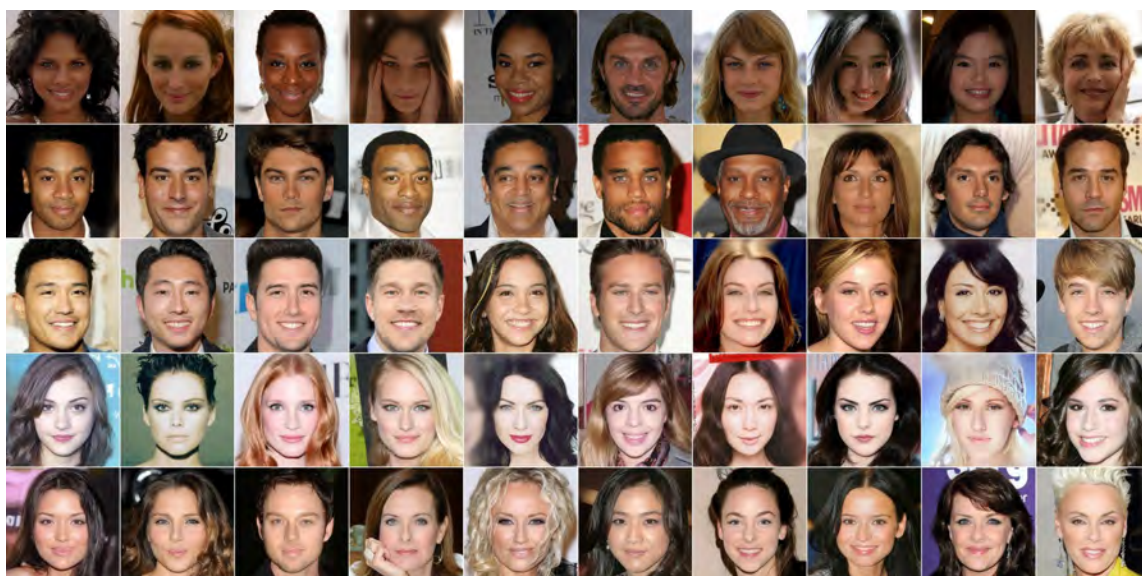


Figure 6.2: Unconditional CIFAR₁₀ progressive sampling quality over time



Figure 6.3: CelebA-HQ 256×256 generated samples



(a) Pixel space nearest neighbors



(b) Inception feature space nearest neighbors

Figure 6.4: CelebA-HQ 256×256 nearest neighbors, computed on a 100×100 crop surrounding the faces. Generated samples are in the leftmost column, and training set nearest neighbors are in the remaining columns.

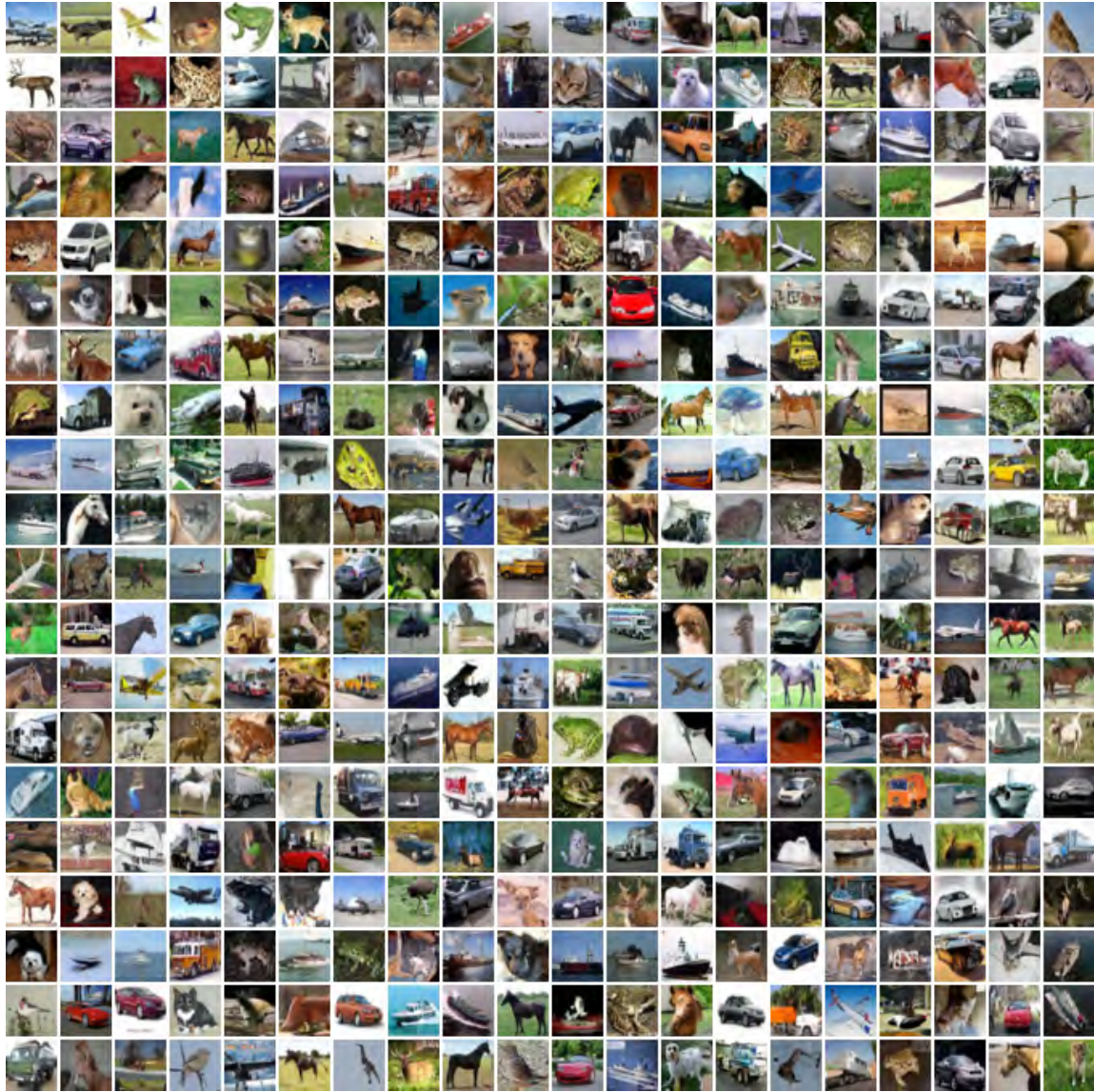


Figure 6.5: Unconditional CIFAR10 generated samples

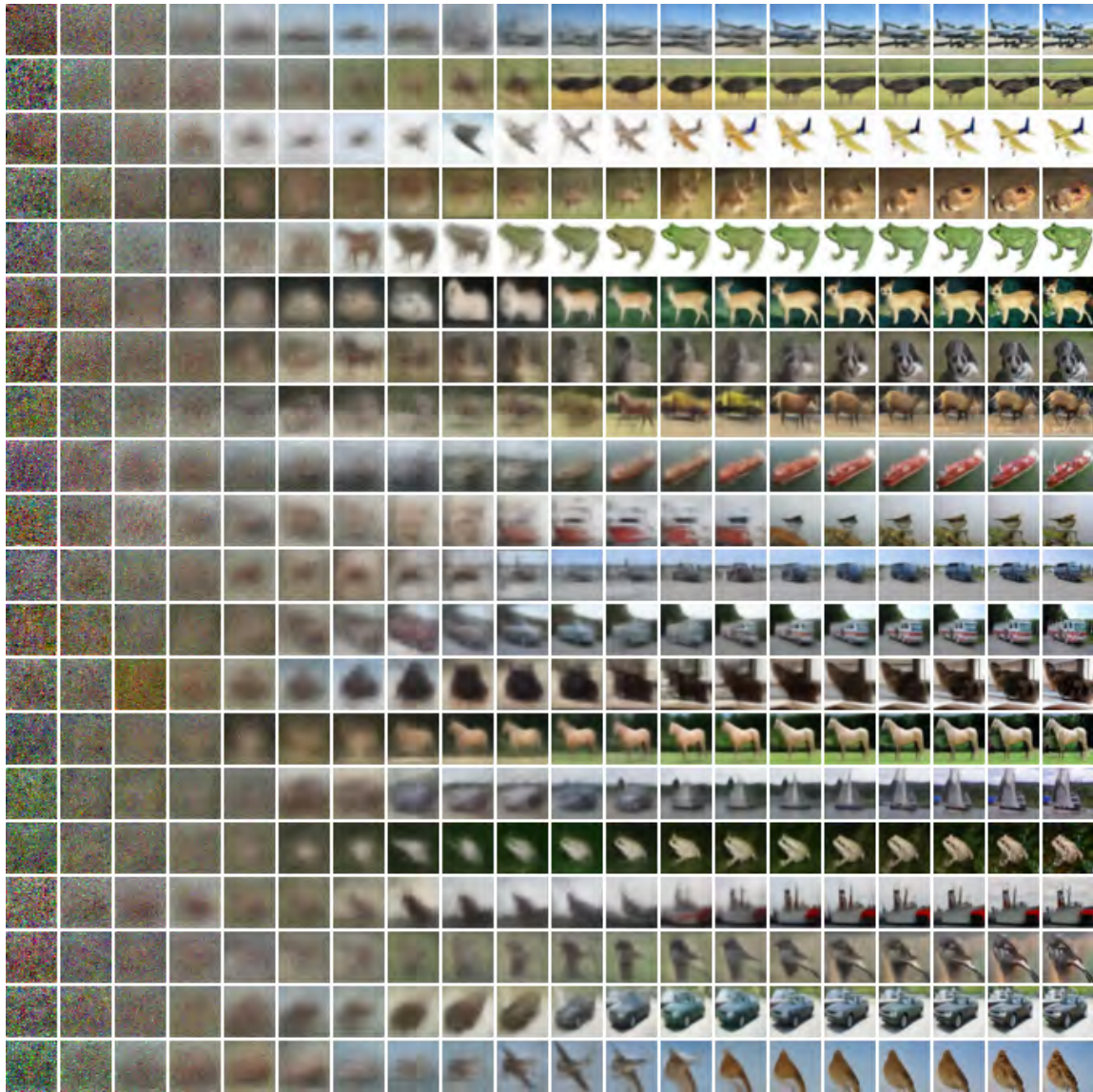
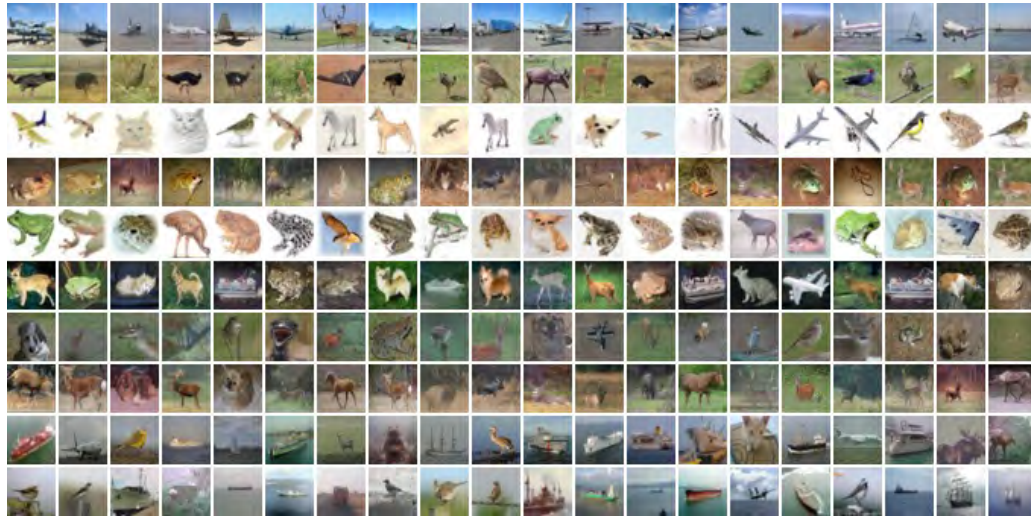
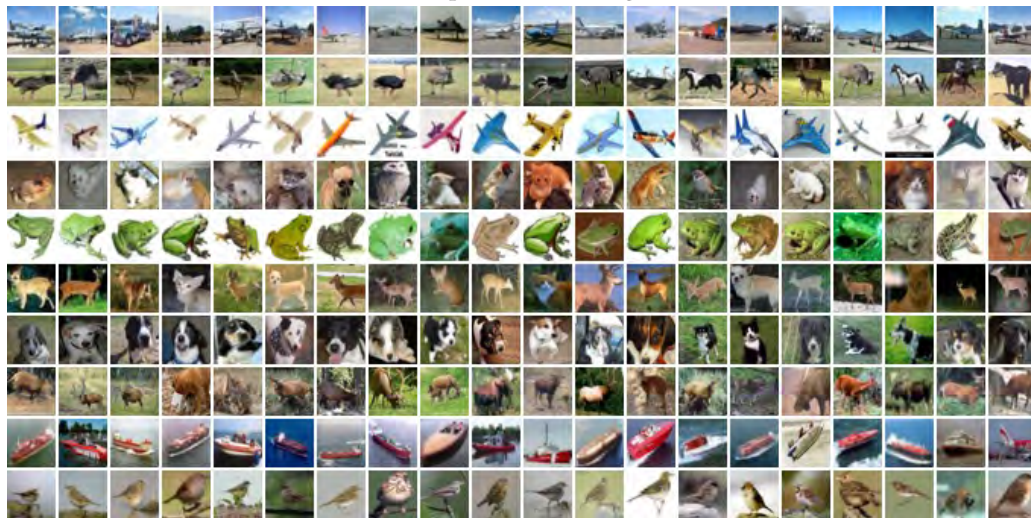


Figure 6.6: Unconditional CIFAR10 progressive generation



(a) Pixel space nearest neighbors



(b) Inception feature space nearest neighbors

Figure 6.7: Unconditional CIFAR10 nearest neighbors. Generated samples are in the leftmost column, and training set nearest neighbors are in the remaining columns.



Figure 6.8: LSUN Church generated samples. FID=7.89



Figure 6.9: LSUN Bedroom generated samples, large model. FID=4.90

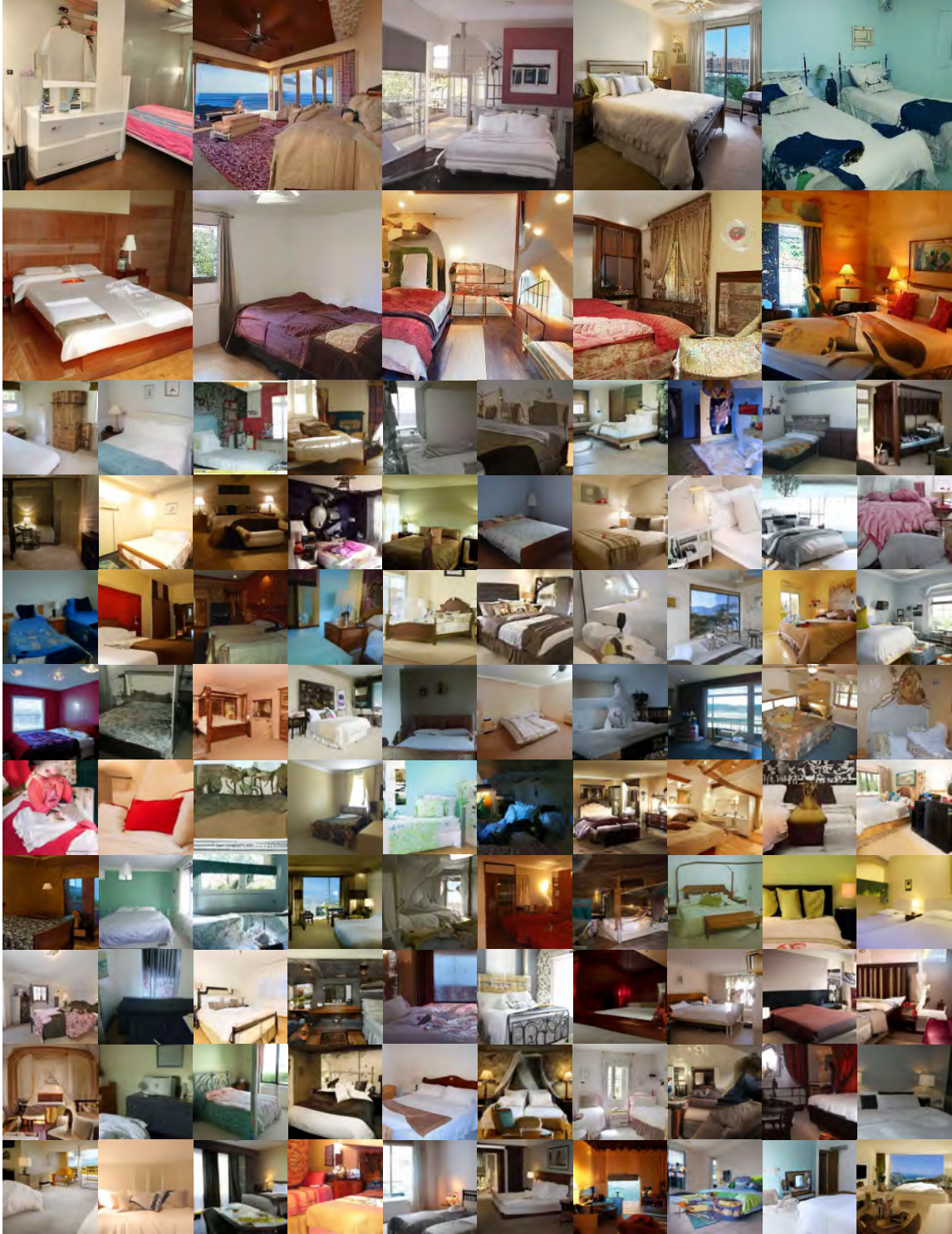


Figure 6.10: LSUN Bedroom generated samples, small model. FID=6.36



Figure 6.11: LSUN Cat generated samples. FID=19.75

7 Appendix: Putting NeRF on a Diet

7.1 Experimental details

View selection For most few-view Realistic Synthetic experiments, we randomly subsample 8 of the available 100 training renders. Views are not manually selected. However, to compare the ability of NeRF and DietNeRF to extrapolate to unseen regions, we manually selected 14 of the 100 views mostly showing the right side of the Lego scene. For DTU experiments where we fine-tune pixelNeRF [157], we use the same source view as [157]. This viewpoint was manually selected and is shared across all 15 scenes.

Simplified NeRF baseline The published version of NeRF [89] can be unstable to train with 8 views, often converging to a degenerate solution. We found that NeRF is sensitive to MLP parameter initialization, as well as hyperparameters that control the complexity of the learned scene representation. For a fair comparison, we tuned the Simplified NeRF baseline on each Realistic Synthetic scene by modifying hyperparameters until object geometry converged. Table 7.1 shows the resulting hyperparameter settings for initial learning rate prior to decay, whether the MLP f_θ is viewpoint dependent, number of samples per ray queried from the fine and coarse networks, and the maximum frequency sinusoidal encoding of spatial position (x, y, z) . The fine and coarse networks are used in [89] for hierarchical sampling. \times denotes that we do not use the fine network.

Implementation Our implementation is based on a PyTorch port [156] of NeRF’s original Tensorflow code. We re-train and evaluate NeRF using this code. For memory efficiency, we use 400×400 images of the scenes as in [156] rather than full-resolution 800×800 images. NV is trained with full-resolution 800×800 views. NV renderings are downsampled with a 2×2 box filter to 400×400 to compute metrics. We train all NeRF, Simplified NeRF and DietNeRF models with the Adam optimizer [71] for 200k iterations.

Metrics Our PSNR, SSIM, and LPIPS metrics use the same implementation as [157] based on the scikit-image Python package [144]. For the DTU dataset, [157] excluded some poses from the

Table 7.1: **Simplified NeRF training details** by scene in the Realistic Synthetic dataset. We tune the initial learning rate, view dependence, number of samples from fine and coarse networks for hierarchical sampling, and the maximum frequency of the (x, y, z) spatial positional encoding.

| Scene | LR | View dep. | Fine | Coarse | Max freq. |
|-----------|--------------------|-----------|------|--------|-----------|
| Full NeRF | 5×10^{-4} | ✓ | 128 | 64 | 2^9 |
| Lego | 5×10^{-5} | ✓ | ✗ | 128 | 2^5 |
| Chair | 5×10^{-5} | ✗ | ✗ | 128 | 2^5 |
| Drums | 5×10^{-5} | ✗ | ✗ | 128 | 2^5 |
| Ficus | 5×10^{-5} | ✗ | ✗ | 128 | 2^5 |
| Mic | 5×10^{-5} | ✗ | ✗ | 128 | 2^5 |
| Ship | 5×10^{-5} | ✗ | ✗ | 128 | 2^5 |
| Materials | 1×10^{-5} | ✗ | ✗ | 128 | 2^5 |
| Hotdog | 1×10^{-5} | ✗ | ✗ | 128 | 2^3 |

validation set as ground truth photographs had excessive shadows due to the physical capture setup. We use the same subset of validation views.

For both Realistic Synthetic and DTU scenes, we also included FID and KID perceptual image quality metrics. While PSNR, SSIM and LPIPS are measured between pairs of pixel-aligned images, FID and KID are measured between two sets of image samples. These metrics compare the *distribution* of image features computed on one set of images to those computed on another set. As distributions are compared rather than individual images, a sufficiently large sample size is needed. For the Realistic Synthetic dataset, we compute the FID and KID between all 3200 ground-truth images (across train, validation and testing splits and across scenes), and 200 rendered test images at the same resolution (25 test views per scene). Aggregating across scenes allows us to have a larger sample size. Due to the setup of the Neural Volumes code, we use additional samples for rendered images for that baseline. For the DTU dataset, we compute FID and KID between 720 rendered images (48 per scene across 15 validation scenes, excluding the viewpoint of the source image provided to pixelNeRF) and 6076 ground-truth images (49 images including the source viewpoint across 124 training and validation scenes). FID and KID metrics are computed using the torch-fidelity Python package [94].

Table 7.2: Quality metrics for each scene in the Realistic Synthetic dataset with 8 observed views.

| PSNR \uparrow | Lego | Chair | Drums | Ficus | Mic | Ship | Materials | Hotdog |
|---|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| NeRF | 9.726 | 21.049 | 17.472 | 13.728 | 26.287 | 12.929 | 7.837 | 10.446 |
| NV [84] | 17.652 | 20.515 | 16.271 | 19.448 | 18.323 | 14.457 | 16.846 | 19.361 |
| Simplified NeRF | 16.735 | 21.870 | 15.021 | 21.091 | 24.206 | 17.092 | 20.659 | 24.060 |
| DietNeRF (ours) | <u>23.897</u> | <u>24.633</u> | 20.034 | 20.744 | <u>26.321</u> | 23.043 | <u>21.254</u> | <u>25.250</u> |
| DietNeRF, \mathcal{L}_{MSE} ft (ours) | 24.311 | 25.595 | <u>20.029</u> | <u>20.940</u> | 26.794 | <u>22.536</u> | 21.621 | 26.626 |
| NeRF, 100 views | 31.618 | 34.073 | 25.530 | 29.163 | 33.197 | 29.407 | 29.340 | 36.899 |
| SSIM \uparrow | Lego | Chair | Drums | Ficus | Mic | Ship | Materials | Hotdog |
| NeRF | 0.526 | 0.861 | 0.770 | 0.661 | <u>0.944</u> | 0.605 | 0.484 | 0.644 |
| NV [84] | 0.707 | 0.795 | 0.675 | 0.815 | 0.816 | 0.602 | 0.721 | 0.796 |
| Simplified NeRF | 0.775 | 0.859 | 0.727 | <u>0.872</u> | 0.930 | 0.694 | 0.823 | 0.894 |
| DietNeRF (ours) | <u>0.863</u> | <u>0.898</u> | <u>0.843</u> | <u>0.872</u> | <u>0.944</u> | 0.758 | <u>0.843</u> | <u>0.904</u> |
| DietNeRF, \mathcal{L}_{MSE} ft (ours) | 0.875 | 0.912 | 0.845 | 0.874 | 0.950 | <u>0.757</u> | 0.851 | 0.924 |
| NeRF, 100 views | 0.965 | 0.978 | 0.929 | 0.966 | 0.979 | 0.875 | 0.958 | 0.981 |
| LPIPS \downarrow | Lego | Chair | Drums | Ficus | Mic | Ship | Materials | Hotdog |
| NeRF | 0.467 | 0.163 | 0.231 | 0.354 | 0.067 | 0.375 | 0.467 | 0.422 |
| NV [84] | 0.253 | 0.175 | 0.299 | 0.156 | 0.193 | 0.456 | 0.223 | 0.203 |
| Simplified NeRF | 0.218 | 0.152 | 0.280 | 0.132 | 0.080 | 0.283 | 0.151 | 0.139 |
| DietNeRF (ours) | <u>0.110</u> | <u>0.092</u> | 0.117 | <u>0.097</u> | <u>0.053</u> | <u>0.204</u> | <u>0.102</u> | <u>0.097</u> |
| DietNeRF, \mathcal{L}_{MSE} ft (ours) | 0.096 | 0.077 | 0.117 | 0.094 | 0.043 | 0.193 | 0.095 | 0.067 |
| NeRF, 100 views | 0.033 | 0.025 | 0.064 | 0.035 | 0.023 | 0.125 | 0.037 | 0.025 |

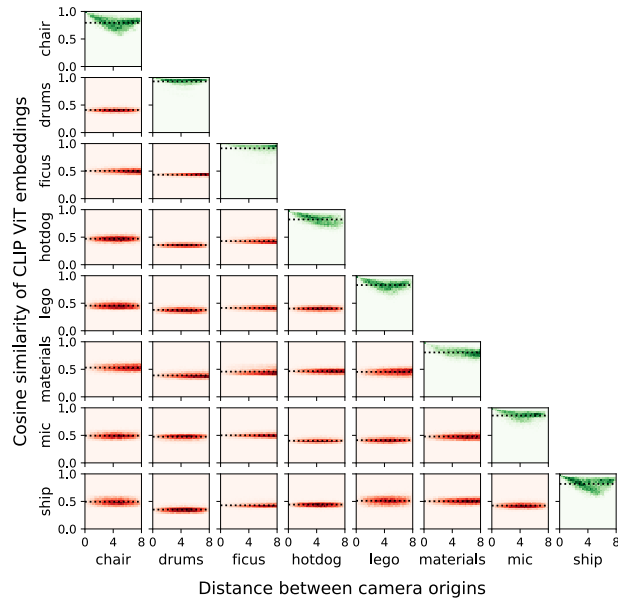


Figure 7.1: **CLIP ViT embeddings are more similar between views of the same scene than across different scenes.** We show a 2D histogram for each pair of Realistic Synthetic scenes comparing ViT embedding similarity and the distance between views. The dashed line shows mean cosine similarity, and green histograms have mean similarity is greater than 0.6. On the diagonal, two views from the upper hemisphere of the same scene are sampled. Embeddings of different views of the same scene are generally highly similar. Nearby (distance 0) and diagonally opposing (distance 8) views are most similar. In comparison, when sampling views from different scenes (lower triangle), embeddings are dissimilar.

7.2 Per-scene metrics

Embedding similarity In Figure 7.1, we compare the cosine similarity of two views with the distance between their camera origins for each pair of scenes in the Realistic Synthetic dataset. When sampling both views from the same scene, views have high cosine similarity (diagonal). For 6 of the 8 scenes, there is some dependence on the relative poses of the camera views, though similarity is high across all camera distances. For views sampled from different scenes, similarity is low (cosine similarity around 0.5).

Quality metrics Table 7.2 shows PSNR, SSIM and LPIPS metrics on a per-scene basis for the Realistic Synthetic dataset. FID and KID metrics are excluded as they need a larger sample size. We bold the best method on each scene, and underline the second-best method. Across all scenes in the few-shot setting, DietNeRF or DietNeRF fine-tuned for 50k iterations with \mathcal{L}_{MSE} performs best or second-best.

7.3 Qualitative results and ground-truth

In this section, we provide additional qualitative results. Figure 7.2 shows the ground-truth training views used for 8-shot Realistic Synthetic experiments. These views are sampled at random from the training set of [89]. Random sampling models challenges with real-world data capture such as uneven view sampling. It may be possible to improve results if views are carefully selected.

In Figure 7.3, we provide additional renderings of Realistic Synthetic scenes from testing poses for baseline methods and DietNeRF. Neural Volumes generally converges to recover coarse object geometry, but has wispy artifacts and distortions. On the Ship scene, Neural Volumes only recovers very low-frequency detail. Simplified NeRF suffers from occluders that are not visible from the 8 training poses. DietNeRF has the highest quality reconstructions without these distortions or occluders, but does miss some high-frequency detail. An interesting artifact is the leakage of green coloration to the back of the chair.

Finally, in Figure 7.4, we show renderings from pixelNeRF and DietPixelNeRF on all DTU dataset validation scenes not included in the main paper. Starting from the same checkpoint, pixelNeRF is fine-tuned using \mathcal{L}_{MSE} for 20k iterations, whereas DietPixelNeRF is fine-tuned using $\mathcal{L}_{MSE} + \mathcal{L}_{SC}$ for 20k iterations. DietPixelNeRF has sharper renderings. On scenes with rectangular objects like bricks and boxes, DietPixelNeRF performs especially well. However, the method struggles to preserve accurate geometry in some cases. Note that the problem is under-determined as only a single view is observed per scene.

7.4 Adversarial approaches

While NeRF is only supervised from observed poses, conceptually, a GAN [37] uses a discriminator to compute a realism loss between real and generated images that need not align pixel-wise. Patch GAN discriminators were introduced for image translation problems [55, 163] and can be useful for high-resolution image generation [29]. SinGAN [118] trains multiscale patch discriminators on a single image, comparable to our single-scene few-view setting. In early experiments, we trained



Figure 7.2: **Training views used for Realistic Synthetic scenes.** These views are randomly sampled from the available 100 views. This is a challenging setting for view synthesis and 3D reconstruction applications as objects are not uniformly observed. Some views are mostly redundant, like the top two Lego views. Other regions are sparsely observed, such as a single side view of Hotdog.

patch-wise discriminators per-scene to supervise f_θ from novel poses in addition to \mathcal{L}_{SC} . However, an auxiliary adversarial loss led to artifacts on Realistic Synthetic scenes, both in isolation and in combination with our semantic consistency loss.

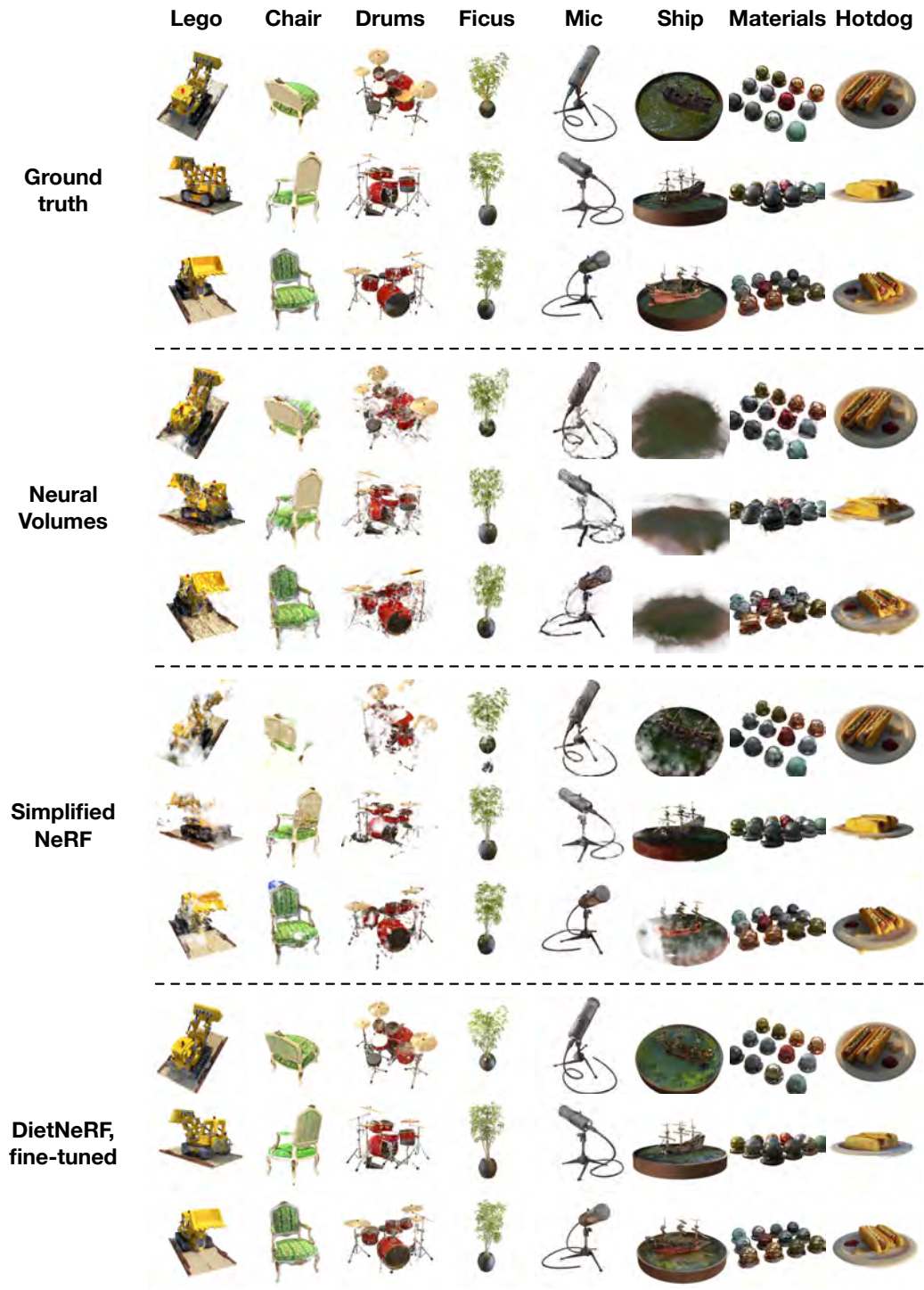


Figure 7.3: Additional renderings of Realistic Synthetic scenes.

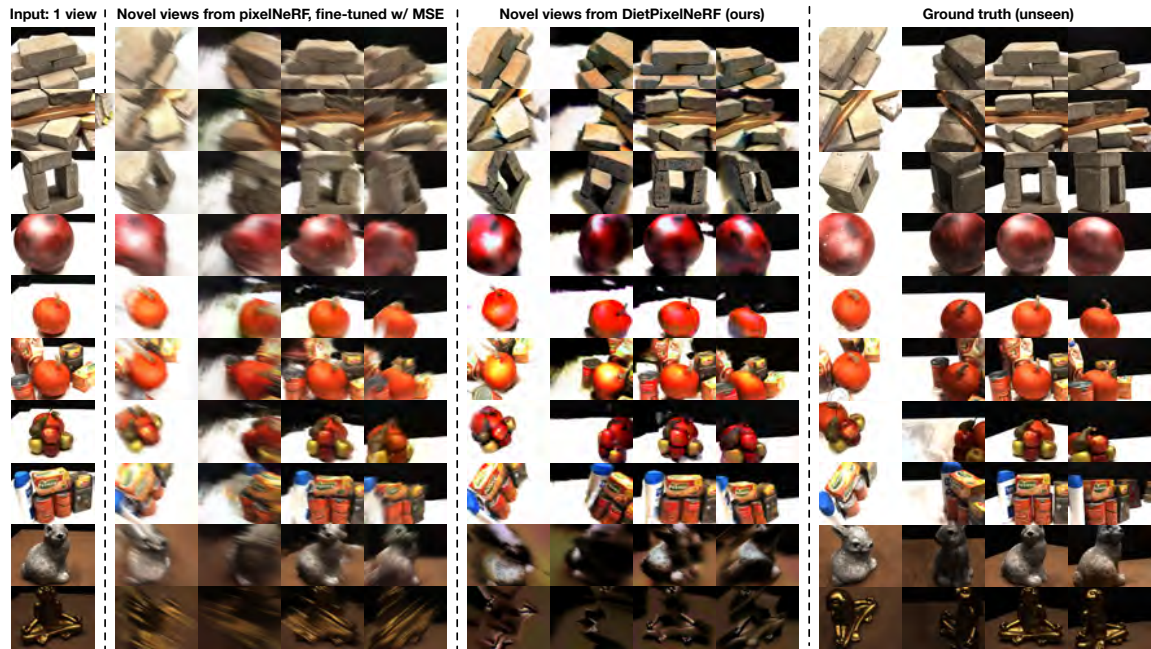


Figure 7.4: **One-shot novel view synthesis:** Additional renderings of DTU scenes generated from a single observed view (left). Ground truth views are shown for reference, but are not provided to the model. pixelNeRF and DietPixelNeRF are pre-trained on the same dataset of other scenes, then fine-tuned on the single input view for 20k iterations with \mathcal{L}_{MSE} alone (pixelNeRF) or $\mathcal{L}_{\text{MSE}} + \mathcal{L}_{\text{SC}}$ (DietPixelNeRF).