Transferable Generative Models

Ajay Jain

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-161
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-161.html

May 12, 2023

Transferable Generative Models

By

Ajay Jain

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy
in
Engineering – Electrical Engineering and Computer Sciences
in the
Graduate Division
of the
University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Professor Alexei A. Efros
Professor Angjoo Kanazawa
Dr. Ben Poole

Spring 2023

Transferable Generative Models

Abstract

Transferable Generative Models

By

Ajay Jain

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

We present progress in developing stable, scalable and transferable generative models for visual data. We first learn expressive image priors using autoregressive models which generate high-quality and diverse images. We then explore transfer learning to generalize visual representations models to new data modalities with limited available data. We propose two methods to generate high quality 3D graphics from sparse input images or natural language descriptions by distilling knowledge from pretrained discriminative vision models. We briefly summarize our work on improving generation quality with a Denoising Diffusion Probabilistic Model, and demonstrate how to transfer it to new modalities including high-quality text-to-3D synthesis using Score Distillation Sampling. Finally, we generate 2D vector graphics from text by optimizing a vector graphics renderer with knowledge distilled from a pretrained text-to-image diffusion model, without vector graphics data. Our models enable high-quality generation across many modalities, and continue to be broadly applied in subsequent work.

To my family.

Contents

1 Introduction **12**

2 Locally Masked Convolution for Autoregressive Models **15**

2.1 Introduction . . . 15
2.2 Background . . . 18
2.3 Image Completion with Maximum Receptive Field . . . 20
2.4 Local Masking . . . 21
2.5 Architecture . . . 23
2.6 Experiments . . . 26
2.6.1 Whole-image Density Estimation . . . 26
2.6.2 Generalization to Novel Orders . . . 28
2.6.3 Image Completion . . . 29
2.6.4 Qualitative Results . . . 30
2.7 Related Work . . . 30
2.8 Conclusion . . . 32

3 Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis **33**

3.1 Introduction . . . 34
3.2 Background on Neural Radiance Fields . . . 36
3.3 NeRF Struggles at Few-Shot View Synthesis . . . 37
3.4 Semantically Consistent Radiance Fields . . . 39
3.4.1 Semantic consistency loss . . . 39
3.4.2 Interpreting representations across views . . . 40
3.4.3 Pose sampling distribution . . . 41
3.4.4 Improving efficiency and quality . . . 41
3.5 Experiments . . . 42
3.5.1 Realistic Synthetic scenes from scratch . . . 43
3.5.2 Single-view synthesis by fine-tuning . . . 44

3.5.3 Reconstructing unobserved regions . . . 45
3.6 Ablations . . . 47
3.7 Related work . . . 48
3.8 Conclusions . . . 49

4 Zero-Shot Text-Guided Object Generation with Dream Fields 51
4.1 Introduction . . . 52
4.2 Related Work . . . 54
4.3 Background . . . 56
4.3.1 Neural Radiance Fields . . . 56
4.3.2 Image-text models . . . 57
4.4 Method . . . 57
4.4.1 Object representation . . . 58
4.4.2 Objective . . . 58
4.4.3 Challenges with CLIP guidance . . . 59
4.4.4 Pose sampling . . . 59
4.4.5 Encouraging coherent objects through sparsity . . . 59
4.4.6 Localizing objects and bounding scene . . . 61
4.4.7 Neural scene representation architecture . . . 61
4.5 Evaluation . . . 62
4.5.1 Experimental setup . . . 62
4.5.2 Analyzing retrieval metrics . . . 63
4.5.3 Compositional generation . . . 65
4.5.4 Model ablations . . . 65
4.6 Limitations . . . 69
4.7 Discussion . . . 69

5 Learning more expressive priors and transferring them to new modalities 70
5.0.1 Denoising Diffusion Probabilistic Models . . . 70
5.0.2 DreamFusion: Text-to-3D using 2D Diffusion . . . 71

6 VectorFusion: Text-to-SVG by Abstracting Image Diffusion Models 73
6.1 Introduction . . . 74
6.2 Related Work . . . 76

6.3 Background . . . 77
6.3.1 Vector representation and rendering pipeline . . . 77
6.3.2 Diffusion models . . . 77
6.3.3 Score distillation sampling . . . 79
6.4 Method: VectorFusion . . . 81
6.4.1 A baseline: text-to-image-to-vector . . . 81
6.4.2 Sampling vector graphics by optimization . . . 82
6.4.3 Reinitializing paths . . . 83
6.4.4 Stylizing by constraining vector representation . . . 83
6.5 Experiments . . . 84
6.5.1 Experimental setup . . . 85
6.5.2 Evaluating caption consistency . . . 86
6.5.3 Comparison with CLIP-based approaches . . . 87
6.5.4 Pixel art generation . . . 87
6.5.5 Sketches and line drawings . . . 88
6.6 Discussion . . . 88

7 Conclusion **89**

Bibliography **91**

8 Appendix: Locally Masked Convolution for Autoregressive Models **107**
8.1 Order visualization . . . 107
8.2 Mask conditioning . . . 107
8.3 Experimental setup . . . 108
8.4 Additional samples . . . 109
8.5 Implementation . . . 110

9 Appendix: Putting NeRF on a Diet **113**
9.1 Experimental details . . . 113
9.2 Per-scene metrics . . . 115
9.3 Qualitative results and ground-truth . . . 117
9.4 Adversarial approaches . . . 117

10 Appendix: Zero-Shot Text-Guided Object Generation with Dream Fields **121**
10.1 Qualitative results and ablations . . . 121

10.2 Object Centric COCO captions dataset . . . 123
10.3 Hyperparameters and training setup . . . 123
10.4 Pixel and voxel baselines . . . 125
10.5 Signed distance field parameterization . . . 125
10.6 Impact of optimization time . . . 125

11 Appendix: VectorFusion: Text-to-SVG by Abstracting Image Diffusion Models 127
11.1 Website with results, videos, and benchmark . . . 127
11.2 Ablation: Reinitializing paths . . . 127
11.3 Ablation: Saturation Penalty . . . 128
11.4 Ablation: Number of paths . . . 129
11.5 Ablation: Number of rejection samples . . . 129
11.6 Ablation: Classifier-Free Guidance . . . 130
11.7 Ablation: SDS + CLIP Hybrid Losses . . . 130
11.8 Pixel Art Results . . . 131
11.9 Experimental hyperparameters . . . 132
11.9.1 Path initialization . . . 132
11.9.2 Data Augmentation . . . 134
11.9.3 Optimization . . . 134
11.10 Perceptual Quality Metric . . . 135

Acknowledgements

I would like to express my gratitude to all the individuals I had the privilege of working with during my four-year journey at UC Berkeley. I had the opportunity to collaborate on various papers with a remarkable group of individuals, including Pieter Abbeel, Jonathan T. Barron, Scott Emmons, Amir Gholami, Joseph E. Gonzalez, Jonathan Ho, Paras Jain, Kurt Keutzer, Misha Laskin, Qiyang Li, Ben Mildenhall, Aniruddha Nrusimha, Deepak Pathak, Ben Poole, Thanard Kurutach, Ion Stoica, Matthew Tancik, Alexey Tumanov, Tianjun Zhang, and Amber Xie (in alphabetical order). Their contributions have greatly impacted the work presented in this report. Additionally, I had the privilege of collaborating with Kumar Krishna Agrawal, Alejandro Escontrela, Ming-Yu Liu, Ruiqi Gao, Durk Kingma, Kevin Murphy, Sean O'Brien, Aravind Srinivas, Jeffrey Tao, and Arash Vahdat.

I extend a special thanks to my advisor, Pieter Abbeel, whose unwavering support and encouragement allowed me to explore diverse research areas. I would also like to acknowledge Joseph Gonzalez, who provided valuable guidance during my early days at Berkeley. A significant portion of my PhD involved close collaboration with Ben Poole and Jonathan Barron at Google, and I am sincerely grateful for their mentorship. I would also like to express my appreciation to Alexei Efros and Angjoo Kanazawa, my other committee members, for their insightful feedback and suggestions for this thesis.

TAs and professors of courses at Berkeley have generously given exceptional guidance. Professors Barna Saha and Laurent El Ghaoui provided valuable advice on one of my first research and course projects at UC Berkeley. To continue the tradition of sharing knowledge, during my program, I had the privilege of teaching classes alongside Ren Ng and Trevor Darrell. Those experiences provided valuable teaching experience and rewarding interactions.

Furthermore, I cherish the wonderful memories shared with my friends—both new and old. The graduate program at Berkeley is so strong due to the caring people in its halls, and the social atmosphere helped make the program a delight. I also fondly remember engaging discussions with my labmates.

I am deeply indebted to my early mentors who played a pivotal role in inspiring my pursuit of a PhD. I am immensely grateful to my undergraduate research advisors from MIT, Saman

Amarasinghe and Charith Mendis, who collaborated with me on compiler and ML systems projects and imparted valuable lessons on rigorous evaluation and debugging techniques. I would also like to express my gratitude to my mentors from Uber, Raquel Urtasun and Renjie Liao, who instilled in me a methodological approach and fueled my passion for tackling profound challenges. Additionally, I would like to thank Pierre Lermusiaux and Tim Chung for their faith in me, even before I obtained my undergraduate degree.

Lastly, I dedicate this thesis to my beloved family—my parents, sister, and brother—who have been unwavering in their support for my educational and research pursuits. Their influence has profoundly shaped my journey, and I am particularly grateful to my brother and co-author, Paras Jain, for introducing me to UC Berkeley's PhD program and for being a constant source of friendship and intellectual discourse.

Each chapter of this thesis contains additional acknowledgments, expressing my sincere appreciation to individuals who provided invaluable feedback, as well as to the funding agencies that supported my research.

Chapter 1

Introduction

I have been lucky to have a very full-stack PhD. In the beginning of my academic journey, I focused on multiple parallel directions, yet aimed to make impossible tasks possible through a combination of learning algorithms and computer systems. My PhD journey has focused on three classes of work: (i) efficient computer systems that enable scaling up learning, (ii) developing more expressive and stable models that can benefit from the scale, and (iii) transfer learning algorithms that enable models to generalize to new modalities. In this thesis, I will focus on the latter two bodies of work.

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 posterior distributions with low computational cost, such as $p_\theta(\mathbf{x}|\tilde{\mathbf{x}})$. General-purpose inference algorithms can sample from the desired posteriors in some cases, 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 thesis is to **learn and transfer expressive generative visual models to many domains**. We tackle the problem by removing architectural limitations in the generative image prior, then by easing data requirements for generative applications by transfering knowledge from large pretrained models. 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 explore a challenging application of image synthesis: the novel view synthesis (NVS) problem. The goal of NVS 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 1-8 observed images. Using prior knowledge from self-supervised models is a promising approach to improve the data efficiency, flexibility and controllability of generative models.

Are any observations required? In Chapter 4, we show that feature space losses can be used to generate a 3D object from scratch given only a caption. We describe a method, Dream Fields, to synthesize a 3D neural radiance field by test-time training. Dream Fields consists of a regularized 3D representation optimized with a loss function based on a pretrained language model and image encoder with aligned feature spaces. Regularizations prove critical for high quality. Our work paves the way to open-domain text-to-3D generation, without using any 3D training data.

DietNeRF and Dream Fields relied on priors from discrimative models like self-supervised Vision Transformers and contrastive language-vision dual encoders. However, discriminative models do not necessarily represent all of the visual details needed for high quality synthesis. Chapter 5 briefly discusses two of our works in generative modeling that enable much higher

fidelity cross-modal generation. We first develop a new Denoising Diffusion Probabilistic Model (DDPM) that achieves state-of-the-art sample quality for image synthesis. DDPM has proven to be a highly scalable, stable prior that can be directly trained in different modalities. Still, there will always be a disparity in the amount of training data available in different formats: image datasets currently are many orders of magnitude larger than the biggest 3D datasets. In subsequent work, we find novel ways to transfer diffusion models out of their training modality. We propose the Score Distillation Sampling loss to unlock this transfer capability, and a high quality text-to-3D method called Dream Fusion as its first application.

Building on our work on diffusion models and Score Distillation Sampling, we develop a text-to-SVG method called VectorFusion in Chapter 6 based on a pretrained text-to-image diffusion model. VectorFusion shows the potential for generative models to create abstract, vectorized graphics from text. Throughout this thesis, we built powerful synthesis tools by combining large scale priors learned on data-rich modalities with differentiable renderers that represent tailored modalities useful for downstream tasks. Chapter 7 provides closing thoughts.

Chapter 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

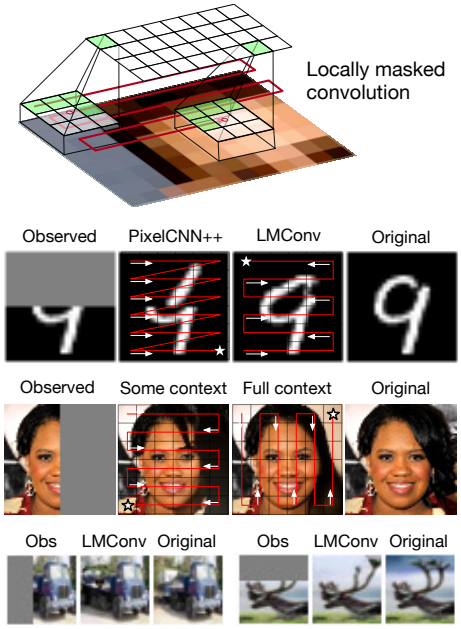


Figure 2.1: The ideal autoregressive joint distribution decomposition and sampling order are task-dependent. We learn to generate images under multiple orderings with the same parameters via *locally masked convolutions* (top), enabling global coherence for image completion (bottom).

frameworks have been proposed. Current state-of-the-art models for high-dimensional image data include (a) autoregressive models [6, 33], (b) normalizing flow density estimators [136], (c) generative adversarial networks (GANs) [43], (d) latent variable models such as the VAE [80, 137] and (e) energy-based models *e.g.* [32, 56, 85, 162]. 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 [33, 129] work could support multiple orders. However, recent CNN-based autoregressive models for images [112, 113, 148] 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 [39], or by masking the input image to hide successor variables [84]. A more efficient approach is to estimate densities in parallel across dimensions by masking network weights [41] 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 [71].

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.

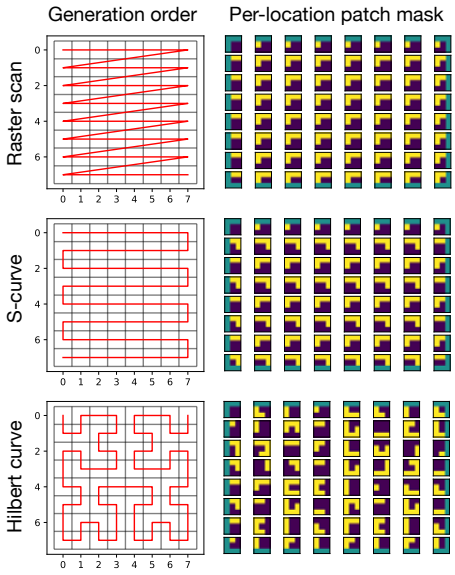


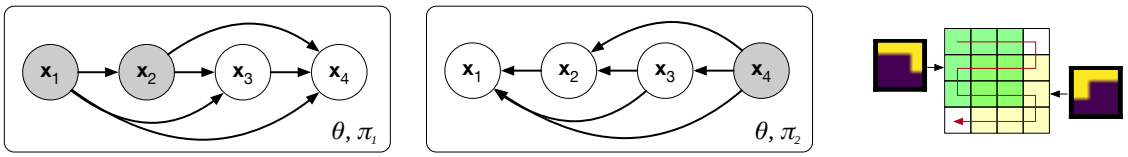
Figure 2.2: The three pixel generation orders and corresponding local masks that we consider in this chapter.

2.2 Background

Deep autoregressive models estimate high-dimensional data distributions using samples from the joint distribution over D-dimensions $p_{\text{data}}(\mathbf{x}_1, \ldots, \mathbf{x}_D)$. In this setting, we wish to approximate the joint with a parametric model $p_\theta(\mathbf{x}_1, \ldots, \mathbf{x}_D)$ 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 [79], these autoregressive (AR) models represent the joint distribution as a product of conditionals,

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

where $\pi : [D] \rightarrow [D]$ is a permutation defining an order over the dimensions, $Pa(\mathbf{x}_{\pi(i)}) = \mathbf{x}_{\pi(1)}, \ldots, \mathbf{x}_{\pi(i-1)}$ 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



(a) Graphical model for suffix completion (b) Graphical model for prefix completion (c) Locally masked convolutions

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.

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 $Pa(\mathbf{x}_{\pi(i)})$.

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 $x_{\pi(2)}, \ldots, x_{\pi(D)}$ 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,

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{x \sim p_{\text{data}}} \sum_{i=1}^{D} \log p_\theta \left(x_{\pi(i)} \mid x_{\pi(1)}, \ldots, x_{\pi(i-1)} \right) \\ \theta^* &= \arg_\theta \max \mathcal{L}(\theta) \end{aligned} \tag{2.2}$$

With some choices of network architecture, the conditionals can be computed in parallel by masking weights [41, 113]. 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 [113], two-stream shift-based convolutional architectures [112], and self-attention combined with convolution [22]. 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 $x_1, \ldots, x_d$, raster scan autoregressive models sequentially sample,

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

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

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

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 $i = 1$,

$$p_\theta(\hat{x}_1 \mid x_{>1}) = \frac{p_\theta(\hat{x}_1, x_{>1})}{\sum_{x_1'} p_\theta(x_1', x_{>1})}, \tag{2.5}$$

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,

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

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 [41, 174]. 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

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

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 $p_\theta(x_{i,j} \mid Pa(p(x_{i,j}))$. 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 [84] 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 [41] 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

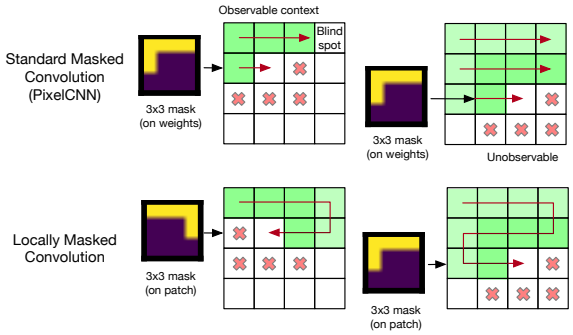


Figure 2.4: A comparison of standard weight masked convolutions and the proposed locally masked convolution.

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 convolutional filter will be applied. Assuming padding and a stride of 1 is used, the rearrangement yields matrix X with $C_{\text{in}} * k^2$ 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 col2im 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 [121]. 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

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: $X = \texttt{im2col}(\texttt{pad}(x), k_1, k_2)$
4: Mask patches: $X = \mathcal{M} \odot X$
5: Perform convolution via batch MM: $Y = \mathcal{W}X + b$
6: Assemble patches: $y = \texttt{col2im}(Y)$
7: **return** y

gradient computation [47, 67]. 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.

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++ [148], the best-in-class density estimator in the fully convolutional autoregressive PixelCNN model family. Convolution operations are masked

Algorithm 2 Create input mask matrix

Input: Generation order $\pi(\cdot)$, constants C_{in}, k_1, k_2, dilation d, is this the first layer?
Start with an empty set of generated coordinates
Initialize $\mathcal{M}$ as $k_1 * k_2 \times H * W$ zero matrix
for i from 1 to $H * W$ **do**
    Let (r, c) be coordinates of dimension $\pi(i)$
    for offsets Δ_r, Δ_c in $k_1 \times k_2$ kernel **do**
        if $(r + d\Delta_r, c + d\Delta_c)$ has been generated **then**
            Allow output location (r, c) to access features at $(r + d\Delta_r, c + d\Delta_c)$ in previous layer: set $\mathcal{M}_{k_2\Delta_r+\Delta_c, Wr+c} = 1$
        end if
    end for
    Add (r, c) to generated coordinates
end for
if not the first layer **then**
    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
end if
Repeat rows of $\mathcal{M}$, C_{in} times
return binary mask matrix $\mathcal{M}$

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 [143] 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 [189] at regular intervals in the model rather than downsampling 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.

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

BINARIZED MNIST, 28x28	**NLL (nats)**
DARN (Intractable) [46]	≈84.13
NADE [174]	88.33
EoNADE 2hl (128 orders) [174]	85.10
EoNADE-5 2hl (128 orders) [131]	84.68
MADE 2hl (32 orders) [41]	86.64
PixelCNN [113]	81.30
PixelRNN [113]	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**

Finally, we normalize the feature map across the channel dimension by applying positional normalization [86]. 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 MNIST dataset [147], 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.

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 [113]	4.70
Attention-based	
Image Transformer [120]	2.90
PixelSNAIL [22]	2.85
Sparse Transformer [23]	**2.80**
Convolutional	
PixelCNN (1 stream) [113]	3.14
Gated PixelCNN (2 stream) [112]	3.03
PixelCNN++ (1 stream)	2.99
PixelCNN++ (2 stream) [148]	2.92
Ours, S-curve (1 stream, 1 order)	2.91
Ours, S-curve (1 stream, 8 orders)	**2.89**

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?*

We estimate the distribution of three image datasets: 28×28 grayscale and binary [147] MNIST digits, 32×32 8-bit color CIFAR10 natural images, and high-resolution CelebA-HQ 5-bit color face photographs [74]. 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.

Table 2.3: Average conditional negative log likelihood for **T**op, **L**eft and **B**ottom 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**

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.

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

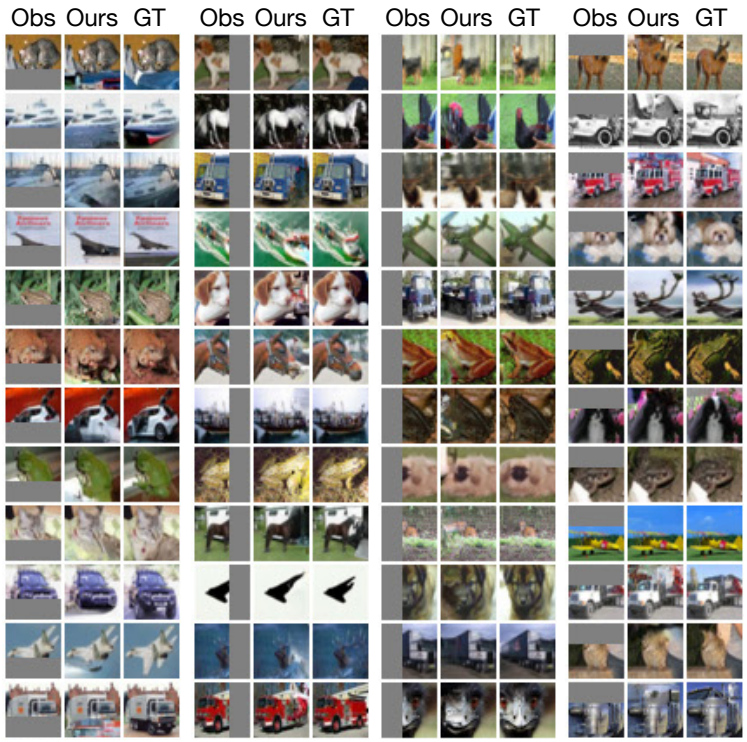


Figure 2.5: CIFAR10 image completions using our locally-masked convolutions with a specialized ordering.

CNNs. These results could further improve with self-attention mechanisms and additional capacity, which have been observed to improve the performance of singe-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 Glow [77], an exact likelihood normalizing flow. In comparison, the state-of-the-art model, SPN [96], 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

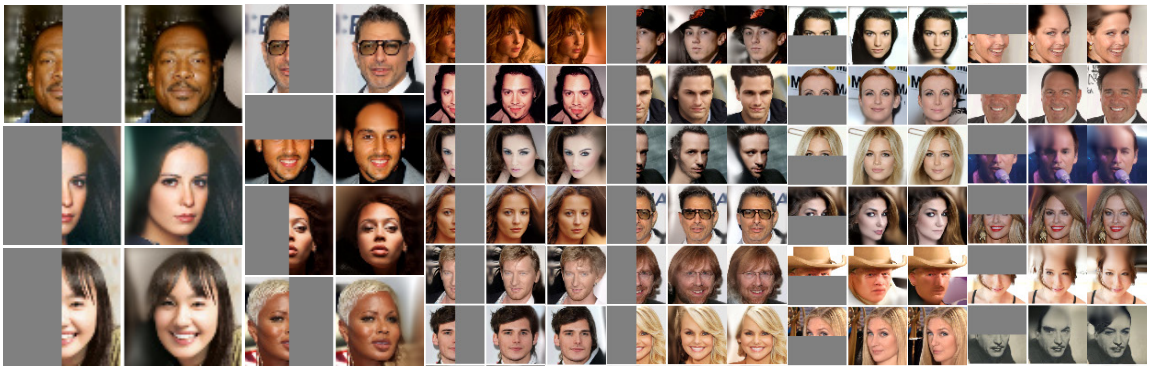


Figure 2.6: Completions of 64 × 64 px CelebA-HQ images at 5-bit color depth. Up to 2 samples are shown to the right of each half-obscured face provided to the model. Missing pixels are generated along an S-curve that first traverses the observed region. Additional samples and ground truth completions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

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.

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. [39] 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. [6] extend this idea, using artificial neural networks to capture conditionals with some parameter sharing. [84] 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. [175] 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. [174] 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 $\pi(1), \ldots, \pi(d)$, sample a training example x and maximize the likelihood of x_d under their model. ConvNADE [173] 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. [41] 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 [98]. [113] 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 [96, 112, 135, 148], 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 [115], and can be sampled in parallel using fixed-point methods [163, 182]. While convolutions process information locally in an image, self-attention mechanisms have been used to gain global receptive field [22, 23, 120] for improved statistical performance.

Normalizing flows [136] 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 [43] have been applied to high resolution image generation [74] and inpainting [124]. Nonparametric approaches have also been successful for inpainting [3, 33, 52]. Partial convolutions [89] improve CNN inpainting quality by rescaling filter responses that access missing pixels, but are not causal unlike LMConv. Latent-variable models like the VAE [80, 137] 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 [8, 108, 159].

2.8 Conclusion

In this chapter, 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, Nilesh 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 chapter are those of the authors and do not necessarily reflect the views of the NSF.

Chapter 3

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

Work by Ajay Jain, Matthew Tancik, and Pieter Abbeel

In Chapter 2, we showed how generative models trained on images can be made more flexible. Our Locally Masked PixelCNNs can outpaint, or extend, regions of images in all directions. Still, even flexible models trained on images are restricted to analyzing, synthesizing and editing images, limiting their application to 2D. How can we extend the utility of 2D image models to new modalities, such as 3D graphics? After all, image datasets contain snapshots of an underlying 3D world, so image models may have greater potential.

In this chapter, 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`.

3.1 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 progress has been made on high-quality view synthesis when many observations are available. A popular approach is to use Neural Radiance Fields (NeRF) [100] 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 3.2B 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 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 3.2D).

Recent work trained NeRF on *multi-view* datasets of similar scenes [153, 167, 171, 180, 188] 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.

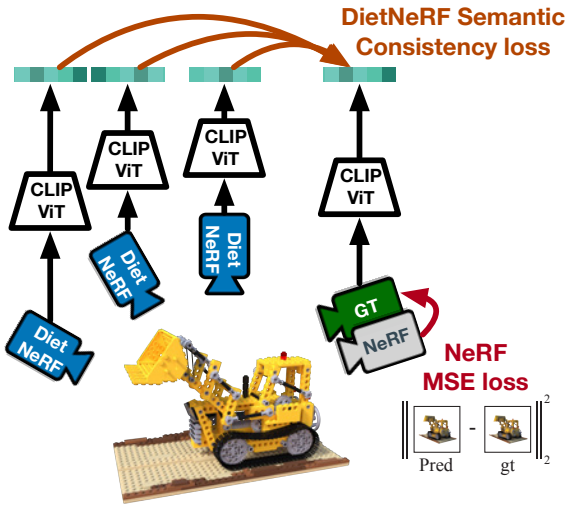


Figure 3.1: Neural Radiance Fields are trained to represent a scene by supervising renderings from the *same pose* as ground-truth observations (**MSE loss**). However, when only a few views are available, the problem is underconstrained. NeRF often finds degenerate solutions unless heavily regularized. 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 [130], 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.

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 [26]. CLIP is a recent multi-modal encoder that is trained to match images with captions in a massive web scrape containing 400M images [130]. 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

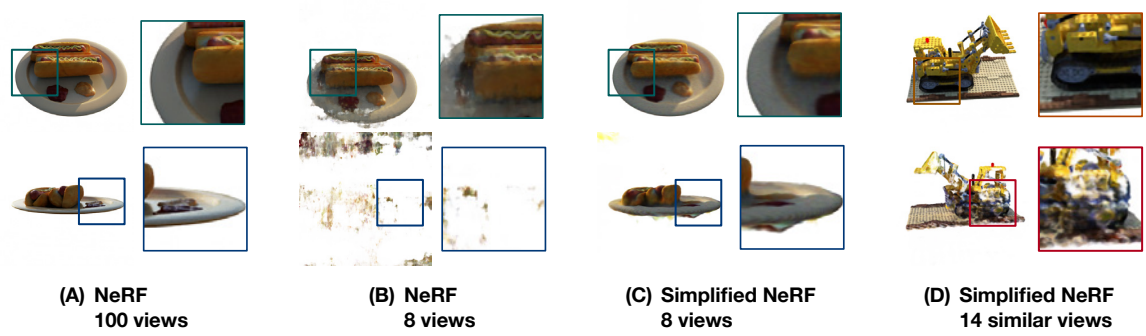


Figure 3.2: **Few-shot view synthesis is a challenging problem for Neural Radiance Fields.** **(A)** When we have 100 observations of an object from uniformly sampled poses, NeRF estimates a detailed and accurate representation that allows for high-quality view synthesis purely from multi-view consistency. **(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.**

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 [30] 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 [188], a generalizable scene representation, and improve perceptual quality.

3.2 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 [100] 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 [72]:

$$\mathbf{C}(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \tag{3.1}$$

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 (3.1). 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 [152] to estimate camera extrinsics (rotations and origins) $\{\mathbf{p}_i\}$, creating a posed dataset $\mathcal{D} = \{(I_i, \mathbf{p}_i)\}$.

3.3 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 (3.1), then minimizing the mean-squared error (MSE) between the images, which should align pixel-wise:

$$\mathcal{L}_{\text{full}}(I, \hat{I}_{\mathbf{p}_i}) = \frac{1}{HW}\|I - \hat{I}_{\mathbf{p}_i}\|_2^2 \tag{3.2}$$

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:

$$\mathcal{L}_{\text{MSE}}(\mathcal{R}) = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} \|\mathbf{C}(\mathbf{r}) - \hat{\mathbf{C}}(\mathbf{r})\|_2^2 \tag{3.3}$$

With many training views, $\mathcal{L}_{\text{MSE}}$ 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 3.2A). Radiance fields with sinusoidal positional embeddings are quite effective at learning high-frequency functions [167], 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. $\mathcal{L}_{\text{MSE}}$ 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 [193] 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 3.2B 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 3.2C shows that these regularizers successfully allow NeRF to recover plausible object geometry. However, high-frequency, fine details are lost compared to 3.2A.

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 3.2D, 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 $\mathcal{L}_{\text{MSE}}$ 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.

3.4 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.

3.4.1 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 $\mathcal{L}_{\text{MSE}}$ 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:

$$\mathcal{L}_{\text{SC},\ell_2}(I, \hat{I}) = \frac{\lambda}{2}\|\phi(I) - \phi(\hat{I})\|_2^2 \tag{3.4}$$

If $\phi(x) = x$, Eq. (3.4) 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 [130]. We then evaluate visual classifiers pre-trained on labeled ImageNet images [30]. 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 [12, 73]. ViT

Algorithm 3 Training DietNeRF on a single scene

Input: Observed views $\mathcal{D} = \{(I, \mathbf{p})\}$, semantic embedding function $\phi(\cdot)$, pose distribution π, consistency interval K, weight λ, rendering size, batch size $|\mathcal{R}|$, lr η_{it}
Result: Trained Neural Radiance Field $f_\theta(\cdot, \cdot)$
Initialize NeRF $f_\theta(\cdot, \cdot)$
Pre-compute target embeddings $\{\phi(I) : I \in \mathcal{D}\}$
for it from 1 to num_iters **do**
    Sample ray batch $\mathcal{R}$, ground-truth colors $\mathbf{C}(\cdot)$
    Render rays $\hat{\mathbf{C}}(\cdot)$ by (3.1)
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{MSE}}(\mathcal{R}, \mathbf{C}, \hat{\mathbf{C}})$
    if it $\% K = 0$ **then**
        Sample target image, pose $(I, \mathbf{p}) \sim \mathcal{D}$
        Sample source pose $\hat{\mathbf{p}} \sim \pi$
        Render image $\hat{I}$ from pose $\hat{\mathbf{p}}$
        $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{\text{SC}}(I, \hat{I})$
    end if
    Update parameters: $\theta \leftarrow Adam(\theta, \eta_{it}, \nabla_\theta \mathcal{L})$
end for

extracts features from non-overlapping image patches in its first layer, then aggregates increasingly abstract representations with Transformer blocks based on global self-attention [176] 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. (3.4) simplifies to cosine similarity up to a constant and a scaling factor that can be absorbed into the loss weight λ:

$$\mathcal{L}_{\text{SC}}(I, \hat{I}) = \lambda \phi(I)^T \phi(\hat{I}) \tag{3.5}$$

We refer to $\mathcal{L}_{\text{SC}}$ (3.5) 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.

3.4.2 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 [27]. For example, the caption "A photo of hotdogs" describes Fig. 3.2A. Language also provides semantically dense learning signal by describing object properties, relationships and appearances [27] 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, [42] 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 3.3, 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.

3.4.3 Pose sampling distribution

We augment the NeRF training loop with $\mathcal{L}_{\text{SC}}$ minimization. Each iteration, we compute $\mathcal{L}_{\text{SC}}$ between a random training image sampled from the observation dataset $I \sim \mathcal{D}$ and rendered image $\hat{I}_{\mathbf{p}}$ 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 $\alpha_1, \alpha_2 \sim \mathcal{U}(0, 1)$.

3.4.4 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 $\mathcal{L}_{\text{SC}}$ converges faster than $\mathcal{L}_{\text{MSE}}$ 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 $\mathcal{L}_{\text{SC}}$ by only minimizing $\mathcal{L}_{\text{SC}}$ every k iterations. DietNeRF is robust to the choice of k, but a value between 10 and 16 worked well in our experiments. StyleGAN2 [75] used a similar strategy for efficiency, referring to periodic application of a loss as *lazy regularization*.

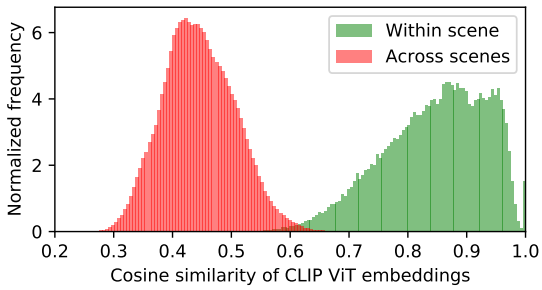


Figure 3.3: CLIP's Vision Transformer learns low-dimensional image representations through language supervision. We find that these representations transfer well to multi-view 3D settings. We sample pairs of ground-truth views of the same scene and of different scenes from NeRF's Realistic Synthetic 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**)

As backpropagation through rendering is memory intensive with reverse-mode automatic differentiation, we render images for $\mathcal{L}_{\text{SC}}$ 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 [20, 66]. All experiments use a single 16 GB NVIDIA V100 or 11 GB 2080 Ti GPU.

Since $\mathcal{L}_{\text{SC}}$ converges before $\mathcal{L}_{\text{MSE}}$, we found it helpful to fine-tune DietNeRF with $\mathcal{L}_{\text{MSE}}$ alone for 20-70k iterations to refine details. Alg. 3 details our overall training process.

3.5 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 §3.5.1. (2) We show that DietNeRF improves perceptual quality of view synthesis from *only a single real photo* §3.5.2. (3) We find that DietNeRF can reconstruct regions that are never observed §3.5.3, and finally (4) run ablations §3.6.

Datasets The Realistic Synthetic benchmark of [99] 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 [68] used by pixelNeRF [188]. DTU is a challenging dataset that includes sparsely sampled real photos of physical objects.

Table 3.1: Quality metrics for novel view synthesis on subsampled splits of the Realistic Synthetic dataset [100]. We randomly sample 8 views from the available 100 ground truth training views to evaluate how DietNeRF performs with limited observations.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	FID ↓	KID ↓
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, $\mathcal{L}_{\text{MSE}}$ ft	**23.591**	**0.874**	**0.097**	**72.0**	**0.004**
NeRF, 100 views	**31.153**	**0.954**	**0.046**	**50.5**	**0.001**

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**) [158]. PSNR expresses mean-squared error in log space. However, SSIM often disagrees with human judgements of similarity [194].

Perceptual metrics Deep CNN activations mirror aspects of human perception. NeRF measures perceptual image quality using **LPIPS** [194], which computes MSE between normalized features from all layers of a pre-trained VGG encoder [155]. Generative models also measure sample quality with feature space distances. The Fréchet Inception Distance (**FID**) [55] computes the Fréchet distance between Gaussian estimates of penultimate Inception v3 [166] 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 [9, 109]. All metrics use a different architecture and data than our CLIP ViT encoder.

3.5.1 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 3.1 shows results. The original NeRF model achieves much poorer quantitative quality with 8 images than with the full 100 image dataset. Neural Volumes [92] 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

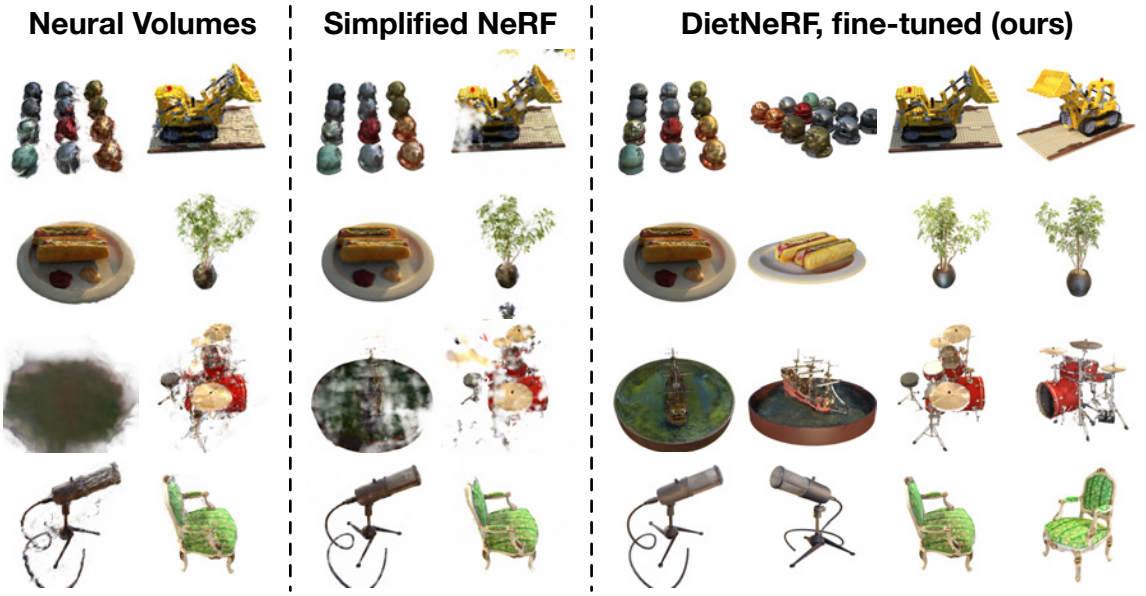


Figure 3.4: Novel views synthesized from eight observations of scenes in the Realistic Synthetic dataset.

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 $\mathcal{L}_{\text{SC}}$ loss. Additionally, fine-tuning with $\mathcal{L}_{\text{MSE}}$ even further improves quality, for a total improvement of +8.5 PSNR, -0.2 LPIPS, and -156 FID over NeRF. This shows that semantic consistency is a valuable prior for high-quality few-shot view synthesis. Figure 3.4 visualizes results.

3.5.2 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 $\mathcal{L}_{\text{MSE}}$, so NeRF performs poorly in the single-view setting (Table 3.2).

To perform single- or few-shot view synthesis, pixelNeRF [188] 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 3.2 shows that pixelNeRF significantly outperforms NeRF given a single photo of a held-out scene. However, novel views are blurry and unrealistic (Figure 3.5). We propose to fine-tune pixelNeRF on a single scene using $\mathcal{L}_{\text{MSE}}$ alone or using both $\mathcal{L}_{\text{MSE}}$ and $\mathcal{L}_{\text{SC}}$. Fine-tuning per-scene with MSE improves local image

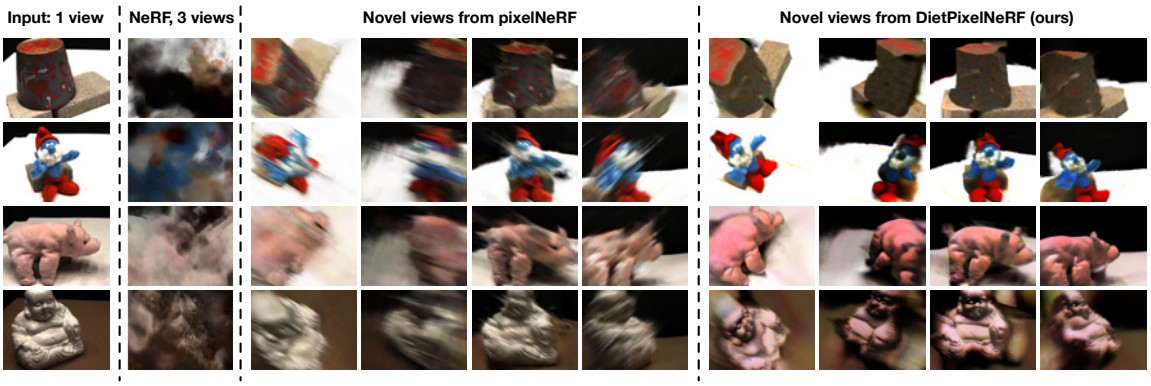


Figure 3.5: **Novel views synthesized from a *single input image*** from the DTU object dataset. Even with 3 input views, NeRF [100] fails to learn accurate geometry or textures (reprinted from [188]). While pixelNeRF [188] has mostly consistent object geometry as the camera pose is varied, renderings 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.

quality metrics, but only slightly helps perceptual metrics. Figure 3.6 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. 3.5, 3.6). DietPixelNeRF outperforms baselines on perceptual LPIPS, FID, and KID metrics (Tab. 3.2). 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.

3.5.3 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. 3.7, 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

Table 3.2: **Single-view novel view synthesis on the DTU dataset**. NeRF and pixelNeRF PSNR, SSIM and LPIPS results are from [188]. 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, $\mathcal{L}_{\text{MSE}}$ ft	**16.048**	**0.564**	0.515	265.2	0.159
DietPixelNeRF	14.242	0.481	**0.487**	**190.7**	**0.066**

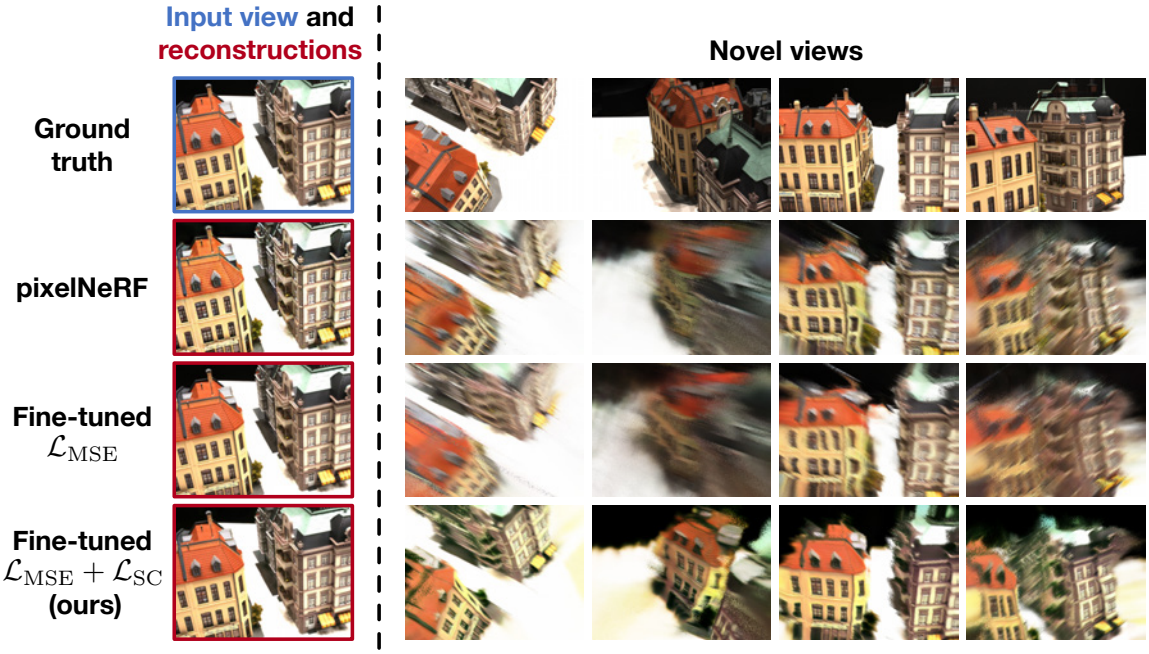


Figure 3.6: **Semantic consistency improves perceptual quality.** Fine-tuning pixelNeRF with $\mathcal{L}_{\text{MSE}}$ slightly improves a rendering of the input view, but does not remove most perceptual flaws like blurriness in *novel views*. Fine-tuning with both $\mathcal{L}_{\text{MSE}}$ and $\mathcal{L}_{\text{SC}}$ (DietPixelNeRF, bottom) improves sharpness of all views.

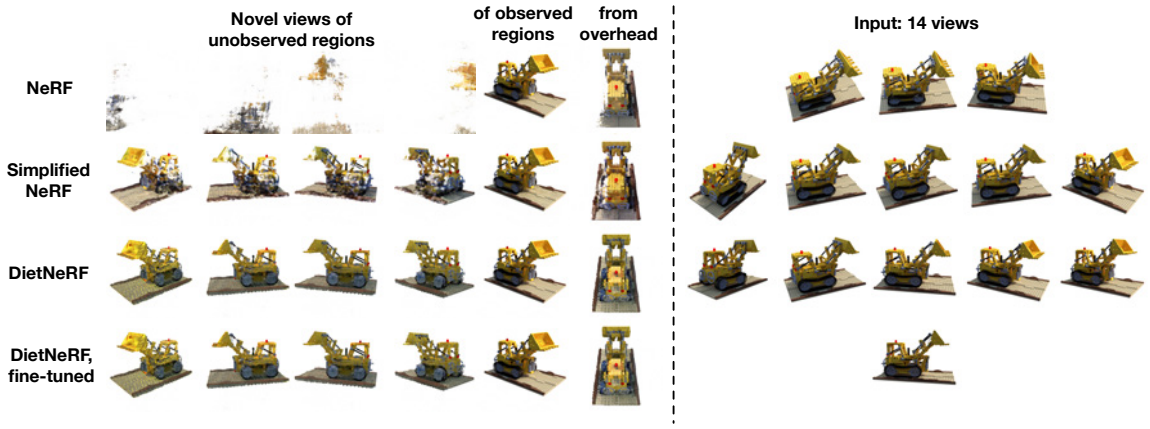


Figure 3.7: **Renderings of occluded regions during training.** 14 images of the right half of the Realistic Synthetic lego scene are used to estimate radiance fields. NeRF either learns high-opacity occlusions blocking the left of the object, or fails to generalize properly to the unseen left side. In contrast, DietNeRF fills in details for a reconstruction that is mostly consistent with the observed half.

Table 3.3: **Extrapolation metrics.** Novel view synthesis with observations of **only one side** of the Realistic Synthetic Lego scene.

Views	Method	PSNR ↑	SSIM ↑	LPIPS ↓
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 + $\mathcal{L}_{\text{MSE}}$ ft	**22.211**	**0.824**	**0.143**
100	NeRF [100]	**31.618**	**0.965**	**0.033**

the scene, while our Simplified NeRF learns unrealistic deformations and incorrect colors (Fig. 3.7, left). Remarkably, DietNeRF learns quantitatively (Tab. 3.3) and qualitatively more accurate colors in the missing regions, suggesting the value of semantic image priors for sparse reconstruction problems. We exclude FID and KID since a single scene has too few samples for an accurate estimate.

3.6 Ablations

Choosing an image encoder Table 3.4 shows quality metrics with different semantic encoder architectures and pre-training datasets. We evaluate on the Lego scene with 8 views. Large ViT

Table 3.4: **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 ↑	SSIM ↑	LPIPS ↓
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 3.5: **Varying the number of iterations that DietNeRF is fine-tuned with $\mathcal{L}_{\text{MSE}}$** on Realistic Synthetic scenes. All models are initially trained for 200k iterations with $\mathcal{L}_{\text{MSE}}$ and $\mathcal{L}_{\text{SC}}$. Further minimizing $\mathcal{L}_{\text{MSE}}$ is helpful, but the model can overfit.

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

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

Varying $\mathcal{L}_{\text{MSE}}$ fine-tuning duration Fine-tuning DietNeRF with $\mathcal{L}_{\text{MSE}}$ can improve quality by better reconstructing fine-details. In Table 3.5, 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.

3.7 Related work

Few-shot radiance fields Several works condition NeRF on latent codes describing scene geometry or appearance rather than estimating NeRF per scene [153, 171, 188]. 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 [153]. Concurrent to our work, IBRNet [180] 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, [40, 167] 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 [156]. Much literature studies single-view reconstruction with other, explicit 3D representations. Notable recent examples include voxel [172], mesh [60] and point-cloud [183] approaches.

Novel view synthesis, image-based rendering Neural Volumes [92] proposes a VAE [80, 138] 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 [111] learn an autoencoder with a 3-dimensional latent that can be rotated to render new perspectives for a single-category. SRNs [158] 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 [83]. Local Light Field Fusion [99] estimates and blends multiple MPI representations for each scene. Free View Synthesis [140] uses geometric approaches to improve view synthesis in unbounded in-the-wild scenes. NeRF++ [193] 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 [7]. Without labels, generative models can learn useful representations for recognition [19], but self-supervised models like CPC [54, 114] tend to be more parameter efficient. Contrastive methods including CLIP learn visual representations by matching similar pairs of items, such as captions and images [70, 130], augmentated variants of an image [21], or video patches across frames [62].

3.8 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 chapter 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.

Chapter 4

Zero-Shot Text-Guided Object Generation with Dream Fields

Work by Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel and Ben Poole

While DietNeRF paved the way toward incorporating image-text representations in 3D reconstruction, it still required several input photographs. In many graphics applications, new assets are digitally designed that have never existed in the real world. In this chapter, we extend DietNeRF to synthesize 3D objects from natural language descriptions alone, without any input images.

We combine neural rendering with multi-modal image and text representations to synthesize diverse 3D objects solely from natural language descriptions. Our method, Dream Fields, can generate the geometry and color of a wide range of objects without 3D supervision. Due to the scarcity of diverse, captioned 3D data, prior methods only generate objects from a handful of categories, such as ShapeNet. Instead, we guide generation with image-text models pre-trained on large datasets of captioned images from the web. Our method optimizes a Neural Radiance Field from many camera views so that rendered images score highly with a target caption according to a pre-trained CLIP model. To improve fidelity and visual quality, we introduce simple geometric priors, including sparsity-inducing transmittance regularization, scene bounds, and new MLP architectures. In experiments, Dream Fields produce realistic, multi-view consistent object geometry and color from a variety of natural language captions. Project website and code: `https://ajayj.com/dreamfields`.

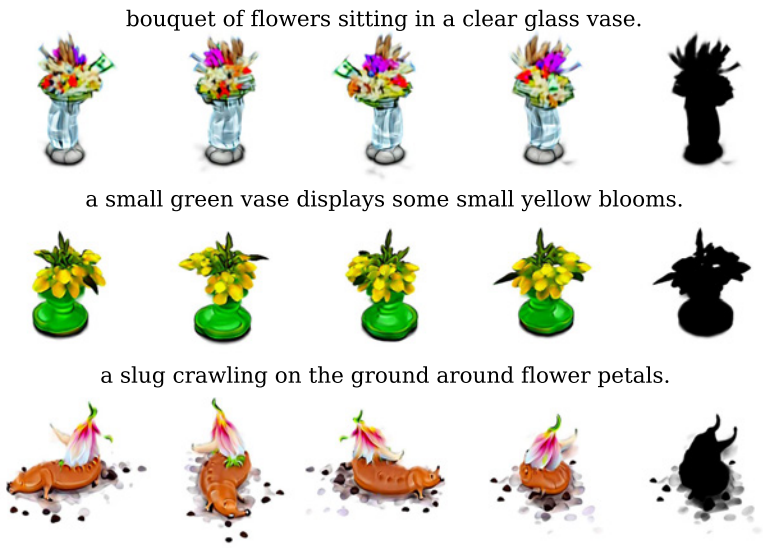


Figure 4.1: Example Dream Fields generated from captions, then rendered from four perspectives. On the right, we show transmittance from the final perspective. We create diverse outputs using the compositionality of language; these captions from MSCOCO describe three flower arrangements with different properties like context and color.

4.1 Introduction

Detailed 3D object models bring multimedia experiences to life. Games, virtual reality applications and films are each populated with thousands of object models, each designed and textured by hand with digital software. While expert artists can author high-fidelity assets, the process is painstakingly slow and expensive. Prior work leverages 3D datasets to synthesize shapes in the form of point clouds, voxel grids, triangle meshes, and implicit functions using generative models like GANs [11, 49, 184, 196]. These approaches only support a few object categories due to small labeled 3D shape datasets. But multimedia applications require a wide variety of content, and need both 3D geometry and texture.

In this work, we propose Dream Fields, a method to automatically generate open-set 3D models from natural language prompts. Unlike prior work, our method does not require any 3D training data, and uses natural language prompts that are easy to author with an expressive interface for specifying desired object properties. We demonstrate that the compositionality of language allows for flexible creative control over shapes, colors and styles.

A Dream Field is a Neural Radiance Field (NeRF) trained to maximize a deep perceptual metric with respect to both the geometry and color of a scene. NeRF and other neural 3D representations

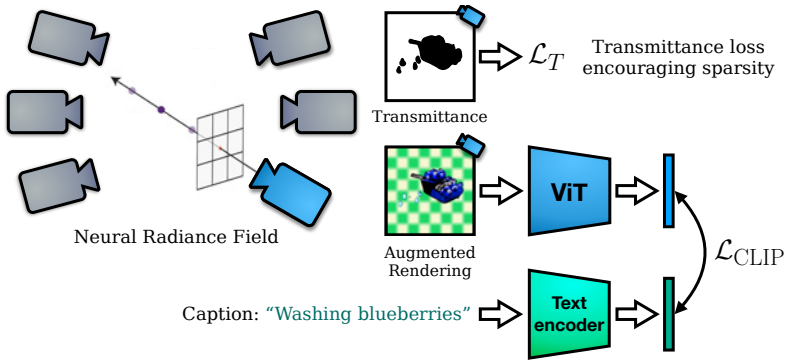


Figure 4.2: Given a caption, we learn a **Dream Field**, a continuous volumetric representation of an object's geometry and appearance learned with guidance from a pre-trained model. We optimize the Dream Field by rendering images of the object from random camera poses that are scored with *frozen* pre-trained **image** and **text** encoders trained on web images and alt-text. 2D views are derived from the same radiance field for 3D consistency.

have recently been successfully applied to novel view synthesis tasks where ground-truth RGB photos are available. NeRF is trained to reconstruct images from multiple viewpoints. As the learned radiance field is shared across viewpoints, NeRF can interpolate between viewpoints smoothly and consistently. Due to its neural representation, NeRF can be sampled at high spatial resolutions unlike voxel representations and point clouds, and are easy to optimize unlike explicit geometric representations like meshes as it is topology-free.

However, existing photographs are not available when creating novel objects from descriptions alone. Instead of learning to reconstruct known input photos, we learn a radiance field such that its renderings have high semantic similarity with a given text prompt. We extract these semantics with pre-trained neural image-text retrieval models like CLIP [130], learned from hundreds of millions of captioned images. As NeRF's volumetric rendering and CLIP's image-text representations are differentiable, we can optimize Dream Fields end-to-end for each prompt. Figure 4.2 illustrates our method.

In experiments, Dream Fields learn significant artifacts if we naively optimize the NeRF scene representation with textual supervision without adding additional geometric constraints (Figure 4.3). We propose general-purpose priors and demonstrate that they greatly improve the realism of results. Finally, we quantitatively evaluate open-set generation performance using a dataset of diverse object-centric prompts.

Our contributions include:

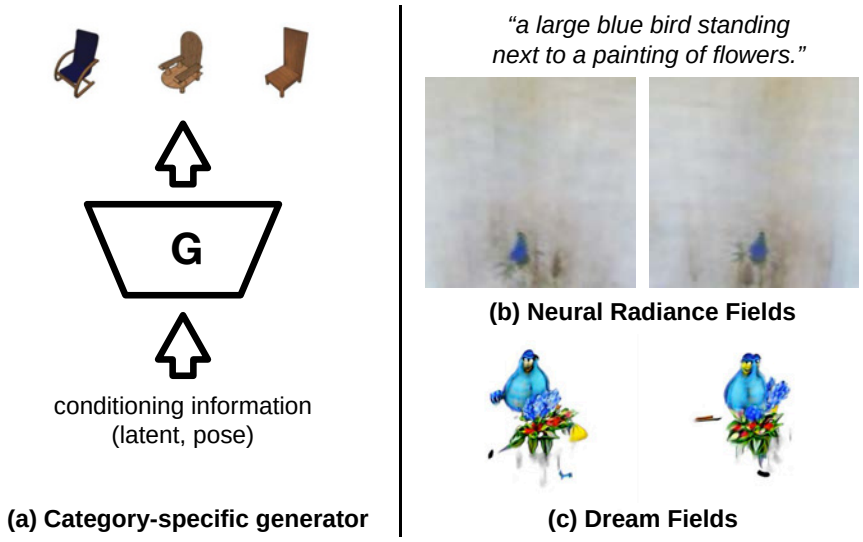


Figure 4.3: **Challenges of text-to-3D synthesis: (a) Poor generalization from limited 3D datasets:** Most 3D generative models are learned on datasets of specific object categories like ShapeNet [16], and won't generalize to novel concepts zero-shot. **(b) Neural Radiance Fields are too flexible without multi-view supervision:** NeRF learns to represent geometry and texture from scene-specific multi-view data, so it does not require a diverse dataset of objects. Yet, when only a source caption is available instead of multi-view images, NeRF produces significant artifacts (*e.g.*, near field occlusions). **(c) Dream Fields:** We introduce general geometric priors that retain much of NeRF's flexibility while improving realism.

- Using aligned image and text models to optimize NeRF without 3D shape or multi-view data,
- Dream Fields, a simple, constrained 3D representation with neural guidance that supports diverse 3D object generation from captions in zero-shot, and
- Simple geometric priors including transmittance regularization, scene bounds, and an MLP architecture that together improve fidelity.

4.2 Related Work

Our work is primarily inspired by DeepDream [102] and other methods for visualizing the preferred inputs and features of neural networks by optimizing in image space [104, 105, 110]. These methods enable the generation of interesting images from a pre-trained neural network without the additional training of a generative model. Closest to our work is [103], which studies differentiable

image parameterizations in the context of style transfer. Our work replaces the style and content-based losses from that era with an image-text loss enabled by progress in contrastive representation learning on image-text datasets [27, 70, 130, 191]. The use of image-text models enables easy and flexible control over the style and content of generated imagery through textual prompt design. We optimize both geometry and color using the differentiable volumetric rendering and scene representation provided by NeRF, whereas [103] was restricted to fixed geometry and only optimized texture. Together these advances enable a fundamentally new capability: open-ended text-guided generation of object geometry and texture.

Concurrently to Dream Fields, a few early works have used CLIP [130] to synthesize or manipulate 3D object representations. CLIP-Forge [149] generates multiple object geometries from text prompts using a CLIP embedding-conditioned normalizing flow model and geometry-only decoder trained on ShapeNet categories. Still, CLIP-Forge generalizes poorly outside of ShapeNet categories and requires ground-truth multi-view images and voxel data. Text2Shape [18] learns a text-conditional Wasserstein GAN [2, 44] to synthesize novel voxelized objects, but only supports finite resolution generation of individual ShapeNet categories. In [24], object geometry is optimized evolutionarily for high CLIP score from a single view then manually colored. ClipMatrix [69] edits the vertices and textures of human SMPL models [93] to create stylized, deformable humanoid meshes. [125] creates an interactive interface to edit signed-distance fields in localized regions, though they do not optimize texture or synthesize new shapes. Text-based manipulation of existing objects is complementary to us.

For images, there has been an explosion of work that leverages CLIP to guide image generation. Digital artist Ryan Murdock (@advadnoun) used CLIP to guide learning of the weights of a SIREN network [157], similar to NeRF but without volume rendering and focused on image generation. Katherine Crowson (@rivershavewings) combined CLIP with optimization of VQ-GAN codes [35] and used diffusion models as an image prior [29]. Recent work from Mario Klingemann (@quasimondo) and [123] have shown how CLIP can be used to guide GAN models like StyleGAN [76]. Some works have optimized parameters of vector graphics, suggesting CLIP guidance is highly general [37, 63, 150]. These methods highlighted the surprising capacity of what image-text models have learned and their utility for guiding 2D generative processes. Direct text to image synthesis with generative models has also improved tremendously in recent years [134, 192], but requires training large generative models on large-scale datasets, making such methods challenging to directly apply to text to 3D where no such datasets exist.

There is also growing progress on generative models with NeRF-based generators trained solely from 2D imagery. However, these models are category-specific and trained on large datasets

of mostly forward-facing scenes [14, 48, 107, 153, 197], lacking the flexibility of open-set text-conditional models. Shape-agnostic priors have been used for 3D reconstruction [4, 185, 195].

4.3 Background

Our method combines Neural Radiance Fields (NeRF) [100] with an image-text loss from [130]. We begin by discussing these existing methods, and then detail our improved approach and methodology that enables high quality text to object generation.

4.3.1 Neural Radiance Fields

NeRF [100] parameterizes a scene's density and color using a multi-layer perceptron (MLP) with parameters θ trained with a photometric loss relying on multi-view photographs of a scene. In our simplified model, the NeRF network takes in a 3D position $\mathbf{x}$ and outputs parameters for an emission-absorption volume rendering model: density $\sigma_\theta(\mathbf{x})$ and color $\mathbf{c}_\theta(\mathbf{x})$. Images can be rendered from desired viewpoints by integrating color along an appropriate ray, $\mathbf{r}(t)$, for each pixel according to the volume rendering equation:

$$\mathbf{C}(r,\theta) = \int_{t_n}^{t_f} T(\mathbf{r},t)\sigma_\theta(\mathbf{r}(t))\mathbf{c}_\theta(\mathbf{r}(t))dt, \tag{4.1}$$

$$\text{where } T(\mathbf{r},\theta,t) = \exp\left(-\int_{t_n}^{t}\sigma_\theta(\mathbf{r}(s))ds\right). \tag{4.2}$$

The integral $T(\mathbf{r},\theta,t)$ is known as "transmittance" and describes the probability that light along the ray will not be absorbed when traveling from t_n (the near scene bound) to t. In practice [100], these two integrals are approximated by breaking up the ray into smaller segments $[t_{i-1}, t_i)$ within which σ and $\mathbf{c}$ are assumed to be roughly constant:

$$\mathbf{C}(\mathbf{r},\theta) \approx \sum_i T_i(1-\exp(-\sigma_\theta(\mathbf{r}(t_i))\delta_i))\mathbf{c}_\theta(\mathbf{r}(t_i)) \tag{4.3}$$

$$T_i = \exp\left(-\textstyle\sum_{j<i}\sigma_\theta(\mathbf{r}(t_j))\delta_j\right), \quad \delta_i = t_i - t_{i-1}\,. \tag{4.4}$$

For a given setting of MLP parameters θ and pose $\mathbf{p}$, we determine the appropriate ray for each pixel, compute rendered colors $C(\mathbf{r},\theta)$ and transmittances, and gather the results to form the rendered image, $I(\theta,\mathbf{p})$ and transmittance $T(\theta,\mathbf{p})$.

In order for the MLP to learn high frequency details more quickly [169], the input $\mathbf{x}$ is preprocessed by a sinusoidal positional encoding γ before being passed into the network:

$$\gamma(\mathbf{x}) = \left[\cos(2^l\mathbf{x}), \sin(2^l\mathbf{x})\right]_{l=0}^{L-1}, \tag{4.5}$$

where L is referred to as the number of "levels" of positional encoding. In our implementation, we specifically apply the integrated positional encoding (IPE) proposed in mip-NeRF to combat aliasing artifacts [5] combined with a random Fourier positional encoding basis [169] with frequency components sampled according to

$$\omega = 2^u\mathbf{d}, \quad \text{where } u \sim \mathcal{U}[0, L], \;\; \mathbf{d} \sim \mathcal{U}(\mathbb{S}^2)\,. \tag{4.6}$$

4.3.2 Image-text models

Large-scale datasets of images paired with associated text have enabled training large-scale models that can accurately score whether an image and an associated caption are likely to correspond [27, 70, 130]. These models consist of an image encoder $\mathbf{g}$, and text encoder $\mathbf{h}$, that map images and text into a shared embedding space. Given a sentence y and an image $\mathbf{I}$, these image-text models produce a scalar score: $\mathbf{g}(\mathbf{I})^{\mathrm{T}}\mathbf{h}(\mathbf{y})$ that is high when the text is a good description of the image, and low when the the image and text are mismatched. Note that the embeddings $\mathbf{g}(\mathbf{I})$ and $\mathbf{h}(\mathbf{y})$ are often normalized, i.e. $\|\mathbf{g}(\mathbf{I})\| = \|\mathbf{h}(\mathbf{y})\| = 1$. Training is typically performed with a symmetric version of the InfoNCE loss [117, 128] that aims to maximize a variational lower bound on the mutual information between images and text. Prior work has shown that once trained, the image and text encoders are useful for a number of downstream tasks [130, 191]. In [134], the image and text encoders are used to score the correspondence of outputs of a generative image model to a target caption [134]. We build on this work by optimizing a volume to produce a high-scoring image, not just reranking.

4.4 Method

In this section, we develop Dream Fields: a zero-shot object synthesis method given only a natural language caption.

4.4.1 Object representation

Building on the NeRF scene representation (Section 4.3.1), a Dream Field optimizes an MLP with parameters θ that produces outputs $\sigma_\theta(\mathbf{x})$ and $\mathbf{c}_\theta(\mathbf{x})$ representing the differential volume density and color of a scene at every 3D point $\mathbf{x}$. This field expresses object geometry via the density network. Our object representation is only dependent on 3D coordinates and not the camera's viewing direction, as we did not find it beneficial. Given a camera pose $\mathbf{p}$, we can render an image $\mathbf{I}(\theta, \mathbf{p})$ and compute the transmittance $T(\theta, \mathbf{p})$ using N segments via (4.4). Segments are spaced at roughly equal intervals with random jittering along the ray. The number of segments, N, determines the fidelity of the rendering. In practice, we fix it to 192 during optimization.

4.4.2 Objective

How can we train a Dream Field to represent a given caption? If we assume that an object can be described similarly when observed from any perspective, we can randomly sample poses and try to enforce that the rendered image matches the caption at all poses. We can implement this idea by using a CLIP network to measure the match between a caption and image given parameters θ and pose $\mathbf{p}$:

$$\mathcal{L}_{\text{CLIP}}(\theta, \text{pose } \mathbf{p}, \text{caption } y) = -\mathbf{g}(\mathbf{I}(\theta, \mathbf{p}))^{\mathrm{T}}\mathbf{h}(\mathbf{y}) \tag{4.7}$$

where $\mathbf{g}(\cdot)$ and $\mathbf{h}(\cdot)$ are aligned representations of image and text semantics, and $\mathbf{I}(\theta, \mathbf{p})$ is a rendered image of the scene from camera pose $\mathbf{p}$. Each iteration of training, we sample a pose $\mathbf{p}$ from a prior distribution, render $\mathbf{I}$, and minimize $\mathcal{L}_{\text{CLIP}}$ with respect to the parameters of the Dream Field MLP, θ. Equation (4.7) measures the similarity of an image and the provided caption in feature space.

We primarily use image and text encoders from CLIP [130], which has a Vision Transformer image encoder $\mathbf{g}(\cdot)$ [31] and masked transformer text encoder $\mathbf{h}(\cdot)$ [176] trained contrastively on a large dataset of 400M captioned 224^2 images. We also use a baseline Locked Image-Text Tuning (LiT) ViT B/32 model from [191] trained via the same procedure as CLIP on a larger dataset of billions of higher-resolution (288^2) captioned images. The LiT training set was collected following a simplified version of the ALIGN web alt-text dataset collection process [70] and includes noisy captions.

Figure 4.2 shows a high-level overview of our method. DietNeRF [65] proposed a related semantic consistency regularizer for NeRF based on the idea that "a bulldozer is a bulldozer from any perspective". The method computed the similarity of a rendered and a real image. In contrast,

(4.7) compares rendered images and a *caption*, allowing it to be used in zero-shot settings when there are no object photos.

4.4.3 Challenges with CLIP guidance

Due to their flexibility, Neural Radiance Fields are capable of high-fidelity novel view synthesis on a tremendous diversity of real-world scenes when supervised with multi-view consistent images. Their reconstruction loss will typically learn to remove artifacts like spurious density when sufficiently many input images are available. However, we find that the NeRF scene representation is too unconstrained when trained solely with $\mathcal{L}_{\text{CLIP}}$ (4.7) alone from a discrete set of viewpoints, resulting in severe artifacts that satisfy $\mathcal{L}_{\text{CLIP}}$ but are not visually compatible according to humans (see Figure 4.3b). NeRF learns high-frequency and near-field [193] artifacts like partially-transparent "floating" regions of density. It also fills the entire camera viewport rather than generating individual objects. Geometry is unrealistic, though textures reflect the caption, reminiscent of the artifacts in Deep Dream feature visualizations [102, 110].

4.4.4 Pose sampling

Image data augmentations such as random crops are commonly used to improve and regularize image generation in DeepDream [102] and related work. Image augmentations can only use in-plane 2D transformations. Dream Fields support 3D data augmentations by sampling different camera pose extrinsics at each training iteration. We uniformly sample camera azimuth in 360° around the scene, so each training iteration sees a different orientation of the object. As the underlying scene representation is shared, this improves the realism of object geometry. For example, sampling azimuth in a narrow interval tended to create flat, billboard geometry.

The camera elevation, focal length and distance from the subject can also be augmented, but we did not find this necessary. Instead, we use a fixed camera focal length during optimization that is scaled by $m_{\text{focal}} = 1.2$ to enlarge the object 20%. Rendering cost is constant in the focal length.

4.4.5 Encouraging coherent objects through sparsity

To remove near-field artifacts and spurious density, we regularize the opacity of Dream Field renderings. Our best results maximize the average transmittance of rays passing through the volume up to a target constant. Transmittance is the probability that light along ray r is not absorbed by participating media when passing between point t along the ray and the near plane at t_n (4.2). We approximate the total transmittance along the ray as the joint probability of light

Figure 4.4: To encourage coherent foreground objects, Dream Fields train with 3 types of background augmentations: blurred Gaussian noise, textures and checkerboards. At test time, we render with a white background. **Prompt:** *"A sculpture of a rooster."*

passing through N discrete segments of the ray according to Eq. (4.4). Then, we define the following transmittance loss:

$$\mathcal{L}_T = -\min(\tau, \text{mean}(T(\theta, \mathbf{p}))) \tag{4.8}$$

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CLIP}} + \lambda \mathcal{L}_T \tag{4.9}$$

This encourages a Dream Field to increase average transmittance up to a target transparency τ. We use $\tau = 88\%$ in experiments. τ is annealed in from $\tau = 40\%$ over 500 iterations to smoothly introduce transparency, which improves scene geometry and is essential to prevent completely transparent scenes. Scaling $1 - \tau \propto f^2/d^2$ preserves object cross sectional area for different focal and object distances.

When the rendering is alpha-composited with a simple white or black background during training, we find that the average transmittance approaches τ, but the scene is diffuse as the optimization populates the background. Augmenting the scene with random background images leads to coherent objects. Dream Fields use Gaussian noise, checkerboard patterns and the random Fourier textures from [103] as backgrounds. These are smoothed with a Gaussian blur with randomly sampled standard deviation. Background augmentations and a rendering during training are shown in Figure 4.4.

We qualitatively compare (4.9) to baseline sparsity regularizers in Figure 4.5. Our loss is inspired by the multiplicative opacity gating used by [103]. However, the gated loss has optimization challenges in practice due in part to its non-convexity. The simplified additive loss is more stable, and both are significantly sharper than prior approaches for sparsifying Neural Radiance Fields.

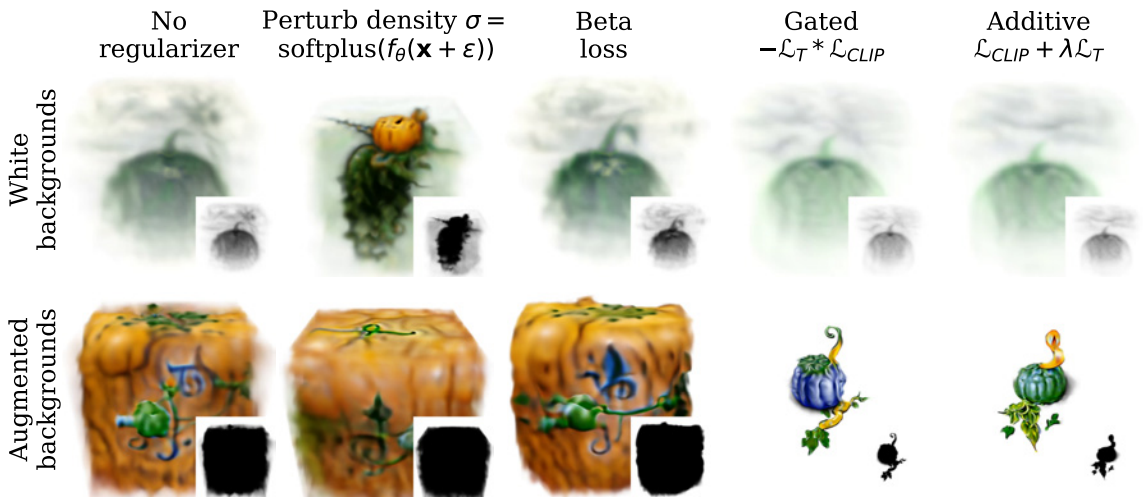


Figure 4.5: Our transmittance losses and background augmentations are complementary. **Top:** Without background augmentations, priors on transmittance (right three columns) do not remove low-density structures. NeRF's density perturbations improve coherence, but cloudy artifacts remain. **Bottom:** When the object is alpha composited with random backgrounds during training, CLIP fills the scene with opaque material to conceal the background. However, gated and our simplified additive transmittance regularizers both limit the opacity of the volume successfully and lead to a sharper object. Inset panels depict transmittance. **Prompt:** *"an illustration of a pumpkin on the vine."*

4.4.6 Localizing objects and bounding scene

When Neural Radiance Fields are trained to reconstruct images, scene contents will align with observations in a consistent fashion, such as the center of the scene in NeRF's Realistic Synthetic dataset [100]. Dream Fields can place density away from the center of the scene while still satisfying the CLIP loss as natural images in CLIP's training data will not always be centered. During training, we maintain an estimate of the 3D object's origin and shift rays accordingly. The origin is tracked via an exponential moving average of the center of mass of rendered density. To prevent objects from drifting too far, we bound the scene inside a cube by masking the density σ_θ.

4.4.7 Neural scene representation architecture

The NeRF network architecture proposed in [100] parameterizes scene density with a simple 8-layer MLP of constant width, and radiance with an additional two layers. We use a residual MLP architecture instead that introduces residual connections around every two dense layers. Within a residual block, we find it beneficial to introduce Layer Normalization at the beginning and increase

the feature dimension in a bottleneck fashion. Layer Normalization improves optimization on challenging prompts. To mitigate vanishing gradient issues in highly transparent scenes, we replace ReLU activations with Swish [132] and rectify the predicted density σ_θ with a softplus function. Our MLP architecture uses 280K parameters per scene, while NeRF uses 494K parameters.

4.5 Evaluation

We evaluate the consistency of generated objects with their captions and the importance of scene representation, then show qualitative results and test whether Dream Fields can generalize compositionally. Ablations analyze regularizers, CLIP and camera poses. Finally, supplementary materials have further examples and videos.

4.5.1 Experimental setup

3D reconstruction methods are evaluated by comparing the learned geometry with a ground-truth reference model, *e.g.* with Chamfer Distance. Novel view synthesis techniques like LLFF [99] and NeRF do not have ground truth models, but compare renderings to pixel-aligned ground truth images from held-out poses with PSNR or LPIPS, a deep perceptual metric [194].

As we do not have access to diverse captioned 3D models or captioned multi-view data, Dream Fields are challenging to evaluate with geometric and image reference-based metrics. Instead, we use the CLIP R-Precision metric [118] from the text-to-image generation literature to measure how well rendered images align with the true caption. In the context of text-to-image synthesis, R-Precision measures the fraction of generated images that a retrieval model associates with the caption used to generate it. We use a different CLIP model for learning the Dream Field and computing the evaluation metric. As with NeRF evaluation, the image is rendered from a held-out pose. Dream Fields are optimized with cameras at a 30° angle of elevation and evaluated at 45° elevation. For quantitative metrics, we render at resolution 168^2 during training as in [65]. For figures, we train with a 50% higher resolution of 252^2.

We collect an object-centric caption dataset with 153 captions as a subset of the Common Objects in Context (COCO) dataset [88] (see supplement for details). Object centric examples are those that have a single bounding box annotation and are filtered to exclude those captioned with certain phrases like “extreme close up". COCO includes 5 captions per image, but only one is used for generation. Hyperparameters were manually tuned for perceptual quality on a set of 20-74 distinct captions from the evaluation set, and are shared across all other scenes. Additional dataset details and hyperparameters are included in the supplement.

4.5.2 Analyzing retrieval metrics

In the absence of 3D training data, Dream Fields use geometric priors to constrain generation. To evaluate each proposed technique, we start from a simplified baseline Neural Radiance Field largely following [100] and introduce the priors one-by-one. We generate two objects per COCO caption using different seeds, for a total of 306 objects. Objects are synthesized with 10K iterations of CLIP ViT B/16 guided optimization of 168×168 rendered images, bilinearly upsampled to the contrastive model's input resolution for computational efficiency. R-Precision is computed with CLIP ViT B/32 [130] and LiT_{uu} B/32 [191] to measure the alignment of generations with the source caption.

Table 4.1 reports results. **The most significant improvements come from sparsity, scene bounds and architecture.** As an oracle, the ground truth images associated with object-centric COCO captions have high R-Precision. The NeRF representation converges poorly and introduces aliasing and banding artifacts, in part from its use of axis-aligned positional encodings.

We instead combine mip-NeRF's integrated positional encodings with random Fourier features, which improves qualitative results and removes a bias toward axis-aligned structures. However, the effect on precision is neutral or negative. The transmittance loss $\mathcal{L}_T$ in combination with background augmentations significantly improves retrieval precision +18% and +15.6%, while the transmittance loss is not sufficient on its own. This is qualitatively shown in Figure 4.5. Our MLP architecture with residual connections, normalization, bottleneck-style feature dimensions and smooth nonlinearities further improves the R-Precision +8% and +2%. Bounding the scene to a cube improves retrieval +13% and +11%. The additional bounds explicitly mask density σ and concentrate samples along each ray.

We also scale up Dream Fields by optimizing with an image-text model trained on a larger captioned dataset of 3.6B images from [191]. We use a ViT B/32 model with image and text encoders trained from scratch. This corresponds to the uu configuration from [191], following the CLIP training procedure to learn both encoders contrastively. The LiT_{uu} ViT encoder used in our experiments takes higher resolution 288^2 images while CLIP is trained with 224^2 inputs. Still, LiT_{uu} B/32 is more compute-efficient than CLIP B/16 due to the larger patch size in the first layer.

LiT_{uu} does not significantly help R-Precision when optimizing Dream Fields with low resolution renderings, perhaps because the CLIP B/32 model used for evaluation is trained on the same dataset as the CLIP B/16 model in earlier rows. Optimizing for longer with higher resolution 252^2 renderings closes the gap. LiT_{uu} improves visual quality and sharpness (Appendix 10.1), suggesting that improvements in multimodal image-text models transfer to 3D generation.

	Method	R-Precision ↑	
		CLIP B/32	LiT_{uu} B/32
Baseline	COCO GT images	77.1±3.4	75.2±3.5
	Simplified NeRF	31.4±2.7	10.8±1.8
Positional encoding	+ mip-NeRF IPE	29.7±2.6	12.4±1.9
	+ Higher freq. Fourier features	24.2±2.5	10.5±1.8
Sparsity, augmentation	+ random crops	25.8±2.5	10.5±1.8
	+ transmittance loss	23.7±2.4	7.6±1.5
	+ background aug.	44.1±2.8	26.1±2.5
Scene parameterization	+ MLP architecture	52.0±2.9	27.8±2.6
	+ scene bounds	65.4±2.7	**38.9±2.8**
	+ track origin	59.8±2.8	34.6±2.7
Scaling	+ LiT_{uu} ViT B/32	59.5±2.8	–
	+ 20K iterations, 252^2 renders	**68.3±2.7**	–

Table 4.1: **When used together, geometric priors improve caption retrieval precision.** We start with a simplified version of the NeRF scene representation and add in one prior at a time until all are used in conjunction. Captions are retrieved from rendered images of the generated objects at held-out camera poses using CLIP's ViT B/32. Objects are generated with $\mathcal{L}_{\text{CLIP}}$ guidance from the pre-trained CLIP ViT B/16 except in scaling experiments where we experiment with the higher-resolution LiT_{uu} B/32 model.

Figure 4.6: Compositional object generation. Dream Fields allow users to express specific artistic styles via detailed captions. **Top two rows:** Similar to text-to-image experiments in [134], we generate objects with the caption *"armchair in the shape of an avocado. armchair imitating avocado."* **Bottom:** Generations vary the texture of a single snail. Captions follow the template *"a snail made of baguette. a snail with the texture of baguette"* Results are not cherry-picked.

4.5.3 Compositional generation

In Figure 4.6, we show non-cherrypicked generations that test the compositional generalization of Dream Fields to fine-grained variations in captions taken from the website of [134]. We independently vary the object generated and stylistic descriptors like shape and materials. DALL-E [134] also had a remarkable ability to combine concepts in prompts out of distribution, but was limited to 2D image synthesis. Dream Fields produces compositions of concepts in 3D, and supports fine-grained variations in prompts across several categories of objects. Some geometric details are not realistic, however. For example, generated snails have eye stalks attached to their shell rather than body, and the generated green vase is blurry.

4.5.4 Model ablations

Ablating sparsity regularizers While we regularize the mean transmittance, other sparsity losses are possible. We compare unregularized Dream Fields, perturbations to the density σ [100], regularization with a beta prior on transmittance [92], multiplicative gating versions of $\mathcal{L}_T$ and our additive $\mathcal{L}_T$ regularizer in Figure 4.5. On real-world scenes, NeRF added Gaussian noise to network predictions of the density prior to rectification as a regularizer. This can encourage sharper boundary definitions as small densities will often be zeroed by the perturbation. The

Method	Loss or parameterization	R-Prec.
No regularizer	$\mathcal{L}_{\text{CLIP}}$ (4.7)	35.3
Perturb σ [100]	$\sigma = \text{softplus}(f_\theta(\mathbf{x}) + \epsilon)$	47.7
Beta prior [92]	(4.10)	50.3
Gated T [103]	$-\text{mean}(T(\theta, \mathbf{p})) \cdot \mathcal{L}_{\text{CLIP}}$	34.6
Clipped gated T	$-\mathcal{L}_T \cdot \mathcal{L}_{\text{CLIP}}$ (4.11)	**62.1**
Clipped additive T	$\mathcal{L}_{\text{CLIP}} + \lambda \mathcal{L}_T$ (4.9)	**62.1**

Table 4.2: Ablating sparsity regularizers. Optimization is done for 10K iterations at 168^2 resolution with LiT_{uu} ViT B/32 and background augmentation, and retrieval uses CLIP ViT B/32. For the purposes of ablation, we run one seed per caption (153 runs).

beta prior from Neural Volumes [92] encourages rays to either pass through the volume or be completely occluded:

$$\mathcal{L}^{\text{beta}}_{\text{total}} = \mathcal{L}_{\text{CLIP}} + \lambda \cdot \text{mean}(\log T(\theta, \mathbf{p}) + \log(1 - T(\theta, \mathbf{p}))) \tag{4.10}$$

The multiplicative loss is inspired by the opacity scaling of [103] for feature visualization. We scale the CLIP loss by a clipped mean transmittance:

$$\mathcal{L}_{\text{total}} = \min(\tau,\ \text{mean}(T(\theta, \mathbf{p}))) \cdot \mathcal{L}_{\text{CLIP}} \tag{4.11}$$

Table 4.2 compares the regularizers, showing that density perturbations and the beta prior improve R-Precision +12.4% and +15%, respectively. Scenes with clipped mean transmittance regularization best align with their captions, +26.8% over the baseline. The beta prior can fill scenes with opaque material even without background augmentations as it encourages both high and low transmittance. Multiplicative gating works well when clipped to a target and with background augmentations, but is also non-convex and sensitive to hyperparameters. Figure 4.7 shows the effect of varying the target transmittance τ with an additive loss.

Varying the image-text model We compare different image and text representations $h(\cdot), g(\cdot)$ used in $\mathcal{L}_{\text{CLIP}}$ (4.7) and for retrieval metrics. Table 4.3 shows the results. CLIP B/32, B/16 and LiT_{uu} B/32 all have high retrieval precision, indicating they can synthesize objects generally aligned with the provided captions. CLIP B/32 performs the best, outperforming the more compute intensive CLIP B/16 model. The architectures differ in the number of pixels encoded in each token supplied to the Transformer backbone, *i.e.* the ViT patch size. A larger patch size may be sufficient due

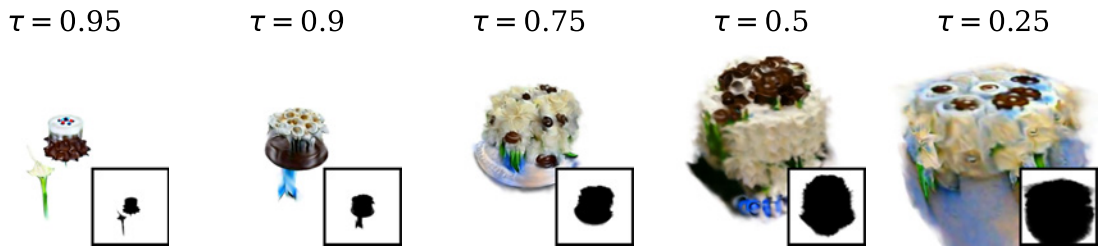


Figure 4.7: The target transmittance τ affects the size of generated objects. Inset panels depict transmittance. **Prompt from Object Centric COCO:** *"A cake toped* [sic] *with white frosting flowers with chocolate centers."*

	Retrieval model R-Precision		
Optimized model	CLIP B/32	CLIP B/16	LiT_{uu} B/32
COCO GT	77.1±3.4	79.1±3.3	75.2±3.5
CLIP B/32 [130]	*(86.6±2.0)*	74.2±2.5	42.8±2.8
CLIP B/16 [130]	59.8±2.8	*(93.5±1.4)*	35.6±2.7
LiT_{uu} B/32	59.5±2.8	66.7±2.7	*(88.9±1.8)*

Table 4.3: The aligned image-text representation used to optimize Dream Fields influences their quantitative validation R-Precision according to a held-out retrieval model. All contrastive models produce high retrieval precision, though qualitatively CLIP B/32 produced overly smooth and simplified objects. We optimize for 10K iterations at 168^2 resolution. *(Italicized)* metrics use the optimized model at a held-out pose and indicate Dream Fields overfit.

to the low resolution of renders: 168^2 cropped to 154^2, then upsampled to CLIP's input size of 224^2. Qualitatively, training with LiT_{uu} B/32 produced the most detailed geometry and textures, suggesting that open-set evaluation is challenging.

Varying optimized camera poses Each training iteration, Dream Fields samples a camera pose $\mathbf{p}$ to render the scene. In experiments, we used a full 360° sampling range for the camera's azimuth, and fixed the elevation. Figure 4.8 shows multiple views of a bird when optimizing with smaller azimuth ranges. In the left-most column, a view from the central azimuth (frontal) is shown, and is realistic for all training configurations. Views from more extreme angles (right, left, rear view columns) have artifacts when the Dream Field is optimized with narrow azimuth ranges. Training with diverse cameras is important for viewpoint generalization.

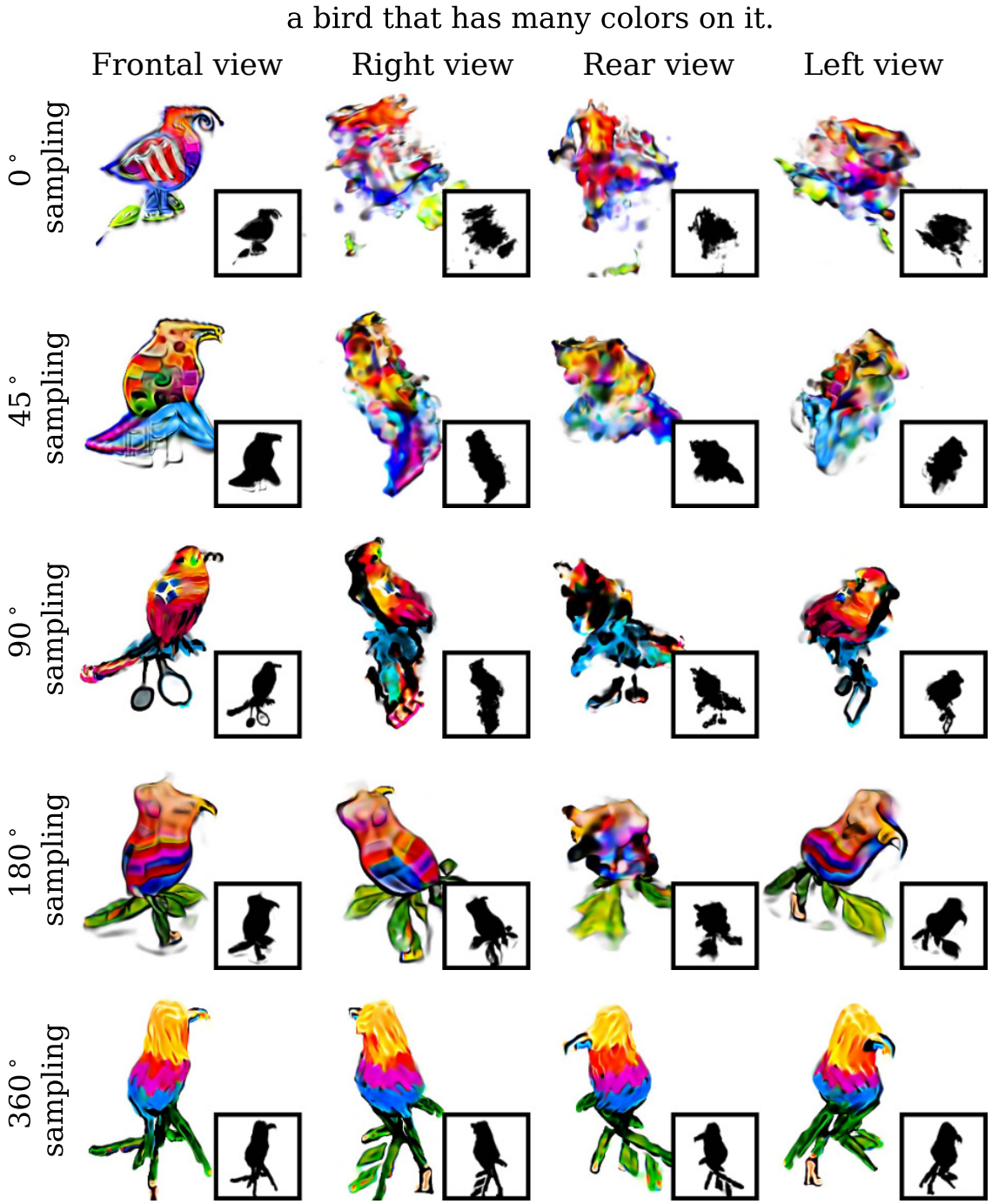


Figure 4.8: Training with diversely sampled camera poses improves generalization across views. In the top row, we sample camera azimuth from a single viewpoint. The rendered view from the same perspective (left column) is realistic, but the object structure is poor as seen from other angles. Qualitative results improve with larger sampling intervals, with the best results from 360° sampling.

4.6 Limitations

There are a number of limitations in Dream Fields. Generation requires iterative optimization, which can be expensive. 2K-20K iterations are sufficient for most objects, but more detail emerges when optimizing longer. Meta-learning [168] or amortization [119] could speed up synthesis.

We use the same prompt at all perspectives. This can lead to repeated patterns on multiple sides of an object. The target caption could be varied across different camera poses. Many of the prompts we tested involve multiple subjects, but we do not target complex scene generation [15, 17, 25, 28] partly because CLIP poorly encodes spatial relations [91, 165]. Scene layout could be handled in a post-processing step.

The image-text models we use to score renderings are not perfect even on ground truth training images, so improvements in image-text models may transfer to 3D generation. Our reliance on pre-trained models inherits their harmful biases. Identifying methods that can detect and remove these biases is an important direction if these methods are to be useful for larger-scale asset generation.

4.7 Discussion

Our work has begun to tackle the difficult problem of object generation from text. By combining scalable multi-modal image-text models and multi-view consistent differentiable neural rendering with simple object priors, we are able to synthesize both geometry and color of 3D objects across a large variety of real-world text prompts. The language interface allows users to control the style and shape of the results, including materials and categories of objects, with easy-to-author prompts. We hope these methods will enable rapid asset creation for artists and multimedia applications.

Acknowledgements

We thank Xiaohua Zhai, Lucas Beyer and Andreas Steiner for providing pre-trained models on the LiT 3.6B dataset, Paras Jain, Kevin Murphy, Matthew Tancik and Alireza Fathi for useful discussions on our work, and many colleagues at Google for building and supporting key infrastructure. Ajay Jain was supported in part by the NSF GRFP under Grant Number DGE 1752814.

Chapter 5

Learning more expressive priors and transferring them to new modalities

As an interlude to contexualize Chapter 6, this chapter briefly summarizes our work on two projects. The first, described in §5.0.1, builds a highly flexible, stable and scalable Denoising Diffusion Probabilistic Model. Then, we propose a novel approach to transfer the knowledge in these models in §5.0.2, achieving state-of-the-art results in text-to-3D generation.

5.0.1 Denoising Diffusion Probabilistic Models

Autoregressive models like the PixelCNNs discussed in Chapter 2 are expressive, previously achieving state-of-the-art likelihoods on image datasets, and stable, admitting large scale training without many tricks. However, their sample quality is relatively poor. As autoregressive models sample data one dimension at a time, they can be quite slow in high dimensions such as high-resolution imagery. An underscaled model also suffers estimation error, which builds up over the course of sampling and can lead to poor quality.

Motivated by the limitations of autoregressive models, we sought to find another class of models that permitted parallel sampling across dimensions, yet retained the stability and scalability benefits of likelihood-based models. Variational approaches are appealing for this task, though modern forms typically estimate an amortized posterior jointly with a decoder and can be expensive or somewhat unstable to train

Instead, we study diffusion probabilistic models, a family of generative models based on a parametric Markov chain, another class of latent variable models inspired by considerations from nonequilibrium thermodynamics. While the Markov chain requires iterative sampling, the number

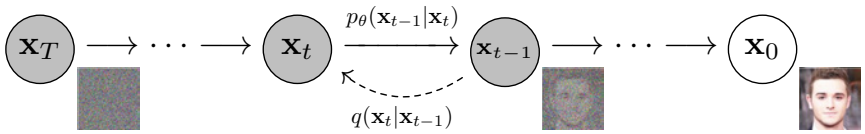


Figure 5.1: The directed graphical model considered by diffusion models.

of steps are a modeling choice independent of the number of dimensions in the data. We show that diffusion probabilistic models generalize autoregressive models but can sample multiple data dimensions in parallel, giving them more flexibility. This allows diffusion models to scale up to high resolution images, with faster sampling than a corresponding autoregressive model.

We propose a variant called the Denoising Diffusion Probabilistic Model (DDPM). 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. We also develop a network architecture that significantly improves sample quality. 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 paper, Denoising Diffusion Probabilistic Models [57] is available at `https://arxiv.org/abs/2006.11239`, and was a collaboration with Jonathan Ho and Pieter Abbeel.

5.0.2 DreamFusion: Text-to-3D using 2D Diffusion

Since the publication of DDPM, diffusion models have shown impressive results in text-to-image synthesis. Using massive datasets of captioned images, diffusion models learn to generate raster images of highly diverse objects and scenes. Recent breakthroughs in text-to-image synthesis have been driven by scaling up denoising diffusion models to billions of image-text pairs. Other subsequent works trained denoising diffusion probabilistic models directly on 3D data, such as point clouds.

However, the same degree of scaling for text-to-3D synthesis with direct training of DDPMs would require large-scale datasets of labeled 3D assets. It also needs new work to develop efficient architectures for denoising 3D data, neither of which existed. In DreamFusion, we circumvent data and architecture limitations by using a pretrained 2D text-to-image diffusion model to perform text-to-3D synthesis.

Figure 5.2: Comparison of 2D sampling methods from a text-to-image diffusion model with text "*a photo of a tree frog wearing a sweater*." For score distillation sampling, as an example we use an image generator that restricts images to be symmetric by having $\mathbf{x} = (\text{flip}(\theta), \theta)$.

We introduce a loss based on probability density distillation that enables the use of a 2D diffusion model as a prior for optimization of a parametric image generator. We refer to the loss as *Score Distillation Sampling*, due to the connection between DDPM and denoising score matching [177], and our derivation based on distillation. Score distillation offers an alternative way to sample diffusion models. While DDPM was sampled with an ancestral sampler defined by the Markov chain, score distillation sampling is based on a gradient-based optimization procedure. We illustrate the difference between the two sampling methods in Figure 5.2.

Similar to DietNeRF and Dream Fields, gradient-based optimization allows the sample to be in a hidden space connected to the observation via a differentiable rendering procedure, with an unknown inverse mapping. Score distillation also permits constraints and regularizers.

Using this loss in a DepDream-like procedure, we optimize a randomly-initialized 3D model (a Neural Radiance Field, or NeRF) via gradient descent such that its 2D renderings from random angles achieve a low loss. The resulting 3D model of the given text can be viewed from any angle, relit by arbitrary illumination, or composited into any 3D environment. Our approach requires no 3D training data and no modifications to the image diffusion model, demonstrating the effectiveness of pretrained image diffusion models as priors.

See `https://dreamfusion3d.github.io` for our paper [127] and qualitative results. This work was a collaboration with Ben Poole, Jonathan T. Barron and Ben Mildenhall.

Chapter 6

VectorFusion: Text-to-SVG by Abstracting Image Diffusion Models

Work by Ajay Jain, Amber Xie and Pieter Abbeel

Chapter 5 earmarked a transition in the quality and generality of visual synthesis methods. DDPM catalyzed significant progress in text-guided synthesis across data rich modalities. The Score Distillation Sampling loss allows scalable diffusion priors to be transfered to a new modality, 3D radiance fields, in contrast to our past work, DietNeRF and Dream Fields that transfer discriminative models.

In this chapter, we continue to explore the limits of transfer learning of diffusion models for text-guided synthesis. We focus on vector graphics, a compact and abstract modality that is difficult to model. Designers frequently use vector representations of images like Scalable Vector Graphics (SVGs) for digital icons or art. Vector graphics can be scaled to any size, and are compact.

We show that a text-conditioned diffusion model trained on raster representations of images can be used to generate SVG-exportable vector graphics. We do so without access to large datasets of captioned SVGs. By optimizing a differentiable vector graphics rasterizer, our method, VectorFusion, distills abstract semantic knowledge out of a pretrained diffusion model. We learn an SVG consistent with a caption using Score Distillation Sampling. To accelerate generation and improve fidelity, VectorFusion also initializes from an image sample. Experiments show greater quality than prior work, and demonstrate a range of styles including pixel art and sketches.

Figure 6.1: Text-to-SVG with VectorFusion. When (a) raster graphics sampled from Stable Diffusion are (b) auto-traced, they lose details that are hard to represent within the constraints of the abstraction. (c-d) VectorFusion improves fidelity and consistency with the caption by directly optimizing paths with a distillation-based diffusion loss. Find videos and more results at https://ajayj.com/vectorfusion.

6.1 Introduction

Graphic designers and artists often express concepts in an abstract manner, such as composing a few shapes and lines into a pattern that evokes the essence of a scene. Scalable Vector Graphics (SVGs) provide a declarative format for expressing visual concepts as a collection of primitives. Primitives include Bézier curves, polygons, circles, lines and background colors. SVGs are the defacto format for exporting graphic designs since they can be rendered at arbitrarily high resolution on user devices, yet are stored and transmitted with a compact size, often only tens of kilobytes. Still, designing vector graphics is difficult, requiring knowledge of professional design tools.

Recently, large captioned datasets and breakthroughs in diffusion models have led to systems capable of generating diverse images from text including DALL-E 2 [133], Imagen [145] and Latent Diffusion [141]. However, the vast majority of images available in web-scale datasets are rasterized, expressed at a finite resolution with no decomposition into primitive parts nor layers. For this reason, existing diffusion models can only generate raster images. In theory, diffusion models could be trained to directly model SVGs, but would need specialized architectures for variable-length hierarchical sequences, and significant data collection work.

Figure 6.2: Given a caption, VectorFusion generates abstract vector graphics in an SVG format. We use a pre-trained diffusion model trained only on rasterized images to guide a differentiable vector renderer. VectorFusion supports diverse objects and styles. To select a style such as flat polygonal vector icons, abstract line drawings or pixel art, we constrain the vector representation to subset of possible primitive shapes and use different prompt modifiers to encourage an appropriate style: * ...minimal flat 2d vector icon. lineal color. on a white background. trending on artstation, ** ...pixel art. trending on artstation, †...minimal 2d line drawing. trending on artstation. Please see videos of the optimization process on our project webpage.

How can we use diffusion models pretrained on pixels to generate high-quality vector graphics? In this work, we provide a method for generating high quality abstract vector graphics from text captions, shown in Fig. 6.1.

We start by evaluating a two phase text-to-image and image-to-vector baseline: generating a raster image with a pretrained diffusion model, then vectorizing it. Traditionally, designers manually convert simple rasterized images into a vector format by tracing shapes. Some ML-based tools [95] can automatically approximate a raster image with an SVG. Unfortunately, we find that text-to-image diffusion models frequently produce complex images that are hard to represent with simple vectors, or are incoherent with the caption (Fig 6.1, Stable Diffusion + LIVE).

To improve quality of the SVG and coherence with the caption, we incorporate the pretrained text-to-image diffusion model in an optimization loop. Our approach, VectorFusion, combines a differentiable vector graphics renderer [87] and the *score distillation sampling* (SDS) loss that we proposed in prior work [127] to iteratively refine shape parameters. Intuitively, score distillation converts diffusion sampling into an optimization problem that allows the image to be represented by an arbitrary differentiable function. In our case, the differentiable function is the forward rasterization process, and the diffusion model provides a signal for improving the raster. To adapt SDS to text-to-SVG synthesis, we make the following contributions:

- We extend score distillation sampling to open source latent space diffusion models,
- improve efficiency and quality by initializing near a raster image sample,
- propose SVG-specific regularization including path reinitialization,
- and evaluate different sets of shape primitives and their impact on style.

In experiments, VectorFusion generates iconography, pixel art and line drawings from diverse captions. VectorFusion also achieves greater quality than CLIP-based approaches that transfer a discriminative vision-language representation.

6.2 Related Work

A few works have used pretrained vision-language models to guide vector graphic generation. VectorAscent [63] and CLIPDraw [38] optimize CLIP's image-text similarity metric [130] to generate vector graphics from text prompts, with a procedure similar to DeepDream [103] and CLIP feature visualization [42]. StyleCLIPDraw [151] extends CLIPDraw to condition on images with an auxiliary style loss with a pretrained VGG16 [154] model. Arnheim [36] parameterizes SVG paths with a neural network, and CLIP-CLOP [101] uses an evolutionary approach to create

image collages. Though we also use pretrained vision-language models, we use a generative model rather than a discriminative model.

Recent work has shown the success of text-to-image generation. DALL-E 2 [133] learns an image diffusion model conditioned on CLIP's text embeddings. Our work uses Stable Diffusion [141] (SD), a text-to-image latent diffusion model. While these models produce high-fidelity images, they cannot be directly transformed into vector graphics.

A number of works generate vector graphics from input images. We extend the work of Layer-wise Image Vectorization (LIVE) [95], which iteratively optimizes closed Bézier paths with a differentiable rasterizer, DiffVG [87].

We also take inspiration from inverse graphics with diffusion models. Diffusion models have been used in zero-shot for image-to-image tasks like inpainting [94]. DDPM-PnP [45] uses diffusion models as priors for conditional image generation, segmentation, and more. DreamFusion [127] uses 2D diffusion as an image prior for text-to-3D synthesis with a more efficient loss than DDPM-PnP, discussed in Section 6.3.3. Following [127], we use diffusion models as transferable priors for vector graphics. Concurrent work [97] also adapts the SDS loss for latent-space diffusion models.

6.3 Background

6.3.1 Vector representation and rendering pipeline

Vector graphics are composed of primitives. For our work, we use paths of segments delineated by control points. We configure the control point positions, shape fill color, stroke width and stroke color. Most of our experiments use closed Bézier curves. Different artistic styles are accomplished with other primitives, such as square shapes for pixel-art synthesis and unclosed Bézier curves for line art.

To render to pixel-based formats, we rasterize the primitives. While many primitives would be needed to express a realistic photogaph, even a few can be combined into recognizable, visually pleasing objects. We use DiffVG [87], a differentiable rasterizer that can compute the gradient of the rendered image with respect to the parameters of the SVG paths. Many works, such as LIVE [95], use DiffVG to vectorize images, though such transformations are lossy.

6.3.2 Diffusion models

Chapter 5 introduced our work on Denoising Diffusion Probabilistic Models. To provide more technical details relevant for VectorFusion, diffusion models are a flexible class of likelihood-based generative models that learn a distribution by denoising. A diffusion model generates data by

learning to gradually map samples from a known prior like a Gaussian toward the data distribution. During training, a diffusion model optimizes a variational bound on the likelihood of real data samples [160], similar to a variational autoencoder [82]. This bound reduces to a weighted mixture of denoising objectives [58]:

$$\mathcal{L}_{\text{DDPM}}(\phi, \mathbf{x}) = \mathbb{E}_{t,\epsilon}\left[w(t)\|\epsilon_\phi(\alpha_t \mathbf{x} + \sigma_t \epsilon) - \epsilon\|_2^2\right] \tag{6.1}$$

where $\mathbf{x}$ is a real data sample and $t \in \{1, 2, \dots T\}$ is a uniformly sampled timestep scalar that indexes noise schedules α_t, σ_t [78]. ϵ is noise of the same dimension as the image sampled from the known Gaussian prior. Noise is added by interpolation to preserve variance. ϵ_ϕ is a learned denoising autoencoder that predicts the noise content of its input. For images, ϵ_ϕ is commonly a U-Net [58, 142], and the weighting function $w(t) = 1$ [58]. Denoising diffusion models can be trained to predict any linear combination of $\mathbf{x}$ and ϵ, such as the clean, denoised image $\mathbf{x}$, though an ϵ parameterization is simple and stable.

At test time, a sampler starts with a draw from the prior $\mathbf{x}_T \sim \mathcal{N}(0, 1)$, then iteratively applies the denoiser to update the sample while decaying the noise level t to 0. For example, DDIM [161] samples with the update:

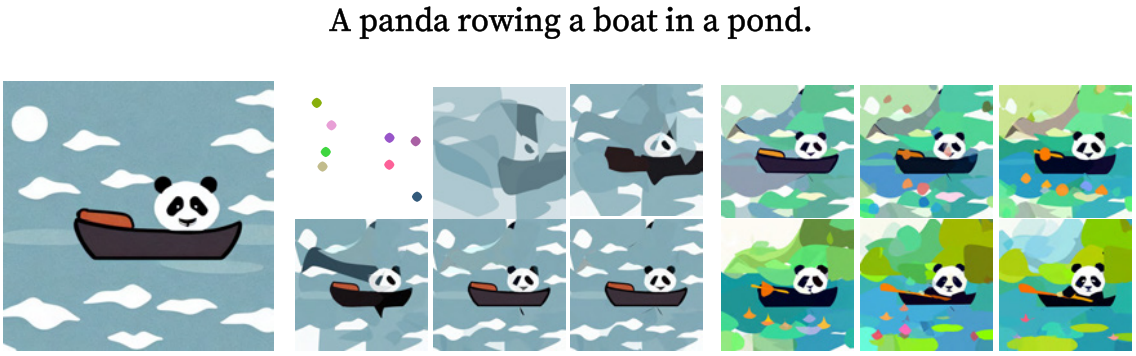
$$\begin{aligned} \hat{\mathbf{x}} &= (\mathbf{x}_t - \sigma_t \epsilon_\phi(\mathbf{x}_t))/\alpha_t, && \text{Predict clean image} \\ \mathbf{x}_{t-1} &= \alpha_{t-1}\hat{\mathbf{x}} + \sigma_{t-1}\epsilon_\phi(\mathbf{x}_t) && \text{Add back noise} \end{aligned} \tag{6.2}$$

For text-to-image generation, the U-Net is conditioned on the caption y, $\epsilon_\phi(\mathbf{x}, y)$, usually via cross-attention layers and pooling of the features of a language model [106]. However, conditional diffusion models can produce results incoherent with the caption since datasets are weakly labeled and likelihood-based models try to explain all possible images. To increase the usage of a label or caption, classifier-free guidance [59] superconditions the model by scaling up conditional model outputs and guiding away from a generic unconditional prior that drops y:

$$\hat{\epsilon}_\phi(\mathbf{x}, y) = (1 + \omega) * \epsilon_\phi(\mathbf{x}, y) - \omega * \epsilon_\phi(\mathbf{x}) \tag{6.3}$$

CFG significantly improves coherence with a caption at the cost of an additional unconditional forward pass per step.

High resolution image synthesis is expensive. Latent diffusion models [141] train on a reduced spatial resolution by compressing 512×512 images into a relatively compact 64×64, 4-channel latent space with a VQGAN-like autoencoder (E, D) [34]. The diffusion model ϵ_ϕ is trained to



(a) Sample raster image with Stable Diffusion

(b) Convert raster image to a vector

(c) VectorFusion: Fine tune by latent score distillation

Figure 6.3: VectorFusion generates SVGs in three stages. **(a)** First, we sample a rasterized image from a text-to-image diffusion model like Stable Diffusion with prompt engineering for iconographic aesthetics. **(b)** Since this image is finite resolution, we approximate it by optimizing randomly initialized vector paths with an L2 loss. The loss is backpropagated through DiffVG, a differentiable vector graphics renderer, to tune path coordinates and color parameters. Paths are added in stages at areas of high loss following [95]. **(c)** However, the diffusion sample often fails to express all the attributes of the caption, or loses detail when vectorized. VectorFusion finetunes the SVG with a latent score distillation sampling loss to improve quality and coherence.

model the latent space, and the decoder D maps back to a high resolution raster image. We use Stable Diffusion, a popular open-source text-to-image model based on latent diffusion.

6.3.3 Score distillation sampling

Diffusion models can be trained on arbitrary signals, but it is easier to train them in a space where data is abundant. Standard diffusion samplers like (6.2) operate in the same space that the diffusion model was trained. While samplers can be modified to solve many image-to-image tasks in zero-shot such as colorization and inpainting [160, 164], until recently, pretrained image diffusion models could only generate rasterized images.

In contrast, image encoders like VGG16 trained on ImageNet and CLIP (Contrastive Language–Image Pre-training) [130] have been transferred to many modalities like mesh texture generation [103], 3D neural fields [64, 65], and vector graphics [38, 63]. Even though encoders are not generative, they can generate data with test time optimization: a loss function in the encoder's feature space is backpropagated to a learned image or function outputting images.

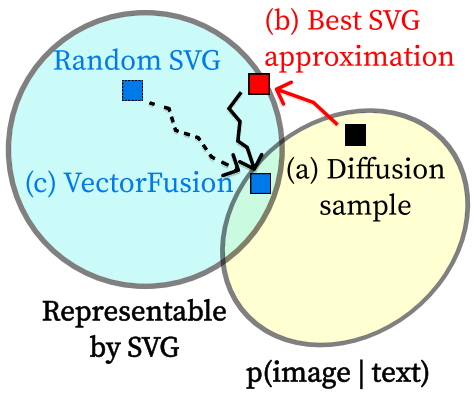


Figure 6.4: Conceptual diagram motivating our approach. While vectorizing a rasterized diffusion sample is lossy, VectorFusion can either finetune the best approximation or optimize a random SVG from scratch to sample an SVG that is consistent with the caption.

DreamFusion [127] proposed an approach to use a pretrained pixel-space text-to-image diffusion model as a loss function. Their proposed Score Distillation Sampling (SDS) loss provides a way to assess the similarity between an image and a caption:

$$\mathcal{L}_{\text{SDS}} = \mathbb{E}_{t,\epsilon}\left[\sigma_t/\alpha_t w(t)\text{KL}(q(\mathbf{x}_t|g(\theta); y, t)\|p_\phi(\mathbf{x}_t; y, t))\right].$$

p_ϕ is the distribution learned by the frozen, pretrained diffusion model. q is a unimodal Gaussian distribution centered at a learned mean image $g(\theta)$. In this manner, SDS turned sampling into an optimization problem: an image or a differentiable image parameterization (DIP) [103] can be optimized with $\mathcal{L}_{\text{SDS}}$ to bring it toward the conditional distribution of the teacher. This is inspired by probability density distillation [116]. Critically, SDS only needs access to a pixel-space prior p_ϕ, parameterized with the denoising autoencoder $\hat{\epsilon}_\phi$. It does not require access to a prior over the parameter space θ. DreamFusion [127] used SDS with the Imagen pixel space diffusion model to learn the parameters of a 3D Neural Radiance Field [100]. In practice, SDS gives access to loss gradients, not a scalar loss:

$$\nabla_\theta \mathcal{L}_{\text{SDS}} = \mathbb{E}_{t,\epsilon}\left[w(t)\left(\hat{\epsilon}_\phi(\mathbf{x}_t; y, t) - \epsilon\right)\frac{\partial \mathbf{x}}{\partial \theta}\right] \tag{6.4}$$

6.4 Method: VectorFusion

In this section, we outline two methods for generating abstract vector representations from pretrained text-to-image diffusion models, including our full VectorFusion approach.

6.4.1 A baseline: text-to-image-to-vector

We start by developing a two stage pipeline: sampling an image from Stable Diffusion, then vectorizing it automatically. Given text, we sample a raster image from Stable Diffusion with a Runge-Kutta solver [90] in 50 sampling steps with guidance scale $\omega = 7.5$ (the default settings in the Diffusers library [126]). Naively, the diffusion model generates photographic styles and details that are very difficult to express with a few constant color SVG paths. To encourage image generations with an abstract, flat vector style, we append a suffix to the text: *"minimal flat 2d vector icon. lineal color. on a white background. trending on artstation"*. This prompt was tuned qualitatively.

Because samples can be inconsistent with captions, we sample K images and select the Stable Diffusion sample that is most consistent with the caption according to CLIP ViT-B/16 [130]. CLIP reranking was originally proposed by [134]. We choose K=4.

Next, we automatically trace the raster sample to convert it to an SVG using the off-the-shelf Layer-wise Image Vectorization program (LIVE) [95]. LIVE produces relatively clean SVGs by initializing paths in stages, localized to poorly recontructed, high loss regions. To encourage paths to explain only a single feature of the image, LIVE weights an L2 reconstruction loss by distance to the nearest path,

$$\mathcal{L}_{\text{UDF}} = \frac{1}{3} \sum_{i=1}^{w \times h} d_i' \sum_{c=1}^{3} (I_{i,c} - \hat{I}_{i,c})^2 \tag{6.5}$$

where I is the target image, $\hat{I}$ is the rendering, c indexes RGB channels in I, d_i' is the unsigned distance between pixel i, and the nearest path boundary, and w, h are width and height of the image. LIVE also optimizes a self-intersection regularizer $\mathcal{L}_{\text{Xing}}$

$$\mathcal{L}_{\text{Xing}} = D_1(\text{ReLU}(-D_2)) + (1 - D_1)(\text{ReLU}(D_2)), \tag{6.6}$$

where D_1 is the characteristic of the angle between two segments of a cubic Bézier path, and D_2 is the value of $\sin(\alpha)$ of that angle. For further clarifications of notation, please refer to LIVE [95].

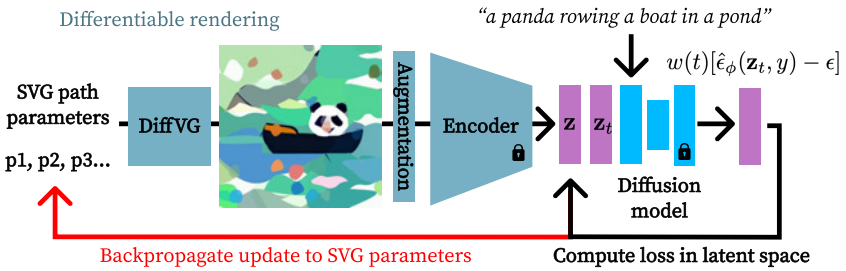


Figure 6.5: An overview of VectorFusion's latent score distillation optimization procedure. We adapt Score Distillation Sampling [127] to support a vector graphics renderer and a latent-space diffusion prior for raster images. First, we rasterize the SVG given path parameters. We apply data augmentations, encode into a latent space, compute the Score Distillation loss on the latents, and backpropagate through the encoding, augmentation and renderering procedure to update paths.

This results in a set of paths $\theta_{\text{LIVE}} = \{p_1, p_2, \dots p_k\}$. Figure 6.3(b) shows the process of optimizing vector parameters in stages that add 8-16 paths at a time. Figure 6.1 shows more automatic conversions. While simple, this pipeline often creates images unsuitable for vectorization.

6.4.2 Sampling vector graphics by optimization

The pipeline in 6.4.1 is flawed since samples may not be easily representable by a set of paths. Figure 6.4 illustrates the problem. Conditioned on text, a diffusion model produces samples from the distribution $p_\phi(\mathbf{x}|y)$. Vectorization with LIVE finds a SVG with a close L2 approximation to that image without using the caption y. This can lose information, and the resulting SVG graphic may no longer be coherent with the caption.

For VectorFusion, we adapt Score Distillation Sampling to support latent diffusion models (LDM) like the open source Stable Diffusion. We initialize an SVG with a set of paths $\theta = \{p_1, p_2, \dots p_k\}$. Every iteration, DiffVG renders a 600 × 600 image $\mathbf{x}$. Like CLIPDraw [38], we augment with perspective transform and random crop to get a 512 × 512 image $\mathbf{x}_{\text{aug}}$. Then, we propose to compute the SDS loss in latent space using the LDM encoder E_ϕ, predicting $\mathbf{z} = E_\phi(\mathbf{x}_{\text{aug}})$. For each iteration of optimization, we diffuse the latents with random noise $\mathbf{z}_t = \alpha_t \mathbf{z} + \sigma_t \epsilon$, denoise with the teacher model $\hat{\epsilon}_\phi(\mathbf{z}_t, y)$, and optimize the SDS loss using a latent-space modification of Equation 6.4:

$$\nabla_\theta \mathcal{L}_{\text{LSDS}} = \mathbb{E}_{t,\epsilon}\left[w(t)\left(\hat{\epsilon}_\phi(\alpha_t \mathbf{z}_t + \sigma_t \epsilon, y) - \epsilon\right) \frac{\partial \mathbf{z}}{\partial \mathbf{x}_{\text{aug}}} \frac{\partial \mathbf{x}_{\text{aug}}}{\partial \theta} \right] \tag{6.7}$$

Figure 6.6: The number of Bézier paths controls the level of detail in generated vector graphics.

Since Stable Diffusion is a discrete time model with $T = 1000$ timesteps, we sample $t \sim \mathcal{U}(50, 950)$ For efficiency, we run the diffusion model $\hat{\epsilon}_\theta$ in half-precision. We found it important to compute the Jacobian of the encoder $\partial \mathbf{z}/\partial \mathbf{x}_{\text{aug}}$ in full FP32 precision for numerical stability. The term $\partial \mathbf{x}_{\text{aug}}/\partial \theta$ is computed with autodifferentiation through the augmentations and differentiable vector graphics rasterizer, DiffVG. $\mathcal{L}_{\text{LSDS}}$ can be seen as an adaptation of $\mathcal{L}_{\text{SDS}}$ where the rasterizer, data augmentation and frozen LDM encoder are treated as a single image generator with optimizable parameters θ for the paths. During optimization, we also regularize self-intersections with (6.6).

6.4.3 Reinitializing paths

In our most flexible setting, synthesizing flat iconographic vectors, we allow path control points, fill colors and SVG background color to be optimized. During the course of optimization, many paths learn low opacity or shrink to a small area and are unused. To encourage usage of paths and therefore more diverse and detailed images, we periodically reinitialize paths with fill-color opacity or area below a threshold. Reinitialized paths are removed from optimization and the SVG, and recreated as a randomly located and colored circle on top of existing paths.

6.4.4 Stylizing by constraining vector representation

Users can control the style of art generated by VectorFusion by modifying the input text, or by constraining the set of primitives and parameters that can be optimized. We explore three settings: iconographic vector art with flat shapes, pixel art, and sketch-based line drawings.

Iconography We use closed Bézier paths with trainable control points and fill colors. Our final vectors have 64 paths, each with 4 segments. For VectorFusion from scratch, we initialize 64 paths randomly and simultaneously, while for SD + LIVE + SDS, we initialize them iteratively during the LIVE autovectorization phase. We include details about initialization parameters in the supplement. Figure 6.6 qualitatively compares generations using 16, 64 and 128 paths (SD + LIVE initialization with K=20 rejection samples and SDS finetuning). Using fewer paths leads to simpler, flatter icons, whereas details and more complex highlights appear with greater numbers of paths.

Pixel art Pixel art is a popular video-game inspired style, frequently used for character and background art. While an image sample can be converted to pixel art by downsampling, this results in blurry, bland, and unrecognizable images. Thus, pixel art tries to maximize use of the available shapes to clearly convey a concept. Pixray [181] uses square SVG polygons to represent pixels and uses a CLIP-based loss following [38, 63]. VectorFusion able to generate meaningful and aesthetic pixel art from scratch and with a Stable Diffusion initialization, shown in Fig. 6.2 and Fig. 6.7. In addition to the SDS loss, we additionally penalize an L2 loss on the image scaled between -1 and 1 to combat oversaturation, detailed in the supplement. We use 32×32 pixel grids.

Sketches Line drawings are perhaps the most abstract representation of visual concepts. Line drawings such as Pablo Picasso's animal sketches are immediately recognizable, but bear little to no pixel-wise similarity to real subjects. Thus, it has been a long-standing question whether learning systems can generate semantic sketch abstractions, or if they are fixated on low-level textures. Past work includes directly training a model to output strokes like Sketch-RNN [50], or optimizing sketches to match a reference image in CLIP feature space [178]. As a highly constrained representation, we optimize only the control point coordinates of a set of fixed width, solid black Bézier curves. We use 16 strokes, each 6 pixels wide with 5 segments, randomly initialized and trained from scratch, since the diffusion model inconsistently generates minimal sketches.

6.5 Experiments

In this section, we quantitatively and qualitatively evaluate the text-to-SVG synthesis capabilities of VectorFusion guided by the following questions. In Section 6.5.2, we ask *(1) Are SVGs generated by VectorFusion consistent with representative input captions?* and *(2) Does our diffusion optimization-based approach help compared to simpler baselines?* In Section 6.5.3, we qualitatively compare VectorFusion's diffusion-based results with past CLIP-based methods. Section 6.5.4 and 6.5.5 describe pixel and sketch art generations. Overall, VectorFusion performs competitively on

Table 6.1: Evaluating the consistency of text-to-SVG generations using 64 primitives with input captions. Consistency is measured with CLIP R-Precision and CLIP similarity score (×100). Higher is better. We compare a CLIP-based approach, CLIPDraw, with diffusion baselines: the best of K raster samples from Stable Diffusion (SD), converting the best of K samples to vectors with LIVE [95], and VectorFusion from scratch or initialized with the LIVE converted SVG. VectorFusion generations are significantly more consistent with captions than Stable Diffusion samples and their automatic vector conversions. CLIPDraw is trained to maximize CLIP score, so it has artificially high scores.

		Caption consistency			
		CLIP L/14		OpenCLIP H/14	
Method	K	R-Prec	Sim	R-Prec	Sim
CLIPDraw (scratch)	–	**85.2**	27.2	77.3	**31.7**
Stable Diff (**raster**)	1	67.2	23.0	69.5	26.7
+ rejection sampling	4	81.3	24.1	80.5	28.2
SD init + LIVE	1	57.0	21.7	59.4	25.8
+ rejection sampling	4	69.5	22.9	65.6	27.6
VectorFusion (scratch)	–	76.6	24.3	69.5	28.5
+ SD init + LIVE	1	78.1	**29.1**	78.1	29.3
+ rejection sampling	4	81.3	24.5	**78.9**	29.4

quantitative caption consistency metrics, and qualitatively produces the most coherent and visually pleasing vectors.

6.5.1 Experimental setup

It is challenging to evaluate text-to-SVG synthesis, since we do not have target, ground truth SVGs to use as a reference. We collect a diverse dataset of captions and evaluate text-SVG coherence with automated CLIP metrics. Our dataset consists of 128 captions from past work and benchmarks for text-to-SVG and text-to-image generation: prompts from CLIPDraw [38] and ES-CLIP [170], combined with captions from PartiPrompts [190], DrawBench [146], DALL-E [134], and DreamFusion [127]. Like previous works, we calculate CLIP R-Precision and cosine similarity.

CLIP Similarity We calculate the average cosine similarity of CLIP embeddings of generated images and the text captions used to generate them. Any prompt engineering is excluded from the reference text. As CLIP Similarity increases, pairs will generally be more consistent with each other. We note that CLIPDraw methods directly optimize CLIP similarity scores and have impressive metrics, but rendered vector graphics are sketch-like and messy unlike the more

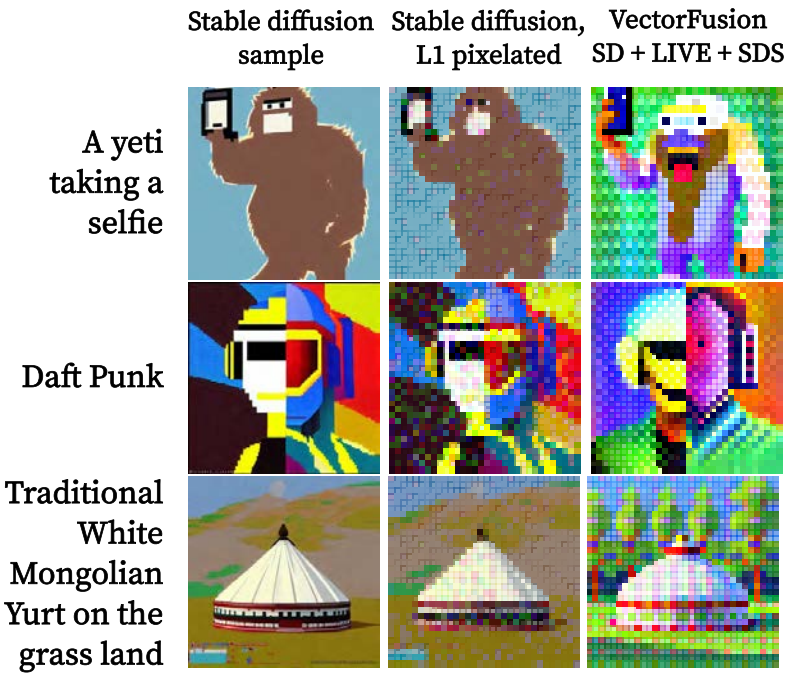


Figure 6.7: **VectorFusion generates coherent, pleasing pixel art.** Stable Diffusion can generate a pixel art style, but has no control over the regularity and resolution of the pixel grid (left). This causes artifacts and blurring when pixelating the sample into a 32x32 grid, even with a robust L1 loss (middle). By finetuning the L1 result, VectorFusion improves quality and generates an abstration that works well despite the low resolution constraint.

cohesive VectorFusion samples. We provide examples in Figure 6.8. To mitigate this effect, we also evaluate the open source Open CLIP ViT-H/14 model, which uses a different dataset for training the representations.

CLIP R-Precision For a more interpretable metric, we also compute CLIP Retrieval Precision. Given our dataset of captions, we calculate CLIP similarity scores for each caption and each rendered image of generated SVGs. R-Precision is the percent of SVGs with maximal CLIP Similarity with the correct input caption, among all 128.

6.5.2 Evaluating caption consistency

As a baseline, we generate an SVG for each caption in our benchmark using CLIPDraw [38] with 64 strokes and their default hyperparameters. We sample 4 raster graphics per prompt from Stable

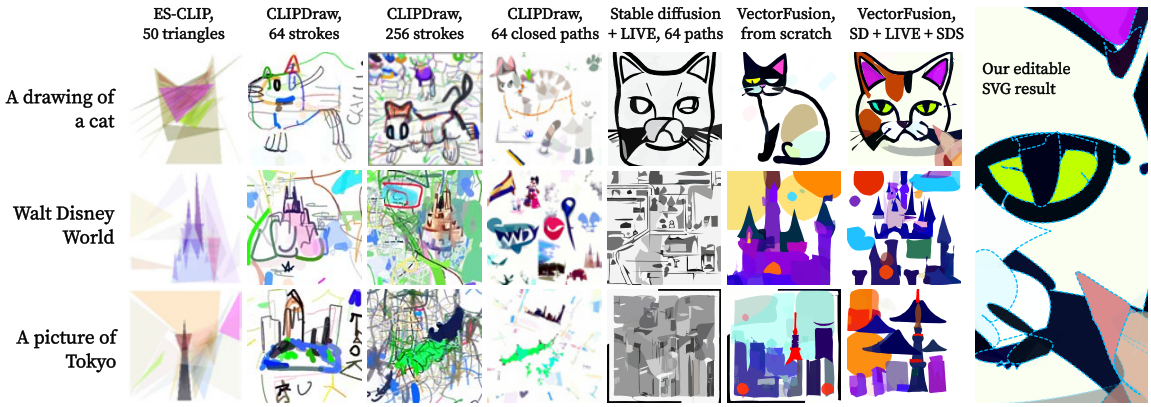


Figure 6.8: VectorFusion produces more coherent vector art than baselines that optimize CLIP score, even with fewer paths (64 shapes). On the right, the resulting SVG can be enlarged to arbitrary scale. Individual paths are highlighted with blue dashed lines. The result can be edited intuitively by the user in design software.

Diffusion as an oracle. These are selected amongst with CLIP reranking (rejection sampling). Stable Diffusion produces rasterized images, not SVGs, but can be evaluated as an oracle with the same metrics. We then autotrace the samples into SVGs using LIVE with 64 strokes, incrementally added in 5 stages of optimization. Finally, we generate with VectorFusion, trained from scratch on 64 random paths per prompt, or initialized with LIVE.

6.5.3 Comparison with CLIP-based approaches

Figure 6.8 qualitatively compares diffusion with CLIP-guided text-to-SVG synthesis. ES-CLIP [170] is an evolutionary search algorithm that searches for triangle abstractions that maximize CLIP score, whereas CLIPDraw uses gradient-based optimization. VectorFusion produces much clearer, cleaner vector graphics than CLIP baselines, because we incorporate a generative prior for image appearance. However, a generative prior is not enough. Optimizing paths with the latent SDS loss $\mathcal{L}_{\text{LSDS}}$ (right two columns) further improves vibrancy and clarity compared to tracing Stable Diffusion samples with LIVE.

6.5.4 Pixel art generation

VectorFusion generates aesthetic and relevant pixel art. Figure 6.2 shows that VectorFusion from scratch can generate striking and coherent samples. Figure 6.7 shows our improvements over

L1-pixelated Stable Diffusion samples, which are pixelated by minimizing an L1 loss with respect to square colors.

6.5.5 Sketches and line drawings

Figure 6.2 includes line drawing samples. VectorFusion produces recognizable and clear sketches from scratch without any image reference, even complex scenes with multiple objects. In addition, it is able to ignore distractor terms irrelevant to sketches, such as *"watercolor"* or *"Brightly colored"* and capture the semantic information of the caption.

6.6 Discussion

We have presented VectorFusion, a novel text-to-vector generative model. Without access to datasets of captioned SVGs, we use pretrained diffusion models to guide generation. The resulting abstract SVG representations can be intuitively used in existing design workflows. Our method shows the effectiveness of distilling generative models compared to using contrastive models like CLIP. In general, we are enthusiastic about the potential of scalable generative models trained in pixel space to transfer to new tasks, with interpretable, editable outputs. VectorFusion provides a reference point for designing such systems.

VectorFusion faces certain limitations. For instance, forward passes through the generative model are more computationally expensive than contrastive approaches due to its increased capacity. VectorFusion is also inherently limited by Stable Diffusion in terms of dataset biases [10] and quality, though we expect that as text-to-image models advance, VectorFusion will likewise continue to improve.

Acknowledgements

We thank Paras Jain, Ben Poole, Aleksander Holynski and Dave Epstein for helpful discussions about this project. VectorFusion relies upon several open source software libraries [51, 87, 95, 122, 126, 139]. The work in this chapter was supported in part by the BAIR Industrial Consortium. We also thank NVIDIA for providing compute resources through the NVIDIA Academic DGX Grant.

Chapter 7

Conclusion

My PhD journey has covered broad ground, from building systems that scale up machine learning, to new machine learning models for improved distribution estimation, to transfer learning algorithms that allow such models to generalize to new modalities. I have also been fortunate to collaborate on a number of other projects during my PhD that have not been included in this document. Instead of covering it all, this thesis focused three bodies of work.

First, in Chapter 2, I introduced Locally Masked Convolutional Networks (LMConv), an efficient technique for training autoregressive image models that support image generation and completion in arbitrary orderings. By sharing parameters across many orderings, LMConv achieved state-of-the-art likelihoods on large image datasets.

Next, in Chapters 3 and 4, I developed methods to transfer knowledge from pretrained 2D image models to challenging 3D tasks with few examples, such as novel view synthesis and text-to-3D generation. By using semantic consistency losses in the feature spaces of models like CLIP, I showed how to guide the learning of continuous volumetric representations to be consistent with natural language prompts. These methods enabled the automatic generation of 3D assets from text descriptions alone.

In Chapter 5, I briefly presented additional work designing stable and expressive generative models for images through Denoising Diffusion Probabilistic Models. I then extended these models to new modalities with the Score Distillation Sampling loss, used by our DreamFusion approach to transfer a 2D text-to-image diffusion model into 3D.

Finally, in Chapter 6, I explored text-to-vector generation by finetuning and distilling a pretrained diffusion model into abstract vector graphics with control over style. VectorFusion achieved compelling results in the challenging text-to-vector space, suggesting the potential for pretrained diffusion models to be used as priors in many settings.

Deep generative models sometimes feel like more of an art than a science. Yes, generative models have rich theoretical backings rooted in statistics and variational inference. Yet, often the qualitative, aesthetic and application questions are more important for their development than the specifics of the underlying learning methodology. The humanistic questions remain the most challenging to answer. How do we define a task? Is it possible to evaluate and compare two models with different styles? What will be the impacts of the work ? Frequently, I found myself surprised—often delighted, and sometimes concerned—about how the community extended our work, and how it was used in practice.

Generative models are also deeply compelling since they enable the creation of new things, especially by people who otherwise may not have been trained to express their ideas in visually rich media. In this regard, generative models are perhaps one of humanities greatest inventions for elevating creativity. Beyond visual modalities, the techniques we discussed in this thesis have found applications in medicine, remote sensing, climate research, simulation, robotics and more domains of great impact. While there are tremendous difficulties ahead in addressing the risks and limitations of generative models, I am excited to see how future work will build upon these contributions to push the boundaries of creation.

Bibliography

1. N. Akoury and A. Nguyen. “Spatial PixelCNN: Generating Images from Patches”. In: *arXiv:1712.00714*, 2017.
2. M. Arjovsky, S. Chintala, and L. Bottou. “Wasserstein Generative Adversarial Networks”. In: *ICML*, 2017.
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. J. T. Barron and J. Malik. “Shape, Illumination, and Reflectance from Shading”. In: *TPAMI*, 2015.
5. J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields”. In: *ICCV*, 2021.
6. Y. Bengio and S. Bengio. “Modeling high-dimensional discrete data with multi-layer neural networks”. In: *NIPS*. 2000.
7. 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.
8. Y. Bengio, E. Laufer, G. Alain, and J. Yosinski. “Deep generative stochastic networks trainable by backprop”. In: *ICML*. 2014.
9. 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=r1lUOzWCW.
10. A. Birhane, V. U. Prabhu, and E. Kahembwe. *Multimodal datasets: misogyny, pornography, and malignant stereotypes*. 2021. DOI: 10.48550/ARXIV.2110.01963. URL: https://arxiv.org/abs/2110.01963.

11. R. Cai, G. Yang, H. Averbuch-Elor, Z. Hao, S. Belongie, N. Snavely, and B. Hariharan. "Learning Gradient Fields for Shape Generation". In: *ECCV*, 2020.
12. 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.
13. J. Červený. *Generalized Hilbert space-filling curve*. 2019.
14. E. Chan, M. Monteiro, P. Kellnhofer, J. Wu, and G. Wetzstein. "pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis". In: *CVPR*, 2021.
15. A. Chang, W. Monroe, M. Savva, C. Potts, and C. D. Manning. "Text to 3D Scene Generation with Rich Lexical Grounding". In: *ACL-IJCNLP*, 2015.
16. A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012*, 2015.
17. A. X. Chang, M. Savva, and C. D. Manning. "Learning Spatial Knowledge for Text to 3D Scene Generation". In: *EMNLP*, 2014.
18. K. Chen, C. B. Choy, M. Savva, A. X. Chang, T. A. Funkhouser, and S. Savarese. "Text2Shape: Generating Shapes from Natural Language by Learning Joint Embeddings". In: *CoRR* abs/1803.08495, 2018. arXiv: 1803.08495. URL: http://arxiv.org/abs/1803.08495.
19. M. Chen, A. Radford, R. Child, J. Wu, H. Jun, P. Dhariwal, D. Luan, and I. Sutskever. "Generative Pretraining from Pixels". In: 2020.
20. T. Chen, B. Xu, C. Zhang, and C. Guestrin. *Training Deep Nets with Sublinear Memory Cost*. 2016. arXiv: 1604.06174 [cs.LG].
21. 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.
22. X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel. "PixelSNAIL: An Improved Autoregressive Generative Model". In: 2018.
23. R. Child, S. Gray, A. Radford, and I. Sutskever. "Generating long sequences with sparse transformers". In: *arXiv*, 2019.
24. E. Chu. "Evolving Evocative 2D Views of Generated 3D Objects". In: *NeurIPS Creativity and Design Workshop*, 2021. URL: https://web.media.mit.edu/~echu/assets/projects/evolving-views/paper.pdf.

25. B. Coyne and R. Sproat. “WordsEye: an automatic text-to-scene conversion system”. In: *Computer graphics and interactive techniques*, 2001.

26. 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.

27. K. Desai and J. Johnson. “VirTex: Learning Visual Representations from Textual Annotations”. In: *CVPR*, 2021.

28. T. DeVries, M. A. Bautista, N. Srivastava, G. W. Taylor, and J. M. Susskind. “Unconstrained Scene Generation with Locally Conditioned Radiance Fields”. In: *arXiv*, 2021.

29. P. Dhariwal and A. Nichol. “Diffusion models beat gans on image synthesis”. In: *arXiv:2105.05233*, 2021.

30. 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.

31. 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: *ICLR*, 2021.

32. Y. Du and I. Mordatch. “Implicit Generation and Modeling with Energy Based Models”. In: *NeurIPS*, 2019.

33. A. A. Efros and T. K. Leung. “Texture synthesis by non-parametric sampling”. In: *ICCV*. 1999.

34. P. Esser, R. Rombach, and B. Ommer. *Taming Transformers for High-Resolution Image Synthesis*. 2020. arXiv: 2012.09841 [cs.CV].

35. P. Esser, R. Rombach, and B. Ommer. “Taming transformers for high-resolution image synthesis”. In: *CVPR*, 2021.

36. C. Fernando, S. M. A. Eslami, J.-B. Alayrac, P. Mirowski, D. Banarse, and S. Osindero. *Generative Art Using Neural Visual Grammars and Dual Encoders*. 2021. arXiv: 2105.00162 [cs.CV].

37. K. Frans, L. B. Soros, and O. Witkowski. “CLIPDraw: Exploring Text-to-Drawing Synthesis through Language-Image Encoders”. In: *CoRR*, 2021.

38. K. Frans, L. B. Soros, and O. Witkowski. “CLIPDraw: Exploring Text-to-Drawing Synthesis through Language-Image Encoders”. In: *CoRR* abs/2106.14843, 2021. arXiv: 2106.14843. URL: https://arxiv.org/abs/2106.14843.

39. B. J. Frey. *Graphical models for machine learning and digital communication*. MIT Press, 1998.

40. 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.

41. M. Germain, K. Gregor, I. Murray, and H. Larochelle. “MADE: Masked autoencoder for distribution estimation”. In: *ICML*. 2015, pp. 881–889.

42. G. Goh, N. C. †, C. V. †, 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.

43. 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.

44. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative Adversarial Nets”. In: *NeurIPS* 27, 2014. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

45. A. Graikos, N. Malkin, N. Jojic, and D. Samaras. “Diffusion models as plug-and-play priors”. In: *arXiv:2206.09012*, 2022.

46. K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra. “Deep AutoRegressive Networks”. In: *ICML*. Vol. 32. 2014.

47. A. Griewank and A. Walther. “Algorithm 799: revolve”. In: *ACM TOMS*, 2000.

48. J. Gu, L. Liu, P. Wang, and C. Theobalt. *StyleNeRF: A Style-based 3D-Aware Generator for High-resolution Image Synthesis*. 2021. arXiv: 2110.08985 [cs.CV].

49. K. Gupta and M. Chandraker. *Neural Mesh Flow: 3D Manifold Mesh Generationvia Diffeomorphic Flows*. 2020. arXiv: 2007.10973 [cs.CV].

50. D. Ha and D. Eck. *A Neural Representation of Sketch Drawings*. 2017. DOI: 10.48550/ARXIV.1704.03477. URL: https://arxiv.org/abs/1704.03477.

51. C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy". In: *Nature* 585:7825, 2020, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

52. J. Hays and A. A. Efros. "Scene Completion Using Millions of Photographs". In: *ACM TOG (SIGGRAPH)* 26:3, 2007.

53. J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee. *Flax: A neural network library and ecosystem for JAX*. Version 0.4.1. 2020. URL: http://github.com/google/flax.

54. 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.

55. 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.

56. G. E. Hinton. "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14:8, 2002.

57. J. Ho, A. Jain, and P. Abbeel. "Denoising Diffusion Probabilistic Models". In: *arXiv preprint arxiv:2006.11239*, 2020.

58. J. Ho, A. Jain, and P. Abbeel. "Denoising Diffusion Probabilistic Models". In: *NeurIPS*, 2020. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin.

59. J. Ho and T. Salimans. "Classifier-free diffusion guidance". In: *arXiv:2207.12598*, 2022.

60. 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.

61. P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks". In: *CVPR*, 2017.

62. A. Jabri, A. Owens, and A. A. Efros. "Space-Time Correspondence as a Contrastive Random Walk". In: *Advances in Neural Information Processing Systems*, 2020.

63. A. Jain. *VectorAscent: Generate vector graphics from a textual description*. 2021. URL: https://github.com/ajayjain/VectorAscent.

64. A. Jain, B. Mildenhall, J. T. Barron, P. Abbeel, and B. Poole. "Zero-Shot Text-Guided Object Generation with Dream Fields". In: *CVPR*, 2022.

65. A. Jain, M. Tancik, and P. Abbeel. "Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis". In: *ICCV*, 2021, pp. 5885–5894.

66. 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.

67. 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.

68. 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.

69. N. Jetchev. *ClipMatrix: Text-controlled Creation of 3D Textured Meshes*. 2021. arXiv: 2109.12922 [cs.LG].

70. C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. Le, Y.-H. Sung, Z. Li, and T. Duerig. "Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision". In: *ICML*, 2021. Ed. by M. Meila and T. Zhang.

71. 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.

72. 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.

73. A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik. "Learning Category-Specific Mesh Reconstruction from Image Collections". In: *ECCV*. 2018.

74. T. Karras, T. Aila, S. Laine, and J. Lehtinen. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *ICLR*. 2018.

75. 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.

76. T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. “Analyzing and improving the image quality of stylegan”. In: *CVPR*, 2020.

77. D. P. Kingma and P. Dhariwal. “Glow: Generative flow with invertible 1x1 convolutions”. In: *NeurIPS*. 2018.

78. D. P. Kingma, T. Salimans, B. Poole, and J. Ho. “Variational Diffusion Models”. In: *NeurIPS*, 2021.

79. 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.

80. D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. In: *ICLR*. Vol. 1. 2014.

81. 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.

82. D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. In: *ICLR*, 2014.

83. A. Kohli, V. Sitzmann, and G. Wetzstein. *Semantic Implicit Neural Scene Representations With Semi-Supervised Training*. 2021. arXiv: 2003.12673 [cs.CV].

84. H. Larochelle and I. Murray. “The neural autoregressive distribution estimator”. In: *AISTATS*. 2011, pp. 29–37.

85. Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. “A tutorial on energy-based learning”. In: 2006.

86. B. Li, F. Wu, K. Q. Weinberger, and S. Belongie. “Positional normalization”. In: *NeurIPS*. 2019.

87. T.-M. Li, M. Lukáč, G. Michaël, and J. Ragan-Kelley. “Differentiable Vector Graphics Rasterization for Editing and Learning”. In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39:6, 2020, 193:1–193:15.

88. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. “Microsoft COCO: Common Objects in Context”. In: *ECCV*, 2014. Ed. by D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars.

89. 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.

90. L. Liu, Y. Ren, Z. Lin, and Z. Zhao. *Pseudo Numerical Methods for Diffusion Models on Manifolds*. 2022. DOI: 10.48550/ARXIV.2202.09778. URL: https://arxiv.org/abs/2202.09778.

91. N. Liu, S. Li, Y. Du, J. B. Tenenbaum, and A. Torralba. "Learning to Compose Visual Relations". In: *Thirty-Fifth Conference on Neural Information Processing Systems*. 2021. URL: https://openreview.net/forum?id=fzwx-pzQGxe.

92. 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.

93. M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. "SMPL: A Skinned Multi-Person Linear Model". In: *SIGGRAPH Asia*, 2015.

94. A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool. *RePaint: Inpainting using Denoising Diffusion Probabilistic Models*. 2022. DOI: 10.48550/ARXIV.2201.09865. URL: https://arxiv.org/abs/2201.09865.

95. X. Ma, Y. Zhou, X. Xu, B. Sun, V. Filev, N. Orlov, Y. Fu, and H. Shi. "Towards Layer-wise Image Vectorization". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2022.

96. J. Menick and N. Kalchbrenner. "Generating high fidelity images with subscale pixel networks and multidimensional upscaling". In: *ICLR*. 2019.

97. G. Metzer, E. Richardson, O. Patashnik, R. Giryes, and D. Cohen-Or. *Latent-NeRF for Shape-Guided Generation of 3D Shapes and Textures*. 2022. DOI: 10.48550/ARXIV.2211.07600. URL: https://arxiv.org/abs/2211.07600.

98. T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur. "Recurrent neural network based language model". In: *INTERSPEECH*. 2010.

99. 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.

100. 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". In: *ECCV*, 2020. arXiv: 2003.08934 [cs.CV].

101. P. Mirowski, D. Banarse, M. Malinowski, S. Osindero, and C. Fernando. "CLIP-CLOP: CLIP-Guided Collage and Photomontage". In: *Proceedings of the Thirteenth International Conference on Computational Creativity*. 2022.

102. A. Mordvintsev, C. Olah, and M. Tyka. “Inceptionism: Going deeper into neural networks”. In: 2015.

103. A. Mordvintsev, N. Pezzotti, L. Schubert, and C. Olah. “Differentiable Image Parameterizations”. In: *Distill*, 2018. https://distill.pub/2018/differentiable-parameterizations. DOI: 10.23915/distill.00012.

104. A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. “Plug & play generative networks: Conditional iterative generation of images in latent space”. In: *CVPR*, 2017.

105. A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. “Synthesizing the preferred inputs for neurons in neural networks via deep generator networks”. In: *NeurIPS*, 2016.

106. A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen. “GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models”. In: *ICML*. 2022.

107. M. Niemeyer and A. Geiger. “GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields”. In: *CVPR*, 2021.

108. 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.

109. 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.

110. C. Olah, A. Mordvintsev, and L. Schubert. “Feature Visualization”. In: *Distill*, 2017. DOI: 10.23915/distill.00007.

111. K. Olszewski, S. Tulyakov, O. Woodford, H. Li, and L. Luo. “Transformable Bottleneck Networks”. In: *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

112. A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. “Conditional image generation with PixelCNN decoders”. In: *NeurIPS*. 2016.

113. A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. “Pixel Recurrent Neural Networks”. In: *ICML*. 2016.

114. A. van den Oord, Y. Li, and O. Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG].

115. A. van den Oord, O. Vinyals, and K. Kavukcuoglu. “Neural discrete representation learning”. In: *NIPS*. 2017.

116. A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. van den Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, N. Casagrande, D. Grewe, S. Noury, S. Dieleman, E. Elsen, N. Kalchbrenner, H. Zen, A. Graves, H. King, T. Walters, D. Belov, and D. Hassabis. "Parallel WaveNet: Fast High-Fidelity Speech Synthesis". In: *ICML*, 2018.

117. A. v. d. Oord, Y. Li, and O. Vinyals. "Representation learning with contrastive predictive coding". In: *arXiv:1807.03748*, 2018.

118. D. H. Park, S. Azadi, X. Liu, T. Darrell, and A. Rohrbach. "Benchmark for Compositional Text-to-Image Synthesis". In: *NeurIPS*, 2021.

119. J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *CVPR*, 2019.

120. N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. "Image Transformer". In: *ICML*. 2018.

121. 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.

122. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

123. O. Patashnik, Z. Wu, E. Shechtman, D. Cohen-Or, and D. Lischinski. "Styleclip: Text-driven manipulation of stylegan imagery". In: *ICCV*, 2021.

124. D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. "Context encoders: Feature learning by Inpainting". In: *CVPR*. 2016.

125. V. Perez, J. Simon, and T. Shiri. "Sculpting with Words". In: 2021. URL: https://www.morphogen.io/research/sculpting_with_words.pdf.

126. P. von Platen, S. Patil, A. Lozhkov, P. Cuenca, N. Lambert, K. Rasul, M. Davaadorj, and T. Wolf. *Diffusers: State-of-the-art diffusion models*. https://github.com/huggingface/diffusers. 2022.

127. B. Poole, A. Jain, J. T. Barron, and B. Mildenhall. “DreamFusion: Text-to-3D using 2D Diffusion”. In: *arXiv*, 2022.

128. B. Poole, S. Ozair, A. Van Den Oord, A. Alemi, and G. Tucker. “On variational bounds of mutual information”. In: *ICML*, 2019.

129. 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.

130. 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”. In: *CoRR* abs/2103.00020, 2021. arXiv: 2103.00020. URL: https://arxiv.org/abs/2103.00020.

131. T. Raiko, Y. Li, K. Cho, and Y. Bengio. “Iterative neural autoregressive distribution estimator NADE-k”. In: *NIPS*. 2014.

132. P. Ramachandran, B. Zoph, and Q. V. Le. “Searching for Activation Functions”. In: *CoRR* abs/1710.05941, 2017. arXiv: 1710.05941. URL: http://arxiv.org/abs/1710.05941.

133. A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. DOI: 10.48550/ARXIV.2204.06125. URL: https://arxiv.org/abs/2204.06125.

134. A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. “Zero-Shot Text-to-Image Generation”. In: *CoRR* abs/2102.12092, 2021. arXiv: 2102.12092. URL: https://arxiv.org/abs/2102.12092.

135. 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.

136. D. Rezende and S. Mohamed. “Variational Inference with Normalizing Flows”. In: *ICML*. 2015.

137. D. J. Rezende, S. Mohamed, and D. Wierstra. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”. In: *ICML*. 2014.

138. 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, Bejing, China, 2014, pp. 1278–1286. URL: http://proceedings.mlr.press/v32/rezende14.html.

139. E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski. “Kornia: an Open Source Differentiable Computer Vision Library for PyTorch”. In: *Winter Conference on Applications of Computer Vision*. 2020. URL: https://arxiv.org/pdf/1910.02190.pdf.

140. 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.

141. R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2021. arXiv: 2112.10752 [cs.CV].

142. O. Ronneberger, P. Fischer, and T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597.

143. O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional networks for biomedical image segmentation”. In: 2015.

144. 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.

145. C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi. “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”. In: *arXiv:2205.11487*, 2022.

146. C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi. *Image Super-Resolution via Iterative Refinement*. 2021. DOI: 10.48550/ARXIV.2104.07636. URL: https://arxiv.org/abs/2104.07636.

147. R. Salakhutdinov and I. Murray. “On the quantitative analysis of deep belief networks”. In: *ICML*. 2008.

148. 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.

149. A. Sanghi, H. Chu, J. G. Lambourne, Y. Wang, C.-Y. Cheng, and M. Fumero. *CLIP-Forge: Towards Zero-Shot Text-to-Shape Generation*. 2021. arXiv: 2110.02624 [cs.CV].

150. P. Schaldenbrand, Z. Liu, and J. Oh. *StyleCLIPDraw: Coupling Content and Style in Text-to-Drawing Synthesis*. 2021. arXiv: 2111.03133 [cs.CV].

151. P. Schaldenbrand, Z. Liu, and J. Oh. *StyleCLIPDraw: Coupling Content and Style in Text-to-Drawing Translation*. 2022. DOI: 10.48550/ARXIV.2202.12362. URL: https://arxiv.org/abs/2202.12362.

152. 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.

153. K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger. "Graf: Generative radiance fields for 3d-aware image synthesis". In: *NeurIPS*, 2020. arXiv: 2007.02442 [cs.CV].

154. K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

155. K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015.

156. V. Sitzmann, E. R. Chan, R. Tucker, N. Snavely, and G. Wetzstein. *MetaSDF: Meta-learning Signed Distance Functions*. 2020. arXiv: 2006.09662 [cs.CV].

157. V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. "Implicit neural representations with periodic activation functions". In: *NeurIPS*, 2020.

158. 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.

159. J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *ICML*. 2015, pp. 2256–2265.

160. J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *ICML*, 2015.

161. J. Song, C. Meng, and S. Ermon. "Denoising Diffusion Implicit Models". In: *CoRR* abs/2010.02502, 2020. arXiv: 2010.02502. URL: https://arxiv.org/abs/2010.02502.

162. Y. Song and S. Ermon. "Generative modeling by estimating gradients of the data distribution". In: *NeurIPS*. 2019.

163. Y. Song, C. Meng, R. Liao, and S. Ermon. *Nonlinear Equation Solving: A Faster Alternative to Feedforward Computation*. 2020. arXiv: 2002.03629 [cs.LG].

164. Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *ICLR*, 2021.

165. S. Subramanian, W. Merrill, T. Darrell, M. Gardner, S. Singh, and A. Rohrbach. “ReCLIP: A Strong Zero-shot Baseline for Referring Expression Comprehension”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Dublin, Ireland, 2022.

166. 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.

167. 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].

168. 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”. In: *CVPR*, 2021.

169. M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”. In: *NeurIPS*, 2020.

170. Y. Tian and D. Ha. *Modern Evolution Strategies for Creativity: Fitting Concrete Images and Abstract Concepts*. 2021. DOI: 10.48550/ARXIV.2109.08857. URL: https://arxiv.org/abs/2109.08857.

171. A. Trevithick and B. Yang. “GRF: Learning a General Radiance Field for 3D Scene Representation and Rendering”. In: *arXiv:2010.04595*. 2020.

172. 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 Regognition (CVPR)*. 2017.

173. B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. “Neural Autoregressive Distribution Estimation”. In: *JMLR*, 2016.

174. B. Uria, I. Murray, and H. Larochelle. “A deep and tractable density estimator”. In: *ICML*. 2014.

175. B. Uria, I. Murray, and H. Larochelle. “RNADE: The real-valued neural autoregressive density-estimator”. In: *NIPS*. 2013.

176. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need”. In: *NeurIPS*, 2017, pp. 5998–6008. arXiv: 1706.03762 [cs.CL].

177. P. Vincent. “A connection between score matching and denoising autoencoders”. In: *Neural computation*, 2011.

178. Y. Vinker, E. Pajouheshgar, J. Y. Bo, R. C. Bachmann, A. H. Bermano, D. Cohen-Or, A. Zamir, and A. Shamir. *CLIPasso: Semantically-Aware Object Sketching*. 2022. arXiv: 2202.05822 [cs.GR].

179. 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.

180. 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.

181. T. White. *Pixray*. URL: https://github.com/pixray/pixray.

182. A. J. Wiggers and E. Hoogeboom. *Predictive Sampling with Forecasting Autoregressive Models*. 2020. arXiv: 2002.09928 [cs.LG].

183. O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson. “SynSin: End-to-end View Synthesis from a Single Image”. In: *CVPR*. 2020.

184. J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. “Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 82–90.

185. J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum. “Learning 3D Shape Priors for Shape Completion and Reconstruction”. In: *European Conference on Computer Vision (ECCV)*. 2018.

186. L. Yariv, J. Gu, Y. Kasten, and Y. Lipman. “Volume Rendering of Neural Implicit Surfaces”. In: *arXiv:2106.12052*, 2021.

187. L. Yen-Chen. *PyTorchNeRF: a PyTorch implementation of NeRF*. 2020. URL: https://github.com/yenchenlin/nerf-pytorch/.

188. A. Yu, V. Ye, M. Tancik, and A. Kanazawa. *pixelNeRF: Neural Radiance Fields from One or Few Images*. 2020.

189. F. Yu and V. Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *ICLR*. 2015.

190. J. Yu, Y. Xu, J. Y. Koh, T. Luong, G. Baid, Z. Wang, V. Vasudevan, A. Ku, Y. Yang, B. K. Ayan, B. Hutchinson, W. Han, Z. Parekh, X. Li, H. Zhang, J. Baldridge, and Y. Wu. “Scaling Autoregressive Models for Content-Rich Text-to-Image Generation”. In: *arXiv:2206.10789*, 2022.

191. X. Zhai, X. Wang, B. Mustafa, A. Steiner, D. Keysers, A. Kolesnikov, and L. Beyer. *LiT: Zero-Shot Transfer with Locked-image Text Tuning*. 2021. arXiv: 2111.07991 [cs.CV].

192. H. Zhang, J. Y. Koh, J. Baldridge, H. Lee, and Y. Yang. “Cross-modal contrastive learning for text-to-image generation”. In: *CVPR*, 2021.

193. K. Zhang, G. Riegler, N. Snavely, and V. Koltun. “NeRF++: Analyzing and Improving Neural Radiance Fields”. In: *arXiv:2010.07492*, 2020.

194. 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.

195. X. Zhang, Z. Zhang, C. Zhang, J. B. Tenenbaum, W. T. Freeman, and J. Wu. “Learning to Reconstruct Shapes From Unseen Classes”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

196. L. Zhou, Y. Du, and J. Wu. “3D Shape Generation and Completion Through Point-Voxel Diffusion”. In: *ICCV*, 2021.

197. P. Zhou, L. Xie, B. Ni, and Q. Tian. “CIPS-3D: A 3D-Aware Generator of GANs Based on Conditionally-Independent Pixel Synthesis”. In: 2021. arXiv: 2110.09788.

198. 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.

Chapter 8

Appendix: Locally Masked Convolution for Autoregressive Models

8.1 Order visualization

Figure 2.2 shows three image generation orders and corresponding local masks used by the first VectorFusion 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 8.1 shows the 8 variants of the S-curve generation order used for order-agnostic training.

8.2 Mask conditioning

ConvNADE [173] 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.

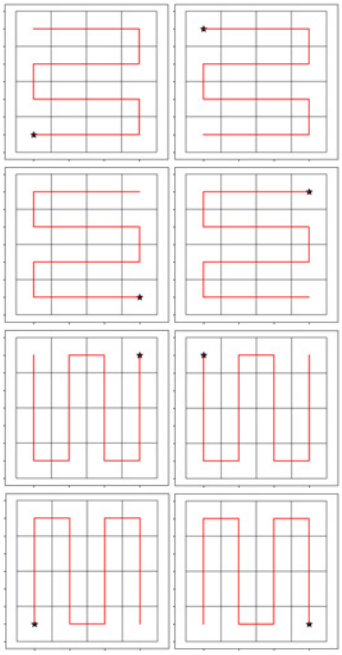


Figure 8.1: Eight variants of the S-curve generation order.

Locally masked convolutions can also condition upon the mask in each layer. Algorithm 4 is an adaptation of Algorithm 1 that supports mask conditioning, with modifications shown in green. Algorithm 4 applies a learned weight matrix $\mathcal{W}_{\mathcal{M}}$ to the first C_{in} rows of the mask matrix as the 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.

8.3 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

Algorithm 4 LMConv with mask conditioning

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

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 [74] contains 30,000 256×256 8-bit color celebrity photos. For experiments, we use the same CelebA-HQ data splits as Glow [77], 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 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 [74], 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.

8.4 Additional samples

Figure 8.2 shows intermediate states of the forward sampling process for unconditional generation of grayscale MNIST digits. We samples 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 [13] 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

Figure 8.2: Unconditionally generating MNIST digits with two Hilbert curve orders, starting at the top or bottom left.

Fig. 8.2. 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 8.3 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.

8.5 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 im2col operation. We provide an abbreviated Python code sample implementing VectorFusion using the PyTorch library in Figure 8.4. The full source including gradient computation, parameter initialization and mask conditioning is available at https://ajayjain.github.io/lmconv.

Figure 8.3: 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 8.4: A memory-efficient PyTorch v1.5.1 implementation of VectorFusion. Gradient calculation is omitted for brevity. See https://ajayjain.github.io/lmconv for full code.

Chapter 9

Appendix: Putting NeRF on a Diet

9.1 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 [188], we use the same source view as [188]. This viewpoint was manually selected and is shared across all 15 scenes.

Simplified NeRF baseline The published version of NeRF [100] 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 9.1 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 [100] for hierarchical sampling. ✗ denotes that we do not use the fine network.

Implementation Our implementation is based on a PyTorch port [187] 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 [187] rather than full-resolution 800×800 images. NV is trained with full-resolution 800×800 views. NV renderings are downsampled with a 2x2 box filter to

Table 9.1: **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

400×400 to compute metrics. We train all NeRF, Simplified NeRF and DietNeRF models with the Adam optimizer [81] for 200k iterations.

Metrics Our PSNR, SSIM, and LPIPS metrics use the same implementation as [188] based on the scikit-image Python package [179]. For the DTU dataset, [188] excluded some poses from the 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

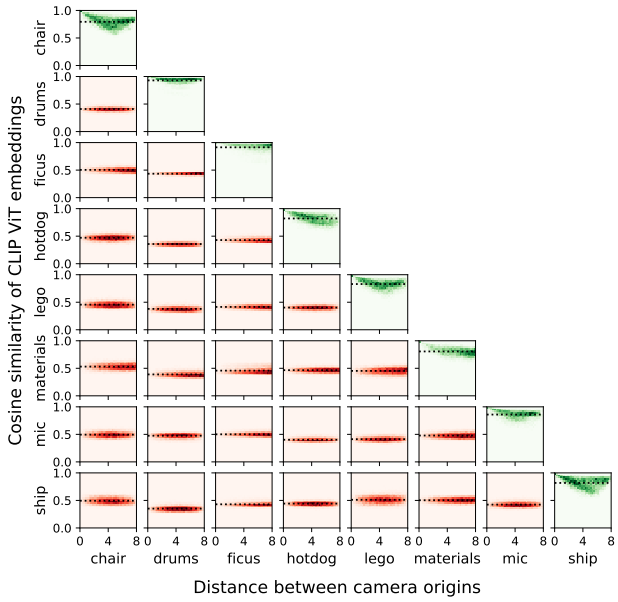


Figure 9.1: **CLIP ViT embeddings are more similar between views of the same scene than across different scenes.** We show a 2D histogram for each pair of Realistic Synthetic scenes comparing ViT embedding similarity and the distance between views. The dashed line shows mean cosine similarity, and green 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.

viewpoint across 124 training and validation scenes). FID and KID metrics are computed using the `torch-fidelity` Python package [109].

9.2 Per-scene metrics

Embedding similarity In Figure 9.1, 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

Table 9.2: **Quality metrics for each scene in the Realistic Synthetic dataset with 8 observed views.**

PSNR ↑	**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 [92]	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)	23.897	24.633	**20.034**	20.744	26.321	**23.043**	21.254	25.250
DietNeRF, $\mathcal{L}_{\text{MSE}}$ ft (ours)	**24.311**	**25.595**	20.029	20.940	**26.794**	22.536	**21.621**	**26.626**
NeRF, 100 views	**31.618**	**34.073**	**25.530**	**29.163**	**33.197**	**29.407**	**29.340**	**36.899**

SSIM ↑	**Lego**	**Chair**	**Drums**	**Ficus**	**Mic**	**Ship**	**Materials**	**Hotdog**
NeRF	0.526	0.861	0.770	0.661	0.944	0.605	0.484	0.644
NV [92]	0.707	0.795	0.675	0.815	0.816	0.602	0.721	0.796
Simplified NeRF	0.775	0.859	0.727	0.872	0.930	0.694	0.823	0.894
DietNeRF (ours)	0.863	0.898	0.843	0.872	0.944	**0.758**	0.843	0.904
DietNeRF, $\mathcal{L}_{\text{MSE}}$ ft (ours)	**0.875**	**0.912**	**0.845**	**0.874**	**0.950**	0.757	**0.851**	**0.924**
NeRF, 100 views	**0.965**	**0.978**	**0.929**	**0.966**	**0.979**	**0.875**	**0.958**	**0.981**

LPIPS ↓	**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 [92]	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)	0.110	0.092	**0.117**	0.097	0.053	0.204	0.102	0.097
DietNeRF, $\mathcal{L}_{\text{MSE}}$ 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**

similarity is high across all camera distances. For views sampled from different scenes, similarity is low (cosine similarity around 0.5).

Quality metrics Table 9.2 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 $\mathcal{L}_{\text{MSE}}$ performs best or second-best.

9.3 Qualitative results and ground-truth

In this section, we provide additional qualitative results. Figure 9.2 shows the ground-truth training views used for 8-shot Realistic Synthetic experiments. These views are sampled at random from the training set of [100]. 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 9.3, 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 9.4, 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 $\mathcal{L}_{\text{MSE}}$ for 20k iterations, whereas DietPixelNeRF is fine-tuned using $\mathcal{L}_{\text{MSE}} + \mathcal{L}_{\text{SC}}$ 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.

9.4 Adversarial approaches

While NeRF is only supervised from observed poses, conceptually, a GAN [44] 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 [61, 198] and can be useful

Figure 9.2: **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.

for high-resolution image generation [34]. SinGAN [144] trains multiscale patch discriminators on a single image, comparable to our single-scene few-view setting. In early experiments, we trained patch-wise discriminators per-scene to supervise f_θ from novel poses in addition to $\mathcal{L}_{\text{SC}}$. However, an auxiliary adversarial loss led to artifacts on Realistic Synthetic scenes, both in isolation and in combination with our semantic consistency loss.

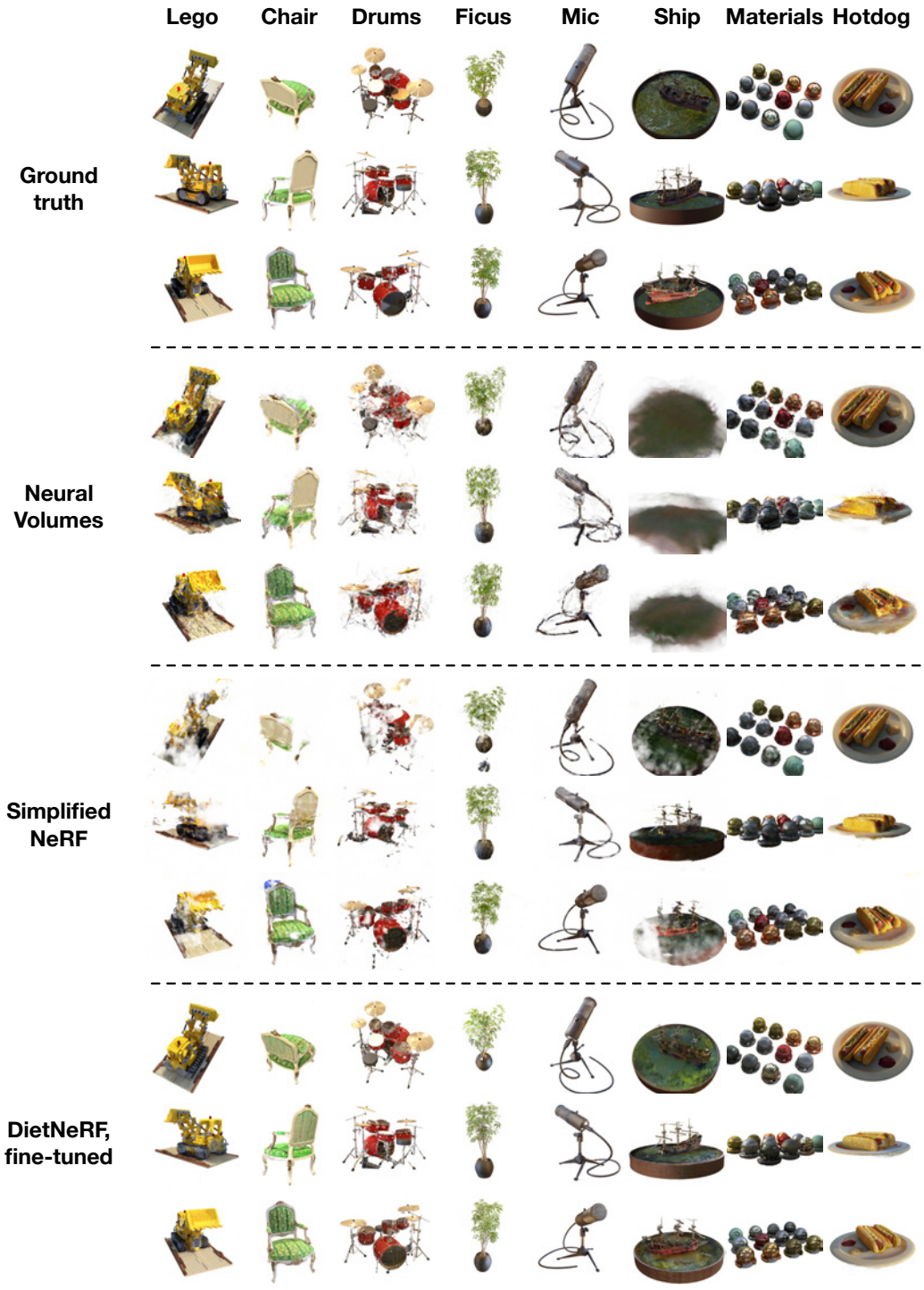


Figure 9.3: **Additional renderings of Realistic Synthetic scenes.**

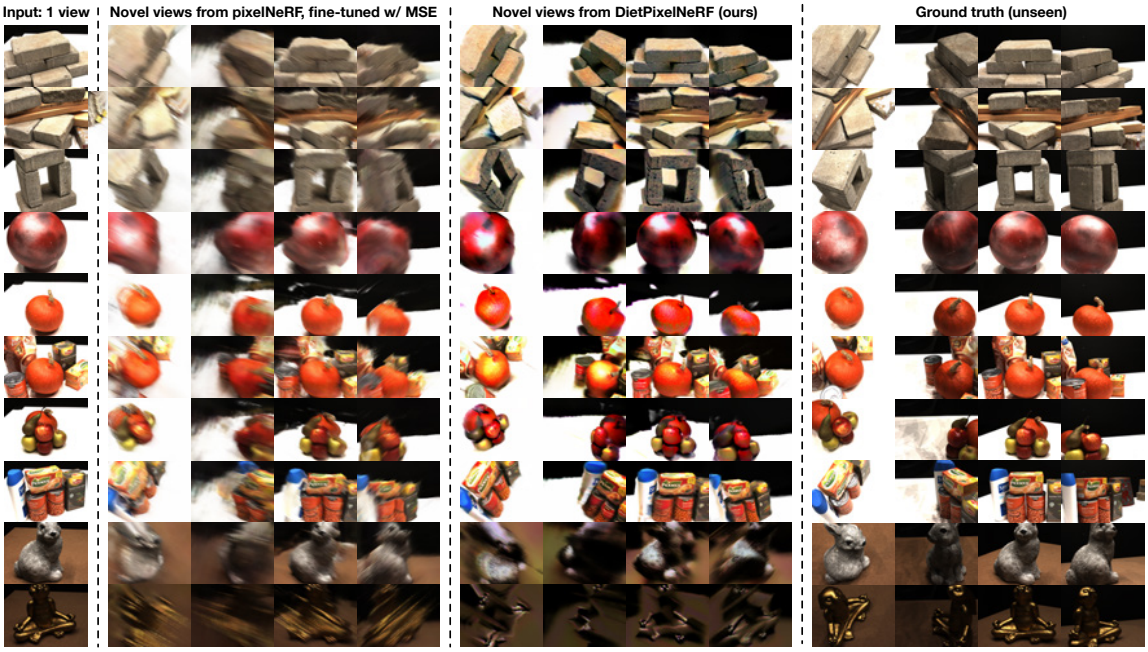


Figure 9.4: **One-shot novel view synthesis:** Additional renderings of DTU scenes generated from a single observed view (left). Ground truth views are shows for reference, but are not provided to the model. pixelNeRF and DietPixelNeRF are pre-trained on the same dataset of other scenes, then fine-tuned on the single input view for 20k iterations with $\mathcal{L}_{\text{MSE}}$ alone (pixelNeRF) or $\mathcal{L}_{\text{MSE}} + \mathcal{L}_{\text{SC}}$ (DietPixelNeRF).

Chapter 10

Appendix: Zero-Shot Text-Guided Object Generation with Dream Fields

10.1 Qualitative results and ablations

An explanatory video with more qualitative results, code, an interactive Colab notebook, and object-centric prompts are available at https://ajayj.com/dreamfields. The video includes 360° renderings where the camera orbits the object, as well as associated depth maps.

Changing the image-text model Figure 10.1 qualitatively compares generations using guidance from three different contrastive image-text models. Dream Fields generated with LiT_{uu} B/32 are generally sharper than those with CLIP models, but all three can produce objects reflecting some aspects of the prompts.

Diversity of synthesized objects In creative applications, users often want to select between multiple synthesized results. Dream Fields can synthesize multiple objects from the same prompt by changing the random seed before optimization. The seed changes the initialization of the NeRF weights, the camera pose sampled each iteration and the random background and crop augmentations. Figure 10.2 shows the effect of changing the seed for four object-centric COCO prompts. Changing the seed changes scene shape and layout. For example, the bus on the left is compressed into a cubical shape, while the bus on the right is elongated. Colors and textures are often similar across the seeds, though can also vary. Changing the seed is another dimension of control in addition to prompt engineering.

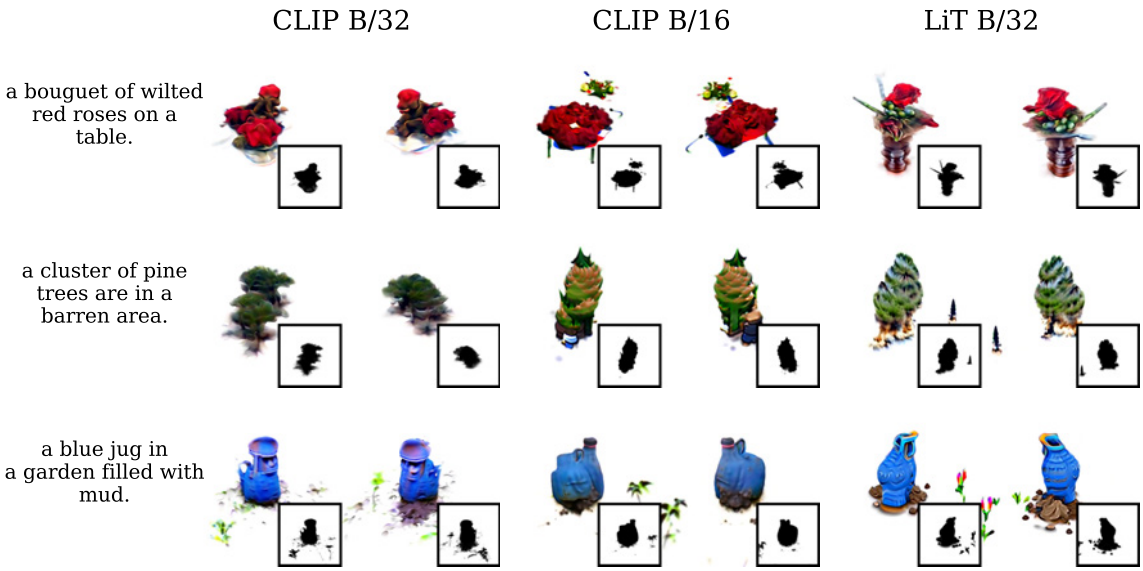


Figure 10.1: **Varying the image-text model** used for Dream Field optimization.

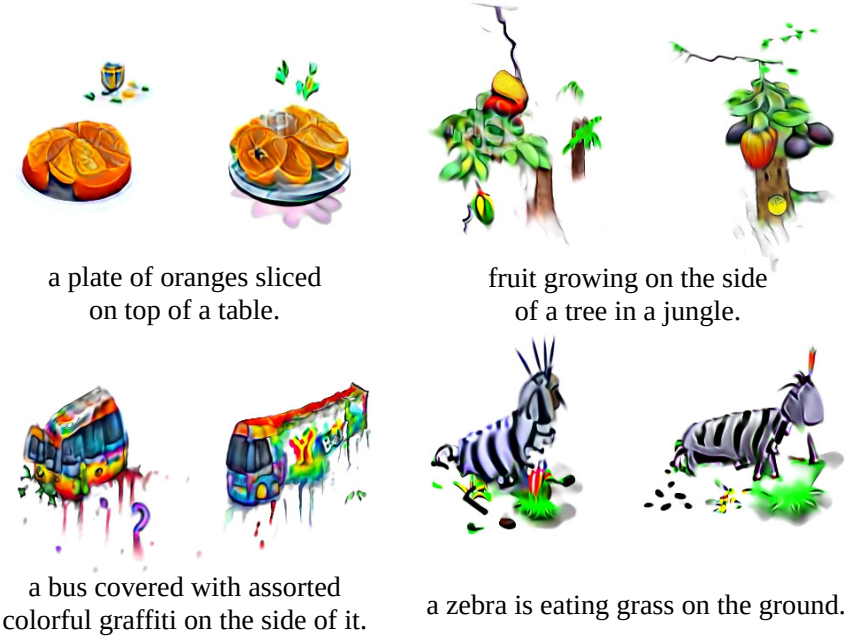


Figure 10.2: **Object diversity:** Objects vary with different seeds, effecting NeRF's weight initialization, camera sampling and augmentations.

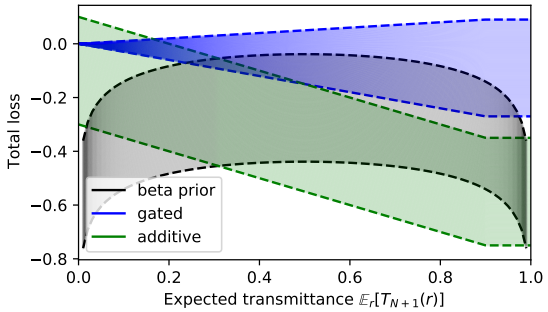


Figure 10.3: Total loss with different sparsity regularizers and constant $\mathcal{L}_{\text{CLIP}} \in [0.1, -0.3]$. Upper and lower bounds of $\mathcal{L}_{\text{CLIP}}$ are shown with dashed lines. The additive loss is convex, and is always minimized by increasing transmittance.

10.2 Object Centric COCO captions dataset

Our Object Centric COCO dataset includes 153 test set prompts and 74 development set prompts. Several additional prompts are used for qualitative results, and are included in the main paper alongside figures. Captions are included along with code on the project website.

10.3 Hyperparameters and training setup

Positional encoding Our Fourier feature positional encodings use $L = 8$ frequency levels, while novel view synthesis applications with image supervision commonly use $L = 10$ to fit high-frequency details in photographs. Low-frequency ablations in Table 1 use $L = 6$, which can improve convergence in the absence of our other geometric priors.

Rendering Scenes are bounded to a cube with side length 2. The camera is sampled at a fixed radius of 4 units from the center of the cubical scene bounds and an elevation of 30° above the equator. Near and far planes are set at $4 \pm \sqrt{3}$ units from the camera based on the minimum and maximum possible distance to the corners of the cube. During training, we sample 192 points along each ray, spaced uniformly and jittered with uniform noise. Rendered 168^2 views are cropped to 154^2 and upsampled to CLIP's input resolution for scenes where we compute qualitative metrics or 252^2 views are cropped to 224^2 for certain higher-quality visualizations. Crop sizes are selected to cover about 80% of the image area. At test time, we sample 512 points along the rays and render at a higher resolution equal to CLIP's input size of 224^2 or LiT's input size of 288^2 for computing R-Precision and 400^2 for visualizations.

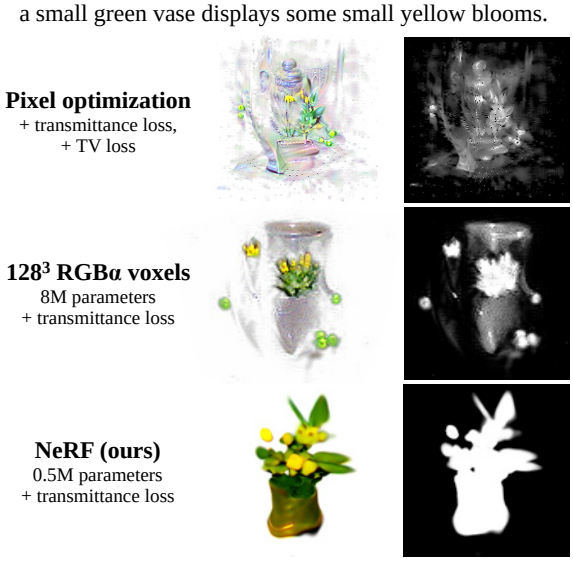


Figure 10.4: **Optimizing a Neural Radiance Field scene representation** (bottom) **leads to fewer artifacts** than optimizing an explicit single-view 2D image (top) or 3D voxel grid (middle), even when the explicit representations are regularized. Our MLP has 16× fewer parameters than the voxel grid, which may contribute to smoother, less noisy objects. We use CLIP B/16 for this experiment.

Optimization MLP parameters are initialized with the Flax [53] defaults: LeCun normal weights and zero bias for linear layers, and unit scaling and zero bias for layer normalization. The MLP is optimized with Adam with $\epsilon = 10^{-5}$. Learning rate warms up exponentially from 10^{-5} to 10^{-4} over 1500 iterations, then is held constant. The camera origin is separately tracked with an exponential moving average with decay rate 0.999 of the center of mass of rendered density. Figure 10.3 visualizes different forms of the transmittance loss.

Hyperparameter selection Hyperparameters are manually tuned for visual quality on a development set of 74 object-centric COCO captions distinct from the test set reported in the paper. Most tuning is done on a smaller subset of 20 of the 74 captions, and hyperparameters are shared across all scenes.

Hardware Optimization is done on 8 preemptible TPU cores, and 10K iterations takes approximately 1 hour 12 minutes. This means each Dream Field costs approximately $3-4 to generate on Google Cloud, which is economical for applications. Training is bottlenecked by MLP inference and backpropagation during volumetric rendering, not CLIP.

10.4 Pixel and voxel baselines

We implemented 2D image optimization with a total variation loss and generative prior (CLIP Guided Diffusion), as well as a 3D voxel baselines to replace NeRF in Fig. 10.4. All results for this ablation optimize CLIP ViT B/16.

The 2D image is an RGBα pixel grid, composited with random backgrounds during optimization similar to Dream Fields. Optimizing a single 2D RGBα image does not produce a multi-view consistent 3D object, so other viewpoints cannot be rendered. Even with transmittance and TV regularization, the resulting image is noisy.

The voxel grid stores 128^3 RGB and alpha values, interpolated trilinearly at ray sample points and composited without a neural network using the PyTorch3D library. Despite the transmittance loss, data augmentations and scene bounds, the voxel grid also has significant low-density artifacts. The voxel baseline has CLIP B/32 R-Precision 37.0%±3.9, while NeRF has 59.8%±2.8 (Tables 4.1, 4.3) with 16× fewer parameters, showing that the neural representation improves consistency with the input caption in a generalizable way. Using a hybrid representation with an explicit voxel grid followed by a smaller MLP head might improve computational efficiency of Dream Fields without degrading quality.

10.5 Signed distance field parameterization

In early experiments, we learned scene density σ with the VolSDF parameterization [186] $\sigma(\mathbf{x}) = \alpha\Phi_\beta(-d_\Omega(\mathbf{x}))$ where $d_\Omega(\mathbf{x})$ is a signed distance function implicitly defining the object surface and Φ is the CDF of the Laplace distribution. This allows normal vector prediction with autodifferentiation and could improve the quality of the surface extracted from the radiance field. Dream Fields successfully train with this alternate parameterization and produce visually compelling objects. The SDF Eikonal loss introduces an additional loss weight hyperparameter, which benefits from some tuning. Alternate 3D representations are an interesting avenue for future work.

10.6 Impact of optimization time

Dream Fields can overfit to the aligned image-text representation used for optimization. Figure 10.5 shows the training losses, $\mathcal{L}_{\text{CLIP}}$ and mean transmittance $\text{mean}(T(\theta, \mathbf{p}))$, as well as the validation R-Precision. Objects are generated with LiT_{uu} ViT B/32 guidance, and R-Precision is computed with a different contrastive image-text model, CLIP ViT B/32. Validation renderings are also done at a held-out elevation angle. Training loss continues to improve over long optimization

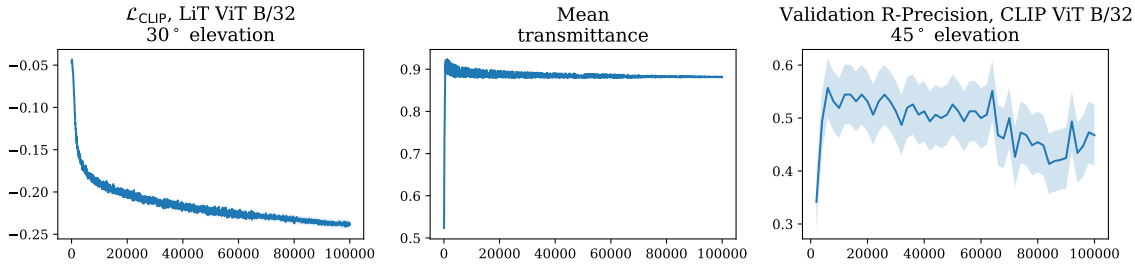


Figure 10.5: Long-run training and validation curves averaged over 79 hand-written prompts. Transmittance remains close to the target τ throughout training. Dream Fields overfit to the image-text representations used for optimization, so in quantitative experiments, we stop training at 10K iterations. The standard error of the mean is shaded.

trajectories, up to 10× longer than reported in the main paper. However, validation retrieval accuracy declines after 5-10K iterations. The metrics are averaged over 79 different hand-written captions that test fine-grained variations in wording and prompt engineering.

Qualitatively, additional details and hyper-realistic effects are added over the course of long runs, shown in our supplementary video. Some details are not realistic, like floating text related to the typographic attacks identified in [42].

More augmentations may help further regularize the optimization. These include more aggressive 2D image augmentations such as smaller random crops, and more 3D data augmentations including varying focal length, varying distance from the subject and varying elevation. 3D data augmentations are supported by our approach.

Chapter 11

Appendix: VectorFusion: Text-to-SVG by Abstracting Image Diffusion Models

11.1 Website with results, videos, and benchmark

Our project website at https://ajayj.com/vectorfusion includes videos of the optimization process and many more qualitative results in SVG format.

See https://ajayj.com/vectorfusion/svg_bench_prompts.txt for the benchmark used for evaluation, consisting of 128 diverse prompts sourced from prior work.

11.2 Ablation: Reinitializing paths

We reinitialize paths below an opacity threshold or area threshold periodically, every 50 iterations. The purpose of reinitializing small, faint paths is to encourage the usage of all paths. Path are not reinitialized for the final 200-500 iterations of optimization, so reinitialized paths have enough time to converge. Reinitialization is only applied during image-text loss computation stages. Note that for SD + LIVE + SDS, we only reinitialize for SDS finetuning, not LIVE image vectorization. For CLIPDraw, we only reinitialize paths for our closed Bézier path version, not for the original CLIPDraw version, which consists of open Bézier paths where it is difficult to measure area. Table 11.1 includes hyperparameters for the threshold, frequency, and number of iterations of reinitialization.

Table 11.1: Path reinitialization hyperparameters

Method	Opacity Thresh.	Area Thresh.	Freq.	Iters
SDS (from scratch)	0.05	0	50	1.5/2K SDS steps
SD + LIVE + SDS	0.05	64 px^2	50	0.8/1K SDS steps
CLIPDraw (icon.)	0.05	64 px^2	50	1.8/2K CLIP steps

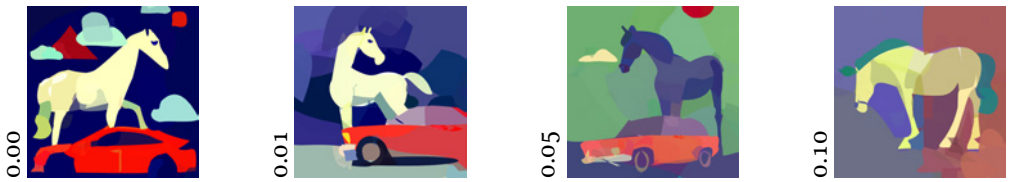
Table 11.2: Evaluating path reinitialization with 64 closed, colored Bézier curves, our iconographic setting. VectorFusion reinitializes paths during optimization to maximize their usage. This improves caption consistency both when training randomly initialized paths with SDS (SDS w/ reinit), and when initializing with a LIVE traced Stable Diffusion sample (SD + LIVE + SDS w/ reinit).

		Caption consistency			
		CLIP L/14		OpenCLIP H/14	
Method	K	R-Prec	Sim	R-Prec	Sim
SDS	0	75.0	24.0	75.0	28.8
w/ reinit	0	78.1	24.1	78.1	**29.3**
SD + LIVE + SDS	4	64.8	22.6	68.8	26.7
w/ reinit	4	**78.9**	**29.4**	**81.3**	24.5

Table 11.2 ablates the use of reinitialization. When optimizing random paths with SDS, reinitialization gives an absolute +3.0% increase in R-Precision according to OpenCLIP H/14 evaluation. When initialized from a LIVE traced sample, reinitialization is quite helpful (+12.5% R-Prec).

11.3 Ablation: Saturation Penalty

We proposed a saturation penalty for pixel art 6.4.4. We did not use it for icongraphy, but it greatly reduces saturation, as shown below.



11.4 Ablation: Number of paths

VectorFusion optimizes path coordinates and colors, but the number of primitive paths is a non-differentiable hyperparameter. Vector graphics with fewer paths will be more abstract, whereas photorealism and details can be improved with many paths. In this ablation, we experiment with different number of paths. We evaluate caption consistency across path counts. For methods that use LIVE, this ablation uses a path schedule that incrementally adds 2, 4, and 10 for a total of 16 paths, and a path schedule of 8, 16, 32, and 72 for 128 total paths. We set K=4 for rejection sampling. Figure 11.1 and Table 11.3 show results. Consistency improves with more paths, but there are diminishing returns.

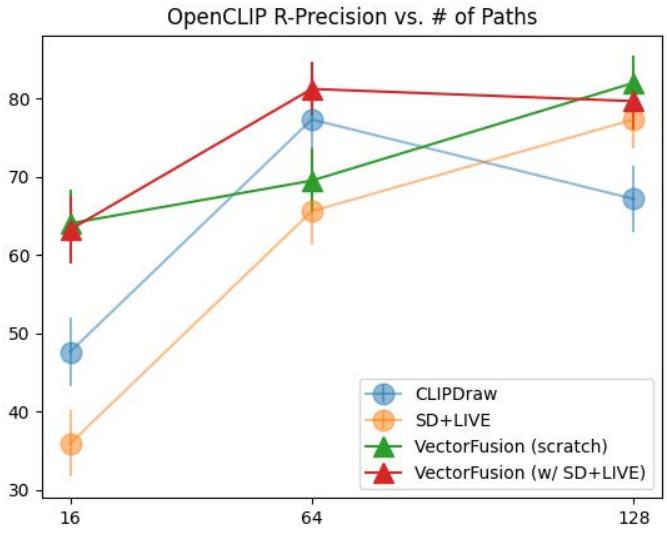


Figure 11.1: Increasing the number of paths generally improves our caption consistency metrics. We find that 64 paths are sufficient to express and optimize SVGs that are coherent with the caption.

11.5 Ablation: Number of rejection samples

In this section, we ablate on the number of Stable Diffusion samples used for rejection sampling. We include results in Figure 11.2 and Table 11.4. Rejection sampling greatly improves coherence of Stable Diffusion raster samples with the caption, since rejection explicitly maximizes a CLIP image-text similarity score. After converting the best raster sample to a vector graphic with LIVE

Table 11.3: **Caption Consistency vs. # Paths**. Increasing the number of paths allows for greater expressivity and caption coherency. However, it also increases memory and time complexity, and we opt for 64 paths to balance between performance and time constraints. We use rejection sampling K=4 for SD+LIVE and SD+LIVE+SDS, and we optimize open Bézier paths for CLIPDraw.

		Caption consistency			
		CLIP L/14		OpenCLIP H/14	
Method	# Paths	R-Prec	Sim	R-Prec	Sim
SDS (scratch)	16	68.0	23.4	64.1	27.4
SD+LIVE	16	33.6	19.7	35.9	22.9
SD+LIVE+SDS	16	63.3	22.9	63.3	27.3
CLIPDraw	16	58.6	23.9	47.7	27.4
SDS (scratch)	64	76.6	24.3	69.5	28.5
SD+LIVE	64	57.0	21.7	59.4	25.8
SD+LIVE+SDS	64	78.9	**29.4**	81.3	24.5
CLIPDraw	64	**85.2**	27.2	77.3	**31.7**
SDS (scratch)	128	83.6	24.8	**82.0**	29.7
SD+LIVE	128	77.3	23.7	77.34	28.7
SD+LIVE+SDS	128	78.1	24.8	79.7	29.7
CLIPDraw	128	73.4	25.7	67.2	30.4

(SD+LIVE), coherence is reduced 10-15% in terms of OpenCLIP H/14 R-Precision. However, using more rejection samples generally improves the SD+LIVE baseline. In contrast, VectorFusion is robust to the number of rejection samples. Initializing with the vectorized result after 1-4 Stable Diffusion samples is sufficient for high SVG-caption coherence.

11.6 Ablation: Classifier-Free Guidance

We compare guidance scales in Table 11.5 and Figure 11.3. This hyperparameter seems fairly robust. While high guidance (>50) can lead to cartoonish generations, it is important for coherence.

11.7 Ablation: SDS + CLIP Hybrid Losses

We investigate combining the SDS and CLIP losses. Adding an additional CLIP loss improves our CLIP-based metrics in Table 11.6. Qualitatively, the additional CLIP loss leads to very different

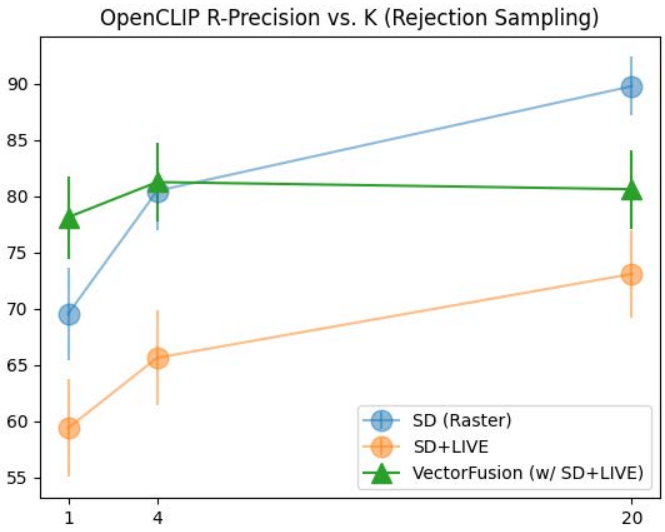


Figure 11.2: Coherence with the caption improves with additional rejection samples. Even with 20 rejection samples, the vectorized Stable Diffusion image baseline (SD+LIVE) still underperforms VectorFusion with no rejection. VectorFusion also slightly benefits from a better initialization, using 4 rejection samples of the SD initialized image.

generations in Figure 11.3. While generated SVGs are less saturated, there are many artifacts characteristic of optimizing the CLIP loss.

11.8 Pixel Art Results

We ablate saturation penalties and different loss objectives in Figure 11.7. We use K=4 rejection samples for the initial Stable Diffusion raster image. Simply pixelating the best of K Stable Diffusion samples (SD+L1) is a straightforward way of generating pixel art, but results are often unrealistic and not as characteristic of pixel art. For example, pixelation results in blurry results since the SD sample does not use completely regular pixel grids.

Finetuning the result of pixelation with an SDS loss, and an additional L2 saturation penalty, improves OpenCLIP's R-Precision +10.2%. Direct CLIP optimization achieves high performance on CLIP R-Precision and CLIP Similarity, but we note that like our iconographic results, CLIP optimization often yields suboptimal samples.

Table 11.4: **Caption Consistency vs. K**. By increasing rejection sampling, we improve Stable Diffusion outputs. This improves both SD and SD+LIVE caption consistency. However, we find that VectorFusion matches Stable Diffusion consistency for K=4 and retains performance for K=20. This suggests that VectorFusion improves upon Stable Diffusion outputs and is robust to different initializations.

		Caption consistency			
		CLIP L/14		OpenCLIP H/14	
Method	K	R-Prec	Sim	R-Prec	Sim
SD (Raster)	1	67.2	23.0	69.5	26.7
SD+LIVE	1	57.0	21.7	59.4	24.8
SD+LIVE+SDS	1	78.1	24.1	78.1	29.3
SD (Raster)	4	81.3	24.1	80.5	28.2
SD+LIVE	4	69.5	22.9	65.6	27.6
SD+LIVE+SDS	4	78.9	**29.4**	81.3	24.5
SD (Raster)	20	**89.1**	25.4	**89.8**	30.1
SD+LIVE	20	71.5	23.6	73.1	28.5
SD+LIVE+SDS	20	79.8	25.0	80.6	**30.3**

11.9 Experimental hyperparameters

In this section, we document experimental settings to foster reproducibility. In general, we find that VectorFusion is robust to the choice of hyperparameters. Different settings can be used to control generation style.

11.9.1 Path initialization

Iconographic Art We initialize our closed Bézier paths with radius 20, random fill color, and opacity uniformly sampled between 0.7 and 1. Paths have 4 segments.

Pixel Art Pixel art is represented with a 32×32 grid of square polygons. The coordinates of square vertices are not optimized. We initialize each square in the grid with a random RGB fill color and an opacity uniformly sampled between 0.7 and 1.

Sketches Paths are open Bézier curves with 5 segments each. In contrast to iconography and pixel art, which have borderless paths, sketch paths have a fixed stroke width of 6 pixels and a fixed black color. Only control point coordinates can be optimized.

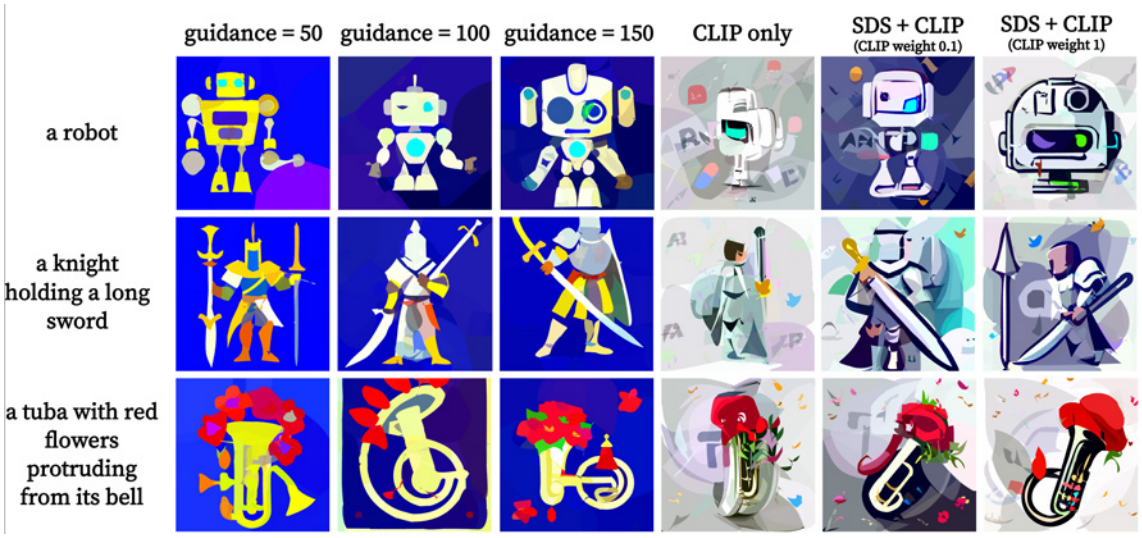


Figure 11.3: **SDS Guidance Scale and CLIP + SDS loss**. Our default guidance is 100. A hybrid CLIP + SDS loss produces samples that are more cohesive than using the CLIP loss only, and samples that are less saturated than using the SDS loss only.

Table 11.5: **SDS Guidance Scale.** We use SDS + rejection sampling (K=4) and LIVE with 64 paths. Our default guidance is 100, and a guidance scale of 50 leads to the best quantitative evaluation metrics.

	CLIP L/14		OpenCLIP H/14	
Guidance	R-Prec	Sim	R-Prec	Sim
5	21.1	18.0	14.8	16.5
50	**80.5**	25.0	**85.2**	**30.0**
100	78.9	**29.4**	81.3	24.5
150	75.8	24.5	81.3	29.7

Table 11.6: **SDS + CLIP.** SD + rejection (K=4), LIVE, and 64 paths.

	CLIP	CLIP L/14		OpenCLIP H/14	
Method	Weight	R-Prec	Sim	R-Prec	Sim
SDS	0	78.9	**29.4**	81.3	24.5
CLIP	1	70.1	23.3	74.0	28.4
SDS+CLIP	0.1	89.1	25.3	**90.6**	**32.0**
SDS+CLIP	1	**90.6**	25.3	87.5	31.8

Table 11.7: **Pixel Art.** We compare CLIP-based optimization and SDS-based optimizations. In addition, we ablate the saturation penalty, which makes pixel art more visually pleasing.

		Caption consistency			
	Sat	CLIP L/14		OpenCLIP H/14	
Method	Penalty	R-Prec	Sim	R-Prec	Sim
SDS (scratch)	0	57.9	21.3	43.8	23.1
SDS (scratch)	0.05	53.9	21.6	42.2	22.7
SD+L1	-	60.9	22.8	52.3	24.5
SD+L1+SDS	0	61.8	23.0	51.6	24.6
SD+L1+SDS	0.05	61.7	21.8	62.5	24.2
CLIP	-	80.5	26.6	73.4	27.5

11.9.2 Data Augmentation

We do not use data augmentations for SD + LIVE + SDS. We only use data augmentations for SDS trained from scratch, and CLIP baselines following [38]. For SDS trained from scratch, we apply a perspective and crop augmentation. Our rasterizer renders images at a 600x600 resolution, and with 0.7 probability, we apply a perspective transform with distortion scale 0.5. Then, we apply a random 512x512 crop. All other hyperparameters are default for Kornia [139].

11.9.3 Optimization

We optimize with a batch size of 1, allowing VectorFusion to run on a single low-end GPU with at least 10 GB of memory. VectorFusion uses the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.9$, $\epsilon = 10^{-6}$. On an NVIDIA RTX 2080ti GPU, VectorFusion (SD + LIVE + SDS) takes 25 minutes per SVG.

For sketches and iconography, the learning rate is linearly warmed from 0.02 to 0.2 over 500 steps, then decayed with a cosine schedule to 0.05 at the end of optimization for control point coordinates. Fill colors use a 20× lower learning rate than control points, and the solid background color has a 200× lower learning rate. A higher learning rate for coordiantes can allow more structural changes.

For pixel art, we use a lower learning rate, warming from 0.00001 to 0.0001 over 1000 iterations. We also add a weighted L2 saturation penalty on the image scaled between [-1, 1], with a loss weight of 0.05:

$$1/3 * \text{mean}(I_r^2 + I_b^2 + I_g^2)$$

Both the lower learning rate and the L2 penalty reduced oversaturation artifacts.

11.10 Perceptual Quality Metric

We measure aesthetic scores in Table 11.8 using the LAION aesthetics classifier, trained on frozen CLIP features of 250k images with human quality labels from 1-10 (Schuhmann et al 2022). VectorFusion with our latent SDS loss has the most aesthetic results, comparable to raster samples.

Table 11.8: VectorFusion from scratch produces the most aesthetic samples, even outperforming Stable Diffusion images.

Method	K	Aesthetic
CLIPDraw (scratch)	–	4.10 ± 0.81
Stable Diff (**raster**)	1	5.39 ± 0.93
+ rejection sampling	4	5.37 ± 0.84
SD init + LIVE	1	4.90 ± 0.91
+ rejection sampling	4	4.93 ± 0.89
VectorFusion (scratch)	–	**5.50 ± 0.79**
+ SD init + LIVE	1	5.45 ± 0.75
+ rejection sampling	4	5.35 ± 0.73