Title
Generative Models of Images and Neural Networks

Author
Peebles, William Smith

Publication Date
2023

Peer reviewed|Thesis/dissertation

Generative Models of Images and Neural Networks

By

William Smith Peebles

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alexei A. Efros, Chair
Professor Jitendra Malik
Professor Sergey Levine
Professor Antonio Torralba

Spring 2023

Generative Models of Images and Neural Networks

Abstract

Generative Models of Images and Neural Networks

by

William Smith Peebles

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Alexei A. Efros, Chair

Large-scale generative models have fueled recent progress in artificial intelligence. Armed with scaling laws that accurately predict model performance as invested compute increases, NLP has become the gold standard for all disciplines of AI. Given a new task, pre-trained generative models can either solve it zero-shot or be efficiently fine-tuned on a small amount of task-specific training examples. However, the widespread adoption of generative models has lagged in other domains—such as vision and meta-learning. In this thesis, we study ways to train improved, scalable generative models of two modalities—images and neural network *parameters*. We also examine how pre-trained generative models can be leveraged to tackle additional downstream tasks.

We begin by introducing a new, powerful class of generative models—Diffusion Transformers (DiTs). We show that transformers—with one small yet critically-important modification—retain their excellent scaling properties for diffusion-based image generation and outperform convolutional neural networks that have previously dominated the area. DiT outperforms all prior generative models on the class-conditional ImageNet generation benchmark.

Next, we introduce a novel framework for *learning to learn* based on building generative models of a new data source—neural network checkpoints. We create datasets containing hundreds of thousands of deep learning training runs and use it

to train generative models of neural network checkpoints. Given a starting parameter vector and a target loss, error or reward, loss-conditional diffusion models trained on this data can sample parameter updates that achieve a desired metric. We apply our framework to problems in vision and reinforcement learning.

Finally, we explore how pre-trained image-level generative models can be used to tackle downstream tasks in vision without requiring task-specific training data. We show that pre-trained GAN generators can be used to create an infinite data stream to train networks for the dense visual correspondence problem—without requiring *any* human-annotated supervision like keypoints. Networks trained on this completely GAN-generated data generalize zero-shot to real images, and they outperform previous self-supervised and keypoint-supervised approaches that train on real data.

To my parents and brother

Contents

List of Figures

List of Tables

Acknowledgments

I am extremely fortunate to have been able to dive into AI research at Berkeley for four years, especially during a time of such rapid progress. Nobody has made those four years more fulfilling and fun than my advisor, Alyosha Efros. Alyosha's enthusiasm is infectious; every three hour chat with him feels like 20 minutes and gets me excited to jump back into my research. I will always look back fondly on our debates over the power and limits of generative modeling (we'll see who ends up being right!). Alyosha has exquisite taste in Trader Joes snacks and paper introductions alike. He is selfless, prioritizing his students' well-being above all else. I'm so grateful for all of his guidance over the years. Actually, I think the existence of Alyosha disproves his claim that everything is "just nearest neighbor mumbo jumbo"; he is a one-of-a-kind advisor, and he definitely has no nearby neighbors!

I am so grateful to Antonio Torralba for taking me on as an undergrad at MIT and giving me complete leeway to pursue any research idea I was excited about. During my time at MIT, the highlight of every week was meeting with Antonio; whatever crazy idea I had in mind, Antonio would dive into the technical details to help make it happen and give encouraging advice.

I couldn't have asked for a better mentor at MIT than Jun-Yan Zhu who was Antonio's postdoc at the time. I was ecstatic when Antonio mentioned that Jun-Yan was starting a postdoc in his group and we could work on a project together. His Pix2Pix and CycleGAN papers inspired my early interest in generative models. As a mentor, Jun-Yan showed me the ropes of deep learning and shaped how I do research from scratch. He was accessible, kind, patient and gave fantastic research advice. I seriously hit the jackpot with him as a mentor and friend. A big thanks as well to David Bau and Sanja Fidler for being phenomenal mentors during my MIT days.

I did three internships during my time at Berkeley (two at Adobe Research and one at FAIR), and I had great mentors each time. Thanks to Eli Shechtman and Richard Zhang for being awesome collaborators at Adobe; they made the two internships exciting and rewarding despite the pandemic. Another big thanks to Saining Xie, my mentor at FAIR. Saining helped me develop more methodical research habits, and it was very exciting to work together on DiT, a project that was

at the exact intersection of our interests (generative modeling and scalable vision).

I'm very grateful to many Berkeley faculty members who have given me great advice (on research and life): Jitendra Malik, Sergey Levine, Pieter Abbeel, Trevor Darrell, Angjoo Kanazawa and Ren Ng. I'm also thankful to my fellow Berkeley PhD students and postdocs for great conversations and support over the years. To list just a few individuals: Anastasios Angelopoulos, Medhini Narasimhan, Hang Gao, Karttikeya Mangalam, Taesung Park, Allan Jabri, Boyi Li, Evonne Ng, Ajay Jain, Amir Bar, Jathushan Rajasegaran, Shiry Ginosar, Michael Janner, Aravind Srinivas, Rudy Corona, Olivia Watkins, Shubham Goel, Ruilong Li, Suzie Petryk, Assaf Shocher, Aleksander Holynski, Matt Tancik, Dave Epstein, Yu Sun, Jasmine Collins, Ashish Kumar, Yossi Gandelsman, Toru Lin, Vongani Maluleke, Sasha Sax, Georgios Pavlakos, Vickie Ye and Ethan Weber. A massive thanks as well to Jean Nguyen and Angie Abbatecola; I would not be able to graduate without their invaluable help navigating BAIR and Berkeley.

I'd like to give a special shout-out to Ilija Radosavovic, Tete Xiao and Tim Brooks for being amazing friends and fellow PhD students over the past few years. As the pandemic was clearing-up, I got dinner with Ilija and Tete almost everyday, and our research discussions were super valuable. Our dinner conversations and debates on scaling, meta-learning, FLOPs, etc. strongly influenced my later research. They are the best collaborators anybody could hope to work with. I can't wait to see what else they do at Berkeley. I'm so grateful that I got to be a PhD student at the same time with Tim. We are very aligned on research directions and interests; it was great to have a soundboard for any generative modeling idea that popped-up. He also throws great dumpling parties. I hope we get to work on more stuff together in the future.

A huge thanks to my parents for their unwavering love, dedication and support over the years. I give an especially big shout-out to my brother, John, for being an awesome collaborator and teaching me Java a decade ago. Finally, thanks to my incredible dogs for participating in some of my research demos, and for getting me through the pandemic with most of my sanity intact: Dakota (golden retriever), Theo (labradoodle) and Winston (boxer mix).

Chapter 1

Introduction

Deep generative models—trained on massive datasets with huge compute investments—have ushered in the era of scaling within artificial intelligence. This paradigm has led to considerable progress on many problems in AI, with the field of natural language processing (NLP) being the largest beneficiary. Generative models of natural language have demonstrated a number of appealing properties. They can be trained on any source of text data with or without additional human-level supervision. They can often solve new tasks zero-shot, without requiring additional task-specific training or fine-tuning [6]. Most importantly, they scale remarkably well with increasing training compute and data [6, 7]. Armed with these properties, deep generative models have fueled the recent meteoric progress in NLP.

Other domains, like vision, have benefited from generative models as well, but not nearly to the same extent as NLP. For example, image-level generative models are usually only used for graphics tasks; approaches to perception remain dominated by task-specific discriminative models supervised with human labels [8, 9]. Additionally, many classes of image-level generative models are not known to scale as effectively as language models [10], and they commonly rely on older, convolution-based architectures instead of modern backbones like transformers [11] which lie at the core of NLP's successes. Resolving these discrepancies is of great interest to the field of computer vision and all other disciplines of AI. In this thesis, we study these issues in the context of vision as well as meta-learning.

In Chapter 2, we introduce a new class of generative models for general continuous data. We study diffusion models [12, 13]—which have previously been shown to be highly effective generative models of images [14]—and modernize them by replacing their convolutional backbones with transformers. Done naively, vanilla transformers yield suboptimal performance as backbones of diffusion models. We introduce one small—yet critically-important—tweak to the standard transformer design that

enables Diffusion Transformers (DiTs) to become an extremely powerful and scalable class of generative models. We analyze DiT as an image-level generative model and show that it scales effectively over a large range of model sizes and token counts. DiT outperforms all prior generative models on the class-conditional ImageNet generation benchmark across several metrics.

In Chapter 3, we explore generative models as a novel framework for learning to learn. We collect a dataset of deep learning training runs and train generative models of neural network parameters. We demonstrate that generative models can act as *learned optimizers* by simply training loss-conditional diffusion models of neural network checkpoints. By querying for a small loss, our generative models can generate parameter updates that achieve the desired metric. Our approach overcomes many of the difficulties of previous meta-learning algorithms [15, 16]—it can optimize non-differentiable objectives and dispenses with unstable unrolled optimization techniques. Unlike iterative, gradient-based optimizers like SGD and Adam which cannot learn from their previous optimization histories, our generative models can optimize neural networks from random initialization with just one generated parameter update. We show that DiTs again scale effectively in both model size and data size in this new regime.

Finally, in Chapter 4 we discuss how image-level generative models can be used to solve a task in vision—dense visual correspondence—without requiring any task-specific training data or supervision. We show that pre-trained generative models already know how to perform certain tasks on *generated* images—like visual alignment—and that this knowledge can be distilled into a discriminative model to directly align *real* images. To this end, we use a pre-trained generative adversarial network (GAN) [17] to generate an infinite stream of paired (x, y) training data: (unaligned image, aligned image). We train a task-specific neural network on this paired data, simultaneously optimizing its parameters with the latent codes that control the alignment of the generated training data. At test time, our *GAN-Supervised* model automatically generalizes to align real images zero-shot. We show that our approach outperforms prior self-supervised and fully-supervised approaches alike which train models using real images and human-labeled annotations.

Chapter 2

Scalable Diffusion Models with Transformers

2.1 Introduction

Machine learning is experiencing a renaissance powered by transformers. Over the past five years, neural architectures for natural language processing [19, 20], vision [3] and several other domains have largely been subsumed by transformers [11]. Many classes of image-level generative models remain holdouts to the trend, though—while transformers see widespread use in autoregressive models [6, 21–23], they have seen less adoption in other generative modeling frameworks. For example, diffusion models have been at the forefront of recent advances in image-level generative models [4, 14]; yet, they all adopt a convolutional U-Net architecture as the de-facto choice of backbone.

The seminal work of Ho *et al.* [13] first introduced the U-Net backbone for diffusion models. Having initially seen success within pixel-level autoregressive models and conditional GANs [24], the U-Net was inherited from PixelCNN++ [25, 26] with a few changes. The model is convolutional, comprised primarily of ResNet [27] blocks. In contrast to the standard U-Net [28], additional spatial self-attention blocks, which are essential components in transformers, are interspersed at lower resolutions. Dhariwal and Nichol [4] ablated several architecture choices for the U-Net, such as the use of adaptive normalization layers [29] to inject conditional information and channel counts for convolutional layers. However, the high-level design of the U-Net from Ho *et al.* has largely remained intact.

This work originally appeared on arXiv [18].

Figure 2.1: **Diffusion models with transformer backbones achieve state-of-the-art image quality.** We show selected samples from two of our class-conditional DiT-XL/2 models trained on ImageNet at 512×512 and 256×256 resolution, respectively.

In this section, we aim to demystify the significance of architectural choices in diffusion models and offer empirical baselines for future generative modeling research. We show that the U-Net inductive bias is *not* crucial to the performance of diffusion models, and they can be readily replaced with standard designs such as transformers. As a result, diffusion models are well-poised to benefit from the recent trend of architecture unification—e.g., by inheriting best practices and training recipes from other domains, as well as retaining favorable properties like scalability, robustness and efficiency. A standardized architecture would also open up new possibilities for cross-domain research.

We focus on a new class of diffusion models based on transformers. We call them *Diffusion Transformers*, or DiTs for short. DiTs adhere to the best practices of Vision Transformers (ViTs) [3], which have been shown to scale more effectively for visual recognition than traditional convolutional networks (e.g., ResNet [27]).

More specifically, we study the scaling behavior of transformers with respect to *network complexity* vs. *sample quality*. We show that by constructing and benchmarking the DiT design space under the *Latent Diffusion Models* (LDMs) [30] framework, where diffusion models are trained within a VAE's latent space, we can successfully replace the U-Net backbone with a transformer. We further show that DiTs are

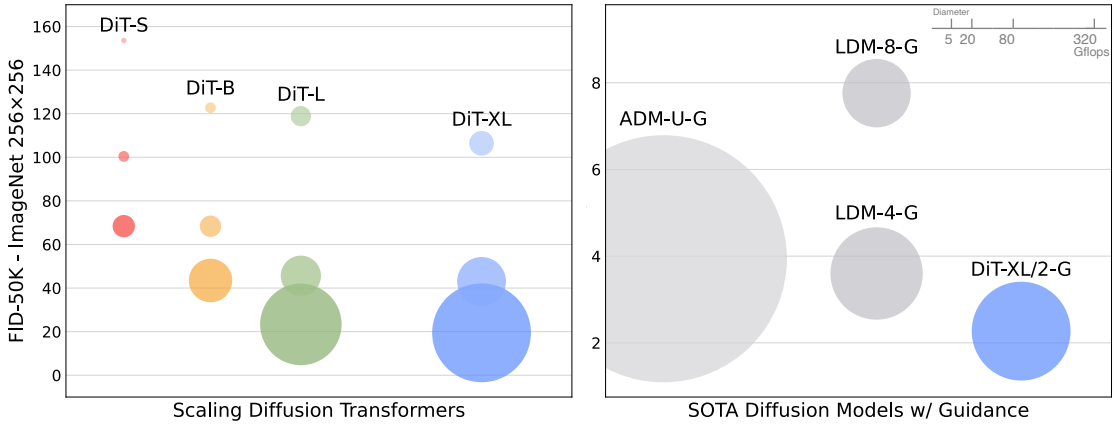

Figure 2.2: **ImageNet generation with Diffusion Transformers (DiTs).** Bubble area indicates the flops of the diffusion model. *Left:* FID-50K (lower is better) of our DiT models at 400K training iterations. Performance steadily improves in FID as model flops increase. *Right:* Our best model, DiT-XL/2, is compute-efficient and outperforms all prior U-Net-based diffusion models, like ADM and LDM.

scalable architectures for diffusion models: there is a strong correlation between the network complexity (measured by Gflops) *vs.* sample quality (measured by FID). By simply scaling-up DiT and training an LDM with a high-capacity backbone (118.6 Gflops), we are able to achieve a state-of-the-art result of 2.27 FID on the class-conditional 256×256 ImageNet generation benchmark.

2.2 Related Work

Transformers. Transformers [11] have replaced domain-specific architectures across language, vision [3], reinforcement learning [31, 32] and meta-learning [33]. They have shown remarkable scaling properties under increasing model size, training compute and data in the language domain [7], as generic autoregressive models [34] and as ViTs [35]. Beyond language, transformers have been trained to autoregressively predict pixels [22, 36, 37]. They have also been trained on discrete codebooks [38] as both autoregressive models [23, 39] and masked generative models [40, 41]; the former has shown excellent scaling behavior up to 20B parameters [42]. Finally, transformers have been explored in DDPMs to synthesize non-spatial data; e.g., to generate CLIP image embeddings in DALL·E 2 [14, 43]. We study the scaling properties of transformers when used as the backbone of diffusion models of images.

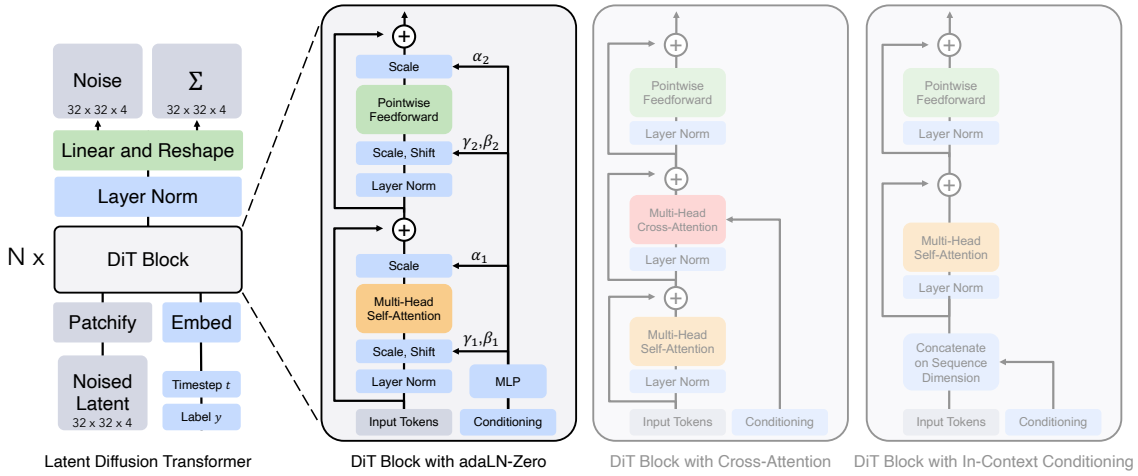

Figure 2.3: **The Diffusion Transformer (DiT) architecture.** *Left:* We train conditional latent DiT models. The input latent is decomposed into patches and processed by several DiT blocks. *Right:* Details of our DiT blocks. We experiment with variants of standard transformer blocks that incorporate conditioning via adaptive layer norm, cross-attention and extra input tokens. Adaptive layer norm works best.

Denoising diffusion probabilistic models (DDPMs). Diffusion [12, 13] and score-based generative models [44, 45] have been particularly successful as generative models of images [10, 14, 30, 46], in many cases outperforming generative adversarial networks (GANs) [17] which had previously been state-of-the-art. Improvements in DDPMs over the past two years have largely been driven by improved sampling techniques [13, 47, 48], most notably classifier-free guidance [49], reformulating diffusion models to predict noise instead of pixels [13] and using cascaded DDPM pipelines where low-resolution base diffusion models are trained in parallel with upsamplers [4, 50]. For all the diffusion models listed above, convolutional U-Nets [28] are the de-facto choice of backbone architecture. Concurrent work [51] introduced a novel, efficient architecture based on attention for DDPMs; we explore pure transformers.

Architecture complexity. When evaluating architecture complexity in the image generation literature, it is fairly common practice to use parameter counts. In general, parameter counts can be poor proxies for the complexity of image models since they do not account for, e.g., image resolution which significantly impacts performance [52, 53]. Instead, much of the model complexity analysis in this section is through the lens of theoretical Gflops. This brings us in-line with the architecture design literature where Gflops are widely-used to gauge complexity. In practice,

the golden complexity metric is still up for debate as it frequently depends on particular application scenarios. Nichol and Dhariwal's seminal work improving diffusion models [4,54] is most related to us—there, they analyzed the scalability and Gflop properties of the U-Net architecture class. Here, we focus on the transformer class.

2.3 Diffusion Transformers

2.3.1 Preliminaries

Diffusion formulation. Before introducing our architecture, we briefly review some basic concepts needed to understand diffusion models (DDPMs) [12,13]. Gaussian diffusion models assume a forward noising process which gradually applies noise to real data x_0: $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I})$, where constants $\bar{\alpha}_t$ are hyperparameters. By applying the reparameterization trick, we can sample $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \mathbf{I})$.

Diffusion models are trained to learn the reverse process that inverts forward process corruptions: $p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t), \Sigma_\theta(x_t))$, where neural networks are used to predict the statistics of p_θ. The reverse process model is trained with the variational lower bound [55] of the log-likelihood of x_0, which reduces to $\mathcal{L}(\theta) = -p(x_0|x_1) + \sum_t \mathcal{D}_{KL}(q^*(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$, excluding an additional term irrelevant for training. Since both q^* and p_θ are Gaussian, $\mathcal{D}_{KL}$ can be evaluated with the mean and covariance of the two distributions. By reparameterizing μ_θ as a noise prediction network ϵ_θ, the model can be trained using simple mean-squared error between the predicted noise $\epsilon_\theta(x_t)$ and the ground truth sampled Gaussian noise ϵ_t: $\mathcal{L}_{simple}(\theta) = ||\epsilon_\theta(x_t) - \epsilon_t||_2^2$. But, in order to train diffusion models with a learned reverse process covariance Σ_θ, the full $\mathcal{D}_{KL}$ term needs to be optimized. We follow Nichol and Dhariwal's approach [54]: train ϵ_θ with $\mathcal{L}_{simple}$, and train Σ_θ with the full $\mathcal{L}$. Once p_θ is trained, new images can be sampled by initializing $x_{t_{\max}} \sim \mathcal{N}(0, \mathbf{I})$ and sampling $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$ via the reparameterization trick.

Classifier-free guidance. Conditional diffusion models take extra information as input, such as a class label c. In this case, the reverse process becomes $p_\theta(x_{t-1}|x_t, c)$, where ϵ_θ and Σ_θ are conditioned on c. In this setting, *classifier-free guidance* can be used to encourage the sampling procedure to find x such that $\log p(c|x)$ is high [49]. By Bayes Rule, $\log p(c|x) \propto \log p(x|c) - \log p(x)$, and hence $\nabla_x \log p(c|x) \propto \nabla_x \log p(x|c) - \nabla_x \log p(x)$. By interpreting the output of diffusion models as the score function, the DDPM sampling procedure can be guided to sample x with high

$p(x|c)$ by: $\hat{\epsilon}_\theta(x_t, c) = \epsilon_\theta(x_t, \emptyset) + s \cdot \nabla_x \log p(x|c) \propto \epsilon_\theta(x_t, \emptyset) + s \cdot (\epsilon_\theta(x_t, c) - \epsilon_\theta(x_t, \emptyset))$, where $s > 1$ indicates the scale of the guidance (note that $s = 1$ recovers standard sampling). Evaluating the diffusion model with $c = \emptyset$ is done by randomly dropping out c during training and replacing it with a learned "null" embedding $\emptyset$. Classifier-free guidance is widely-known to yield significantly improved samples over generic sampling techniques [14, 46, 49], and the trend holds for our DiT models.

Latent diffusion models. Training diffusion models directly in high-resolution pixel space can be computationally prohibitive. *Latent diffusion models* (LDMs) [30] tackle this issue with a two-stage approach: (1) learn an autoencoder that compresses images into smaller spatial representations with a learned encoder E; (2) train a diffusion model of representations $z = E(x)$ instead of a diffusion model of images x (E is frozen). New images can then be generated by sampling a representation z from the diffusion model and subsequently decoding it to an image with the learned decoder $x = D(z)$.

As shown in Figure 2.2, LDMs achieve good performance while using a fraction of the Gflops of pixel space diffusion models like ADM. Since we are concerned with compute efficiency, this makes them an appealing starting point for architecture exploration. In this work, we apply DiTs to latent space, although they could be applied to pixel space without modification as well. This makes our image generation pipeline a hybrid-based approach; we use off-the-shelf convolutional VAEs and transformer-based DDPMs.

2.3.2 Diffusion Transformer Design Space

We introduce Diffusion Transformers (DiTs), a new architecture for diffusion models. We aim to be as faithful to the standard transformer architecture as possible to retain its scaling properties. Since our focus is training DDPMs of images (specifically, spatial representations of images), DiT is based on the Vision Transformer (ViT) architecture which operates on sequences of patches [3]. DiT retains many of the best practices of ViTs. Figure 2.3 shows an overview of the complete DiT architecture. In this section, we describe the forward pass of DiT, as well as the components of the design space of the DiT class.

Patchify. The input to DiT is a spatial representation z (for $256 \times 256 \times 3$ images, z has shape $32 \times 32 \times 4$). The first layer of DiT is "patchify," which converts the spatial input into a sequence of T tokens, each of dimension d, by linearly embedding each patch in the input. Following patchify, we apply standard ViT frequency-based positional embeddings (the sine-cosine version) to all input tokens. The number

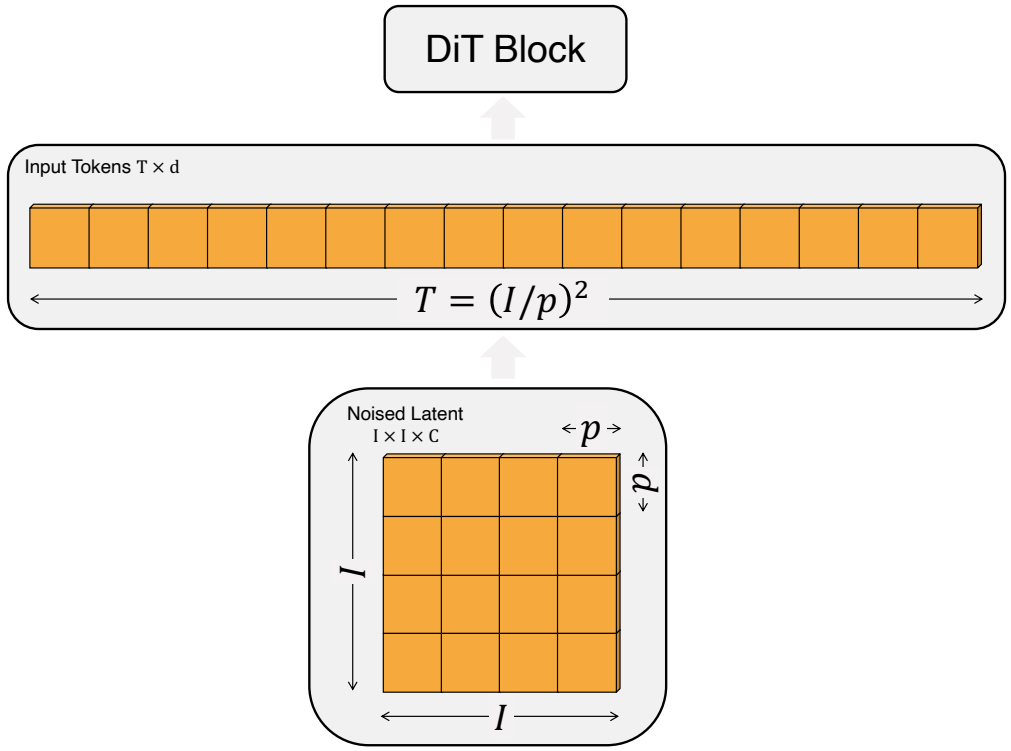

Figure 2.4: **Input specifications for DiT.** Given patch size $p \times p$, a spatial representation (the noised latent from the VAE) of shape $I \times I \times C$ is "patchified" into a sequence of length $T = (I/p)^2$ with hidden dimension d. A smaller patch size p results in a longer sequence length and thus more Gflops.

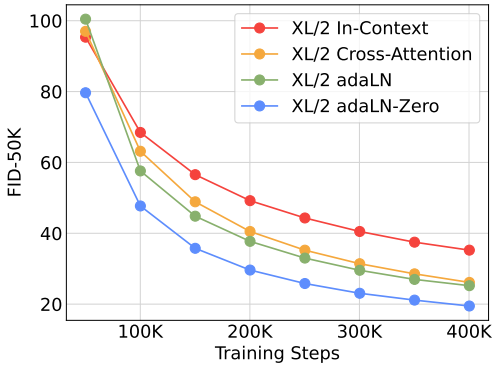

Figure 2.5: **Comparing different conditioning strategies.** adaLN-Zero outperforms cross-attention and in-context conditioning at all stages of training.

of tokens T created by patchify is determined by the patch size hyperparameter p. As shown in Figure 2.4, halving p will quadruple T, and thus *at least* quadruple total transformer Gflops. Although it has a significant impact on Gflops, note that changing p has no meaningful impact on downstream parameter counts.

We add $p = 2, 4, 8$ to the DiT design space.

DiT block design. Following patchify, the input tokens are processed by a sequence of transformer blocks. In addition to noised image inputs, diffusion models sometimes process additional conditional information such as noise timesteps t, class labels c, natural language, etc. We explore four variants of transformer blocks that process conditional inputs differently. The designs introduce small, but important, modifications to the standard ViT block design. The designs of all blocks are shown in Figure 2.3.

- *In-context conditioning.* We simply append the vector embeddings of t and c as two additional tokens in the input sequence, treating them no differently from the image tokens. This is similar to `cls` tokens in ViTs, and it allows us to use standard ViT blocks without modification. After the final block, we remove the conditioning tokens from the sequence. This approach introduces negligible new Gflops to the model.

- *Cross-attention block.* We concatenate the embeddings of t and c into a length-two sequence, separate from the image token sequence. The transformer block is modified to include an additional multi-head cross-attention layer following the multi-head self-attention block, similar to the original design from Vaswani *et al.* [11], and also similar to the one used by LDM for conditioning on class

Model	Layers N	Hidden size d	Heads	Gflops $_{(I=32,\ p=4)}$
DiT-S	12	384	6	1.4
DiT-B	12	768	12	5.6
DiT-L	24	1024	16	19.7
DiT-XL	28	1152	16	29.1

Table 2.1: **Details of DiT models.** We follow ViT [3] model configurations for the Small (S), Base (B) and Large (L) variants; we also introduce an XLarge (XL) config as our largest model.

labels. Cross-attention adds the most Gflops to the model, roughly a 15% overhead.

– *Adaptive layer norm (adaLN) block.* Following the widespread usage of adaptive normalization layers [29] in GANs [56, 57] and diffusion models with U-Net backbones [4], we explore replacing standard layer norm layers in transformer blocks with adaptive layer norm (adaLN). Rather than directly learn dimension-wise scale and shift parameters γ and β, we regress them from the sum of the embedding vectors of t and c. Of the three block designs we explore, adaLN adds the least Gflops and is thus the most compute-efficient. It is also the only conditioning mechanism that is restricted to apply the *same function* to all tokens.

– *adaLN-Zero block.* Prior work on ResNets has found that initializing each residual block as the identity function is beneficial. For example, Goyal *et al.* found that zero-initializing the final batch norm scale factor γ in each block accelerates large-scale training in the supervised learning setting [58]. Diffusion U-Net models use a similar initialization strategy, zero-initializing the final convolutional layer in each block prior to any residual connections. We explore a modification of the adaLN DiT block which does the same. In addition to regressing γ and β, we also regress dimension-wise scaling parameters α that are applied immediately prior to any residual connections within the DiT block. We initialize the MLP to output the zero-vector for all α; this initializes the full DiT block as the identity function. As with the vanilla adaLN block, adaLN-Zero adds negligible Gflops to the model.

We include the in-context, cross-attention, adaptive layer norm and adaLN-Zero blocks in the DiT design space.

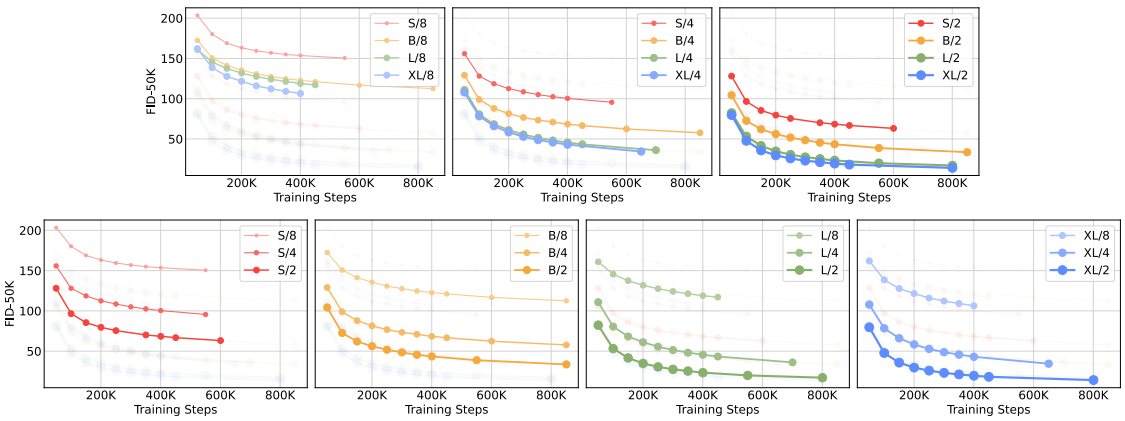

Figure 2.6: **Scaling the DiT model improves FID at all stages of training.** We show FID-50K over training iterations for 12 of our DiT models. *Top row:* We compare FID holding patch size constant. *Bottom row:* We compare FID holding model size constant. Scaling the transformer backbone yields better generative models across all model sizes and patch sizes.

Model size. We apply a sequence of N DiT blocks, each operating at the hidden dimension size d. Following ViT, we use standard transformer configs that jointly scale N, d and attention heads [3,35]. Specifically, we use four configs: DiT-S, DiT-B, DiT-L and DiT-XL. They cover a wide range of model sizes and flop allocations, from 0.3 to 118.6 Gflops, allowing us to gauge scaling performance. Table 2.1 gives details of the configs.

We add B, S, L and XL configs to the DiT design space.

Transformer decoder. After the final DiT block, we need to decode our sequence of image tokens into an output noise prediction and an output diagonal covariance prediction. Both of these outputs have shape equal to the original spatial input. We use a standard linear decoder to do this; we apply the final layer norm (adaptive if using adaLN) and linearly decode each token into a $p \times p \times 2C$ tensor, where C is the number of channels in the spatial input to DiT. Finally, we rearrange the decoded tokens into their original spatial layout to get the predicted noise and covariance.

The complete DiT design space we explore is patch size, transformer block architecture and model size.

2.4 Experimental Setup

We explore the DiT design space and study the scaling properties of our model class. Our models are named according to their configs and latent patch sizes p; for example, DiT-XL/2 refers to the XLarge config and $p = 2$.

Training. We train class-conditional latent DiT models at 256×256 and 512×512 image resolution on the ImageNet dataset [8], a highly-competitive generative modeling benchmark. We initialize the final linear layer with zeros and otherwise use standard weight initialization techniques from ViT. We train all models with AdamW [59, 60]. We use a constant learning rate of 1×10^{-4}, no weight decay and a batch size of 256. The only data augmentation we use is horizontal flips. Unlike much prior work with ViTs [61, 62], we did not find learning rate warmup nor regularization necessary to train DiTs to high performance. Even without these techniques, training was highly stable across all model configs and we did not observe any loss spikes commonly seen when training transformers. Following common practice in the generative modeling literature, we maintain an exponential moving average (EMA) of DiT weights over training with a decay of 0.9999. All results reported use the EMA model. We use identical training hyperparameters across all DiT model sizes and patch sizes. Our training hyperparameters are almost entirely retained from ADM. *We did not tune learning rates, decay/warm-up schedules, Adam β_1/β_2 or weight decays.*

Diffusion. We use an off-the-shelf pre-trained variational autoencoder (VAE) model [55] from Stable Diffusion [30]. The VAE encoder has a downsample factor of 8—given an RGB image x with shape $256 \times 256 \times 3$, $z = E(x)$ has shape $32 \times 32 \times 4$. Across all experiments in this section, our diffusion models operate in this $\mathcal{Z}$-space. After sampling a new latent from our diffusion model, we decode it to pixels using the VAE decoder $x = D(z)$. We retain diffusion hyperparameters from ADM [4]; specifically, we use a $t_{\max} = 1000$ linear variance schedule ranging from 1×10^{-4} to 2×10^{-2}, ADM's parameterization of the covariance Σ_θ and their method for embedding input timesteps and labels.

Evaluation metrics. We measure scaling performance with Fréchet Inception Distance (FID) [63], the standard metric for evaluating generative models of images. We follow convention when comparing against prior works and report FID-50K using 250 DDPM sampling steps. FID is known to be sensitive to small implementation details [64]; to ensure accurate comparisons, all values reported are obtained by exporting samples and using ADM's TensorFlow evaluation suite [4]. FID numbers reported in this section do *not* use classifier-free guidance except where otherwise stated.

We additionally report Inception Score [65], sFID [66] and Precision/Recall [67] as secondary metrics.

Compute. We implement all models in JAX [68] and train them using TPU-v3 pods. DiT-XL/2, our most compute-intensive model, trains at roughly 5.7 iterations/second on a TPU v3-256 pod with a global batch size of 256.

2.5 Experiments

DiT block design. We train four of our highest Gflop DiT-XL/2 models, each using a different block design—in-context (119.4 Gflops), cross-attention (137.6 Gflops), adaptive layer norm (adaLN, 118.6 Gflops) or adaLN-zero (118.6 Gflops). We measure FID over the course of training. Figure 2.5 shows the results. The adaLN-Zero block yields lower FID than both cross-attention and in-context conditioning while being the most compute-efficient. At 400K training iterations, the FID achieved with the adaLN-Zero model is nearly half that of the in-context model, demonstrating that the conditioning mechanism critically affects model quality. Initialization is also important—adaLN-Zero, which initializes each DiT block as the identity function, significantly outperforms vanilla adaLN. *For the rest of this section, all models will use adaLN-Zero DiT blocks.*

Scaling model size and patch size. We train 12 DiT models, sweeping over model configs (S, B, L, XL) and patch sizes (8, 4, 2). Note that DiT-L and DiT-XL are significantly closer to each other in terms of relative Gflops than other configs. Figure 2.2 (left) gives an overview of the Gflops of each model and their FID at 400K training iterations. In all cases, we find that increasing model size and decreasing patch size yields considerably improved diffusion models.

Figure 3.7 (top) demonstrates how FID changes as model size is increased and patch size is held constant. Across all four configs, significant improvements in FID are obtained over all stages of training by making the transformer deeper and wider. Similarly, Figure 3.7 (bottom) shows FID as patch size is decreased and model size is held constant. We again observe considerable FID improvements throughout training by simply scaling the number of tokens processed by DiT, holding parameters approximately fixed.

DiT Gflops are critical to improving performance. The results of Figure 3.7 suggest that parameter counts do not uniquely determine the quality of a DiT model. As model size is held constant and patch size is decreased, the transformer's total parameters are effectively unchanged (actually, total parameters slightly *decrease*),

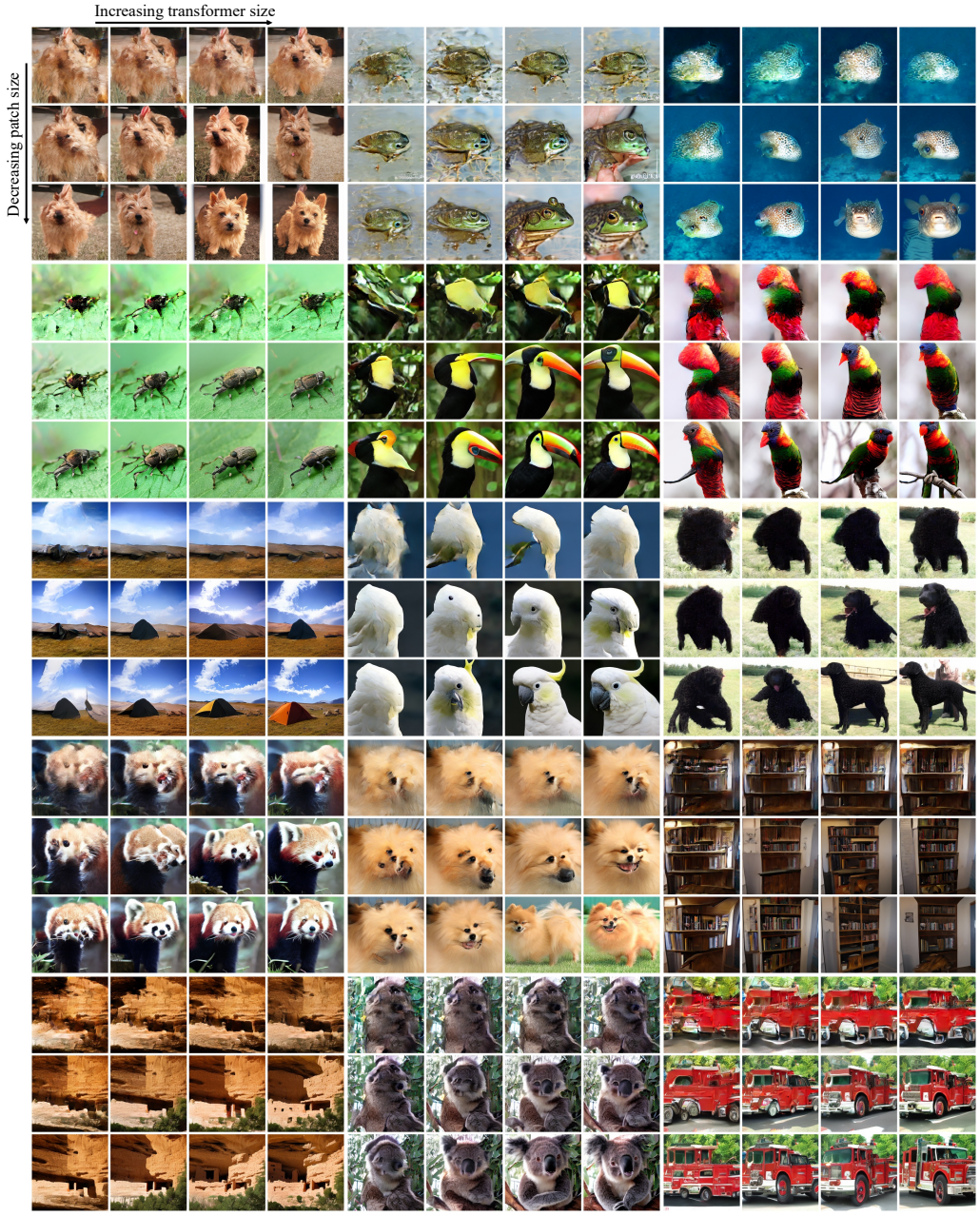

Figure 2.7: **Increasing transformer forward pass Gflops increases sample quality.** *Best viewed zoomed-in.* We sample from all 12 of our DiT models after 400K training steps using the same input latent noise and class label. Increasing the Gflops in the model—either by increasing transformer depth/width or increasing the number of input tokens—yields significant improvements in visual fidelity.

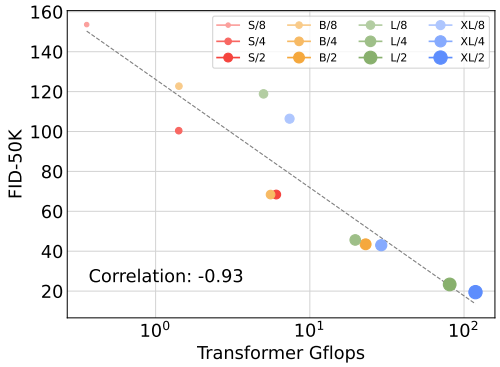

Figure 2.8: **Transformer Gflops are strongly correlated with FID.** We plot the Gflops of each of our DiT models and each model's FID-50K after 400K training steps.

and only Gflops are increased. These results indicate that scaling model *Gflops* is actually the key to improved performance. To investigate this further, we plot the FID-50K at 400K training steps against model Gflops in Figure 2.8. The results demonstrate that different DiT configs obtain similar FID values when their total Gflops are similar (e.g., DiT-S/2 and DiT-B/4). We find a strong negative correlation between model Gflops and FID-50K, suggesting that additional model compute is the critical ingredient for improved DiT models. In Figure 2.9, we find that this trend holds for other metrics such as Inception Score.

Larger DiT models are more compute-efficient. In Figure 2.10, we plot FID as a function of total training compute for all DiT models. We estimate training compute as model Gflops · batch size · training steps · 3, where the factor of 3 roughly approximates the backwards pass as being twice as compute-heavy as the forward pass. We find that small DiT models, even when trained longer, eventually become compute-inefficient relative to larger DiT models trained for fewer steps. Similarly, we find that models that are identical except for patch size have different performance profiles even when controlling for training Gflops. For example, XL/4 is outperformed by XL/2 after roughly 10^{10} Gflops.

Visualizing scaling. We visualize the effect of scaling on sample quality in Figure 2.7. At 400K training steps, we sample an image from each of our 12 DiT models using *identical* starting noise $x_{t_{\max}}$, sampling noise and class labels. This lets us visually interpret how scaling affects DiT sample quality. Indeed, scaling both model size and the number of tokens yields notable improvements in visual quality.

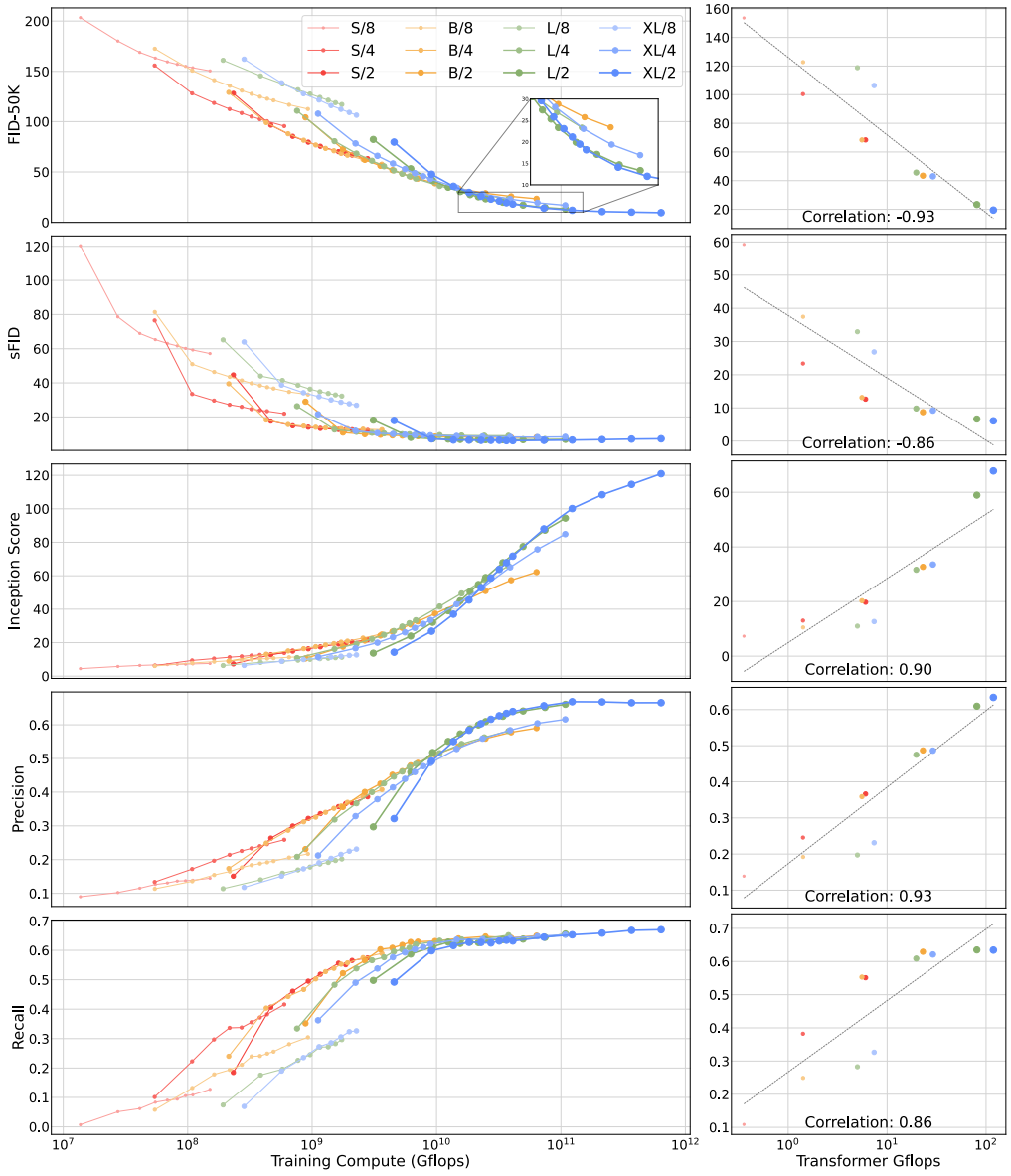

Figure 2.9: **DiT scaling behavior on several generative modeling metrics.** *Left:* We plot model performance as a function of total training compute for FID, sFID, Inception Score, Precision and Recall. *Right:* We plot model performance at 400K training steps for all 12 DiT variants against transformer Gflops, finding strong correlations across metrics. All values were computed using the ft-MSE VAE decoder.

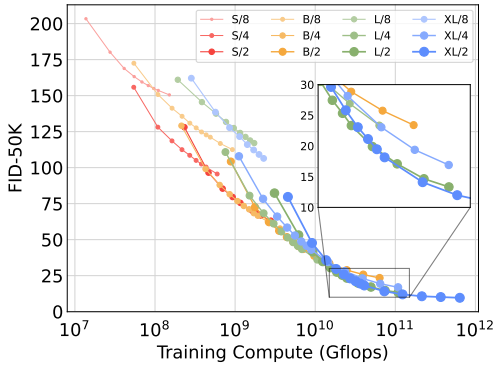

Figure 2.10: **Larger DiT models use large compute more efficiently.** We plot FID as a function of total training compute.

2.5.1 State-of-the-Art Diffusion Models

256×256 ImageNet. Following our scaling analysis, we continue training our highest Gflop model, DiT-XL/2, for 7M steps. We show samples from the model in Figures 4.1, and we compare against state-of-the-art class-conditional generative models. We report results in Table 2.2. When using classifier-free guidance, DiT-XL/2 outperforms all prior diffusion models, decreasing the previous best FID-50K of 3.60 achieved by LDM to 2.27. Figure 2.2 (right) shows that DiT-XL/2 (118.6 Gflops) is compute-efficient relative to latent space U-Net models like LDM-4 (103.6 Gflops) and substantially more efficient than pixel space U-Net models such as ADM (1120 Gflops) or ADM-U (742 Gflops). Our method achieves the lowest FID of all prior generative models, including the previous state-of-the-art StyleGAN-XL [69]. Finally, we also observe that DiT-XL/2 achieves higher recall values at all tested classifier-free guidance scales compared to LDM-4 and LDM-8. When trained for only 2.35M steps (similar to ADM), XL/2 still outperforms all prior diffusion models with an FID of 2.55.

512×512 ImageNet. We train a new DiT-XL/2 model on ImageNet at 512×512 resolution for 3M iterations with identical hyperparameters as the 256×256 model. With a patch size of 2, this XL/2 model processes a total of 1024 tokens after patchifying the $64 \times 64 \times 4$ input latent (524.6 Gflops). Table 2.3 shows comparisons against state-of-the-art methods. XL/2 again outperforms all prior diffusion models at this resolution, improving the previous best FID of 3.85 achieved by ADM to 3.04. Even with the increased number of tokens, XL/2 remains compute-efficient. For example, ADM uses 1983 Gflops and ADM-U uses 2813 Gflops; XL/2 uses 524.6 Gflops. We show samples from the high-resolution XL/2 model in Figure 4.1 and

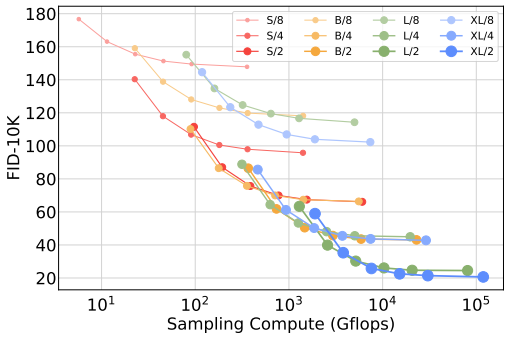

Figure 2.11: **Scaling-up *sampling* compute does not compensate for a lack of *model* compute.** For each of our DiT models trained for 400K iterations, we compute FID-10K using [16, 32, 64, 128, 256, 1000] sampling steps. For each number of steps, we plot the FID as well as the Gflops used to sample each image. Small models cannot close the performance gap with our large models, even if they sample with more test-time Gflops than the large models.

Figures 2.12 through 2.15.

2.5.2 Scaling Model vs. Sampling Compute

Diffusion models are unique in that they can use additional compute after training by increasing the number of sampling steps when generating an image. Given the impact of model Gflops on sample quality, in this section we study if smaller-*model compute* DiTs can outperform larger ones by using more *sampling compute*. We compute FID for all 12 of our DiT models after 400K training steps, using [16, 32, 64, 128, 256, 1000] sampling steps per-image. The main results are in Figure 2.11. Consider DiT-L/2 using 1000 sampling steps versus DiT-XL/2 using 128 steps. In this case, L/2 uses 80.7 Tflops to sample each image; XL/2 uses 5× less compute—15.2 Tflops—to sample each image. Nonetheless, XL/2 has the better FID-10K (23.7 vs 25.9). In general, scaling-up sampling compute *cannot* compensate for a lack of model compute.

2.6 Conclusion

We introduce Diffusion Transformers (DiTs), a simple transformer-based backbone for diffusion models that outperforms prior U-Net models and inherits the excellent scaling properties of the transformer model class. Given our promising scaling results, future work should continue to scale DiTs to larger models and token counts. DiT

Figure 2.12: **DiT-XL/2 samples.**
Classifier-free guidance scale = 4.0
Label = “arctic wolf" (270)
Resolution = 512 × 512

Figure 2.13: **DiT-XL/2 samples.**
Classifier-free guidance scale = 4.0
Label = “sulphur-crested cockatoo" (89)
Resolution = 512 × 512

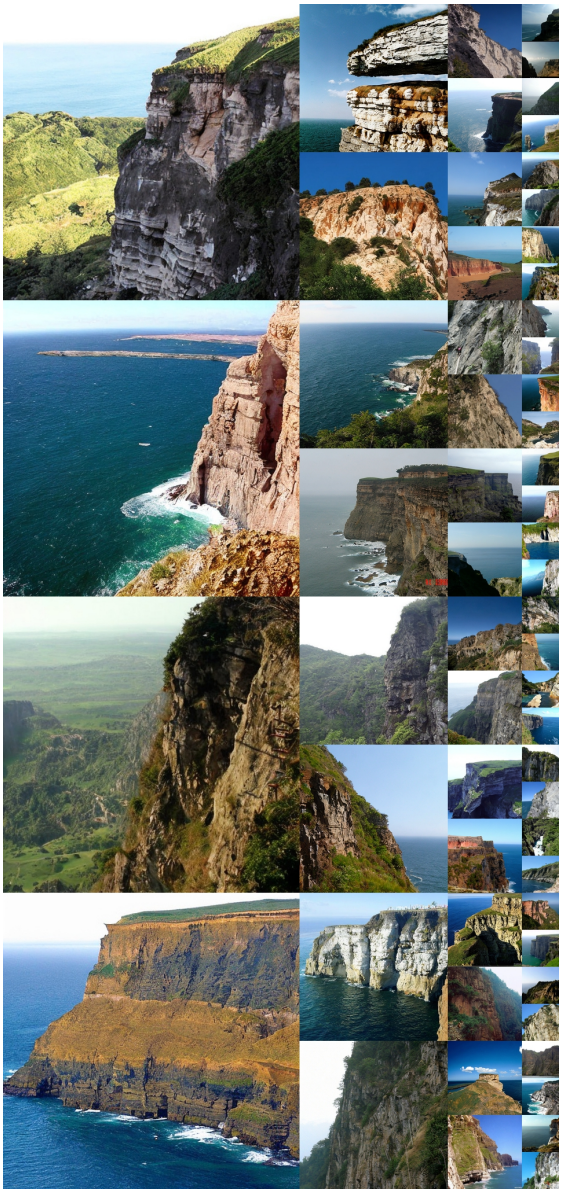

Figure 2.14: **DiT-XL/2 samples.**
Classifier-free guidance scale = 4.0
Label = “cliff drop-off" (972)
Resolution = 512 × 512

Figure 2.15: **DiT-XL/2 samples.**
Classifier-free guidance scale = 4.0
Label = “balloon" (417)
Resolution = 512 × 512

could also be explored as a drop-in backbone for text-to-image models like DALL·E 2 and Stable Diffusion.

Class-Conditional ImageNet 256×256					
Model	FID↓	sFID↓	IS↑	Precision↑	Recall↑
BigGAN-deep [56]	6.95	7.36	171.4	0.87	0.28
StyleGAN-XL [69]	2.30	4.02	265.12	0.78	0.53
ADM [4]	10.94	6.02	100.98	0.69	0.63
ADM-U	7.49	5.13	127.49	0.72	0.63
ADM-G	4.59	5.25	186.70	0.82	0.52
ADM-G, ADM-U	3.94	6.14	215.84	0.83	0.53
CDM [50]	4.88	-	158.71	-	-
LDM-8 [30]	15.51	-	79.03	0.65	0.63
LDM-8-G	7.76	-	209.52	0.84	0.35
LDM-4	10.56	-	103.49	0.71	0.62
LDM-4-G (cfg=1.25)	3.95	-	178.22	0.81	0.55
LDM-4-G (cfg=1.50)	3.60	-	247.67	**0.87**	0.48
DiT-XL/2	9.62	6.85	121.50	0.67	**0.67**
DiT-XL/2-G (cfg=1.25)	3.22	5.28	201.77	0.76	0.62
DiT-XL/2-G (cfg=1.50)	**2.27**	**4.60**	**278.24**	0.83	0.57

Table 2.2: **Benchmarking class-conditional image generation on ImageNet 256×256.** DiT-XL/2 achieves state-of-the-art FID.

Class-Conditional ImageNet 512×512					
Model	FID↓	sFID↓	IS↑	Precision↑	Recall↑
BigGAN-deep [56]	8.43	8.13	177.90	0.88	0.29
StyleGAN-XL [69]	2.41	4.06	267.75	0.77	0.52
ADM [4]	23.24	10.19	58.06	0.73	0.60
ADM-U	9.96	5.62	121.78	0.75	**0.64**
ADM-G	7.72	6.57	172.71	**0.87**	0.42
ADM-G, ADM-U	3.85	5.86	221.72	0.84	0.53
DiT-XL/2	12.03	7.12	105.25	0.75	**0.64**
DiT-XL/2-G (cfg=1.25)	4.64	5.77	174.77	0.81	0.57
DiT-XL/2-G (cfg=1.50)	**3.04**	**5.02**	**240.82**	0.84	0.54

Table 2.3: **Benchmarking class-conditional image generation on ImageNet 512×512.** Note that prior work [4] measures Precision and Recall using 1000 real samples for 512 × 512 resolution; for consistency, we do the same.

Chapter 3

Learning to Learn with Generative Models of Neural Network Checkpoints

3.1 Introduction

Gradient-based optimization is the fuel of modern deep learning. Techniques of this class, such as SGD [70] and Adam [60], are easy to implement, scale reasonably well and converge to surprisingly good solutions—even in high-dimensional, non-convex neural network loss landscapes. Over the past decade, they have enabled impressive results in computer vision [8, 9], natural language processing [11, 19] and audio generation [71].

While these *manual* optimization techniques have led to large advances, they suffer from an important limitation: they are unable to improve from past experience. For example, SGD will not converge any faster when used to optimize the same neural network architecture from the same initialization the 100th time versus the first time. *Learned* optimizers capable of leveraging their past experiences have the potential to overcome this limitation and may accelerate future progress in deep learning.

Of course, the concept of learning improved optimizers is not new and dates back to the 1980s, if not earlier, following early work from Schmidhuber [72] and Bengio *et. al* [73]. In recent years, significant effort has been spent on designing algorithms that

This work originally appeared on arXiv [33].

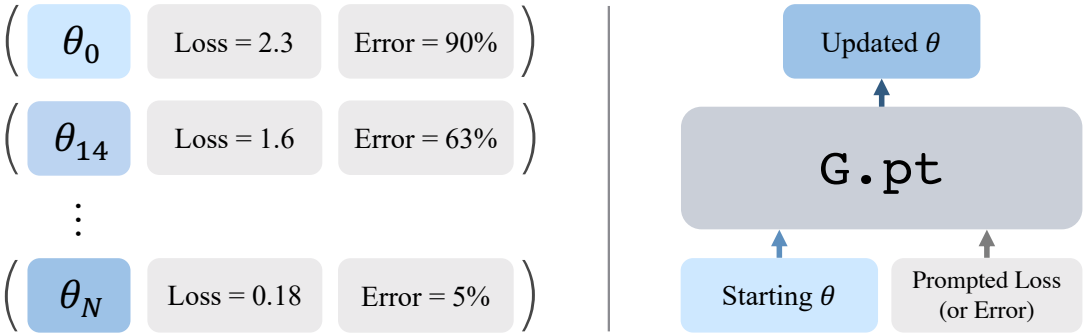

Figure 3.1: **Generative pre-training from checkpoints.** *Left:* We build a dataset of neural network checkpoints from many training runs. Each checkpoint includes the neural network's parameters and relevant metadata (test losses and test errors for supervised learning tasks, returns for RL tasks). *Right:* `G.pt`, a generative model of checkpoints. `G.pt` takes a parameter vector and a loss/error/return prompt as input and predicts the distribution over updated parameters that achieve the prompt.

learn via nested meta-optimization, where the inner loop optimizes the task-level objective and the outer loop learns the optimizer [15, 16, 74]. In some instances, these approaches outperform manual optimizers. However, they are challenging to train in practice due to a reliance on unrolled optimization and reinforcement learning.

Taking a modern deep learning perspective suggests a simple, scalable and data-driven approach to this problem. Over the past decade, our community has trained a massive number of checkpoints. These checkpoints contain a wealth of information: diverse parameter configurations and rich metrics such as test losses, classification errors and RL returns that describe the quality of the checkpoint. Instead of leveraging large-scale datasets of images or text, we propose learning from large-scale datasets of *checkpoints* recorded over the course of many training runs.

To this end, we create a dataset of neural network checkpoints (Figure 3.1, left). Our dataset consists of 23 million checkpoints from over a hundred thousand training runs. We collect data from supervised learning tasks (MNIST, CIFAR-10) as well as reinforcement learning tasks (Cartpole), and across different neural network architectures (MLPs, CNNs). In addition to parameters, we record relevant task-level metrics in each checkpoint, such as test losses and classification errors.

Given this data, we explore generative pre-training directly in parameter space (Figure 3.1, right). Specifically, we train transformer-based diffusion models of neural network parameters. Given an initial input parameter vector and a target loss,

error or return, these models are trained to predict the distribution over updated parameter vectors for a single network architecture that achieve the target metric. Our method is trained with standard generative modeling techniques instead of unrolled optimization and reinforcement learning algorithms. We call our model `G.pt`[1].

We show that our approach has a number of favorable properties. First, it is able to rapidly train neural networks from unseen initializations with just one parameter update (Figure 3.3). Second, it can generate parameters that achieve a wide range of prompted losses, errors and returns (Figure 3.5). Third, it is able to generalize to out-of-distribution weight initialization algorithms (Figure 3.6). Fourth, as a generative model, it is able to sample diverse solutions (Figure 3.8). Finally, it can optimize non-differentiable objectives, such as RL returns or classification errors.

3.2 Generative Pre-training from Neural Network Checkpoints

We pre-train a generative model `G.pt` on neural network checkpoints. At test time, we use it to generate parameters for neural networks that solve a downstream task.

3.2.1 A Dataset of Neural Network Checkpoints

In order to train `G.pt`, we build a dataset of neural network checkpoints. Each checkpoint contains neural network parameters and relevant task-level metrics like train losses, test errors or returns. We use standard optimizers like Adam and SGD with momentum to generate the parameters, and we randomly save a subset of checkpoints from each training run. Our methodology for generating each individual training run is explained in detail in Algorithm 1. See Section 3.3 for additional details.

Augmenting datasets of neural networks. To offset the computational cost of collecting checkpoints, we use data augmentation in neural network parameter space. Given a checkpoint (θ, ℓ), we construct augmented tuples $(\mathcal{T}(\theta), \ell)$, where $\mathcal{T}(\cdot)$ is the parameter-level augmentation. In order for these augmented tuples to be valid, we need $f_{\mathcal{T}(\theta)}(x) = f_\theta(x)$ for all parameter vectors θ and all inputs to the neural network x. One type of augmentation that meets this criteria is *permutation augmentation*. Consider an MLP. If we apply some permutation to the outgoing

[1] `G` and `.pt` refer to generative models and checkpoint extensions, respectively.

Algorithm 1 Checkpoint Data Generation

Input: Dataset or simulator D, neural network f, loss function L, task metric, meta data store S.
Initialize: Learnable parameters θ for f
for $t = 1, 2, ..., N_{\text{iter}}$ **do**
    # Sample a mini-batch of data
    $\{\text{inputs}, \text{labels}\}_t \sim D$
    # Compute the predictions
    $\text{predictions} \leftarrow f_\theta(\text{inputs})$
    # Compute the loss
    $\text{loss} \leftarrow L(\text{predictions}, \text{labels})$
    # Update the model's parameters
    $\theta_{t+1} \leftarrow \text{update}(\text{loss}; \theta)$
    # Compute the task metric
    $\ell_t \leftarrow \text{metric}(\text{predictions}, \text{labels})$
    # Save the checkpoint
    $S \leftarrow S \cup \{\theta_t, \ell_t\}$
end for

Algorithm 2 Pre-training from Checkpoints

Input: Number of training runs K, checkpoint dataset runs $\{S_k\}_{k=1}^{K}$, `G.pt`, diffusion process length J, diffusion cumulative variance schedule $\bar{\alpha}$.
Initialize: Learnable parameters ϕ for G
for $i = 1, 2, ..., N_{\text{iter}}$ **do**
    # Sample a mini-batch of data
    $\{\theta_{t_1}, \theta_{t_2}, \ell_{t_1}, \ell_{t_2}\}_i \sim S_k$
    # Noise future parameters
    $j \sim U(\{1, ..., J\})$
    $\tilde{\theta}_{t_2} \sim \mathcal{N}(\sqrt{\bar{\alpha}_j}\theta_{t_2}, (1 - \bar{\alpha}_j)I)$
    # Compute the predictions
    $\hat{\theta}_{t_2} \leftarrow G_\phi(\tilde{\theta}_{t_2}, \theta_{t_1}, \ell_{t_2}, \ell_{t_1}, j)$
    # Compute the loss
    $\text{loss} \leftarrow ||\hat{\theta}_{t_2} - \theta_{t_2}||_2^2$
    # Update `G.pt`'s parameters
    $\phi_{i+1} \leftarrow \text{update}(\text{loss}; \phi)$
end for

weights (and biases) of the input layer and to the incoming weights of the next layer, the output of the neural network will be preserved [75, 76]. Different permutations can be sampled for each layer up to the output layer. This technique is generic and can be applied to MLPs and CNNs alike. We apply the same permutation to both the input and target parameters during pre-training.

3.2.2 Generative Models of Neural Network Checkpoints

Using our dataset of checkpoints, we train a generative model G that learns to rapidly train other neural networks. Specifically, G predicts the distribution of updated parameters $p_G(\theta^*|\theta, \ell^*, \ell)$, where θ is the starting (potentially random) neural network parameters, ℓ is the starting loss/error/return and ℓ^* is a user's prompted loss/error/return. Conditioning on ℓ^* allows `G.pt` to learn from checkpoints with good and bad performance alike. In this section, we describe an instantiation of our approach based on diffusion and transformers.

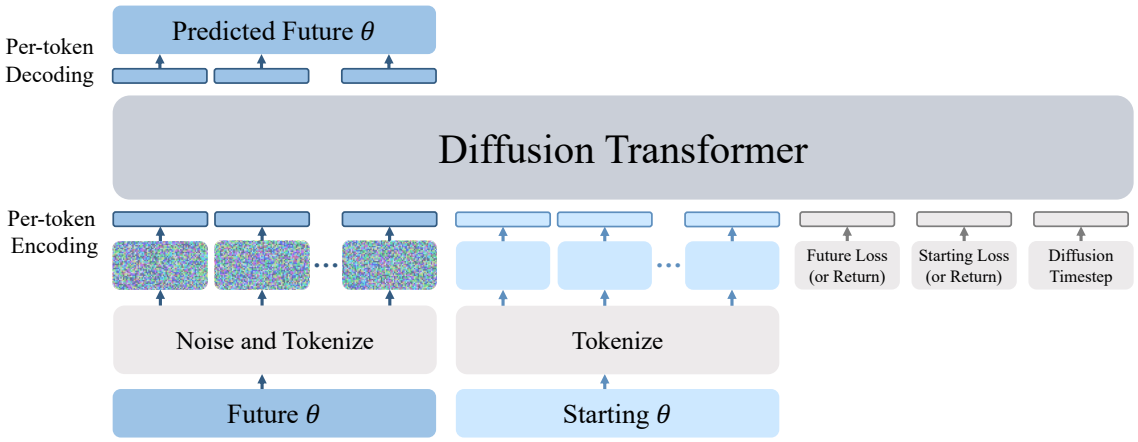

Figure 3.2: **The `G.pt` architecture.** During training, we sample two checkpoints from the same run—a "starting" network's parameters and a "future" network's parameters from later in the run—as well as their losses/errors/returns. Each layer's parameters are flattened and linearly encoded. The future network's parameters are noised via a diffusion forward process prior to encoding.

Pre-training Objective: Diffusion of Neural Network Checkpoints

We use diffusion [12] as our generative pre-training task. Diffusion is a good generative modeling framework for neural network parameters since the number of forward passes required to sample a novel parameter vector is set by the length of the diffusion process J as opposed to the dimensionality of the data. This instantiation of `G.pt` samples parameters by gradually denoising the future (updated) parameters θ^*.

Parameterization. Given an input corrupted with noise, diffusion models can be parameterized to predict either the signal or the noise [13, 54]. Prior work in the image domain has shown that noise prediction outperforms signal prediction. We find that signal prediction works better in our setting empirically, and so we parameterize G to output parameters. We use fixed variances as in [13].

Training. Our model takes two parameter vectors as input: a starting θ and a noised future parameter vector θ^*_j, where j denotes the timestep in the diffusion forward noising process. We minimize the simplified variational lower bound, which reduces to predicting the denoised future parameters:

$$\mathcal{L}(G) = \mathbb{E}\left[||\theta^* - G(\theta^*_j, \theta, \ell^*, \ell, j)||_2^2\right] \tag{3.1}$$

Algorithm 2 details our full training procedure. Note that we need tuples of data $(\theta^*, \theta, \ell^*, \ell)$ to compute $\mathcal{L}$. We sample these tuples from our checkpoint dataset. First, we sample a training run uniformly at random. Then, θ and θ^* are sampled

uniformly at random from the checkpoints saved within the selected training run. We enforce that θ is always from an earlier training step than θ^*. Note that θ and θ^* can be arbitrarily distant, even the initial and final checkpoints from a run.

Sampling. After pre-training, we sample updated parameters θ^* by querying G with an input parameter vector θ, its loss/error/return ℓ and a *prompted* loss/error/return ℓ^*. Sampling begins by feeding-in Gaussian noise as the θ^* input and gradually denoising it. We use DDPM sampling.

Architecture

Our generative model is a transformer [11] that operates over parameter tokens from both θ and θ^* (Figure 3.2). It uses few domain-specific inductive biases beyond tokenization.

Parameter tokenizers. Before being processed by `G.pt`, the two input parameter vectors θ and θ_j^* each need to be decomposed into several tokens. In general, a task-level network f_θ will contain many unique layers, each with a potentially different number of parameters. We define the i-th token as the flattened parameter vector of the i-th layer. Layers with multiple parameter groups (e.g., layers with both a weight and a bias) are decomposed into separate tokens. Note that these tokens will usually be of different dimensionality. We call this *layer-by-layer* tokenization.

Parameter tokenizers for big neural networks. For larger networks, we find that it is beneficial to decompose a single layer's parameters into multiple tokens. We do this with *layer chunking*. We define a hyperparameter M, the maximum number of parameters a single token can have. Layers containing more than M total parameters are flattened and chunked into multiple tokens, each with at-most M parameters. For example, if $M = 1000$, a weight matrix with 10×768 parameters will be decomposed into eight tokens, seven containing 1000 parameters and one containing 680 parameters. We set M to be smaller than the hidden size of the Transformer to avoid lossy compression.

Metric tokenizers. We also feed the scalar input metrics ℓ and ℓ^* (loss, error, return, etc.) and diffusion timestep j as individual tokens to the transformer. We project each scalar to a vector representation using a frequency-based encoding scheme [77].

Per-token encoders. After tokenizing f_θ's layers and the input scalars, we project each token to the hidden size of the transformer. We explored more complicated encoders, but find that a simple linear layer works well. Each token's encoder has a unique set of weights.

Transformer. The core of the `G.pt` architecture is a transformer which operates on the set of input parameters and metrics, linearly-encoded into tokens. Our

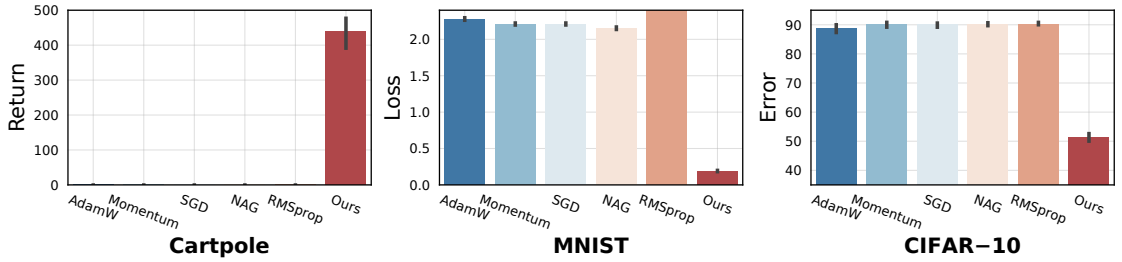

Figure 3.3: `G.pt` **optimizes unseen network parameters in one step.** We compare performance after a single update from `G.pt` versus a single step of gradient-based optimizers. Error bars are computed over five input parameter vectors, all of which are randomly-initialized.

transformer is a version of GPT-2 [21]. We omit causal masking as our model is not autoregressive across tokens.

Per-token decoders. The final layer of `G.pt` is a decoder from the transformer's output to the future parameter vector. The i-th token is linearly decoded from the transformer's hidden size back to the original size of the i-th layer's flattened parameter vector. Note that only the output tokens for the noised future parameter vector θ_j^* are decoded to predictions. Our decoders do not share weights.

Global residual connection. Finally, we find that it is beneficial to add a residual connection [27] to the input θ at the very end of `G.pt`. This amounts to predicting the parameter *update* $\theta^* - \theta$ instead of directly predicting θ^* itself. This residual connection also allows us to initialize G to perform the identity function by initializing the decoder weights to zero. Empirically, the global residual connection in conjunction with the identity initialization significantly accelerates training.

3.3 Implementation Details

We consider both supervised and reinforcement learning tasks. In all cases, we generate a large collection of training runs in order to pre-train a generative model.

Pre-training data for supervised learning. We create datasets of MNIST and CIFAR-10 network checkpoints. For MNIST, the task-level model is a two-layer MLP with 10 hidden units; for CIFAR-10, the model has two conv layers followed by global average pooling and a fully-connected layer. Both models use ReLU activations. We train the MNIST models for 25 epochs and CIFAR-10 models for 50 epochs, each with half-period cosine annealing. We use SGD with momentum of 0.9, a learning rate of 0.1 and a weight decay of 5e-4. We train approximately 10K MNIST models and 55K CIFAR-10 models from different random initializations. We

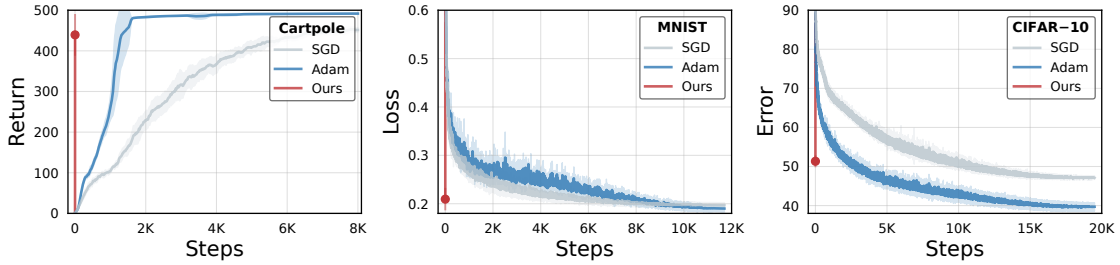

Figure 3.4: **Optimization curves.** We compare one step of `G.pt` optimization to training curves produced by SGD and Adam. Error bars are computed over five initializations.

select 200 checkpoints to save each run: the initial checkpoint (before training), the final checkpoint and intermediate checkpoints at random iterations. In total, this results in 2M trained MNIST MLPs and 11M trained CIFAR-10 CNNs.

Pre-training data for reinforcement learning. For our reinforcement learning (RL) experiments, we train policies for the Cartpole task using the IsaacGym simulator [78]. Our policy is a three-layer MLP with 32 hidden units and SeLU activations. We also train a separate critic network with the same architecture as the policy; we only model the policy's parameters in our `G.pt` experiments. We train for 500 iterations using PPO [79] and Adam [60] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We train 50K models and record 200 checkpoints in each. This results in a dataset of 10M trained policies.

Model pre-training. We train `G.pt` with AdamW [59]. We maintain an exponential moving average (EMA) of `G.pt` weights over the course of training. Our transformer uses a hidden dimension between 1536 and 2048 depending on dataset and has 12 hidden layers with 12-16 heads for self-attention. We use learned positional embeddings across all tokens, initialized to zero. We train one `G.pt` model per-metric, dataset and architecture (e.g., an error-conditional MNIST MLP model).

Parameter normalization. We follow DALL·E 2's [14] normalization scheme, where the data is scaled such that the variance of the marginal distribution matches the variance of ImageNet pixels scaled to [-1, 1], for which diffusion hyperparameters have been tuned. We find that this normalization ensures the forward noising process destroys nearly all signal in θ_j^*; the KL divergence against a standard normal is roughly 8×10^{-6} bits/dim across our experiments.

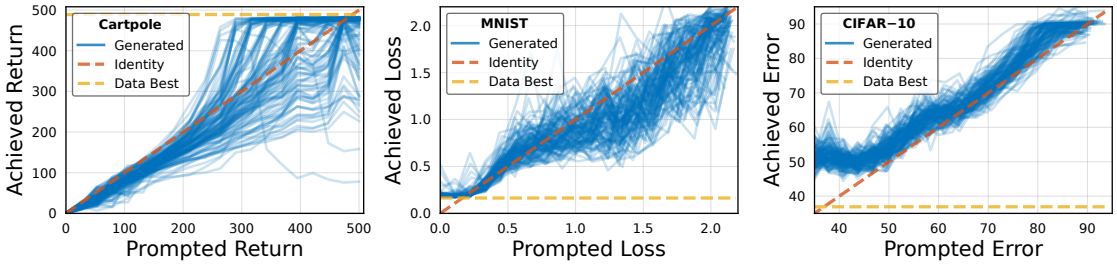

Figure 3.5: **Achieved returns, losses and errors across a range of input `G.pt` prompts.** `G.pt` can train unseen neural network parameters to a range of desired values in one update. Each blue curve corresponds to a different randomly-initialized input parameter vector. We also show the best value of each metric present in the training split of the checkpoint dataset.

3.4 Experiments

We compare our method to hand-designed optimizers and study the properties of our approach. We always report optimization of `G.pt` on unseen network parameters.

3.4.1 Comparison to Hand-Designed Optimizers

Training in one step. Figure 3.3 demonstrates `G.pt`'s ability to train unseen neural network parameters in one update. This property is unique compared to gradient-based optimizers like SGD and Adam which usually require thousands, if not millions, of updates to achieve good performance. We compare against several of these traditional optimizers with tuned learning rates and weight decays[2]. Note that we did not systematically tune training hyperparameters for checkpoints in our dataset. For each method, we measure performance after applying one update to randomly-initialized network parameters and average results over five seeds. We prompt `G.pt` by setting ℓ^* near the best return/loss/error in our dataset (for some tasks, asking for a value slightly above or below the best value in the dataset works better). `G.pt` outperforms gradient-based optimizers in this regime across tasks (control, image classification), datasets (Cartpole, MNIST, CIFAR-10) and conditioning metrics (return, test loss, test error). Additionally, `G.pt` successfully optimizes non-differentiable metrics (CIFAR-10 test error) whereas baseline optimizers must use smoothed surrogates.

[2]We perform a grid search over three learning rates (the PyTorch default and 10× above/below) and three weight decay values (0, 5×10^{-5}, 5×10^{-4}) for each baseline optimizer.

Training in multiple steps. We compare one step of `G.pt` to multiple steps of SGD and Adam in Figure 3.4. SGD and Adam use tuned learning rates and weight decays. The baseline optimizers require thousands of iterations to match the performance of one step of `G.pt`. With tuned hyperparameters and a sufficiently large number of updates, gradient-based optimizers supersede one-step `G.pt` optimization. Our model can also be used as an iterative optimizer with recursive prompting. In this setting, we repeatedly feed `G.pt`'s predicted θ^* back in as its input θ and ask for low loss/error or high returns. Interestingly, we find that the best performance is usually realized with one-step prompting (recursive prompting usually brings only minor improvements). However, we find that recursive prompting leads to considerably better results when the input neural network comes from an out-of-distribution initialization algorithm not present in our checkpoint dataset (see Figure 3.6 below).

3.4.2 Prompting for Losses, Errors and Returns

By prompting for various desired losses, errors, or returns, `G.pt` can sample different parameter updates that achieve a range of performance levels. In Figure 3.5, we show that `G.pt` successfully learns to generate parameters corresponding to a large range of prompted values. We pass `G.pt` randomly-initialized neural network parameters and ask it to optimize them in one step to a range of losses/errors/returns. We show results for several different starting parameters. Across different tasks and metrics, `G.pt` generates parameter updates that are well-correlated with the prompted value. While our model is able to achieve a range of prompted values, we note that it currently shows limited ability to extrapolate to values beyond the limits of the pre-training dataset.

3.4.3 Generalization to Out-of-Distribution Initializations

The networks in our checkpoint dataset are initialized with a single weight initialization scheme. For MNIST, they are sampled $\theta \sim U[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$, where n is the fan-in of a layer. In Figure 3.6, we evaluate `G.pt`'s ability to generalize to randomly-initialized input parameter vectors θ, where the weights are sampled from different distributions [80–82] not present in our dataset. While one step prompting performance is degraded, recursive prompting significantly improves results. `G.pt` is able to rapidly optimize out-of-distribution weights in ten or fewer parameter updates.

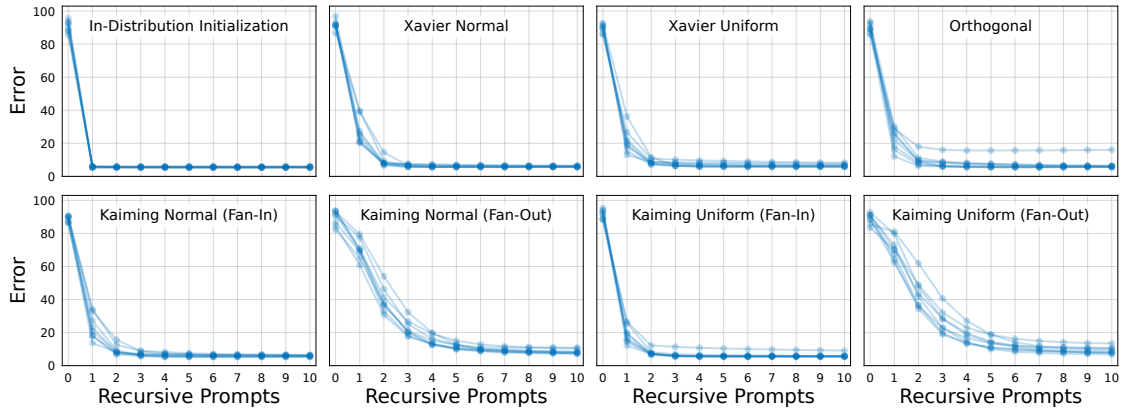

Figure 3.6: `G.pt` **generalizes to out-of-distribution parameter initializations.** We query `G.pt` with randomly-initialized weights sampled from a different distribution than those in our MNIST checkpoint dataset. By recursively applying `G.pt` to its own output and prompting for low test error, we rapidly optimize out-of-distribution random initializations.

3.4.4 Scaling Model and Data Size

Performance metric. We use prompt alignment to measure scaling performance. We define it as the R^2 coefficient of determination between a set of input loss/error/return prompts and the actual loss/error/return achieved by the parameters sampled from `G.pt`. We compute R^2 values over 20 regularly-sampled prompts and average results over 128 neural networks. The optimal score is $+1$, which indicates that `G.pt` perfectly listens to loss prompts. Randomly-initialized `G.pt` score around -2.7. We use unseen, randomly-initialized input networks in order to gauge generalization capabilities. Empirically, we find that prompt alignment is a more reliable quality metric than diffusion mean-squared error on unseen parameter vectors.

Model scale. We analyze the impact of increasing the number of `G.pt` parameters in Figure 3.7 (top). We train six models with transformer hidden sizes in [64, 128, 256, 512, 1024, 2048]; the smallest model is approximately 2M parameters while the largest is 858M parameters. We evaluate the `G.pt` checkpoint that attains the highest prompt alignment score over training. We find that larger models generalize much more effectively than smaller models. Small models (<60M parameters) largely fail to generalize to unseen parameter vectors. Even at roughly 10^9 parameters, we find that `G.pt` has not saturated its model scaling curve.

Data scale. Next, we analyze the impact of increasing the number of training checkpoints in Figure 3.7 (bottom). We train our largest 858M parameter model on [500, 5K, 10K, 25K, 55K] runs, with each run containing 200 checkpoints. Performance improves substantially as the number of training checkpoints is scaled from 100K to 5M. We do not observe significant improvement when further increasing from 5M to 10M checkpoints. This may be a result of `G.pt` requiring additional model scale to benefit from a larger pre-training dataset.

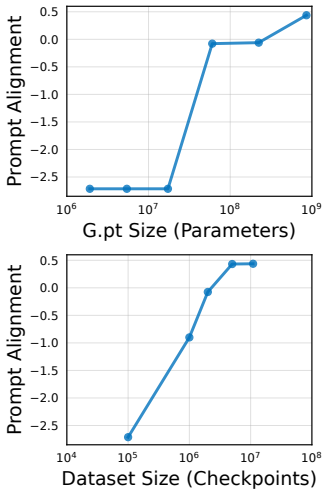

Figure 3.7: **`G.pt` Scaling.**

3.4.5 Diversity of Generated Parameters

The mapping of loss values to neural network parameters is one-to-many. As a generative model, `G.pt` is able to sample diverse parameter solutions given a loss prompt. By fixing all `G.pt` inputs (including θ) and varying sampling noise, we can sample multimodal solutions that cover distinct error minima (Figure 3.8). Visual inspection of generated first-layer weights suggests that sampling noise controls subtle variations in individual filters as well as the specific ordering of filters.

Intuitively, conditioning on a starting θ should narrow the space of possible parameter solutions (in other words, initialization should have some bearing on where optimization converge). Indeed, we find that the most significant variation is obtained by re-sampling the starting θ.

3.4.6 Dataset Design Decisions

Parameter augmentation aids generalization. In the absence of permutation augmentation, we observe that `G.pt` can aggressively overfit the training set and fail to generalize to new networks (i.e., it exhibits poor prompt alignment when taking unseen networks as input). Training with parameter augmentation alleviates overfitting in our Cartpole `G.pt` model.

Training on intermediate checkpoints improves one step training. Given the redundancy in neural network parameters over a single training run, it is worth asking if there is value in training `G.pt` on 200 intermediate checkpoints per-run. Instead, we could train `G.pt` exclusively on the initial and final checkpoint from each run. We find that this setup degrades one step training capabilities by over 50%: average test loss when prompting with $\ell^* = 0$ worsens from 0.2 to 0.32. `G.pt`

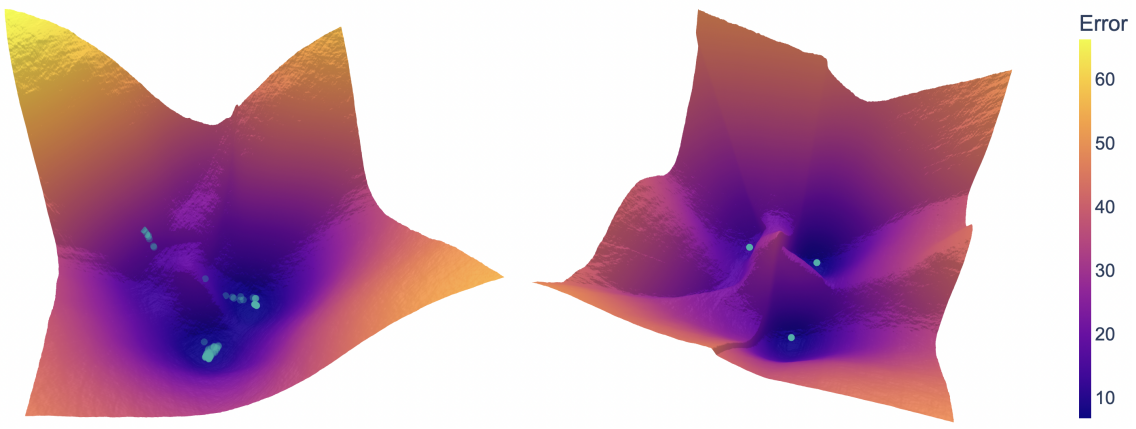

Figure 3.8: `G.pt` **learns a multimodal distribution over local error minima.** We visualize the test error landscape for an MNIST MLP via parameter space PCA directions [1]. The dots are samples from `G.pt` when prompted for low test error; the two plots use different MLP initializations. With fixed inputs, `G.pt` samples diverse solutions that cover distinct positive-curvature regions of the error landscape. We show `G.pt` samples that reconstruct accurately from PCA encoding.

significantly benefits from training on a large number of checkpoints, even those from the same run.

3.5 Memorization Versus Generalization

Next, we investigate the extent to which `G.pt` memorizes solutions from the training set. This is a challenging topic to address for any generative model, and there is no universally-accepted methodology to measure it. For generative models of images, one popular methodology is to visualize the nearest neighbors of generated images in the training set. Visualizing parameters is challenging for deep networks beyond the first layer, so we instead provide a basic way to quantify memorization.

Experimental setup. Our approach is also based on nearest neighbors. We feed `G.pt` an unseen, randomly-initialized parameter vector from a test run and sample a corresponding solution θ^* from our model. If `G.pt` is memorizing parameters from the training set, then the sampled θ^* should be closer to one of the millions of parameter vectors across *all* runs in the training split than the 200 "ground truth" parameter vectors in the same test run from which we took the randomly-initialized input parameters. On the other hand, if θ^* is closer to one of these 200 held-out checkpoints, it suggests that it is accurately predicting the outcome of gradient-based optimization (i.e., there is some level of meaningful generalization). For simplicity,

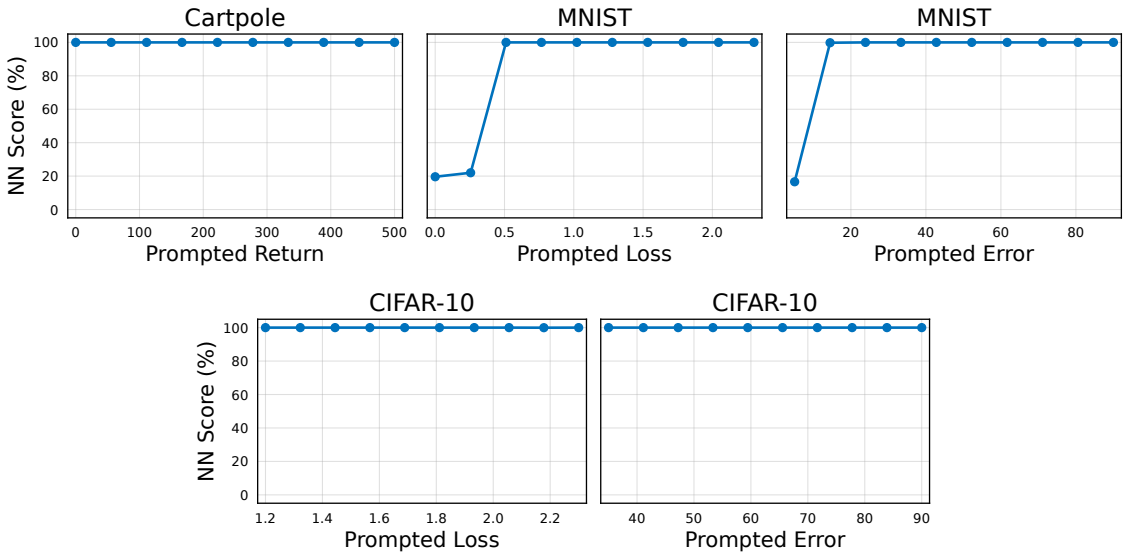

Figure 3.9: `G.pt` **predictions on held-out (unseen) random initializations tend to lie closer to the ground truth outcome of SGD/Adam than any parameter vector from our checkpoint dataset's training split.** For each test run in our dataset, we feed the initial parameters and a metric prompt to `G.pt`, and we sample a prediction. We count the percentage of runs for which the prediction is closer to one of the 200 checkpoints in that same test run than all checkpoints in the training split (Cartpole has 10M training split checkpoints, CIFAR-10 has 11.3M and MNIST has 2.1M). Each plot corresponds to a different `G.pt` model, and we repeat the test for a wide range of prompts.

we compute distances in Euclidean space. We count the percentage of test runs for which `G.pt` generates a solution closer to any of the 200 checkpoints in the test run than all checkpoints in the training split of our dataset (Cartpole has 10M training split checkpoints, CIFAR-10 has 11.3M and MNIST has 2.1M). We call this percentage the *nearest neighbor score*. A score of 100% suggests `G.pt` is perfectly generalizing. We repeat this test for a range of loss, error and return prompts.

Results. Figure 3.9 shows nearest neighbor scores for all five of our `G.pt` models. Our models appear to accurately generalize under a large number of loss, error and return prompts. Our Cartpole and CIFAR-10 models exhibit perfect scores (100%) for all input prompts. Interestingly, while our MNIST models also have perfect scores for the majority of loss/error prompts, they have lower scores for smaller prompts (decreasing to about 15-20%). A speculative explanation is that our MNIST

models were trained with about a fifth of the number of training runs compared to our Cartpole and CIFAR-10 models; this could possibly degrade generalization capabilities. Overall, this test provides some initial evidence that `G.pt` is generalizing and not just memorizing training set parameters.

3.6 Related Work

3.6.1 Pre-training from Large-Scale Data

Transformers for X. Transformers [11] were initially developed for language but have been shown to be well-suited for a wide range of domains. They have achieved strong results in vision [3], language modeling [6, 19, 21], coding [83, 84], , reinforcement learning [31, 32], image synthesis [23, 39, 42] and protein folding [85]. Likewise, we show that transformers can be used for learning to learn by generative pre-training from neural network parameters.

Diffusion. Diffusion models [12] have recently been shown to be highly effective for images [4, 10, 13, 14, 45, 50, 54]. In this section, we show that diffusion models can be used for meta-learning by generating neural network parameters.

Pre-training. Large scale pre-training has led to significant advances in vision [8, 9], natural language processing [6,19–21] and audio understanding [71,86]. We explore pre-training from datasets of neural networks instead of datasets of images and text.

Datasets of neural networks. Past works have constructed datasets of neural networks and used them in various settings: analyzing population-level trends [52,53], benchmarking neural architecture search [87], training hypernetworks [88], predicting model properties [76] and dataset distillation [89, 90]. We share the goal of using datasets of neural networks, but for the novel meta-learning approach of pre-training a generative model from trained neural network checkpoints.

3.6.2 Learning to Learn

Learning optimizers. Past works have explored parameterizing optimization update rules with neural networks in place of hand-designed rules like Adam. These rules can be parameterized implicitly as neural networks that take gradients as input and output an improved parameter update. They are typically trained with unrolled optimization [15, 91–98] or reinforcement learning [74, 99].

Hypernetworks. Rather than parameterizing update rules, neural networks can be used to directly output or modify other neural networks' parameters [100–102]. For example, hypernetworks [103] train parameter regression networks end-to-end

with the task objective. Hypernetworks have subsequently been extended to support sampling different parameter solutions [104–106].

Model-agnostic meta-learning. MAML learns a parameter initialization that is rapidly adaptable to new tasks [16]. Subsequent work has built simple probabilistic models over learned MAML initializations [107]. These methods possess similar characteristics as learned optimizers—they rely on unrolled optimization and require differentiable task-level objectives.

Learning hyperparameters. A large body of prior work has explored learning hyperparameters of standard optimizers [108,109]. For example, learning rates, weight decays and weight initializations can all be learned via hypergradient descent [110–112], Bayesian optimization [113] and reinforcement learning [114–117].

Learning to learn as pre-training. In contrast to learned optimizers, hypernetworks and MAML, `G.pt` pre-trains from vast amounts of trained neural network checkpoints. Our method does not backpropagate through task-level losses and, as a result, does not require the task metric being optimized for to be differentiable. This allow us to train with standard generative modeling techniques instead of reinforcement learning or unrolled optimization which can be unstable [118].

3.7 Discussion

Limitations. The current instantiation of our method has several limitations. First, the model sometimes exhibits signs of underfitting the full loss/error landscape, such as with CIFAR-10. Second, our current `G.pt` models struggle to extrapolate to losses and errors not present in the pre-training data. Third, our work only pre-trains from single-architecture and single-task data. Finally, we consider relatively simple datasets of neural networks with static optimizer hyperparameters.

Conclusion. We propose generative pre-training from neural network checkpoints. We show that our approach enables rapid optimization of neural networks across tasks (supervised and reinforcement learning) and metrics (losses, errors, returns). Learning algorithms designed by humans have led to large advancements across different areas of artificial intelligence. We hope that our work serves as a step towards learning learning algorithms from data using modern deep learning techniques.

Chapter 4

Perception from Pre-trained Generative Models

In the previous sections, we explored methods for building improved generative models of various modalities. In this section, we explore how to leverage powerful pre-trained generative models to tackle *downstream* tasks. There are many ways one could go about using, e.g., pre-trained image-level generative models to tackle tasks in perception. We explore one promising approach in this section—generating infinite datasets of highly-customizable (and end-to-end learnable) inputs and labels for the dense visual alignment/correspondence task. A notable advantage of this approach is that it obviates the need to collect expensive task-specific annotations from humans while producing unlimited amounts of training data. As we will see, our approach outperforms fully-supervised and self-supervised baselines that train models on *real* data.

4.1 Introduction

Visual alignment, also known as the correspondence or registration problem, is a critical element in much of computer vision, including optical flow, 3D matching, medical imaging, tracking and augmented reality. While much recent progress has been made on pairwise alignment (aligning image A to image B) [120–132], the problem of global joint alignment (aligning *all* images across a dataset) has not received as much attention. Yet, joint alignment is crucial for tasks requiring a common reference frame, such as automatic keypoint annotation, augmented reality or edit propagation (see Figure 4.1 bottom row). There is also evidence that training

This work originally appeared in CVPR 2022 [119].

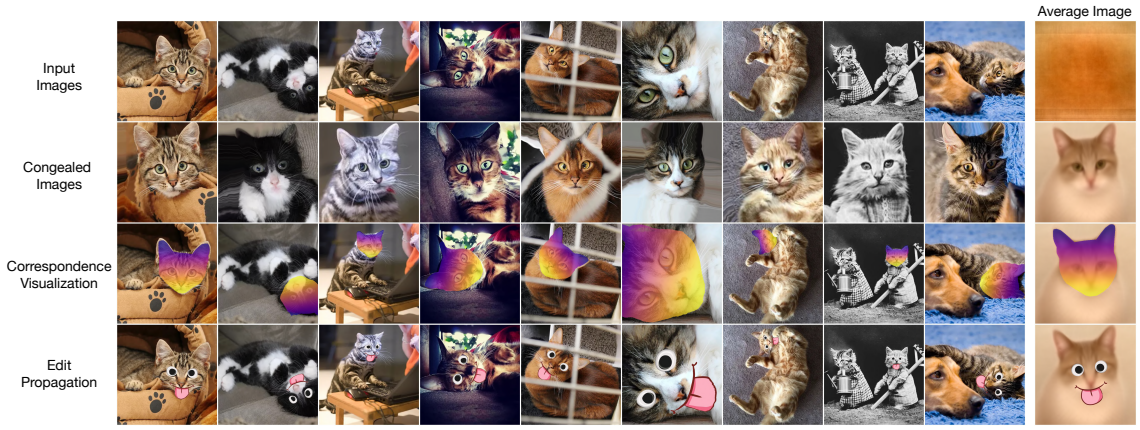

Figure 4.1: Given an input dataset of unaligned images, our GANgealing algorithm discovers dense correspondences between all images. **Top row**: Images from LSUN Cats and the dataset's average image. **Second row**: Our learned transformations of the input images. **Third row**: Dense correspondences learned by GANgealing. **Bottom row**: By annotating the average transformed image, we can propagate user edits to images and videos. **Please see our project page for detailed video results:** `www.wpeebles.com/gangealing`.

on jointly aligned datasets (such as FFHQ [57], AFHQ [133], CelebA-HQ [134]) can produce higher quality generative models than training on unaligned data.

In this section, we take inspiration from a series of classic works on automatic joint image set alignment. In particular, we are motivated by the seminal unsupervised *Congealing* method of Learned-Miller [135] which showed that a set of images could be brought into alignment by continually warping them toward a common, updating mode. While Congealing can work surprisingly well on simple binary images, such as MNIST digits, the direct pixel-level alignment is not powerful enough to handle most datasets with significant appearance and pose variation.

To address these limitations, we propose GANgealing: a *GAN-Supervised* algorithm that learns transformations of input images to bring them into better joint alignment. The key is in employing the latent space of a GAN (trained on the unaligned data) to automatically generate paired training data for a Spatial Transformer [136]. Crucially, in our proposed GAN-Supervised Learning framework, *both* the Spatial Transformer and the target images are learned jointly. Our Spatial Transformer is trained *exclusively* with GAN images and generalizes to real images at test time.

We show results spanning eight datasets—LSUN Bicycles, Cats, Cars, Dogs, Horses and TVs [137], In-The-Wild CelebA [138] and CUB [139]—that demonstrate

our GANgealing algorithm is able to discover accurate, dense correspondences across datasets. We show our Spatial Transformers are useful in image editing and augmented reality tasks. Quantitatively, GANgealing significantly outperforms past self-supervised dense correspondence methods, nearly doubling key point transfer accuracy (PCK [140]) on many SPair-71K [141] categories. Moreover, GANgealing sometimes matches and even exceeds state-of-the-art correspondence-*supervised* methods.

4.2 Related Work

Pre-Trained GANs for Vision. Prior work has explored the use of GANs [142, 143] in vision tasks such as classification [144–148], segmentation [149–152] and representation learning [153–157], as well as 3D vision and graphics tasks [158–161]. Likewise, we share the goal of leveraging the power of pre-trained deep generative models for vision tasks. However, the relevant past methods follow a common two-stage paradigm of (1) synthesizing a GAN-generated dataset and (2) training a discriminative model on the fixed dataset. In contrast, our GAN-Supervised Learning approach learns *both* the discriminative model as well as the GAN-generated data jointly end-to-end. We do not rely on hand-crafted pixel space augmentations [144, 155], human-labeled data [151,152,158,160,161] or post-processing of GAN-generated datasets using domain knowledge [146, 149, 150, 160].

Joint Image Set Alignment. Average images have long been used to visualize joint alignment of image sets of the same semantic content (e.g., [162, 163]), with the seminal work of Congealing [135, 164] establishing unsupervised joint alignment as a research problem. Congealing uses sequential optimization to gradually minimize the entropy of the intensity distribution of a set of images by continuously warping each image via a parametric transformation (e.g., affine). It produces impressive results on well-structured datasets, such as digits, but struggles on more complex data. Subsequent work assumes the data lies on a low-rank subspace [165, 166] or factorizes images as a composition of color, appearance and shape [167] to establish dense correspondences between instances of the same object category. FlowWeb [168] uses cycle consistency constraints to estimate a fully-connected correspondence flow graph. Every method above assumes that it is possible to align all images to a single central mode in the data. Joint visual alignment and clustering was proposed in AverageExplorer [163] but as a user-driven data interaction tool. Bounding box supervision has been used to align and cluster multiple modes within object categories [169]. Automated transformation-invariant clustering methods [170–172] can align images in a collection before comparing them but work only in limited

domains. Recently, Monnier et al. [173] showed that warps could be predicted with a network instead, removing the need for per-image optimization; this opened the door for simultaneous alignment and clustering of large-scale collections. Unlike our approach, these methods assume images can be aligned with simple (e.g., affine) color transformations; this assumption breaks down for complex datasets like LSUN.

Spatial Transformer Networks (STNs). A Spatial Transformer module [136] is one way to incorporate learnable geometric transformations in a deep learning framework. It regresses a set of warp parameters, where the warp and grid sampling functions are differentiable to enable backpropagation. STNs have seen success in discriminative tasks (e.g., classification) and applications such as robust filter learning [174, 175], view synthesis [176–178] and 3D representation learning [179–181]. Inverse Compositional STNs (IC-STNs) [182] advocate an iterative image alignment framework in the spirit of the classical Lukas-Kanade algorithm [183, 184]. Prior work has incorporated STNs in generative models for geometry-texture disentanglement [185] and image compositing [186]. In contrast, we use a generative model to directly produce training data *for* STNs.

4.3 GAN-Supervised Learning

In this section, we present GAN-Supervised Learning. Under this framework, $(\boldsymbol{x}, \boldsymbol{y})$ pairs are sampled from a pre-trained GAN generator, where $\boldsymbol{x}$ is a random sample from the GAN and $\boldsymbol{y}$ is the sample obtained by applying a *learned* latent manipulation to $\boldsymbol{x}$'s latent code. These pairs are used to train a network $f_\theta : \boldsymbol{x} \to \boldsymbol{y}$. This framework minimizes the following loss:

$$\mathcal{L}(f_\theta, \boldsymbol{y}) = \ell(f_\theta(\boldsymbol{x}), \boldsymbol{y}), \tag{4.1}$$

where ℓ is a reconstruction loss. In vanilla supervised learning, f_θ is learned on *fixed* $(\boldsymbol{x}, \boldsymbol{y})$ pairs. In contrast, in GAN-Supervised Learning, *both* f_θ and the targets $\boldsymbol{y}$ are learned jointly end-to-end. At test time, we are free to evaluate f_θ on *real* inputs.

4.3.1 Dense Visual Alignment

Here, we show how GAN-Supervised Learning can be applied to Congealing [135]—a classic unsupervised alignment algorithm. In this instantiation, f_θ is a Spatial Transformer Network [136] T, and we describe our parameterization of inputs $\boldsymbol{x}$ and learned targets $\boldsymbol{y}$ below. We call our algorithm *GANgealing*. We present an overview in Figure 4.2.

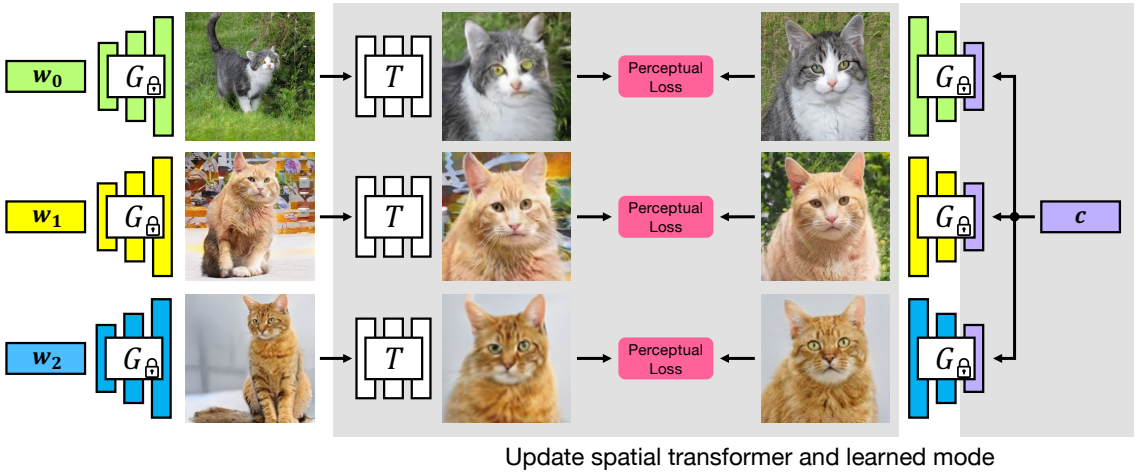

Figure 4.2: **GANgealing Overview**. We first train a generator G on unaligned data. We create a *synthetically-generated* dataset for alignment by learning a mode $\mathbf{c}$ in the generator's latent space. We use this dataset to train a Spatial Transformer Network T to map from unaligned to corresponding aligned images using a perceptual loss [2]. The Spatial Transformer generalizes to align *real* images automatically.

GANgealing begins by training a latent variable generative model G on an unaligned input dataset. We refer to the input latent vector to G as $\mathbf{w} \in \mathbb{R}^{512}$. With G trained, we are free to draw samples from the unaligned distribution by computing $\boldsymbol{x} = G(\mathbf{w})$ for randomly sampled $\mathbf{w} \sim \mathcal{W}$, where $\mathcal{W}$ denotes the distribution over latents. Now, consider a fixed latent vector $\mathbf{c} \in \mathbb{R}^{512}$. This vector corresponds to a fixed synthetic image $G(\mathbf{c})$ from the original unaligned distribution. A simple idea in the vein of traditional Congealing is to use $G(\mathbf{c})$ as the target mode $\boldsymbol{y}$—i.e., we learn a Spatial Transformer T that is trained to warp every random unaligned image $\boldsymbol{x} = G(\mathbf{w})$ to the same target image $\boldsymbol{y} = G(\mathbf{c})$. Since G is differentiable in its input, we can optimize $\mathbf{c}$ and hence learn the target we wish to congeal towards. Specifically, we can optimize the following loss with respect to both T's parameters and the target image's latent vector $\mathbf{c}$ jointly:

$$\mathcal{L}_{\text{align}}(T, \mathbf{c}) = \ell(T(G(\mathbf{w})), G(\mathbf{c})), \tag{4.2}$$

where ℓ is some distance function between two images. By minimizing $\mathcal{L}$ with respect to the target latent vector $\mathbf{c}$, GANgealing encourages $\mathbf{c}$ to find a pose that makes T's job as easy as possible. If the current value of $\mathbf{c}$ corresponds to a pose that cannot be reached from most images via the transformations predicted by T, then it can

be adjusted via gradient descent to a different vector that is “reachable" by more images.

This simple approach is reasonable for datasets with limited diversity; however, in the presence of significant appearance and pose variation, it is not reasonable to expect that every unaligned sample can be aligned to the exact same target image. Hence, optimizing the above loss does not produce good results in general (see Table 4.3). Instead of using the same target $G(\mathbf{c})$ for every randomly sampled image $G(\mathbf{w})$, it would be ideal if we could construct a *per-sample target* that retains the appearance of $G(\mathbf{w})$ but where the pose and orientation of the object in the target image is roughly identical across targets. To accomplish this, given $G(\mathbf{w})$, we produce the corresponding target by setting just a portion of the $\mathbf{w}$ vector equal to the target vector $\mathbf{c}$. Specifically, let $\text{mix}(\mathbf{c}, \mathbf{w}) \in \mathbb{R}^{512}$ refer to the latent vector whose first entries are taken from $\mathbf{c}$ and remaining entries are taken from $\mathbf{w}$. By sampling new $\mathbf{w}$ vectors, we can create an infinite pool of paired data where the input is the unaligned image $\boldsymbol{x} = G(\mathbf{w})$ and the target $\boldsymbol{y} = G(\text{mix}(\mathbf{c}, \mathbf{w}))$ shares the appearance of $G(\mathbf{w})$ but is in a learned, fixed pose. This gives rise to the GANgealing loss function:

$$\mathcal{L}_{\text{align}}(T, \mathbf{c}) = \ell(T(\underbrace{G(\mathbf{w})}_{\boldsymbol{x}}), \underbrace{G(\text{mix}(\mathbf{c}, \mathbf{w}))}_{\boldsymbol{y}}), \tag{4.3}$$

where ℓ is a perceptual loss function [2]. In this work, we opt to use StyleGAN2 [187] as our choice of G, but in principle other GAN architectures could be used with our method. An advantage of using StyleGAN2 is that it possesses some innate style-pose disentanglement that we can leverage to construct the per-image target described above. Specifically, we can construct the per-sample targets $G(\text{mix}(\mathbf{c}, \mathbf{w}))$ by using style mixing [57]—$\mathbf{c}$ is supplied to the first few inputs to the synthesis generator that roughly control pose and $\mathbf{w}$ is fed into the later layers that roughly control texture. See Table 4.3 for a quantitative ablation of the mixing "cutoff point" where we begin to feed in $\mathbf{w}$ (i.e., the cutoff point is chosen as a layer index in $\mathcal{W}^+$ space [188]).

Spatial Transformer Parameterization. Recall that a Spatial Transformer T takes as input an image and regresses and applies a (reverse) sampling grid $G \in \mathbb{R}^{H \times W \times 2}$ to the input image. Hence, one must choose how to constrain the G regressed by T. Here, we explore a T that performs similarity transformations (rotation, uniform scale, horizontal shift and vertical shift). We also explore an arbitrarily expressive T that directly regresses unconstrained per-pixel flow fields G. Our final T is a composition of the similarity Spatial Transformer into the

unconstrained Spatial Transformer, which we found worked best. In contrast to prior work [173, 186], we do not find multi-stage training necessary and train our composed T end-to-end. Finally, our Spatial Transformer is also capable of performing horizontal flips at test time.

When using the unconstrained T, it can be beneficial to add a total variation regularizer that encourages the predicted flow to be smooth to mitigate degenerate solutions: $\mathcal{L}_{\text{TV}}(T) = \mathcal{L}_{\text{Huber}}(\Delta_x G) + \mathcal{L}_{\text{Huber}}(\Delta_y G)$, where $\mathcal{L}_{\text{Huber}}$ denotes the Huber loss and Δ_x and Δ_y denote the partial derivative w.r.t. x and y coordinates under finite differences. We also use a regularizer that encourages the flow to not deviate from the identity transformation: $\mathcal{L}_{\text{I}}(T) = ||G||_2^2$.

Parameterization of c. In practice, we do not backpropagate gradients directly into $\mathbf{c}$. Instead, we parameterize $\mathbf{c}$ as a linear combination of the top-N principal directions of $\mathcal{W}$ space [189, 190]:

$$\mathbf{c} = \bar{\mathbf{w}} + \sum_{i=1}^{N} \alpha_i \mathbf{d}_i, \tag{4.4}$$

where $\bar{\mathbf{w}}$ is the empirical mean $\mathbf{w}$ vector, $\mathbf{d}_i$ is the i-th principal direction and α_i is the learned scalar coefficient of the direction. Instead of optimizing $\mathcal{L}$ w.r.t. $\mathbf{c}$ directly, we optimize it w.r.t. the coefficients $\{\alpha_i\}_{i=1}^N$. The motivation for this reparameterization is that StyleGAN's $\mathcal{W}$ space is highly expressive. Hence, in the absence of additional constraints, naive optimization of $\mathbf{c}$ can yield poor target images off the manifold of natural images. Decreasing N keeps $\mathbf{c}$ on the manifold and prevents degenerate solutions. See Table 4.3 for an ablation of N.

Our final GANgealing objective is given by:

$$\mathcal{L}(T, \mathbf{c}) = \mathbb{E}_{\mathbf{w}\sim\mathcal{W}}[\mathcal{L}_{\text{align}}(T, \mathbf{c}) + \lambda_{\text{TV}}\mathcal{L}_{\text{TV}}(T) + \lambda_I \mathcal{L}_{\text{I}}(T)]. \tag{4.5}$$

We set the loss weighting λ_{TV} at either 1000 or 2500 (depending on choice of ℓ) and the loss weighting λ_I at 1.

4.3.2 Joint Alignment and Clustering

GANgealing as described so far can handle highly-multimodal data (e.g., LSUN Bicycles, Cats, etc.). Some datasets, such as LSUN Horses, feature extremely diverse poses that cannot be represented well by a single mode in the data. To handle this situation, GANgealing can be adapted into a clustering algorithm by simply learning more than one target latent $\mathbf{c}$. Let K refer to the number of $\mathbf{c}$ vectors (clusters) we wish to learn. Since each $\mathbf{c}$ captures a specific mode in the data, learning multiple

$\{\mathbf{c}_k\}_{k=1}^K$ would enable us to learn multiple modes. Now, each $\mathbf{c}_k$ will learn its own set of α coefficients. Similarly, we will now have K Spatial Transformers, one for each mode being learned. This variant of GANgealing amounts to simultaneously clustering the data and learning dense correspondence between all images within each cluster. To encourage each $\mathbf{c}_k$ and T_k pair to specialize in a particular mode, we include a hard-assignment step to assign unaligned synthetic images to modes:

$$\mathcal{L}_{\text{align}}^K(T, \mathbf{c}) = \min_k \mathcal{L}_{\text{align}}(T_k, \mathbf{c}_k) \tag{4.6}$$

Note that the $K = 1$ case is equivalent to the previously described unimodal case. At test time, we can assign an input fake image $G(\mathbf{w})$ to its corresponding cluster index $k^* = \arg\min_k \mathcal{L}_{\text{align}}(T_k, \mathbf{c}_k)$. Then, we can warp it with the Spatial Transformer T_{k^*}. However, a problem arises in that we cannot compute this cluster assignment for input *real* images—the assignment step requires computing $\mathcal{L}_{\text{align}}$, which itself requires knowledge of the input image's corresponding $\mathbf{w}$ vector. The most obvious solution to this problem is to perform GAN inversion [191–193] on input real images $\boldsymbol{x}$ to obtain a latent vector $\mathbf{w}$ such that $G(\mathbf{w}) \approx \boldsymbol{x}$. However, accurate GAN inversion for non-face datasets remains somewhat challenging and slow, despite recent progress [194, 195]. Instead, we opt to train a classifier that directly predicts the cluster assignment of an input image. We train the classifier using a standard cross-entropy loss on (input fake image, target cluster) pairs $(G(\mathbf{w}), k^*)$, where k^* is obtained using the above assignment step. We initialize the classifier with the weights of T (replacing the warp head with a randomly-initialized classification head). As with the Spatial Transformer, the classifier generalizes well to real images despite being trained exclusively on fake samples.

4.4 Experiments

In this section, we present quantitative and qualitative results of GANgealing on eight datasets: LSUN Bicycles, Cats, Cars, Dogs, Horses and TVs [137], In-The-Wild CelebA [138] and CUB-200-2011 [139]. These datasets feature significant diversity in appearance, pose and occlusion of objects. Only LSUN Cars and Horses use clustering ($K = 4$)[1]; for all other datasets we use unimodal GANgealing ($K = 1$). Note that all figures except Figure 4.2 show our method applied to real images—not GAN samples. Please see `www.wpeebles.com/gangealing` for full results.

[1] K is a hyperparameter that can be set by the user. We found $K = 4$ to be a good default choice for our clustering models.

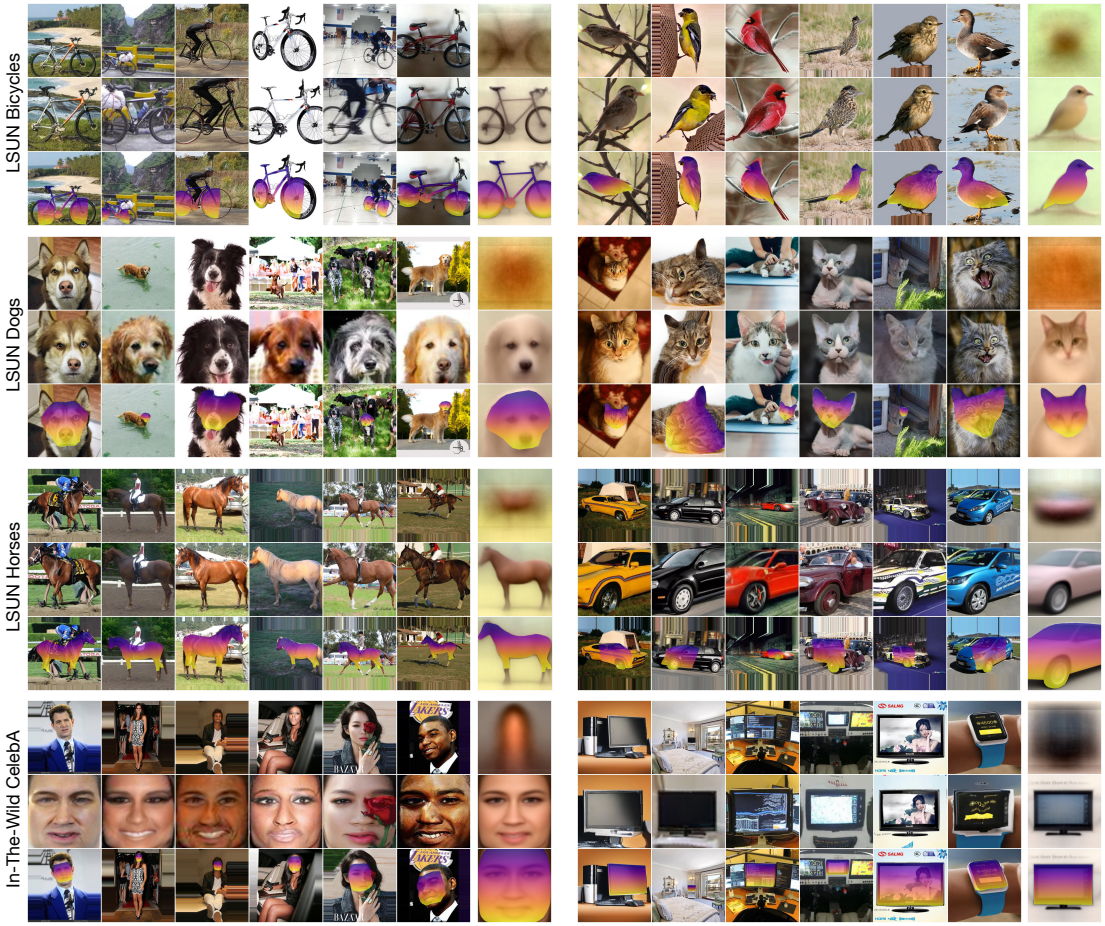

Figure 4.3: **Dense correspondence results on eight datasets**. For each dataset, the top row shows unaligned images and the dataset average image. The middle row shows our learned alignment of the input images. The bottom row shows dense correspondences between the images. For our clustering models (LSUN Horses and Cars), we show results for one selected cluster.

4.4.1 Propagation from Congealed Space

With the Spatial Transformer T trained, it is trivial to identify dense correspondences between real input images $\boldsymbol{x}$. A particularly convenient way to find dense correspondences between a set of images is by propagating from our *congealed coordinate space*. As described earlier, T both regresses and applies a sampling grid G to an input image. Because we use reverse sampling, this grid tells us where each point in the congealed image $T(\boldsymbol{x})$ maps to in the original image $\boldsymbol{x}$. This

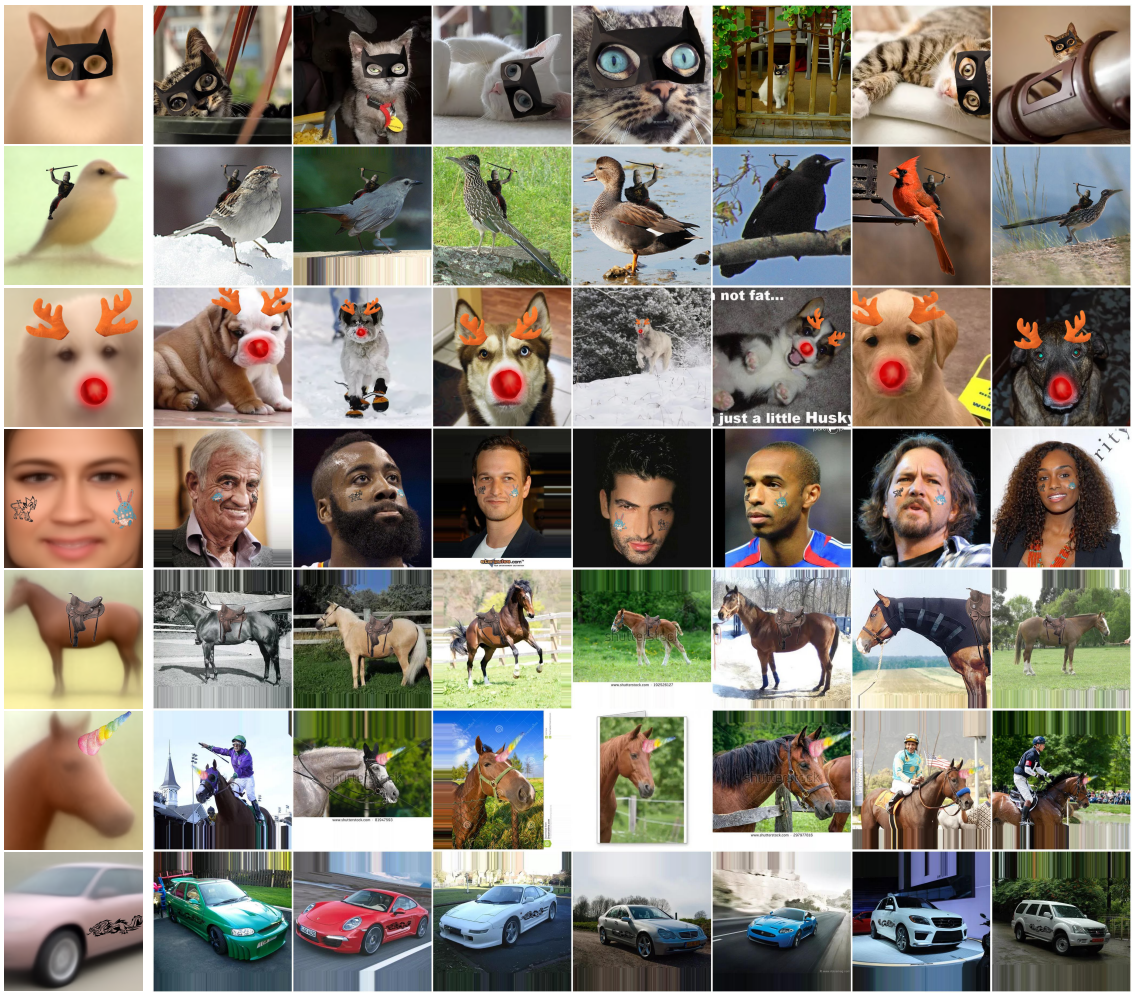

Figure 4.4: **Image editing with GANgealing**. By annotating just a single image per-category (our average transformed image), a user can propagate their edits to any image or video in the same category.

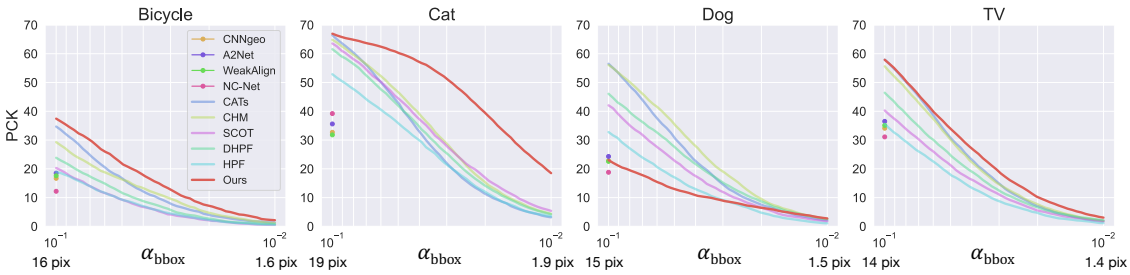

Figure 4.5: **PCK@α_{bbox} on various SPair-71K categories for α_{bbox} between 10^{-1} and 10^{-2}.** We report the average threshold (maximum distance for a correspondence to be deemed correct) in pixels for 256×256 images beneath each plot. GANgealing outperforms state-of-the-art supervised methods for very precise thresholds (< 2 pixel error tolerance), sometimes by substantial margins.

enables us to propagate *anything* from the congealed coordinate space—dense labels, sparse keypoints, etc. If a user annotates a *single* congealed image (or the average congealed image) they can then propagate those labels to an entire dataset by simply predicting the grid G for each image $\boldsymbol{x}$ in their dataset via a forward pass through T. Figures 4.1 and 4.3 show visual results for all eight datasets—our method can find accurate dense correspondences in the presence of significant appearance and pose diversity. GANgealing accurately handles diverse morphologies of birds, cats with varying facial expressions and bikes in different orientations.

Image Editing. Our average congealed image is a template that can propagate any user edit to images of the same category. For example, by drawing cartoon eyes or overlaying a Batman mask on our average congealed cat, a user can effortlessly propagate their edits to massive numbers of cat images with forward passes of T. We show editing results on several datasets in Figures 4.4 and 4.1.

Augmented Reality. Just as we can propagate dense correspondences to images, we can also propagate to individual video frames. Surprisingly, we find that GANgealing yields remarkably smooth and consistent results when applied *out-of-the-box* to videos per-frame without leveraging any temporal information. This enables mixed reality applications like dense tracking and filters. GANgealing can outperform supervised methods like RAFT [132]—please see `www.wpeebles.com/gangealing` for results.

4.4.2 Direct Image-to-Image Correspondence

In addition to propagating correspondences from congealed space to unaligned images, we can also find dense correspondences directly between any pair of images $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$. At a high level, this merely involves applying the forward warp that maps points in $\boldsymbol{x}_A$ to points in $T(\boldsymbol{x}_A)$ and composing it with the reverse warp that maps points in the congealed coordinate space back to $\boldsymbol{x}_B$.

Quantitative Results. We evaluate GANgealing with PCK-Transfer. Given a source image $\boldsymbol{x}_A$, target image $\boldsymbol{x}_B$ and ground-truth keypoints for both images, PCK-Transfer measures the percentage of keypoints transferred from $\boldsymbol{x}_A$ to $\boldsymbol{x}_B$ that lie within a certain radius of the ground-truth keypoints in $\boldsymbol{x}_B$.

We evaluate PCK on SPair-71K [141] and CUB. For SPair, we use the α_{bbox} threshold in keeping with prior works. Under this threshold, a predicted keypoint is deemed to be correctly transferred if it is within a radius $\alpha_{\text{bbox}} \max(H_{\text{bbox}}, W_{\text{bbox}})$ of the ground truth, where H_{bbox} and W_{bbox} are the height and width of the object bounding box in the target image. For each SPair category, we train a StyleGAN2 on the corresponding LSUN category[2]—the GANs are trained on 256×256 center-cropped images. We then train a Spatial Transformer using GANgealing and directly evaluate on SPair. For CUB, we first pre-train a StyleGAN2 with ADA [196] on the NABirds dataset [197] and fine-tune it with FreezeD [198] on the training split of CUB, using the same image pre-processing and dataset splits as ACSM [5] for a fair comparison. When T performs a horizontal flip for one image in a pair, we permute our model's predictions for keypoints with a left versus right distinction.

SPair-71K Results. We compare against several self-supervised and state-of-the-art supervised methods on the challenging SPair-71K dataset in Table 4.1, using the standard $\alpha_{\text{bbox}} = 0.1$ threshold. Our method significantly outperforms prior self-supervised methods on several categories, nearly doubling the best prior self-supervised method's PCK on SPair Bicycles and Cats. *GANgealing performs on par with and even outperforms state-of-the-art correspondence-supervised methods on several categories.* We increase the previous best PCK on Bicycles achieved by Cost Aggregation Transformers [127] from 34.7% to 37.5% and perform comparably on Cats and TVs.

High-Precision SPair-71K Results. The usual $\alpha_{\text{bbox}} = 0.1$ threshold reported by most papers using SPair deems a correspondence correct if it is localized within

[2]We use off-the-shelf StyleGAN2 models for LSUN Cats, Dogs and Horses. Note that we do not evaluate PCK on our clustering models (LSUN Cars and Horses) as these models can only transfer points between images in the same cluster.

Method	Correspondence Supervision	SPair-71K Category			
		Bicycle	Cat	Dog	TV
HPF [125]	matching pairs + keypoints	18.9	52.9	32.8	35.6
DHPF [126]	matching pairs + keypoints	23.8	61.6	46.1	46.5
SCOT [123]	matching pairs + keypoints*	20.7	63.1	42.5	40.8
CHM [129]	matching pairs + keypoints	29.3	64.9	56.1	55.6
CATs [127]	matching pairs + keypoints	34.7	66.5	56.5	58.0
WeakAlign [121]	matching image pairs	17.6	31.8	22.6	35.1
NC-Net [122]	matching image pairs	12.2	39.2	18.8	31.1
CNNgeo [120]	self-supervised	16.7	32.7	22.8	34.1
A2Net [124]	self-supervised	18.5	35.6	**24.3**	36.5
GANgealing	**GAN-supervised**	**37.5**	**67.0**	23.1	**57.9**

Table 4.1: **PCK-Transfer@$\alpha_{\text{bbox}} = 0.1$ results on SPair-71K categories** (test split).

roughly 10 to 20 pixels of the ground truth for 256×256 images (depending on the SPair category). In Figure 4.5, we evaluate performance over a range of thresholds between 0.1 and 0.01 (the latter of which affords a roughly 1 to 2 pixel error tolerance, again depending on category). GANgealing outperforms all supervised methods at these high-precision thresholds across all four categories tested. Notably, our LSUN Cats model improves the previous best SPair Cats PCK@$\alpha_{\text{bbox}} = 0.01$ achieved by SCOT [123] from 5.4% to 18.5%. On SPair TVs, we improve the best supervised PCK achieved by Dynamic Hyperpixel Flow [126] from 2.1% to 3.0%. Even on SPair Dogs, where GANgealing is outperformed by every supervised method at low-precision thresholds, we marginally outperform all baselines at the 0.01 threshold.

CUB Results. Table 4.2 shows PCK results on CUB, comparing against several 2D and 3D correspondence methods that use varying amounts of supervision. GANgealing achieves 57.5% PCK, outperforming all past methods that require instance mask supervision and performing comparably with the best correspondence-supervised baseline (58.5%).

Ablations. We ablate several components of GANgealing in Table 4.3. We find that learning the target mode $\mathbf{c}$ is critical for complex datasets; fixing $\mathbf{c} = \bar{\mathbf{w}}$ dramatically degrades PCK from 67% to 10.6% for our LSUN Cats model. This highlights the value of our GAN-Supervised Learning framework where *both the discriminative model and targets are learned jointly*. We additionally find that our baseline inspired by traditional Congealing (using a single learned target $G(\mathbf{c})$ for

Figure 4.6: **GANgealing alignment improves downstream GAN training**. We show random, untruncated samples from StyleGAN2 trained on LSUN Cats versus our aligned LSUN Cats (both models trained from scratch). Our method improves visual fidelity.

all inputs) is highly unstable and degrades PCK to as little as 7.7%. This result demonstrates the importance of our *per-input* alignment targets. We also ablate two choices of the perceptual loss ℓ: an off-the-shelf supervised option (LPIPS [199]) and a fully-unsupervised VGG-16 [200] pre-trained with SimCLR [201] on ImageNet-1K [202] (SSL)—there is no significant difference in performance between the two ($\pm 0.2\%$). Please see Table 4.3 for more ablations.

4.4.3 Automated GAN Dataset Pre-Processing

An exciting application of GANgealing is automated dataset pre-processing. Dataset alignment is an important yet costly step for many machine learning methods. GAN training in particular benefits from carefully-aligned and filtered datasets, such as FFHQ [57], AFHQ [133] and CelebA-HQ [134]. We can align input datasets using our similarity Spatial Transformer T to train generators with higher visual fidelity. We show results in Figure 4.6: training StyleGAN2 from scratch with our learned pre-processing of LSUN Cats yields high-quality samples reminiscent of AFHQ. Empirically, we find that our pre-processing accelerates GAN training significantly.

Method	Supervision Required		PCK@0.1
	Inst. Mask	Keypoints	
Rigid-CSM (with keypoints) [203]	✓	✓	45.8
ACSM (with keypoints) [5]	✓	✓	51.0
IMR (with keypoints) [204]	✓	✓	58.5
Dense Equivariance [205]	✓		33.5
Rigid-CSM [203]	✓		36.4
ACSM [5]	✓		42.6
IMR [204]	✓		53.4
Neural Best Buddies [128]			35.1
Neural Best Buddies (with flip heuristic)			37.8
GANgealing			**57.5**

Table 4.2: **PCK-Transfer@0.1 on CUB**. Numbers for the 3D methods are reported from [5]. We sample 10,000 random pairs from the CUB validation split as in [5].

Ablation Description	Loss (ℓ)	$\mathcal{W}^+$ cutoff	λ_{TV}	N	PCK
Don't learn $\mathbf{c}$ (fix $\mathbf{c} = \bar{\mathbf{w}}$)	SSL	5	1000	0	10.6
Unconstrained $\mathbf{c}$ optimization	SSL	5	1000	512	0.34
Early style mixing cutoff	SSL	4	1000	1	60.5
Late style mixing cutoff	SSL	6	1000	1	65.0
No style mixing	SSL	14	1000	1	25.9
No style mixing (LPIPS)	LPIPS	14	1000	1	7.74
No $\mathcal{L}_{TV}$ regularizer	SSL	5	0	1	59.0
Lower λ_{TV} (LPIPS)	LPIPS	5	1000	1	66.7
Complete model (SSL)	SSL	5	1000	1	**67.2**
Complete model (LPIPS)	LPIPS	5	2500	1	**67.0**

Table 4.3: **GANgealing ablations for LSUN Cats**. We evaluate on SPair-71K Cats using $\alpha_{\text{bbox}} = 0.1$. SSL refers to using a self-supervised VGG-16 as the perceptual loss ℓ. N refers to the number of $\mathcal{W}$ space PCA coefficients learned when optimizing $\mathbf{c}$. Note that the LSUN Cats StyleGAN2 generator has 14 layers.

Figure 4.7: **Various failure modes**: significant out-of-plane rotation and complex poses poorly modeled by GANs.

4.5 Limitations and Discussion

Our Spatial Transformer has a few notable failure modes as demonstrated in Figure 4.7. One limitation with GANgealing is that we can only reliably propagate correspondences that are visible in our learned target mode. For example, the learned mode of our LSUN Dogs model is the upper-body of a dog—this particular model is thus incapable of finding correspondences between, e.g., paws. A potential solution to this problem is to initialize the learned mode with a user-chosen image via GAN inversion that covers all points of interest. Despite this limitation, we obtain competitive results on SPair for some categories where many keypoints are not visible in the learned mode (e.g., cats).

In this section, we showed that GANs can be used to train highly competitive dense correspondence algorithms from scratch with our proposed GAN-Supervised Learning framework. We hope this work will lead to increased adoption of GAN-Supervision for other challenging tasks.

Chapter 5

Discussion

In this thesis, we introduced a scalable new generative modeling framework and explored ways to leverage pre-trained generative models to tackle problems in vision and meta-learning. To wrap-up, we discuss a few potential further avenues for research.

Generative Models as Broadly-Useful, Generic Meta-Learners While we found that loss-conditional diffusion models can serve as a powerful class of learned optimizers, our initial experiments in this direction considered single-architecture and single-task settings. In order to make this method broadly useful, a single diffusion model should be trained to optimize any downstream neural network for any task. The two main obstacles to realizing this goal are data collection and model scale. It is unclear if there are currently a sufficient number of neural network checkpoints floating around the internet to train such a model, and how large the resulting diffusion model would need to be to successfully optimize, e.g., 175 billion parameter downstream language models. However, there are many reasons to believe this is a promising research direction. Training remains the most resource-intensive aspect of deep learning pipelines, and future training runs do not benefit from the massive quantity of intermediate data generated from prior runs. Learned optimizers that can leverage their past experiences are clearly the solution to this problem. Second, generative models have been shown to work exceptionally well across a breadth of domains—provided sufficient scale and data. History suggests the same will hold true for generative models of neural networks.

The Scaling Behavior of Synthetic Datasets from Generative Models Although we've shown that pre-trained image models are capable of generating high quality training data for a downstream task-specific network, it remains to be shown

how such a setup can most effectively be scaled. In this regime, both the downstream task-level model as well as the dataset-producing generative model can be scaled. Understanding how to allocate a fixed compute budget to scaling the two models is an important direction for future work in this space. Similarly, future work should study how quality metrics of the generative model (like FID, training loss, etc.) translate to performance of the downstream task-specific model.

Formal Scaling Laws for Image Synthesis and Meta-Learning In this thesis, we quantified the compute-based scaling behavior of Diffusion Transformer (DiT) models in a couple of modalities. While initial results are promising, data points at larger Gflop scale may be needed before reliable scaling laws can be developed. In general, a promising avenue for future work is scaling-up DiT by several additional orders of magnitude.

Bibliography

[1] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *NeurIPS*, 2018.

[2] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *ECCV*, 2016.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2020.

[4] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," in *NeurIPS*, 2021.

[5] N. Kulkarni, A. Gupta, D. F. Fouhey, and S. Tulsiani, "Articulation-aware canonical surface mapping," in *CVPR*, 2020.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," in *NeurIPS*, 2020.

[7] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv:2001.08361*, 2020.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.

[10] 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," *arXiv:2205.11487*, 2022.

[11] 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.

[12] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *ICML*, 2015.

[13] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *NeurIPS*, 2020.

[14] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv:2204.06125*, 2022.

[15] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *NeurIPS*, 2016.

[16] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[17] 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.

[18] W. Peebles and S. Xie, "Scalable diffusion models with transformers," *arXiv preprint arXiv:2212.09748*, 2022.

[19] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HCT*, 2019.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," 2019.

[22] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *ICML*, 2020.

[23] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *ICML*, 2021.

[24] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[25] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "PixelCNN++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[26] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," *Advances in neural information processing systems*, vol. 29, 2016.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[28] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[29] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," in *AAAI*, 2018.

[30] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *CVPR*, 2022.

[31] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," in *NeurIPS*, 2021.

[32] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *NeurIPS*, 2021.

[33] W. Peebles, I. Radosavovic, T. Brooks, A. Efros, and J. Malik, "Learning to learn with generative models of neural network checkpoints," *arXiv preprint arXiv:2209.12892*, 2022.

[34] T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, J. Jackson, H. Jun, T. B. Brown, P. Dhariwal, S. Gray, *et al.*, "Scaling laws for autoregressive generative modeling," *arXiv preprint arXiv:2010.14701*, 2020.

[35] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling vision transformers," in *CVPR*, 2022.

[36] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *International conference on machine learning*, pp. 4055–4064, PMLR, 2018.

[37] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[38] A. Van Den Oord, O. Vinyals, *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[39] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," 2020.

[40] H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman, "Maskgit: Masked generative image transformer," in *CVPR*, pp. 11315–11325, 2022.

[41] S. Gu, D. Chen, J. Bao, F. Wen, B. Zhang, D. Chen, L. Yuan, and B. Guo, "Vector quantized diffusion model for text-to-image synthesis," in *CVPR*, pp. 10696–10706, 2022.

[42] J. Yu, Y. Xu, J. Y. Koh, T. Luong, G. Baid, Z. Wang, V. Vasudevan, A. Ku, Y. Yang, B. K. Ayan, *et al.*, "Scaling autoregressive models for content-rich text-to-image generation," *arXiv:2206.10789*, 2022.

[43] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *ICML*, 2021.

[44] A. Hyvärinen and P. Dayan, "Estimation of non-normalized statistical models by score matching.," *Journal of Machine Learning Research*, vol. 6, no. 4, 2005.

[45] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," in *NeurIPS*, 2019.

[46] 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," *arXiv:2112.10741*, 2021.

[47] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv:2010.02502*, 2020.

[48] T. Karras, M. Aittala, T. Aila, and S. Laine, "Elucidating the design space of diffusion-based generative models," in *Proc. NeurIPS*, 2022.

[49] J. Ho and T. Salimans, "Classifier-free diffusion guidance," in *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[50] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans, "Cascaded diffusion models for high fidelity image generation," *arXiv:2106.15282*, 2021.

[51] A. Jabri, D. Fleet, and T. Chen, "Scalable adaptive computation for iterative generation," *arXiv preprint arXiv:2212.11972*, 2022.

[52] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollár, "On network design spaces for visual recognition," in *ICCV*, 2019.

[53] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *CVPR*, 2020.

[54] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *ICML*, 2021.

[55] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[56] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *ICLR*, 2019.

[57] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *CVPR*, 2019.

[58] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv:1706.02677*, 2017.

[59] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv:1711.05101*, 2017.

[60] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[61] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer, "How to train your ViT? data, augmentation, and regularization in vision transformers," *TMLR*, 2022.

[62] T. Xiao, P. Dollar, M. Singh, E. Mintun, T. Darrell, and R. Girshick, "Early convolutions help transformers see better," in *NeurIPS*, 2021.

[63] 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," 2017.

[64] G. Parmar, R. Zhang, and J.-Y. Zhu, "On aliased resizing and surprising subtleties in gan evaluation," in *CVPR*, 2022.

[65] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training GANs," in *NeurIPS*, 2016.

[66] C. Nash, J. Menick, S. Dieleman, and P. W. Battaglia, "Generating images with sparse representations," *arXiv preprint arXiv:2103.03841*, 2021.

[67] T. Kynkäänniemi, T. Karras, S. Laine, J. Lehtinen, and T. Aila, "Improved precision and recall metric for assessing generative models," in *NeurIPS*, 2019.

[68] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[69] A. Sauer, K. Schwarz, and A. Geiger, "Stylegan-xl: Scaling stylegan to large diverse datasets," in *SIGGRAPH*, 2022.

[70] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, 1951.

[71] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio.," *SSW*, 2016.

[72] J. Schmidhuber, "Evolutionary principles in self-referential learning," 1987.

[73] Y. Bengio, S. Bengio, and J. Cloutier, "Learning a synaptic learning rule," in *IJCNN*, 1991.

[74] K. Li and J. Malik, "Learning to optimize," *arXiv:1606.01885*, 2016.

[75] G. Roeder, L. Metz, and D. Kingma, "On linear identifiability of learned representations," in *ICML*, 2021.

[76] K. Schürholt, D. Kostadinov, and D. Borth, "Self-supervised representation learning on neural network weights for model characteristic prediction," *NeurIPS*, 2021.

[77] 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.

[78] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," in *NeurIPS*, 2021.

[79] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[80] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.

[81] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv:1312.6120*, 2013.

[82] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.

[83] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, "Evaluating large language models trained on code," *arXiv:2107.03374*, 2021.

[84] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, *et al.*, "Competition-level code generation with alphacode," *arXiv:2203.07814*, 2022.

[85] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, 2021.

[86] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv:2005.00341*, 2020.

[87] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *ICML*, 2019.

[88] B. Knyazev, M. Drozdzal, G. W. Taylor, and A. Romero-Soriano, "Parameter prediction for unseen deep architectures," in *NeurIPS*, 2021.

[89] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," *arXiv:1811.10959*, 2018.

[90] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu, "Dataset distillation by matching training trajectories," in *CVPR*, 2022.

[91] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *ICANN*, 2001.

[92] A. S. Younger, S. Hochreiter, and P. R. Conwell, "Meta-learning with back-propagation," in *IJCNN*, 2001.

[93] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *ICML*, 2010.

[94] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *ICLR*, 2017.

[95] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *ICML*, 2017.

[96] K. Lv, S. Jiang, and J. Li, "Learning gradient descent: Better generalization and longer horizons," in *ICML*, 2017.

[97] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein, "Understanding and correcting pathologies in the training of learned optimizers," in *ICML*, 2019.

[98] L. Metz, C. D. Freeman, J. Harrison, N. Maheswaranathan, and J. Sohl-Dickstein, "Practical tradeoffs between memory, compute, and performance in learned optimizers," *arXiv:2203.11860*, 2022.

[99] K. Li and J. Malik, "Learning to optimize neural nets," *arXiv:1703.00441*, 2017.

[100] J. Schmidhuber, "Learning to control fast-weight memories: An alternative to dynamic recurrent networks," *Neural Computation*, 1992.

[101] J. Schmidhuber, "A 'self-referential' weight matrix," in *ICANN*, 1993.

[102] G. E. Hinton and D. C. Plaut, "Using fast weights to deblur old memories," in *In Proceedings of the 9th Annual Conference of the Cognitive Science Society*, 1987.

[103] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *ICLR*, 2017.

[104] D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville, "Bayesian hypernetworks," *arXiv:1710.04759*, 2017.

[105] L. Deutsch, E. Nijkamp, and Y. Yang, "A generative model for sampling high-performance and diverse weights for neural networks," *arXiv preprint arXiv:1905.02898*, 2019.

[106] N. Ratzlaff and L. Fuxin, "Hypergan: A generative model for diverse, performant neural networks," in *ICML*, 2019.

[107] C. Finn, K. Xu, and S. Levine, "Probabilistic model-agnostic meta-learning," in *NeurIPS*, 2018.

[108] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *NeurIPS*, 2011.

[109] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated machine learning*, 2019.

[110] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *ICML*, 2015.

[111] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, "Online learning rate adaptation with hypergradient descent," *arXiv:1703.04782*, 2017.

[112] H. Drucker and Y. Le Cun, "Improving generalization performance using double backpropagation," *IEEE Transactions on Neural Networks*, 1992.

[113] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *NeurIPS*, 2012.

[114] C. Daniel, J. Taylor, and S. Nowozin, "Learning step size controllers for robust neural network training," 2016.

[115] C. Xu, T. Qin, G. Wang, and T.-Y. Liu, "Reinforcement learning for learning rate control," *arXiv:1705.11159*, 2017.

[116] Z. Xu, A. M. Dai, J. Kemp, and L. Metz, "Learning an adaptive learning rate schedule," *arXiv:1909.09712*, 2019.

[117] D. Almeida, C. Winter, J. Tang, and W. Zaremba, "A generalizable approach to learning optimizers," *arXiv:2106.00958*, 2021.

[118] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman, "Gradients are not all you need," *arXiv:2111.05803*, 2021.

[119] W. Peebles, J.-Y. Zhu, R. Zhang, A. Torralba, A. Efros, and E. Shechtman, "Gan-supervised dense visual alignment," in *CVPR*, 2022.

[120] I. Rocco, R. Arandjelovic, and J. Sivic, "Convolutional neural network architecture for geometric matching," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6148–6157, 2017.

[121] I. Rocco, R. Arandjelović, and J. Sivic, "End-to-end weakly-supervised semantic alignment," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6917–6925, 2018.

[122] I. Rocco, M. Cimpoi, R. Arandjelović, A. Torii, T. Pajdla, and J. Sivic, "Neighbourhood consensus networks," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[123] Y. Liu, L. Zhu, M. Yamada, and Y. Yang, "Semantic correspondence as an optimal transport problem," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[124] P. H. Seo, J. Lee, D. Jung, B. Han, and M. Cho, "Attentive semantic alignment with offset-aware correlation kernels," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 349–364, 2018.

[125] J. Min, J. Lee, J. Ponce, and M. Cho, "Hyperpixel flow: Semantic correspondence with multi-layer neural features," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3395–3404, 2019.

[126] J. Min, J. Lee, J. Ponce, and M. Cho, "Learning to compose hypercolumns for visual correspondence," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV 16*, pp. 346–363, Springer, 2020.

[127] S. Cho, S. Hong, S. Jeon, Y. Lee, K. Sohn, and S. Kim, "Cats: Cost aggregation transformers for visual correspondence," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[128] K. Aberman, J. Liao, M. Shi, D. Lischinski, B. Chen, and D. Cohen-Or, "Neural best-buddies: Sparse cross-domain correspondence," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 69, 2018.

[129] J. Min and M. Cho, "Convolutional hough matching networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2940–2950, June 2021.

[130] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2758–2766, 2015.

[131] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2462–2470, 2017.

[132] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *European Conference on Computer Vision*, pp. 402–419, Springer, 2020.

[133] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha, "Stargan v2: Diverse image synthesis for multiple domains," in *CVPR*, 2020.

[134] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," in *ICLR*, 2018.

[135] E. G. Learned-Miller, "Data driven image models through continuous joint alignment," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 2, pp. 236–250, 2006.

[136] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," pp. 2017–2025, 2015.

[137] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[138] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *ICCV*, December 2015.

[139] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011.

[140] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pp. 3686–3693, 2014.

[141] J. Min, J. Lee, J. Ponce, and M. Cho, "Spair-71k: A large-scale benchmark for semantic correspondence," *arXiv preprint arXiv:1908.10543*, 2019.

[142] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *NeurIPS*, 2014.

[143] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *ICLR*, 2016.

[144] L. Chai, J.-Y. Zhu, E. Shechtman, P. Isola, and R. Zhang, "Ensembling with deep generative views," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14997–15007, 2021.

[145] C. Mao, A. Cha, A. Gupta, H. Wang, J. Yang, and C. Vondrick, "Generative interventions for causal learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3947–3956, 2021.

[146] V. Besnier, H. Jain, A. Bursuc, M. Cord, and P. Pérez, "This dataset does not exist: training models from generated images," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, IEEE, 2020.

[147] F. H. K. d. S. Tanaka and C. Aranha, "Data augmentation using gans," *arXiv preprint arXiv:1904.09135*, 2019.

[148] E. Wu, K. Wu, D. Cox, and W. Lotter, "Conditional infilling gans for data augmentation in mammogram classification," in *Image analysis for moving organ, breast, and thoracic images*, pp. 98–106, Springer, 2018.

[149] A. Voynov, S. Morozov, and A. Babenko, "Big gans are watching you: Towards unsupervised object segmentation with off-the-shelf generative models," *arXiv preprint arXiv:2006.04988*, 2020.

[150] L. Melas-Kyriazi, C. Rupprecht, I. Laina, and A. Vedaldi, "Finding an unsupervised image segmenter in each of your deep generative models," *arXiv preprint arXiv:2105.08127*, 2021.

[151] Y. Zhang, H. Ling, J. Gao, K. Yin, J.-F. Lafleche, A. Barriuso, A. Torralba, and S. Fidler, "Datasetgan: Efficient labeled data factory with minimal human effort," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10145–10155, 2021.

[152] N. Tritrong, P. Rewatbowornwong, and S. Suwajanakorn, "Repurposing gans for one-shot semantic part segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4475–4485, 2021.

[153] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," in *ICLR*, 2017.

[154] J. Donahue and K. Simonyan, "Large scale adversarial representation learning," *arXiv preprint arXiv:1907.02544*, 2019.

[155] A. Jahanian, X. Puig, Y. Tian, and P. Isola, "Generative models as a data source for multiview representation learning," *arXiv preprint arXiv:2106.05258*, 2021.

[156] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville, "Adversarially learned inference," in *ICLR*, 2017.

[157] M. Baradad, J. Wulff, T. Wang, P. Isola, and A. Torralba, "Learning to see by looking at noise," *arXiv preprint arXiv:2106.05963*, 2021.

[158] Y. Shi, D. Aggarwal, and A. K. Jain, "Lifting 2d stylegan for 3d-aware face generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6258–6266, 2021.

[159] X. Pan, B. Dai, Z. Liu, C. C. Loy, and P. Luo, "Do 2d gans know 3d shape? unsupervised 3d shape reconstruction from 2d image gans," in *International Conference on Learning Representations*, 2021.

[160] Y. Zhang, W. Chen, H. Ling, J. Gao, Y. Zhang, A. Torralba, and S. Fidler, "Image gans meet differentiable rendering for inverse graphics and interpretable 3d neural rendering," *arXiv preprint arXiv:2010.09125*, 2020.

[161] Z. Hao, A. Mallya, S. Belongie, and M.-Y. Liu, "GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds," in *ICCV*, 2021.

[162] A. Torralba, "http://people.csail.mit.edu/torralba/gallery/," 2001.

[163] J.-Y. Zhu, Y. J. Lee, and A. A. Efros, "Averageexplorer: Interactive exploration and alignment of visual data collections," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–11, 2014.

[164] G. B. Huang, V. Jain, and E. Learned-Miller, "Unsupervised joint alignment of complex images," in *2007 IEEE 11th International Conference on Computer Vision*, 2007.

[165] Y. Peng, A. Ganesh, J. Wright, W. Xu, and Y. Ma, "RASL: robust alignment by sparse and low-rank decomposition for linearly correlated images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2233–2246, 2012.

[166] I. Kemelmacher-Shlizerman and S. M. Seitz, "Collection flow," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1792–1799, IEEE, 2012.

[167] H. Mobahi, C. Liu, and W. T. Freeman, "A compositional model for low-dimensional image set representation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pp. 1322–1329, IEEE Computer Society, 2014.

[168] T. Zhou, Y. J. Lee, S. Yu, and A. A. Efros, "Flowweb: Joint image set alignment by weaving consistent, pixel-wise correspondences," in *CVPR*, 2015.

[169] S. Divvala, A. Efros, and M. Hebert, "Object instance sharing by enhanced bounding box correspondence," in *BMVC*, 2012.

[170] J. B. Frey and N. Jojic, "Estimating mixture models of images and inferring spatial transformations using the em algorithm," in *CVPR*, 1999.

[171] B. J. Frey and N. Jojic, "Transformation-invariant clustering using the EM algorithm," vol. 25, no. 1, pp. 1–17, 2003.

[172] M. A. Mattar, A. R. Hanson, and E. G. Learned-Miller, "Unsupervised joint alignment and clustering using bayesian nonparametrics," in *UAI*, 2012.

[173] T. Monnier, T. Groueix, and M. Aubry, "Deep transformation-invariant clustering," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 7945–7955, Curran Associates, Inc., 2020.

[174] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," *arXiv preprint arXiv:1703.06211*, 2017.

[175] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool, "Dynamic filter networks," in *Advances in Neural Information Processing Systems*, pp. 667–675, 2016.

[176] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, "View synthesis by appearance flow," *ECCV*, 2016.

[177] Y. Ganin, D. Kononenko, D. Sungatullina, and V. Lempitsky, "Deepwarp: Photorealistic image resynthesis for gaze manipulation," in *European Conference on Computer Vision*, pp. 311–326, Springer, 2016.

[178] E. Park, J. Yang, E. Yumer, D. Ceylan, and A. C. Berg, "Transformation-grounded image generation network for novel 3d view synthesis," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[179] A. Kanazawa, D. W. Jacobs, and M. Chandraker, "Warpnet: Weakly supervised matching for single-view reconstruction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3253–3261, 2016.

[180] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee, "Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision," in *Advances in Neural Information Processing Systems*, pp. 1696–1704, 2016.

[181] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," *arXiv preprint arXiv:1704.07813*, 2017.

[182] C.-H. Lin and S. Lucey, "Inverse compositional spatial transformer networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[183] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pp. 674–679, 1981.

[184] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.

[185] X. Xing, R. Gao, T. Han, S.-C. Zhu, and Y. N. Wu, "Deformable generator network: Unsupervised disentanglement of appearance and geometry," *arXiv preprint arXiv:1806.06298*, 2018.

[186] C.-H. Lin, E. Yumer, O. Wang, E. Shechtman, and S. Lucey, "St-gan: Spatial transformer generative adversarial networks for image compositing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9455–9464, 2018.

[187] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110–8119, 2020.

[188] R. Abdal, Y. Qin, and P. Wonka, "Image2stylegan: How to embed images into the stylegan latent space?," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4432–4441, 2019.

[189] E. Härkönen, A. Hertzmann, J. Lehtinen, and S. Paris, "Ganspace: Discovering interpretable gan controls," *arXiv preprint arXiv:2004.02546*, 2020.

[190] Y. Tian, J. Ren, M. Chai, K. Olszewski, X. Peng, D. N. Metaxas, and S. Tulyakov, "A good image generator is what you need for high-resolution video synthesis," in *International Conference on Learning Representations*, 2021.

[191] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, "Generative visual manipulation on the natural image manifold," in *ECCV*, 2016.

[192] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Neural photo editing with introspective adversarial networks," in *ICLR*, 2017.

[193] D. Bau, H. Strobelt, W. Peebles, J. Wulff, B. Zhou, J. Zhu, and A. Torralba, "Semantic photo manipulation with a generative image prior," *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, vol. 38, no. 4, 2019.

[194] Y. Alaluf, O. Patashnik, and D. Cohen-Or, "Restyle: A residual-based stylegan encoder via iterative refinement," *arXiv preprint arXiv:2104.02699*, 2021.

[195] M. Huh, R. Zhang, J.-Y. Zhu, S. Paris, and A. Hertzmann, "Transforming and projecting images to class-conditional generative networks," in *ECCV*, 2020.

[196] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, "Training generative adversarial networks with limited data," *arXiv*, 2020.

[197] G. Van Horn, S. Branson, R. Farrell, S. Haber, J. Barry, P. Ipeirotis, P. Perona, and S. Belongie, "Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 595–604, 2015.

[198] S. Mo, M. Cho, and J. Shin, "Freeze the discriminator: a simple baseline for fine-tuning gans," *arXiv preprint arXiv:2002.10964*, 2020.

[199] 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.

[200] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2014.

[201] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, PMLR, 2020.

[202] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[203] N. Kulkarni, A. Gupta, and S. Tulsiani, "Canonical surface mapping via geometric cycle consistency," *ICCV*, 2019.

[204] S. Tulsiani, N. Kulkarni, and A. Gupta, "Implicit mesh reconstruction from unannotated image collections," 2020.

[205] J. Thewlis, A. Vedaldi, and H. Bilen, "Unsupervised object learning from dense equivariant image labelling," 2017.