Learning to Imitate Datasets by Long-Range Matching of Expert Training Trajectories

George Cazenavette

CMU-RI-TR-22-45

August 1, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Jun-Yan Zhu, co-chair
Simon Lucey, co-chair
Vincent Sitzmann, (MIT)
Nathaniel Chodosh

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Copyright © 2022 George Cazenavette. All rights reserved.
To Ashton, James, Matt, Max, and Pablo.
Abstract

Dataset distillation is the task of synthesizing a small dataset such that a model trained on the synthetic set will match the test accuracy of the model trained on the full dataset. In this thesis, we present 2 methods. First, our base method of Matching Training Trajectories: given a network, we train it for several iterations on our distilled data and optimize the distilled data with respect to the distance between the synthetically trained parameters and the parameters trained on real data. To efficiently obtain the initial and target network parameters for large-scale datasets, we pre-compute and store training trajectories of expert networks trained on the real dataset. Our method handily outperforms existing methods and also allows us to distill higher-resolution visual data. This method can also be augmented to produce infinitely tileable class-based textures.

Our second method introduces a Deep Generative Prior to the distillation process. Rather than directly optimizing pixels, this method now distills into the latent space of a pre-trained generative model. This greatly reduces overfitting to the backbone model and allows our distilled data to generalize much better to unseen architectures.
Acknowledgments

Simon, my first advisor, for getting me started in my research career. Calvin, for helping me write my first paper. Jun-Yan, for taking me on as a third-year masters student. Tongzhou, Alyosha, and Antonio, for being fantastic collaborators. Kangle, Sheng-Yu, Gaurav, Nupur, Ruihan, and Andrew, for being great lab-mates and friends.
Funding

Throughout his masters studies, George Cazenavette was supported by the NSF Graduate Research Fellowship under Grant No. DGE1745016
## Contents

1 Dataset Distillation 1  
1.1 Motivation 1  
1.2 Problem Setting 3  
1.3 The First Dataset Distillation Method 3  
1.4 Gradient Matching 5  
1.5 Differentiable Siamese Augmentation 6  
1.6 Distribution Matching 6  
1.7 Kernel Inducing Points 7  

2 Matching Training Trajectories 8  
2.1 Introduction 8  
2.2 Expert Trajectories 10  
2.3 Long-Range Parameter Matching 10  
2.4 Memory Constraints 13  
2.5 Experiments 13  
  2.5.1 Low-Resolution Data (32×32) 15  
  2.5.2 Short-Range vs. Long-Range Matching 17  
  2.5.3 Tiny ImageNet (64×64) 20  
  2.5.4 ImageNet Subsets (128×128) 21  
2.6 Discussion and Limitations 22  

3 Tileable Texture Synthesis 24  
3.1 Introduction 24  
3.2 Preliminaries 26  
3.3 Tileable Texture Distillation 26  
3.4 Experiments 28  
  3.4.1 Results 30  
3.5 Discussion 30  

4 Generative Latent Distillation 32  
4.1 Introduction 32  
4.2 Method 35  
  4.2.1 Matching Training Trajectories 36  
  4.2.2 Deep Generative Prior 36
# List of Figures

1.1 Illustration of the Dataset Distillation problem. Large datasets are distilled into small synthetic sets such that a model trained on the synthetic set will have similar test performance as a model trained on the full real dataset.  

2.1 We perform long-range parameter matching between training on distilled synthetic data and training on real data. Starting from the same initial parameters, we train distilled data $S$ such that $N$ training steps on them match the same result (in parameter space) from much more $M$ steps on real data.  

2.2 CIFAR-10: The 1 image per class images are more abstract but also more information-dense while the 10 images per class images are more expressive and contain more structure.  

2.3 CIFAR-100 (1 image / class). Left: Smaller $M$ and $N$ match shorter-range behavior, which performs worse than longer-range matching. Right: Over 1000 training steps on distilled data, we track the closest distance in parameter space (normalized MSE in Eqn. 2.2) to a target set of parameters, obtained with $\Delta_t$ training steps on real data. Matching long-range behavior, our method better approximate real data training for longer ranges ($\Delta_t$).  

2.4 Selected samples distilled from Tiny ImageNet, one image per class. Despite the higher resolution, our method still produces high-fidelity images. (Can you guess which classes these images represent? Check your answers in the footnote!\textsuperscript{1})  

2.5 Our method is the first capable of distilling higher-resolution ($128 \times 128$) images, allowing us to explore the ImageNet [15] dataset.  

3.1 Our method takes random crops of a padded distilled canvas instead of discrete images. Note: Our method distills all classes simultaneously, although only one (penguin) is shown in detail in this figure for simplicity.  

3.2 Our new method builds upon recent dataset distillation work [9] and lets us distill classes into tilable textures. These textures can then be used for downstream tasks, such as pattern swatches for clothing. (Visualizations made with FAB3D: https://fabric.tri3d.in/)
3.3 Traditional distillation methods take the real training data (A) and create disjoint synthetic training images (B). Our new method builds upon recent dataset distillation work [9] and allows for the synthesis of class-based tileable textures (C).

3.4 We initialize our synthetic canvases from Gaussian noise. Structure emerges after several hundred iterations, but it takes tens of thousands until convergence.

3.5 ImageSquawk Distilled Textures Tiled 3x3

3.6 By distilling classes into tileable textures, we can apply our synthetic data to cases where such tileability is required, such as fabric pattern swatches. From left to right, we see our flamingo, eagle, macaw, and penguin textures applied to clothing using FAB3D [1].

4.1 Rather than directly distilling a dataset into synthetic pixels (like all previous methods), our new work instead distills into the latent space of a deep generative prior. This enforces a tune-able amount of coherence in the output synthetic images, leading to far better generalization to new architectures.

4.2 Visualizations of all current methods capable of distilling high-resolution datasets applied to the Birds-10 subset of ImageNet. We encourage the reader to zoom in and notice the absence of high-frequency noise in our method (c) compared to MTT (b).

4.3 Different distillation optimization spaces induce different levels of generative priors. Using spaces of earlier layers (left side) imposes stronger priors and yields more realistic images, but is not as expressive as using spaces of later layers (right side). This varying visual granularity can be seen in the both the final distilled images (bottom row) and the initialization of these images (second bottom row). A proper amount of generative prior is helpful for cross-architecture generalization, while too strong a prior limits expressivity and thus hurts distillation performance (top bar chart). An intermediate space (e.g., F20) balances this trade-off.

4.4 ImageNet Birds-10 distilled using just one expert trajectory. Our standard F20 space (a) handily outperforms MTT (b) on both backbone and cross-architecture accuracy. Using a stronger generative prior of F8 space (c) yields even better backbone and cross-architecture performance in the single-expert setting.
“Macaw” distilled from 512×512 ImageNet Birds-10. While MTT can still form somewhat coherent images at 128×128 resolution, 512×512 MTT images are completely littered with high-frequency noise. Conversely, our weak generative prior allows our method to still produce high-fidelity images even at this much higher resolution. (See the rest of the classes in the supplementary material.)

A.1 **Left:** ZCA Ablation. **Right:** Distillation Time.

A.2 **Left:** We see logarithmic performance improvement with respect to the number of expert trajectories used, quickly saturating near 200. **Right:** The upper bound on the expert epoch at which the synthetic data starts working cannot be too high or low to ensure quality learning signal.

A.3 **Left:** Our learned synthetic step size $\alpha$ seems to scale inversely with the number of synthetic steps $N$ to cover the same distance in parameter space as the expert steps $M$. **Right:** Having a learnable step size $\alpha$ saves us from having to search for an appropriate fixed $\alpha_0$.

A.4 CIFAR-100, 1 Image Per Class

A.5 Most-correct distilled images for Tiny ImageNet ($\geq 30\%$)

A.6 Least-correct distilled images for Tiny ImageNet ($\leq 4\%$)

A.7 CIFAR-100, Initialized as Random Noise

A.8 CIFAR-100, No ZCA Whitening

A.9 CIFAR-100, No Differentiable Augmentation

A.10 CIFAR-100, Only 1 Expert Trajectory

A.11 Tiny ImageNet, 1 Image Per Class (Classes 1-100)

A.12 Tiny ImageNet, 1 Image Per Class (Classes 101-200)

A.13 ImageNette, 10 Images Per Class

A.14 ImageWoof, 10 Images Per Class

A.15 ImageSquawk, 10 Images Per Class

A.16 ImageMeow, 10 Images Per Class

A.17 ImageFruit, 10 Images Per Class

A.18 ImageYellow, 10 Images Per Class

B.1 ImageFruit Distilled Textures Tiled 3x3

C.1 High-resolution distillation methods applied to the Fruits-10 subset of ImageNet.

C.2 High-resolution distillation methods applied to the Food-10 subset of ImageNet.

C.3 High-resolution distillation methods applied to the Marine-10 subset of ImageNet.
C.4 High-resolution distillation methods applied to the Herptiles-10 subset of ImageNet. ................................................................. 78
C.5 High-resolution distillation methods applied to the Mammals-10 subset of ImageNet. ................................................................. 79
C.6 High-resolution distillation methods applied to the Cats-10 subset of ImageNet. ................................................................. 79
C.7 High-resolution distillation methods applied to the Earth-10 subset of ImageNet. ................................................................. 80
C.8 High-resolution distillation methods applied to the Sports-10 subset of ImageNet. ................................................................. 80
C.9 High-resolution distillation methods applied to the ImageNette [19] subset of ImageNet. ................................................................. 81
C.10 CIFAR-10 distilled into different spaces with varying levels of prior. Unlike in the 50 images/class setting (Table 4.2), choice of distillation space seems to have a significant impact on CIFAR-10 cross-architecture generalization in the 1 image/class setting. ................................................................. 82
C.11 Birds-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 83
C.12 Fruits-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 84
C.13 Food-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 85
C.14 Marine-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 86
C.15 Herptiles-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 87
C.16 Mammals-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 88
C.17 Cats-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 89
C.18 Earth-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 90
C.19 Sports-10 subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 91
C.20 ImageNette [19] subset of ImageNet distilled into different spaces with varying levels of prior. ................................................................. 92
C.21 Different distillation spaces in the single-expert setting on the Fruits-10 subset of ImageNet. ................................................................. 93
C.22 Different distillation spaces in the single-expert setting on the Food-10 subset of ImageNet. ................................................................. 93
C.23 Different distillation spaces in the single-expert setting on the Marine-10 subset of ImageNet. ........................................... 94
C.24 Different distillation spaces in the single-expert setting on the Herptiles-10 subset of ImageNet. ........................................... 94
C.25 Different distillation spaces in the single-expert setting on the Mammals-10 subset of ImageNet. ........................................... 95
C.26 Different distillation spaces in the single-expert setting on the Cats-10 subset of ImageNet. ........................................... 95
C.27 Different distillation spaces in the single-expert setting on the Earth-10 subset of ImageNet. ........................................... 96
C.28 Different distillation spaces in the single-expert setting on the Sports-10 subset of ImageNet. ........................................... 96
C.30 512×512: Peacock, Flamingo, Pelican ................................... 98
C.31 512×512: King Penguin, Bald Eagle, Toucan .......................... 99
C.32 512×512: Ostrich, Black Swan, Cockatoo ............................ 100
## List of Tables

2.1 Comparing distillation and coreset selection methods. As in previous work, we distill the given number of images per class using the training set, train a neural network on the synthetic set, and evaluate on the test set. To get $\bar{x} \pm s$, we train 5 networks from scratch on the distilled dataset. Note that the earlier works DD$^\dagger$ and LD$^\dagger$ use different architectures, i.e., LeNet [33] for MNIST and AlexNet [31] for CIFAR-10. All others use a 128-width ConvNet. CIFAR values marked by (∗) signify best results were obtained with ZCA whitening. ................................................................. 12

2.2 Kernel Inducing Point (KIP) [36] performs distillation using the infinite-width network limit. We consistently outperform KIP when evaluating on the same finite-width network, and almost always outperform KIP applied on wider networks. ................................................................. 16

2.3 Despite being trained for a specific architecture, our synthetic images do not seem to suffer from much over-fitting to that model. This is evaluated on CIFAR-10 with 10 images per class. ................................................................. 16

2.4 Applying our method to 128×128 resolution ImageNet subsets. On this higher resolution, across various subsets, our method continues to produce high-quality distilled images. ................................................................. 19

4.1 Cross-architecture generalization among dataset distillation methods. Our method reaches comparable accuracy on the backbone architecture used to distill the dataset (ConvNet) with MTT [10] (unlike DM [52]) while also generalizing much better to unseen architectures (averaged across ResNet-18, VGG-11, and ViT). Results reported as $\bar{x} \pm s$ over 5 random initializations. Only our method reaches good performance on both the distilling architecture (ConvNet) and unseen architectures (Cross-Arch.). ................................................................. 41

4.2 Cross-architecture generalization for CIFAR-10, 50 img/class. Images distilled using ConvNetD3 and evaluated on unseen architectures. Our new method generalizes just as well as MTT on CIFAR-10, but we do not see the same gap as with the high-resolution data, implying the generative prior is less critical at low resolution. ................................................................. 42
4.3 Single-expert distillation results. With our generative prior, we significantly improve performance of prior state of the art (MTT) in the challenging single-expert setting (where distillation is performed w.r.t. a fixed initialization but tested on random unseen ones) and thus greatly reduce computation and memory cost required to reach good performance. Compared to the many-expert setting (Table 4.1), this single-expert setting has far less supervision and thus a relatively stronger prior (F8 space) becomes optimal.

A.1 As we ablate our categorical hyper-parameters, we still achieve state-of-the-art performance (compared to DSA: 13.9%). This is evaluated on CIFAR-100 with 1 image per class. Each design choice in our final method improves the performance of distilled images. Here we use the default set of hyper-parameters for these ablations.

A.2 Hyper-parameters used for our best-performing distillation experiments. A synthetic batch size of "-" indicates that we used the full support set at each synthetic step. Note: instead of number of expert updates ($M$), here we list number of expert epochs ($M^*$) for simplicity across datasets.

A.3 Distillation time for each dataset and support size.

A.4 Optimal hyper-parameters for our reported width-128 KIP to NN results. These were obtained via grid search using the notebook provided by the KIP authors.

C.1 Class listings for our ImageNet subsets. Visualizations show classes in the same order given here.

C.2 Whole-dataset validation accuracy for $128 \times 128$ ImageNet subsets. These values represent the upper-bound of the dataset distillation problem given our evaluation training protocol. For consistency, we include “Cross-Arch” here as the average of “ResNet-18,” “VGG-11,” and “ViT-b/16.”
Chapter 1

Dataset Distillation

In this chapter, we introduce the problem of Dataset Distillation and review previously proposed methods.

1.1 Motivation

In the seminal 2015 paper, Hinton et al. [25] proposed model distillation, which aims to distill the knowledge of a complex model into a simpler one. Dataset distillation, proposed by Wang et al. [51], is a related but orthogonal task: rather than distilling the model, the idea is to distill the dataset. As shown in Figure 1.1, the goal is to distill the knowledge from a large training dataset into a very small set of synthetic training images (as low as one image per class) such that training a model on the distilled data would give a similar test performance as training one on the original dataset. Dataset distillation has become a lively research topic in machine learning [4, 35, 36, 44, 52, 53, 54] with various applications, such as continual learning, neural architecture search, and privacy-preserving ML. Still, the problem has so far been of mainly theoretical interest, since most prior methods focus on toy datasets, like MNIST [32] and CIFAR [30], while struggling on real, higher-resolution images. In this work, we present a new approach to dataset distillation that not only outperforms previous work in performance, but is also applicable to large-scale datasets.

Unlike classical data compression, dataset distillation aims for a small synthetic dataset
that still retains adequate task-related information so that models trained on it can generalize to unseen test data, as shown in Figure 1.1. Thus, the distilling algorithm must strike a delicate balance by heavily compressing information without completely obliterating the discriminative features. To do this, dataset distillation methods attempt to discover exactly which aspects of the real data are critical for learning said discrimination. Several methods consider end-to-end training [35, 36, 51] but often require huge compute and memory and suffer from inexact relaxation [35, 36] or training instability of unrolling many iterations [34, 51]. To reduce the optimization difficulty, other methods [53, 54] focus on short-range behavior, enforcing a single training step on distilled data to match that on real data. However, error may accumulate in evaluation, where the distilled data is applied over many steps. We confirm this hypothesis experimentally in Section 2.5.2.

To address the above challenges, we sought to directly imitate the long-range training dynamics of networks trained on real datasets. In particular, we match segments of parameter trajectories trained on synthetic data with segments of pre-recorded trajectories from models trained on real data and thus avoid being short-sighted (i.e., focusing on single steps) or difficult to optimize (i.e., modeling the full trajectories). Treating the real dataset as the gold standard for guiding the network’s training dynamics, we can consider the induced sequence of network parameters to be an expert trajectory. If our distilled dataset were to induce a network’s training dynamics to follow these expert trajectories, then the synthetically trained network would land at a place close to the model trained on real data (in the parameter space) and achieve similar test performance.

In our method, our loss function directly encourages the distilled dataset to guide the network optimization along a similar trajectory (Figure 2.1). We first train a set of models from scratch on the real dataset and record their expert training trajectories. We then initialize a new model with a random time step from a randomly chosen expert trajectory and train for several iterations on the synthetic dataset. Finally, we penalize the distilled data based on how far this synthetically trained network deviated from the expert trajectory and back-propagate through the training iterations. Essentially, we transfer the knowledge from many expert training trajectories to the distilled images.

Extensive experiments show that our method handily outperforms existing dataset distillation methods as well as coreset selection methods on standard datasets, including CIFAR-10, CIFAR-100, and Tiny ImageNet. For example, we achieve 46.3% with a single image per class and 71.5% with 50 images per class on CIFAR-10, compared to the previous
state of the art (28.8% / 63.0% from [52, 53] and 36.1% / 46.5% from [36]). Furthermore, our method also generalizes well to larger data, allowing us to see high $128 \times 128$-resolution images distilled from ImageNet [15] for the first time. Finally, we analyze our method through additional ablation studies and visualizations. Code and models are also available on our webpage.

1.2 Problem Setting

As introduced by Wang et al. [51], the task of “Dataset Distillation” concerns the synthesis of a small, synthetic dataset such that a model trained from scratch to convergence on only this synthetic data will have similar test performance as a model trained on the full, real dataset.

Formally, given a real dataset $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{|T|}, y_{|T|})\}$ made up of $|T|$ data-label pairs, we want to learn a synthetic dataset $S^*$ such that

$$S^* := \arg\min_S \mathbb{E}_{\theta_0 \sim P_\theta_0} \left[ \ell(Opt(\theta_0, S), T) \right] \quad (1.1)$$

where $\theta_0$ are the randomly initialized network parameters, $\ell$ is the classification loss, and $Opt$ returns a neural network trained on the given dataset.

For linear models, this objective can be solved in closed-form via a simple ridge regression. However, in the case of deep neural networks, this objective on its own is intractable due to the many iterations required to train such models. Instead, methods of Dataset Distillation typically introduce some surrogate objective such that aims to approximate Eq. 1.1. We illustrate the objective of the Dataset Distillation problem in Figure 1.1.

1.3 The First Dataset Distillation Method

Along with posing the problem itself, Wang et al. [51] proposed the first solution to dataset distillation. This method optimizes the real classification loss of a model trained end-to-end
Figure 1.1: Illustration of the Dataset Distillation problem. Large datasets are distilled into small synthetic sets such that a model trained on the synthetic set will have similar test performance as a model trained on the full real dataset.
1. Dataset Distillation

on the synthetic data. Formally,

\[ \mathcal{L} := \ell(\theta_n, T) \]
\[ \theta_{i+1} := \theta_i - \eta_i \nabla_{\theta} \ell(\theta_i, S_i) \]  \hspace{1cm} (1.2)

where \( n \) is the total number of synthetic steps and \( \eta_i \) and \( S_i \) are separate step sizes and synthetic data learned for each training iteration.

While this method (DD) attempts to directly optimize Eq. 1.1 without any surrogate loss, this turns out to be its greatest weakness. Due to the computational power needed to back-propagate through many synthetic updates, the maximum number of synthetic iterations, \( n \), is severely limited. Thus, the synthetically-trained networks are never actually trained to convergence, as is implied in Eq. 1.1. This results in the synthetic data not containing information required to mimic the middle and end stages of training on real data.

1.4 Gradient Matching

A few years later, Zhao et al. [54] proposed a new method of dataset distillation by means of single-step gradient matching. Rather than back-propagating through the entire synthetic training process, this method (DC) only encourages the synthetic data to induce similar gradients as the real data at individual parameter states, thus alleviating the memory issues present in DD [51] and causing the synthetic data to include information required for the later stages of training. After training a randomly initialized network for \( t \) iterations on the synthetic data, the gradient matching loss is calculated as

\[ \mathcal{L} := D(\nabla_{\theta} \ell(\theta_t, S), \nabla_{\theta} \ell(\theta_t, T)) \]  \hspace{1cm} (1.3)

where \( D \) is a distance metric between the vectorized parameters, such as the \( \ell_2 \) norm or cosine distance, and \( \theta_t \) are the synthetically-trained parameters. Since DC only considers single-instance gradients, there is no enforcement that the long-term synthetically-trained parameter trajectories over time match those induced by the real data.
1.5 Differentiable Siamese Augmentation

Next, Zhao and Bilen [53] applied Differentiable Siamese Augmentation (DSA) to the DC method. In addition to the standard gradient matching of DC [54], DSA enforces that real and synthetic data should also induce similar gradients when transformed by the same differentiable augmentation operation. The distillation loss then becomes

$$\mathcal{L} := D(\nabla_\theta \ell_t(\theta_t, A(S)), \nabla_\theta \ell_t(\theta_t, A(T)))$$

(1.4)

where $A$ is the differentiable augmentation function. The addition of differentiable augmentations provided significant improvements over DC alone. However, DSA still suffers from the same lack of long-range consistency as DC.

1.6 Distribution Matching

Zhao and Bilen [52] (DM) foregoes the bi-level optimization of previous methods and instead matches features rather than gradients. Instead of calculating the gradients of the loss with respect to the parameters, DM simply takes a randomly initialized feature extractor and minimizes the average difference between real and synthetic outputs for each class

$$\mathcal{L} := \sum_{c=1}^{C} \left\| \frac{1}{|T|} \sum_{i=1}^{|T_c|} \mathcal{F}(T_c^k) - \frac{1}{|S|} \sum_{j=1}^{|S_c|} \mathcal{F}(S_c^k) \right\|^2$$

(1.5)

where $\mathcal{F}$ is the randomly initialized feature extractor and $T_c^k$ is the $k^{th}$ element of class $c$ of the real dataset and $S_c^k$ is likewise the $k^{th}$ element of class $c$ of the synthetic dataset. This method is unique among dataset distillation techniques in that it is task-agnostic. Since the feature extractors are randomly initialized, the loss is not at all dependent on the $C$-way classification problem. Thus, the images distilled for each class via this method would be the same regardless of how many total or which classes are considered during the distillation process, unlike all other methods. While this task-agnosticism makes the images more useful for other tasks, such as continual learning [6], they suffer reduced performance when training classification models.
1. Dataset Distillation

1.7 Kernel Inducing Points

Rather than a finitely-wide neural network, Nguyen et al. [35] and Nguyen et al. [36] consider the infinitely-wide Neural Tangent Kernel (NTK) [26] induced by the network architecture. Given a neural tangent kernel $K$, this method simply minimizes the Kernel Ridge-Regression loss trained on the synthetic dataset $S$ and evaluated on the real dataset $T$

$$L := \| \mathcal{T}_y - K_{\mathcal{T}_x}S_x(K_{S_x}S_x + \lambda I)^{-1}S_y \|^2$$

where for sets $A$ and $B$, $K_{AB}$ is the matrix of kernel elements $(K(a,b))_{a \in A, b \in B}$ and $\lambda$ is a regularization term. Unfortunately, computing these kernel matrices requires massive amounts of computing power such that thousands of GPU hours are required to distill even a small-scale dataset like CIFAR [30]. Furthermore, the images also show degraded performance when used to train finitely-wide networks rather than the infinitely-wide NTK.
Chapter 2

Matching Training Trajectories

2.1 Introduction

In the seminal 2015 paper, Hinton et al. [25] proposed model distillation, which aims to distill the knowledge of a complex model into a simpler one. Dataset distillation, proposed by Wang et al. [51], is a related but orthogonal task: rather than distilling the model, the idea is to distill the dataset. As shown in Figure 1.1, the goal is to distill the knowledge from a large training dataset into a very small set of synthetic training images (as low as one image per class) such that training a model on the distilled data would give a similar test performance as training one on the original dataset. Dataset distillation has become a lively research topic in machine learning [4, 35, 36, 44, 52, 53, 54] with various applications, such as continual learning, neural architecture search, and privacy-preserving ML. Still, the problem has so far been of mainly theoretical interest, since most prior methods focus on toy datasets, like MNIST and CIFAR, while struggling on real, higher-resolution images. In this work, we present a new approach to dataset distillation that not only outperforms previous work in performance, but is also applicable to large-scale datasets, as shown in Figure 3.2.

Unlike classical data compression, dataset distillation aims for a small synthetic dataset that still retains adequate task-related information so that models trained on it can generalize to unseen test data. Thus, the distilling algorithm must strike a delicate balance by heavily compressing information without completely obliterating the discriminative features. To
2. Matching Training Trajectories

do this, dataset distillation methods attempt to discover exactly which aspects of the real
data are critical for learning said discrimination. Several methods consider end-to-end
training [35, 36, 51] but often require huge compute and memory and suffer from inexact
relaxation [35, 36] or training instability of unrolling many iterations [34, 51]. To reduce
the optimization difficulty, other methods [53, 54] focus on short-range behavior, enforcing
a single training step on distilled data to match that on real data. However, error may
accumulate in evaluation, where the distilled data is applied over many steps. We confirm
this hypothesis experimentally in Section 2.5.2.

To address the above challenges, we sought to directly imitate the long-range training
dynamics of networks trained on real datasets. In particular, we match segments of parameter
trajectories trained on synthetic data with segments of pre-recorded trajectories from models
trained on real data and thus avoid being short-sighted (i.e., focusing on single steps) or
difficult to optimize (i.e., modeling the full trajectories). Treating the real dataset as the gold
standard for guiding the network’s training dynamics, we can consider the induced sequence
of network parameters to be an expert trajectory. If our distilled dataset were to induce
a network’s training dynamics to follow these expert trajectories, then the synthetically
trained network would land at a place close to the model trained on real data (in the
parameter space) and achieve similar test performance.

In our method, our loss function directly encourages the distilled dataset to guide the
network optimization along a similar trajectory (Figure 2.1). We first train a set of models
from scratch on the real dataset and record their expert training trajectories. We then
initialize a new model with a random time step from a randomly chosen expert trajectory
and train for several iterations on the synthetic dataset. Finally, we penalize the distilled
data based on how far this synthetically trained network deviated from the expert trajectory
and back-propagate through the training iterations. Essentially, we transfer the knowledge
from many expert training trajectories to the distilled images.

Extensive experiments show that our method handily outperforms existing dataset
distillation methods as well as coreset selection methods on standard datasets, including
CIFAR-10, CIFAR-100, and Tiny ImageNet. For example, we achieve 46.3% with a single
image per class and 71.5% with 50 images per class on CIFAR-10, compared to the previous
state of the art (28.8%/63.0% from [52, 53] and 36.1%/46.5% from [36]). Furthermore,
our method also generalizes well to larger data, allowing us to see high 128 × 128-resolution
images distilled from ImageNet [15] for the first time. Finally, we analyze our method
2. Matching Training Trajectories through additional ablation studies and visualizations.

2.2 Expert Trajectories

The core of our method involves using expert trajectories $\tau^*$ to guide the distillation of our synthetic dataset. By expert trajectories, we mean the time sequence of parameters $\{\theta^*_t\}_{0}^{T}$ obtained during the training of a neural network on the full, real dataset. To generate these expert trajectories, we simply train a large number of networks on the real dataset and save their snapshot parameters at every epoch. We call these sequences of parameters “expert trajectories” because they represent the theoretical upper bound for the dataset distillation task: the performance of a network trained on the full, real dataset. Similarly, we define student parameters $\hat{\theta}_t$ as the network parameters trained on synthetic images at the training step $t$. Our goal is to distill a dataset that will induce a similar trajectory (given the same starting point) as that induced by the real training set such that we end up with a similar model.

Since these expert trajectories are computed using only real data, we can pre-compute them before distillation. All of our experiments for a given dataset were performed using the same pre-computed set of expert trajectories, allowing for rapid distillation and experimentation.

2.3 Long-Range Parameter Matching

Our distillation process learns from the generated sequences of parameters making up our expert trajectories $\{\theta^*_t\}_{0}^{T}$. Unlike previous work, our method directly encourages the long-range training dynamics induced by our synthetic dataset to match those of networks trained on the real data.

At each distillation step, we first sample parameters from one of our expert trajectories at a random timestep $\theta^*_t$ and use these to initialize our student parameters $\hat{\theta}_t := \theta^*_t$. Placing an upper bound $T^+$ on $t$ lets us ignore the less informative later parts of the expert trajectories where the parameters do not change much.

With our student network initialized, we then perform $N$ gradient descent updates on
2. Matching Training Trajectories

Algorithm 1 Dataset Distillation via Trajectory Matching

Input: \( \{\tau^*_i\} \): set of expert parameter trajectories trained on \( D_{\text{real}} \).

Input: \( M \): # of updates between starting and target expert params.

Input: \( N \): # of updates to student network per distillation step.

Input: \( A \): Differentiable augmentation function.

Input: \( T^+ < T \): Maximum start epoch.

1: \( S \sim \mathcal{N}(0, I) \) \( \triangleright \) Initialize distilled data
2: \( \alpha := \alpha_0 \) \( \triangleright \) Initialize trainable learning rate
3: for each distillation step... do
4: \( \tau^* \sim \{\tau^*_i\} \) with \( \tau^* = \{\theta^*_t\}^T_0 \) \( \triangleright \) Sample expert trajectory
5: \( t \sim \text{Uniform}[0 \ldots T^+] \) \( \triangleright \) Choose random start epoch
6: \( \hat{\theta}_t := \theta^*_t \) \( \triangleright \) Initialize student network with expert params
7: for \( n = 0 \rightarrow N - 1 \) do
8: \( b_{t+n} \sim \mathcal{D}_{\text{syn}} \) \( \triangleright \) Sample a mini-batch of distilled images
9: \( \hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha \nabla \ell(A(b_{t+n}); \hat{\theta}_{t+n}) \) \( \triangleright \) Update student network w.r.t. classification loss
10: end for
11: \( \mathcal{L} = \|\hat{\theta}_{t+N} - \theta^*_t\|_2^2 / \|\theta^*_{t} - \theta^*_{t+M}\|_2^2 \) \( \triangleright \) Compute loss between student and expert params
12: \( S, \alpha \leftarrow \text{SGD}(S, \alpha; \mathcal{L}) \) \( \triangleright \) Update \( S \) and \( \alpha \) with respect to \( \mathcal{L} \)
13: end for

Output: distilled data \( S \) and learning rate \( \alpha \)

the student parameters with respect to the classification loss of the synthetic data:

\[
\hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha \nabla \ell(A(S); \hat{\theta}_{t+n}),
\]

where \( A \) is the differentiable augmentation technique [28, 47, 55, 56] used in previous work [53], and \( \alpha \) is the (trainable) learning rate used to update the student network. Any data augmentation used during distillation must be differentiable so that we can back-propagate through the augmentation layer to our synthetic data. Our method does not use differentiable Siamese augmentation since there is no real data used during the distillation process; we are only applying the augmentations to synthetic data at this time. However, we do use the same types of differentiable augmentations on real data during the generation of the expert trajectories.

From this point, we return to our expert trajectory and retrieve the expert parameters from \( M \) training updates after those used to initialize the student network \( \theta^*_{t+M} \). Finally, we update our distilled images according to the weight matching loss: i.e., the normalized squared \( L_2 \) error between the updated student parameters \( \hat{\theta}_{t+N} \) and the known future expert...
2. Matching Training Trajectories

parameters $\theta^*_t$: 

$$\mathcal{L} = \frac{\|\hat{\theta}_{t+N} - \theta^*_t\|^2}{\|\theta_t^* - \theta^*_t\|^2},$$  \hspace{1cm} (2.2)$$

where we normalize the $L_2$ error by the expert distance traveled so that we still get a strong signal from later training epochs where the expert does not move as much. This normalization also helps self-calibrate the magnitude difference across neurons and layers.

We have also experimented with other choices of loss functions such as a cosine distance, but find our simple $L_2$ loss works better empirically. We also tried to match the network’s output logits between expert trajectory and student network but did not see a clear improvement. We speculate that backpropagating from the network output to the weights introduces additional optimization difficulty.

We then minimize this objective to update the pixels of our distilled dataset, along with our trainable learning rate $\alpha$, by back-propagating through all $N$ updates to the student network. The optimization of trainable learning rate $\alpha$ serves as automatic adjusting for the number of student and expert updates (hyperparameters $M$ and $N$). We use SGD with momentum to optimize $\mathcal{S}$ and $\alpha$ with respect to the above objective. Algorithm 3 illustrates our main algorithm.

<table>
<thead>
<tr>
<th>Img/Cls</th>
<th>Ratio %</th>
<th>Training Set Synthesis</th>
<th>Full Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>1 0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>1 0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiny ImageNet</td>
<td>1 0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Comparing distillation and coreset selection methods. As in previous work, we distill the given number of images per class using the training set, train a neural network on the synthetic set, and evaluate on the test set. To get $\bar{x} \pm s$, we train 5 networks from scratch on the distilled dataset. Note that the earlier works DD† and LD† use different architectures, i.e., LeNet [33] for MNIST and AlexNet [31] for CIFAR-10. All others use a 128-width ConvNet. CIFAR values marked by (*) signify best results were obtained with ZCA whitening.
2. Matching Training Trajectories

2.4 Memory Constraints

Given that we are back-propagating through many gradient descent updates, memory consumption quickly becomes an issue when our distilled dataset is sufficiently large, as we have to jointly optimize all the images of all the classes at each optimization step. To reduce memory consumption and ease the learning problem, previous methods distill one class at a time [52, 53, 54], but this may not be an ideal strategy for our method since the expert trajectories are trained on all classes simultaneously.

We could potentially circumvent this memory constraint by sampling a new mini-batch at every distillation step (the outer loop in Line 3 of Algorithm 3). Unfortunately, this comes with its own issues, as redundant information could be distilled into multiple images across the synthetic dataset, degrading to catastrophic mode collapse in the worst case.

Instead, we can sample a new mini-batch \( b \) for every update of the student network (i.e., the inner loop in Line 9 of Algorithm 3) such that all distilled images will have been seen by the time the final weight matching loss (Eqn. 2.2) is calculated. The mini-batch \( b \) still contains images from different classes but has much fewer images per class. In this case, our student network update then becomes

\[
\hat{\theta}_{t+n} = \hat{\theta}_{t+n} - \alpha \nabla \ell(A(b_{t+n}); \hat{\theta}_{t+n}).
\]  

(2.3)

This method of batching allows us to distill a much larger synthetic dataset while ensuring some amount of heterogeneity among the distilled images of the same class.

2.5 Experiments

We evaluate our method on various datasets, including

- 32 × 32 CIFAR-10 and CIFAR-100 (Section 2.5.1), two commonly used datasets in dataset distillation literature,
- 64 × 64 Tiny ImageNet (Section 2.5.3), a recent benchmark by the concurrent work [52], and
- our new 128 × 128 ImageNet subsets (Section 2.5.4).

We provide additional visualizations and ablation studies in the supplementary material.
2. Matching Training Trajectories

Figure 2.1: We perform long-range parameter matching between training on distilled synthetic data and training on real data. Starting from the same initial parameters, we train distilled data $\mathcal{S}$ such that $N$ training steps on them match the same result (in parameter space) from much more $M$ steps on real data.

**Evaluation and Baselines.** We evaluate various methods according to the standard protocol: training a randomly initialized neural network from scratch on distilled data and evaluating on the validation set.

To generate the distilled images for our method, we employ the distillation process detailed in the previous section and Algorithm 3, using the same suite of differentiable augmentations as done in previous work [52, 53]. The hyperparameters used for each setting (real epochs per iteration, synthetic updates per iteration, image learning rate, etc.) can be found in the supplemental material.

We compare to several recent methods including Dataset Distillation [51] (DD), Flexible Dataset Distillation [4] (LD), Dataset Condensation [54] (DC), and Differentiable Siamese Augmentation [53] (DSA), along with a method based on the infinite-width kernel limit [35, 36] (KIP) and concurrent works Distribution Matching [52] (DM) and Aligning Features [50] (CAFE). We also compare our methods with instance selection algorithms including random selection (random), herding methods [12] (herding), and example forgetting [45] (forgetting).
Network Architectures. Staying with precedent [36, 52, 53, 54], we mainly employ a simple ConvNet architecture designed by Gidaris and Komodakis [21] for our distillation tasks. The architecture consists of several convolutional blocks, each containing a $3 \times 3$ convolution layer with 128 filters, Instance normalization [49], RELU, and $2 \times 2$ average pooling with stride 2. After the convolution blocks, a single linear layer produces the logits. The exact number of such blocks is decided by the dataset resolution and is specified below for each dataset. Staying with this simple architecture allows us to directly analyze the effectiveness of our core method and remain comparable with previous works.

2.5.1 Low-Resolution Data ($32 \times 32$)

For low-resolution tasks, we begin with the $32 \times 32$ CIFAR-10 and CIFAR-100 datasets [30]. For these datasets, we employ ZCA whitening as done in previous work [35, 36], using the Kornia [40] implementation with default parameters. Staying with precedent, we use a depth-3 ConvNet taken directly from the open-source code [53, 54].

As seen in Table 2.1, our method significantly outperforms all baselines in every setting. In fact, on the one image per class setting, we improve the next best method (DSA [53]) to almost twice test accuracy, on both datasets. For CIFAR-10, these distilled images can be seen in Figure 2.2. CIFAR-100 images are visualized in the supplementary material

In Table 2.2, we also compare with a recent method KIP [35, 36], where the distilled data is learned with respect to the Neural Tangent Kernel. Because KIP training is agnostic to actual network width, we test their result on both a ConvNet of the same width as us and other methods (128) and a ConvNet of larger width (1024) (which is shown in KIP paper [36]). Based on the the infinite-width network limit, KIP may exhibit a gap with practical finite-width networks. Our method does not suffer from this limitation and generally achieves better performance. In all settings, our method, trained on the 128-width network, outperforms KIP results evaluated on both widths, except for just one setting where KIP is applied on the much wider 1024-width network.

As noted in the previous methods [51], we also see significant diminishing returns when allowing more images in our synthetic datasets. For instance, on CIFAR-10, we see an increase from 46.3% to 65.3% classification accuracy when increasing from 1 to 10 images per class, but only an increase from 65.3% to 71.5% when increasing the number of distilled images per class from 10 to 50.
2. Matching Training Trajectories

<table>
<thead>
<tr>
<th>Img/Cls</th>
<th>Ratio %</th>
<th>KIP to NN (1024-width)</th>
<th>KIP to NN (128-width)</th>
<th>Ours (128-width)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.02</td>
<td>49.9</td>
<td>38.3</td>
<td>46.3</td>
</tr>
<tr>
<td>10</td>
<td>0.1</td>
<td>62.7</td>
<td>57.6</td>
<td>65.3</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>68.6</td>
<td>65.8</td>
<td>71.5</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.2</td>
<td>15.7</td>
<td>18.2</td>
<td>24.3</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>28.3</td>
<td>32.8</td>
<td>39.4</td>
</tr>
</tbody>
</table>

Table 2.2: Kernel Inducing Point (KIP) [36] performs distillation using the infinite-width network limit. We consistently outperform KIP when evaluating on the same finite-width network, and almost always outperform KIP applied on wider networks.

<table>
<thead>
<tr>
<th>Evaluation Model</th>
<th>ConvNet</th>
<th>ResNet</th>
<th>VGG</th>
<th>AlexNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Ours</td>
<td>64.3 ± 0.7</td>
<td>46.4 ± 0.6</td>
<td>50.3 ± 0.8</td>
</tr>
<tr>
<td></td>
<td>DSA</td>
<td>52.1 ± 0.4</td>
<td>42.8 ± 1.0</td>
<td>43.2 ± 0.5</td>
</tr>
<tr>
<td></td>
<td>KIP</td>
<td>47.6 ± 0.9</td>
<td>36.8 ± 1.0</td>
<td>42.1 ± 0.4</td>
</tr>
</tbody>
</table>

Table 2.3: Despite being trained for a specific architecture, our synthetic images do not seem to suffer from much over-fitting to that model. This is evaluated on CIFAR-10 with 10 images per class.

If we look at the one image per class visualizations in Figure 2.2 (top), we see very abstract, yet still recognizable, representations of each class. When we limit the task to just one synthetic image per class, the optimization is forced to squeeze as much of the class’s distinguishing information as possible into just one sample. When we allow more images in which to disperse the class’s information, the optimization has the freedom to spread the class’s discriminative features among the multiple samples, resulting in a diverse set of structured images we see in Figure 2.2 (bottom) (e.g., different types of cars and horses with different poses).

Cross-Architecture Generalization. We also evaluate how well our synthetic data performs on architectures different from the one used to distill it on the CIFAR-10, 1 image per class task. In Table 2.3, we show our baseline ConvNet performance and evaluate on ResNet [24], VGG [42], and AlexNet [31].

For KIP, instead of the Kornia [40] ZCA implementation, we use the authors’ custom ZCA implementation for evaluation of their method. Our method is solidly the top performer.
on all the transfer models except for AlexNet where we lie within one standard deviation of 
\( \text{DSA} \). This could be attributed to our higher baseline performance, but it still shows that our 
method is robust to changes in architecture.

### 2.5.2 Short-Range vs. Long-Range Matching

Unlike some prior works (\( \text{DC} \) and \( \text{DSA} \)), our method performs long-range parameter matching, where \( N \) training steps on distilled data match a much larger \( M \) steps on real data. Methods that optimize over entire training processes (e.g., \( \text{DD} \) and \( \text{KIP} \)) can be viewed as even longer range matching. However, their performances fall short of our method (e.g., in Table 2.1), likely due to related instabilities or inexact approximations. Here, we experimentally confirm our hypothesis that long-range matching achieved by larger \( M \) and \( N \) in our method is superior to the short-range counterparts (such as small \( M \) and \( N \) and \( \text{DSA} \)).

In Figure 2.3 (left), we evaluate our method on different settings of \( M \) and \( N \). Really short-range matching (with \( N = 1 \) and small \( M \)) generally exhibits worse performance than long-range matching, with the best performance attained when both \( N \) and \( M \) are relatively large. Furthermore, as we increase \( N \), the power of \( N \) combined steps (on distilled data) becomes stronger and can approximate longer-range behavior, leading to the optimal \( M \) values shifting to greater values correspondingly.

In Figure 2.3 (right), we evaluate our method and a short-range matching work (\( \text{DSA} \)) on their abilities to approximate real training behavior over short and long ranges. Starting from a set of initial parameters, we set the target parameters to be the result of \( \Delta t \) training steps on real data (i.e., the long-range behavior that distilled data should mimic). A small (or large) \( \Delta t \) means evaluating matching over a short (or long) range. For both methods, we test how close they can train the network (using distilled data) from the same initial parameters to the target parameters. \( \text{DSA} \) is only optimized to match short-range behavior, and thus errors may accumulate during longer training. Indeed, as \( \Delta t \) grows larger, \( \text{DSA} \) fails to mimic the real data behavior over longer ranges. In comparison, our method is optimized for long-range matching and thus performs much better.
Figure 2.2: CIFAR-10: The 1 image per class images are more abstract but also more information-dense while the 10 images per class images are more expressive and contain more structure.
2. Matching Training Trajectories

Figure 2.3: CIFAR-100 (1 image / class). **Left:** Smaller $M$ and $N$ match shorter-range behavior, which performs worse than longer-range matching. **Right:** Over 1000 training steps on distilled data, we track the closest distance in parameter space (normalized MSE in Eqn. 2.2) to a target set of parameters, obtained with $\Delta_t$ training steps on real data. Matching long-range behavior, our method better approximate real data training for longer ranges (large $\Delta_t$).

<table>
<thead>
<tr>
<th></th>
<th>ImageNette</th>
<th>ImageWoof</th>
<th>ImageFruit</th>
<th>ImageMeow</th>
<th>ImageSquawk</th>
<th>ImageYellow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Img/Cls</td>
<td>47.7 ± 0.9</td>
<td>28.6 ± 0.8</td>
<td>26.6 ± 0.8</td>
<td>30.7 ± 1.6</td>
<td>39.4 ± 1.5</td>
<td>45.2 ± 0.8</td>
</tr>
<tr>
<td>10 Img/Cls</td>
<td>63.0 ± 1.3</td>
<td>35.8 ± 1.8</td>
<td>40.3 ± 1.3</td>
<td>40.4 ± 2.2</td>
<td>52.3 ± 1.0</td>
<td>60.0 ± 1.5</td>
</tr>
<tr>
<td>Full Dataset</td>
<td>87.4 ± 1.0</td>
<td>67.0 ± 1.3</td>
<td>63.9 ± 2.0</td>
<td>66.7 ± 1.1</td>
<td>87.5 ± 0.3</td>
<td>84.4 ± 0.6</td>
</tr>
</tbody>
</table>

Table 2.4: Applying our method to 128×128 resolution ImageNet subsets. On this higher resolution, across various subsets, our method continues to produce high-quality distilled images.
2. Matching Training Trajectories

Figure 2.4: Selected samples distilled from Tiny ImageNet, one image per class. Despite the higher resolution, our method still produces high-fidelity images. (Can you guess which classes these images represent? Check your answers in the footnote!)

2.5.3 Tiny ImageNet (64×64)

Introduced to the dataset distillation task by the concurrent work, Distribution Matching (DM) [52], we also show the effectiveness of our algorithm on the 200 class, 64×64 Tiny ImageNet [14] dataset (a downscaled subset of ImageNet [15]). To account for the higher image resolution, we move up to a depth-4 ConvNet, similar to DM [52].

Most dataset distillation methods (other than DM) are unable to handle this larger resolution due to extensive memory or time requirement, as the DM authors also observed [52]. In Table 2.1, our method consistently outperforms the only viable such baseline, DM. Notably, on the 10 images per class task, our method improves the concurrent work DM from 12.9% and 23.2%. A subset of our results is shown in Figure 2.4. The supplementary material contains the rest of the images.

At 200 classes and 64×64 resolution, Tiny ImageNet certainly poses a much harder task than previous datasets. Despite this, many of our distilled images are still recognizable, with a clear color, texture, or shape pattern.
Figure 2.5: Our method is the first capable of distilling higher-resolution (128 × 128) images, allowing us to explore the ImageNet [15] dataset.

### 2.5.4 ImageNet Subsets (128 × 128)

Next, we push the boundaries of dataset distillation even further by running our method on yet higher resolution images in the form of 128 × 128 subsets of ImageNet [15]. Again, due to the higher resolution, we increase the depth of our architecture and use a depth-5 ConvNet for the 128 × 128 ImageNet subsets.

ImageNette (assorted objects) and ImageWoof (dog breeds) are existing subsets [19] designed to be easy and hard to learn respectively. We also introduce ImageFruit (fruits), ImageMeow (cats), ImageSquawk (birds), and ImageYellow (yellow-ish things) to further illustrate our algorithm.

Similar to Tiny ImageNet, most dataset distillation baselines do not scale up to our

---

1 Answers for Figure 2.4: **First Row:** African Elephant, Jellyfish, Kimono, Lampshade, Monarch **Second Row:** Organ, Pizza, Pretzel, Teapot, Teddy
2. Matching Training Trajectories

ImageNet subset settings. As the code of DM [52] is not publicly available now, we choose to only compare to the networks trained on the full dataset. We wish to show that our method transfers well to large images and still produces meaningful results at a higher resolution. Validation set accuracies are presented in Table 2.4.

While all of the generated images are devoid of high-frequency noise, the tasks still differ in the type of distilled image they induce. For tasks where all the classes have a similar structure but unique textures like ImageSquawk (Figure 2.5), the distilled images may not have much structure but instead store discriminating information in the textures.

Conversely, for tasks where all classes have similar color or textures like ImageYellow (Figure 2.5), the distilled images seem to diverge from their common trait and accentuate the structure or secondary color that makes them unique. Specifically, note the differences between the distilled “Banana” images for the ImageFruit and ImageYellow (bottom row, Figure 2.5). Although the expert trajectory-generating networks saw the same “Banana” training images, the distilled images differ between the two tasks. The distilled “Banana” for the ImageYellow task is clearly much “greener” than the equivalent image for the ImageFruit task. This implies that the expert networks identify different features by which to identify classes based on the other classes in the task.

2.6 Discussion and Limitations

In this work, we introduced a dataset distillation algorithm by means of directly optimizing the synthetic data to induce similar network training dynamics as the real data. The main difference between ours and prior approaches is that we are neither limited to the short-range single-step matching nor subject to instability and compute intensity of optimizing over the full training process. Our method balances these two regimes and shows improvement over both.

Unlike prior methods, ours is the first to scale to $128 \times 128$ ImageNet images, which not only allows us to gain interesting insights of the dataset (e.g., in Section 2.5.4) but also may serve as an important step towards practical applications of dataset distillation on real-world datasets.

Limitations. Our use of pre-computed trajectories allows for significant memory saving, at the cost of additional disk storage and computational cost for expert model training.
2. Matching Training Trajectories

The computational overhead of training and storing expert trajectories is quite high. For example, CIFAR experts took \(~3\) seconds per epoch (8 GPU hours total for all 200 CIFAR experts) while each ImageNet (subset) expert took \(~11\) seconds per epoch (15 GPU hours total for all 100 ImageNet experts). Storage-wise, each CIFAR expert took up \(~60\)MB of storage while each ImageNet expert took up \(~120\)MB.
Chapter 3

Tileable Texture Synthesis

3.1 Introduction

The task of dataset distillation [9, 35, 36, 50, 51, 52, 53, 54] involves creating a small synthetic dataset (as little as one image per class) such that a model trained on this synthetic dataset will have similar test-time performance as a model trained on the full original training set. This problem is particularly challenging, as a good solution must strike a delicate balance between compressing the visual information into just a few images, while still preserving the learnable discriminative features of each class. Unexpectedly, the results of the most recent method [9] have shown an additional property of being visually intriguing (see Figure 3.3B). In this short paper, we further explore dataset distillation as a tool for synthesizing visually interesting textures.

Until now, all existing methods of dataset distillation have focused on the synthesis of standard-sized distilled image(s) corresponding to each class. Here, we propose an alternative method that yields a more artistic yet still functional result. We consider a “canvas” of pixels for each class, which is twice as large as the input training images. At distillation time, we apply circular padding and take random crops from this canvas. By optimizing all such random crops to serve as effective training data, we see the emergence of category-based textures that preserve continuity across their borders. While being aesthetically pleasing in their own right, the resulting images can also be applied to practical tasks, such as pattern swatches used to create clothing or wallpaper. We call our method Tileable Texture...
3. Tileable Texture Synthesis

Figure 3.1: Our method takes random crops of a padded distilled *canvas* instead of discrete images. Note: Our method distills all classes simultaneously, although only one (penguin) is shown in detail in this figure for simplicity.

Figure 3.2: Our new method builds upon recent dataset distillation work [9] and lets us distill classes into *tilable textures*. These textures can then be used for downstream tasks, such as pattern swatches for clothing. (Visualizations made with FAB3D: https://fabric.tri3d.in/)

Distillation.

Our method can be thought of as a type of texture synthesis, but while classic patch-based synthesis methods, e.g. [18], create a texture from a single source image, our method produces a result that captures the “essence” of an entire dataset class. Our way of synthesizing tileable textures is inspired by Wang tiles work of Cohen et al. [13] (and their algorithm can easily be used to extend our work to non-periodic tiled textures). Our other source of inspiration is the classic Image Epitomes work [27], but while epitomes capture the statistics necessary for reconstruction of an image, our method instead focuses on data needed for discrimination.
3. Tileable Texture Synthesis

3.2 Preliminaries

Here, we briefly review our underlying distillation method: Dataset Distillation by Matching Training Trajectories (MTT) [9]. Similar to previous methods, MTT seeks to distill synthetic data that can still train a well-performing model.

While other methods attempt to optimize the synthetic data with respect to either a single gradient step [53, 54] or the entire training process [35, 36, 51], MTT considers many-step trajectories starting from varying points along pre-computed expert trajectories (those obtained by training on the full, real training set). On a high-level, MTT encourages models trained on synthetic data to mimic the behavior of the expert trajectory.

Specifically, MTT initialize initializes a student network at some point along an expert trajectory. The student network is then trained for many \( N \approx 50 \) iterations on the synthetic data, and the “distillation loss” is calculated as the relative error between the student network’s current parameters and those of a future timestep of the expert trajectory. The motivation behind this method is an effort to directly optimize the synthetic data such that it induces a similar training trajectory as the real training set, resulting in a final model that lies at a similar point in parameter space. For further details on the distillation method from which ours was adapted, we refer the reader to [9].

3.3 Tilable Texture Distillation

Our method, Tilable Texture Distillation, builds upon MTT and offers a new mode of distillation. Instead of distilling individual, disjoint training images, we instead optimize a toroidal canvas of pixels such that any random crop of the canvas is a good training sample. As illustrated in Figure 4.1, our optimization objective is to ensure that a model trained on random crops of our distilled canvases will perform similarly to a model trained on the real training set.

To achieve this effect, we first apply circular padding in both the \( x \) and \( y \) directions. By always considering the image to be circular padded, we effectively induce a toroidal topology on our synthetic canvas. Where MTT would use the disjoint synthetic samples to train the student trajectory, we instead train our student trajectory on random crops of our padded synthetic canvases, including such crops that would span the seams of the torus.

While the random cropping alone induces a texturized appearance in our synthetic
3. Tileable Texture Synthesis

Figure 3.3: Traditional distillation methods take the real training data (A) and create disjoint synthetic training images (B). Our new method builds upon recent dataset distillation work [9] and allows for the synthesis of class-based tileable textures (C).
3. Tileable Texture Synthesis

Figure 3.4: We initialize our synthetic canvases from Gaussian noise. Structure emerges after several hundred iterations, but it takes tens of thousands until convergence.

canvas, the circular padding ensures that they will also be *tileable*. Our loss function naturally encourages each “image” fed through the student network to be continuous. By using circular padding, we enforce that the patches that span the seams of the torus (i.e. the edges of the image) to also be continuous. This has the corollary effect of making our images tileable, as seen in Figure 3.5. Note that both padding and cropping are differentiable operations.

Figure 3.5: ImageSquawk Distilled Textures Tiled 3x3

3.4 Experiments

Since we are focusing on visual (qualitative) results in this work instead of classification performance, we chose to distill higher-resolution imagery than in the MTT paper [9]. Specifically, for each class, we distilled a $512 \times 512$ toroidal canvas wherein each $256 \times 256$ patch is optimized to be a good training sample. For comparison, the largest images distilled
Algorithm 2 Texture Synthesis via Trajectory Matching

Input: \( \{ \tau_i^* \} \): set of expert parameter trajectories trained on \( T \).
Input: \( M \): # of updates between starting and target expert params.
Input: \( N \): # of updates to student network per distillation step.
Input: \( A \): Differentiable augmentation function.
Input: \( T^+ \leq T \): Maximum start epoch.

1: \( S \sim \mathcal{N}(0, I) \)  \( \triangleright \) Initialize distilled data
2: \( \alpha := \alpha_0 \)  \( \triangleright \) Initialize trainable learning rate
3: \textbf{for each} distillation step... \textbf{do}
4: \( \tau^* \sim \{ \tau_i^* \} \) with \( \tau^* = \{ \theta_i^* \}_{t=0}^{T} \)  \( \triangleright \) Sample expert trajectory
5: \( t \sim \text{Uniform}[0 \ldots T^+] \)  \( \triangleright \) Choose random start epoch
6: \( \hat{\theta}_t := \theta_t^* \)  \( \triangleright \) Initialize student network with expert params
7: \textbf{for} \( n = 0 \rightarrow N - 1 \) \textbf{do}
8: \( b_{t+n} \sim S \)  \( \triangleright \) Sample a mini-batch of distilled images
9: \( b_{t+n}^* = \text{crop_and_pad}(b_{t+n}) \)  \( \triangleright \) Take random crop with circular padding
10: \( \hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha \nabla \ell(A(b_{t+n}^*); \hat{\theta}_{t+n}) \)  \( \triangleright \) Update student network w.r.t. classification loss
11: \textbf{end for}
12: \( \mathcal{L} = \| \hat{\theta}_{t+N} - \theta_{t+M}^* \|_2^2 / \| \theta_t^* - \theta_{t+M}^* \|_2^2 \)  \( \triangleright \) Compute loss between student and expert params
13: \( S, \alpha \leftarrow \text{SGD}(S, \alpha; \mathcal{L}) \)  \( \triangleright \) Update \( S \) and \( \alpha \) with respect to \( \mathcal{L} \)
14: \textbf{end for}

Output: distilled data \( S \) and learning rate \( \alpha \)

Figure 3.6: By distilling classes into tileable textures, we can apply our synthetic data to cases where such tileability is required, such as fabric pattern swatches. From left to right, we see our flamingo, eagle, macaw, and penguin textures applied to clothing using FAB3D [1]

in MTT were 128 × 128.

We stay with the precedent set by other dataset distillation work and use ConvNet models [21]. Since our patches are 256 × 256, we use a depth-6 ConvNet for our distillation,
3. Tileable Texture Synthesis

following the pattern seen in DM [52] and MTT [9].

We distill the ImageSquawk subset of ImageNet [15] as introduced in [9]. We chose this dataset with the thought that the various colors and patterns of the birds’ plumage would make for visually appealing textures. Specifically, ImageSquawk is composed of the peacock, flamingo, macaw, pelican, penguin, eagle, toucan, ostrich, black swan, and cockatoo classes. We also distill ImageFruit, which contains pineapple, banana, strawberry, orange, lemon, pomegranate, fig, bell pepper, cucumber, and granny smith.

3.4.1 Results

We present the result of our main experiment in Figure 3.2. Our method (b) can synthesize texture images with no centering bias while still resembling the distinct features of their respective classes (a), which makes them potentially useful in designing new clothes (c).

In Figure 3.5, we see that our distilled textures seamlessly tile together into a larger, continuous image. It is possible to create larger tiles without increasing memory cost simply by distilling a larger canvas while keeping the underlying patch size the same. However, this method would significantly increase training time since each patch would be sampled less frequently, requiring more total iterations for convergence.

As far as training (distillation) dynamics, we see coarse structure emerge relatively early in the optimization, within the first several hundred iterations. However, it takes many more iterations, on the order of tens of thousands, to synthesize a more defined structure in the texture and remove the high-frequency noise components, as seen in Figure 3.4.

Once we have our converged toroidal textures, we can apply them to domain-relevant tasks, such as pattern swatches for fabrics. Using FAB3D [1], we visualize our textures as applied to clothing. FAB3D has knowledge of the clothing article’s underlying topology, so it makes full use of our textures’ tileability, as seen in Figure 3.6.

3.5 Discussion

In this work, we introduced an extension to a recent dataset distillation algorithm that allows us to distill tileable, class-based textures. While all distillation methods to date have solely focused on classification performance [35, 36, 50, 51, 52, 53, 54], our new method is the first to change the overall objective to synthesize something other than disjoint training
3. Tileable Texture Synthesis

samples.

Tileable Texture Distillation takes random crops across an induced toroidal canvas and enforces them to all be good training examples for discriminating the respective class. It gives us a unique way of synthesizing tileable class-based textures for down-stream use. We also include code allowing users to easily distill textures for their own classes of choice.

By presenting a dataset distillation method with an end goal other than training discriminative models, we hope to encourage the creative use of dataset distillation methods for other novel tasks.

Limitations. As we continue to distill higher-resolution images, the process becomes more computationally costly, both spatially and temporally. For MTT specifically, training larger models to obtain the expert trajectories (i.e., depth-6 ConvNet versus depth-3 ConvNet) takes much longer, and the checkpoints take up significantly more space when stored on disk. Furthermore, distilling quadratically more pixels (i.e., doubling the resolution) while keeping the patch size the same requires many more iterations before convergence, especially when performing our new random crop method.
Chapter 4

Generative Latent Distillation

4.1 Introduction

Many recent exciting advancements in machine learning come from combining large networks and big data. Such trained models have shown strong capabilities to perform on a wide range of diverse tasks [8, 16, 39] and are considered by some as an ongoing paradigm shift [5]. While such approaches show great potential to improve the frontier of AI, we, as a scientific community, are also curious about the underlying principles and limitations. Do networks have to be large to express the functions of interest? Do datasets have to be big? Can training on “small data” be equally successful?

The seminal work on Knowledge Distillation [25] and recent discoveries such as Lottery Ticket Hypothesis [20] have revealed small models are often sufficient to express large trained models (which are sometimes useful in optimization). Dataset Distillation, proposed by Wang et al. [51], investigates the analogous yet orthogonal question on datasets: is there a small succinct dataset that is sufficient for training models? In other words, Dataset Distillation aims to distill a large dataset into a small (synthetic) one, such that training on the small dataset yields comparable performance (Figure 4.1).

Dataset Distillation is an interesting investigation on the lower limit of dataset size, and also potentially produces synthetic data useful for dataset understanding. Additionally, several applications have been explored, including continual learning, neural architecture search, privacy-preserving machine learning, and even computer-aided fashion design. Since
its proposal, Dataset Distillation has gained much attention in the research community [9, 10, 43, 50, 51, 52, 53, 54]. Usually, such methods optimize for a small synthetic vision dataset, i.e., raw pixels of a fixed number of images, so that it can train randomly initialized networks to good performance on real test data.

Unfortunately, these methods face two major challenges, limiting both their scientific value and empirical applications. First, the distilled synthetic dataset is often trained w.r.t. a specific network architecture, but does not generalize well to other architectures. Second, while producing insightful distilled images on toy datasets, these methods generally fail to work well on realistic datasets (e.g., $\geq 128 \times 128$ resolution) and tend to distill visually noisy images with subpar performance.

In this work, we argue that both issues are caused by parameterizing the synthetic dataset in pixel space. Directly optimizing pixels can be susceptible to learning high-frequency patterns that overfit the specific architecture used in training. To address this, we consider regularizing the distillation process to some prior that may help cross-architecture generalization. However, how and where to perform this regularization poses a delicate balance. For example, restricting our synthetic set to the real data manifold can significantly reduce the cross-architecture performance gap but is too strong a regularization to learn good distilled datasets. In the limit, it reduces to dataset/coreset selection [3, 7, 23, 48], which is known to not work as well [10, 51, 53, 54].

We propose Generative Latent Distillation, which utilizes a deep generative prior by parameterizing the synthetic dataset in the intermediate feature space of generative models, such as Generative Adversarial Networks (GANs) [22]. It encourages the learned datasets to be more generalizable to novel architectures but is also lax enough to not prohibitively restrict the expressiveness of the distilled dataset. Our analysis on a spectrum of dataset parameterization spaces highlights the benefits of such a prior over no prior at all (pixel space; current practice) and too restrictive a prior (e.g., real data manifold).

Empirically, we perform extensive experiments on ImageNet subsets at resolutions up to $512 \times 512$. Our proposed method greatly improves cross-architecture generalization over the current state-of-the-art, Matching Training Trajectories (MTT) [10], often leading to a roughly $1.5 \times$ improved test accuracy on unseen architectures while still maintaining comparable performance on the backbone (distilling) architecture. Additionally, our method drastically reduces the high-frequency noise present high-resolution datasets distilled by MTT.
4. Generative Latent Distillation

Figure 4.1: Rather than directly distilling a dataset into synthetic pixels (like all previous methods), our new work instead distills into the latent space of a *deep generative prior*. This enforces a tune-able amount of coherence in the output synthetic images, leading to far better generalization to new architectures.

Our method enables us to scale and generalize Dataset Distillation to more realistic settings. In summary, our contributions are:

- We propose *Generative Latent Distillation* to add a *deep generative prior* to Dataset Distillation by distilling into an intermediate feature space of a generative model trained on real data. (Section 4.2)
- Our extensive analysis and ablations highlight the importance of a *deep generative prior* in solving the two major challenges of Dataset Distillation: cross-architecture generalization and high-resolution data. (Section 4.3.1)
- Our method significantly improves cross-architecture generalization of the distilled dataset (often $\geq 1.5 \times$ test accuracy) (Section 4.3.2), and yields better high-resolution results (Section 4.3.4) on ImageNet subsets and other standard datasets, outperforming prior works and baselines.
- By adding a prior, our method requires less supervision from *real training trajectories* (the “supervising” signal of MTT) to reach good performance and reduces the associated expensive pre-computation and storage cost of the previous state of the art, MTT. (Section 4.3.3)
Algorithm 3 Generative Latent Distillation

**Input:** \{\tau^*_i\}: set of expert parameter trajectories trained on real dataset \(T\).
**Input:** \(M\): # of updates between starting and target expert params.
**Input:** \(N\): # of updates to student network per distillation step.
**Input:** \(A\): Differentiable augmentation function.
**Input:** \(T^+ < T\): Maximum start epoch.
**Input:** \(G\): Pre-trained generator.
**Input:** \(P_z\): Distribution of latent initializations.

1: \(Z \sim P_z\) ▷ Initialize distilled latents
2: \(\alpha := \alpha_0\) ▷ Initialize trainable learning rate
3: for each distillation step... do
4: \(\tau^* \sim \{\tau^*_i\}\) with \(\tau^* = \{\theta^*_t\}_T\) ▷ Sample expert trajectory
5: \(t \sim \text{Uniform}[0...T^+]\) ▷ Choose random start epoch
6: \(\hat{\theta}_t := \theta^*_t\) ▷ Initialize student network with expert params
7: \(S = G(Z)\) ▷ Get synthetic images from latents
8: for \(n = 0 \rightarrow N - 1\) do
9: \(\hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha \nabla \ell(A(I); \hat{\theta}_{t+n})\) ▷ Update student network
10: end for
11: \(L = \|\theta_{t+N} - \theta^*_{t+M}\|_2^2 / \|\theta^*_t - \theta^*_{t+M}\|_2^2\) ▷ Compute loss between student and expert params
12: \(Z, \alpha \leftarrow \text{SGD}(Z, \alpha; L)\) ▷ Update \(Z\) and \(\alpha\) with respect to \(L\)
13: end for

**Output:** distilled latents \(Z\)

### 4.2 Method

To date, all existing methods of dataset distillation [4, 10, 35, 36, 50, 51, 53, 54] have relied on a “backbone” architecture to formulate the distillation objective. Since optimizing distilled images in pixel space allows too much freedom to over-fit to the backbone architecture, we propose introducing a deep generative prior to the distillation process as a form of regularization by optimizing latent codes of a pre-trained generative model rather than raw pixels.

In Section 4.2.1, we review the previous method on which ours is built. In Section 4.2.2, we introduce our Generative Latent Distillation technique and discuss some of the problems it addresses. Lastly, in Section 4.2.3 we describe how to further reduce memory consumption by regenerating the synthetic data.
4. Generative Latent Distillation

4.2.1 Matching Training Trajectories

For completeness, we briefly review the trajectory matching method [10], the backbone to our new generative latent distillation method. Like all other dataset distillation methods, MTT seeks to distill a small synthetic dataset $S$ such that a model trained from scratch on $S$ will have similar performance as a model trained on the full, real dataset $T$. Prior to distillation, many expert trajectories $\{\theta^*_t\}_{0}^{T}$ are obtained by training networks from scratch on the full real dataset and storing parameter snapshots at given intervals. At each distillation step, a random expert trajectory and starting timestamp $\theta^*_t$ are sampled. A student network is then initialized at the given expert timestamp $\hat{\theta}_t := \theta^*_t$ and trained for $N$ iterations on the synthetic data. The distillation loss is then calculated as the normalized mean-squared error between the final parameters of the student network $\hat{\theta}_{t+N}$ and those of a future timestep ($M$ steps ahead) of the expert trajectory $\theta^*_{t+M}$:

$$L = \frac{\|\hat{\theta}_{t+N} - \theta^*_{t+M}\|_2^2}{\|\theta^*_t - \theta^*_{t+M}\|_2^2}.$$ 

This loss is then back-propagated through all the iterations of the student network to the synthetic images. For a more detailed explanation, we refer the reader to the original paper [10].

4.2.2 Deep Generative Prior

Rather than naively optimizing the synthetic pixels directly (as in all previous methods [4, 10, 35, 36, 50, 51, 53, 54]), we propose applying a deep generative prior to our distillation process by means of distilling into the latent space of a pre-trained generative model. We find that such a prior greatly increases the generalization of the synthetic dataset to architectures other than the “backbone” model used in the distillation process (the lack of such generalization being one of the largest limitations of previous methods). Furthermore, our new method facilitates the distillation of even larger resolution synthetic data devoid of the high-frequency noise induced by previous distillation methods. Lastly, the deep generative prior addresses another major limitation of MTT [10] by reaching good performance with a far lower number (as low as 1) of expert trajectories, alleviating the massive pre-training and disk storage requirements of MTT.
4. Generative Latent Distillation

We fill the role of our deep generative prior with the recently proposed StyleGAN-XL [41], a modified version of StyleGAN3 [29]. Distilling into the latent space of StyleGAN-XL can be thought of as a pseudo-inversion task. However, we found that distilling into even extended latent space $W^+$ [2], the most flexible of the traditional StyleGAN inversion spaces, was too restrictive for our objective. Here $W$ space refers to the output space of StyleGAN-XL’s MLP “mapping” network, and $W^+$ concatenates different $W$ space vectors of different layers. As such, we distill into “$F$” space [37, 57] where “$F_n$” space signifies optimizing the $n$-th hidden layer of StyleGAN-XL’s “synthesis” network’s latent representation along with all subsequent $W^+$ modulation codes.

We initialize our $W^+$ codes as the average output of 1000 random Gaussian vectors passed through the class-conditioned “mapping” network [29, 41]. We initialize our $F$ latents with random noise where the mean and variance are obtained from the same latent generated by a forward pass of $W^+$ through the “synthesis” network. For simplicity, we will henceforth refer to a single latent code as “$z$,” our generator $G$ as a function of this $z$, and the set of all distilled latent codes as $Z$, regardless of distillation space. Please see the supplement for details of the mapping and synthesis networks.

At distillation time, we feed our latent codes through the pre-trained StyleGAN-XL generator $G$ to recover our synthetic images: $S = G(Z)$. These synthetic images are then used to calculate the distillation loss $L$ as in MTT. The distillation loss is then back-propagated through all the student network updates, as well as the generator, $G$, to calculate $\partial L/\partial Z$ and update our latent codes accordingly. Please see Algorithm 3 for a complete write-up of our method.

4.2.3 Image Regeneration

As the forward pass through StyleGAN-XL [41] requires copious amounts of VRAM, our method (if implemented naively) becomes even harder to run on a limited number of GPUs than the already memory-hungry method of MTT [10]. To circumvent this issue, we employ a technique inspired by gradient checkpointing [11]. At each distillation iteration, we first obtain our synthetic images $S = G(Z)$ without tracking any gradients. We then calculate our trajectory matching loss $L$ (just as in MTT [10]), calculate the gradient of this loss with respect to our synthetic images ($\partial L/\partial S$), and delete the computation graph used to compute $L$ and its gradient. To compute $\partial L/\partial Z$, we re-compute the forward pass through
4. Generative Latent Distillation

- **Distribution Matching (DM) [52]**: Cross-arch. validation accuracy: 19.9%
- **Matching Training Trajectories (MTT) [10]**: Cross-arch. validation accuracy: 19.0%
- **Our Method: Generative Latent Distillation**: Cross-arch. validation accuracy: 27.2%

Figure 4.2: Visualizations of all current methods capable of distilling high-resolution datasets applied to the Birds-10 subset of ImageNet. We encourage the reader to zoom in and notice the absence of high-frequency noise in our method (c) compared to MTT (b).

StyleGAN-XL, \( S = G(Z) \), this time tracking gradients such that we know \( \partial S / \partial Z \). From here, application of the chain rule gives us \( \partial L / \partial Z = (\partial L / \partial S)(\partial S / \partial Z) \) which we use to update the latent codes for our synthetic data. This memory-saving trick allows us to comfortably distill our 128x128 synthetic sets on a single 24GB GPU.

4.3 Experiments

We evaluate our method for distilling CIFAR-10 [30] and 10-class subsets of ImageNet 1k [15], utilizing StyleGAN-XL [41] generators trained on these datasets (obtained from the official released model checkpoints). Specifically, we evaluate our method on 10 different 10-class subsets of ImageNet (some of which appeared in the MTT paper [10]) which we call Birds-10, Fruits-10, Food-10, Marine-10, Herptiles-10, Mammals-10, Cats-10, Earth-10, Sports-10, and ImageNette [19]. For the sake of space, the specific classes of each of these subsets are enumerated in the supplementary material. All visualizations in the main paper are on Birds-10 for consistency and ease of comparison, but analogous visualizations for all other subsets can be found in the supplementary material.

We mostly compare our method against Matching Training Trajectories (MTT) [10] and Distribution Matching (DM) [52] as these are the only other methods currently capable of distilling high-resolution datasets. Any reported results for MTT or DM were generated.
Figure 4.3: Different distillation optimization spaces induce different levels of generative priors. Using spaces of earlier layers (left side) imposes stronger priors and yields more realistic images, but is not as expressive as using spaces of later layers (right side). This varying visual granularity can be seen in the both the final distilled images (bottom row) and the initialization of these images (second bottom row). A proper amount of generative prior is helpful for cross-architecture generalization, while too strong a prior limits expressivity and thus hurts distillation performance (top bar chart). An intermediate space (e.g., F20) balances this trade-off.
using the respective project’s open-source code. For results from MTT and our new method, we use the same set of hyper-parameters \((N, M, T^+, \#\text{iterations}, \text{etc.})\) to ensure a fair comparison. Please see the supplementary material for hyper-parameter details.

**Evaluation Protocol.** After distilling our synthetic datasets with their respective algorithm, we then evaluate them on a set of unseen architectures. To evaluate a synthetic dataset on a given architecture, we train a network from scratch on the distilled dataset and then evaluate it on the validation set.

The training regiment is the same for all networks and datasets: SGD with momentum, \(\ell^2\) weight decay, 1000 total epochs, and 90\% LR decay after the first 500 epochs. An appropriate (fixed) starting learning rate is used for each architecture. This process is repeated 5 times, and the mean validation accuracy \(\pm 1\) standard deviation is reported. Further details can be found in the supplement.

**Network Architectures.** As with prior dataset distillation works [35, 36, 50, 51, 52, 53, 54], we use the ConvNet [21] architecture as our backbone network. A Depth-\(n\) ConvNet consists of \(n\) blocks followed by a fully-connected layer where each block consists of a \(3 \times 3\) convolutional layer with 128 filters, instance normalization [49], ReLU non-linearity, and \(2 \times 2\) average pooling with stride 2.

We used two sets of models for our cross-architecture generalization experiments. For our ImageNet subsets, we used the ResNet-18 [24], VGG-11 [42] (with batch-norm), and ViT-b/16 [17] models provided directly by the torchvision module of PyTorch [38]. For our CIFAR-10 experiments, we use the same models as CAFE [50]: the modified AlexNet, VGG-11, ResNet, and MLP included with the open-source code for DC [54], DSA [53], and DM [52].

Our experiments are enumerated below to highlight the contributions of our proposed method. In Section 4.3.2, we show that our method produces data that generalizes much better to architectures apart from the “backbone” used in the distillation process. In Section 4.3.3, we show that our method greatly out-performs the previous state-of-the-art (MTT [10]) when training on a limited number of expert trajectories. Lastly, in Section 4.3.4, we visualize high-resolution data \((512 \times 512)\) distilled by our method and observe the lack of high-frequency noise when compared to MTT’s results.
Table 4.1: Cross-architecture generalization among dataset distillation methods. Our method reaches comparable accuracy on the backbone architecture used to distill the dataset (ConvNet) with MTT [10] (unlike DM [52]) while also generalizing much better to unseen architectures (averaged across ResNet-18, VGG-11, and ViT). Results reported as $\bar{x} \pm s$ over 5 random initializations. Only our method reaches good performance on both the distilling architecture (ConvNet) and unseen architectures (Cross-Arch.).

<table>
<thead>
<tr>
<th>Method</th>
<th>Birds</th>
<th>Fruits</th>
<th>Food</th>
<th>Marine</th>
<th>Herptiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLD (F20)</td>
<td>38.7±1.4</td>
<td>27.2±1.4</td>
<td>25.7±1.4</td>
<td>28.0±0.9</td>
<td>20.2±0.8</td>
</tr>
<tr>
<td>MTT</td>
<td>38.9±1.7</td>
<td>19.0±1.6</td>
<td>25.9±1.0</td>
<td>29.2±1.0</td>
<td>34.3±0.8</td>
</tr>
<tr>
<td>DM</td>
<td>28.4±1.4</td>
<td>19.9±2.0</td>
<td>21.8±1.9</td>
<td>22.5±1.7</td>
<td>16.8±1.4</td>
</tr>
<tr>
<td>Avg. W</td>
<td>23.7±1.5</td>
<td>24.8±1.3</td>
<td>15.7±0.4</td>
<td>22.5±1.2</td>
<td>17.2±1.3</td>
</tr>
<tr>
<td>Rand. Real</td>
<td>15.3±2.6</td>
<td>16.9±3.1</td>
<td>12.3±1.2</td>
<td>17.6±3.1</td>
<td>12.5±1.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Mammals</th>
<th>Cats</th>
<th>Earth</th>
<th>Sports</th>
<th>ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLD (F20)</td>
<td>20.4±0.9</td>
<td>27.7±1.1</td>
<td>28.4±2.5</td>
<td>21.3±1.3</td>
<td>31.1±0.6</td>
</tr>
<tr>
<td>MTT</td>
<td>40.0±0.8</td>
<td>15.7±1.6</td>
<td>30.2±0.9</td>
<td>14.8±1.2</td>
<td>28.6±1.5</td>
</tr>
<tr>
<td>DM</td>
<td>31.0±0.8</td>
<td>17.0±2.3</td>
<td>20.1±0.1</td>
<td>17.2±1.3</td>
<td>25.3±0.5</td>
</tr>
<tr>
<td>Avg. W</td>
<td>22.0±0.7</td>
<td>25.8±1.1</td>
<td>16.5±0.6</td>
<td>21.3±1.4</td>
<td>22.6±0.5</td>
</tr>
<tr>
<td>Rand. Real</td>
<td>16.4±2.5</td>
<td>16.0±2.7</td>
<td>12.1±1.7</td>
<td>16.0±2.7</td>
<td>14.2±2.1</td>
</tr>
</tbody>
</table>

4.3.1 Finding a Suitable Latent Space

Given the depth of StyleGAN-XL [41], there are many possible latent spaces into which we can distill our dataset. To find the best latent space for cross-architecture generalization, we experiment with the ImageNet Birds-10 subset. In Figure 4.3, we see the random initializations of the class-conditioned generator (top) and final distilled images (bottom) for the “macaw” class. Earlier latent spaces enforce a stronger prior on the distilled image, while later latent spaces offer more flexibility for the optimization to fit to the trajectory matching objective. Examining Figure 4.3, we see that the images distilled into F20 space (a relatively weak prior) result in the best cross-architecture generalization. As such, we use F20 space for the rest of our 128 × 128 experiments.

4.3.2 Improving Cross-Architecture Generalization

Arguably the most lacking point of all previous dataset distillation methods, cross-architecture generalization gives a good understanding of how well the distillation method “understands” the classification task rather than simply overfitting to a given architecture. In Table 4.1, we show cross-architecture results for DM, MTT, and our new method on 10 subsets of
4. Generative Latent Distillation

Table 4.2: Cross-architecture generalization for CIFAR-10, 50 img/class. Images distilled using ConvNetD3 and evaluated on unseen architectures. Our new method generalizes just as well as MTT on CIFAR-10, but we do not see the same gap as with the high-resolution data, implying the generative prior is less critical at low resolution.

<table>
<thead>
<tr>
<th>Method</th>
<th>ConvNet</th>
<th>AlexNet</th>
<th>VGG11</th>
<th>ResNet18</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC [54]</td>
<td>53.9±0.5</td>
<td>28.8±0.7</td>
<td>38.8±1.1</td>
<td>20.9±1.0</td>
<td>28.7±0.7</td>
</tr>
<tr>
<td>CAFE [50]</td>
<td>55.5±0.4</td>
<td>34.0±0.6</td>
<td>40.6±0.8</td>
<td>25.3±0.9</td>
<td>36.7±0.6</td>
</tr>
<tr>
<td>MTT [10]</td>
<td>63.6±0.5</td>
<td>53.9±0.8</td>
<td>61.5±0.4</td>
<td>56.2±0.8</td>
<td>42.0±0.2</td>
</tr>
<tr>
<td>GLD (F8)</td>
<td>62.0±0.3</td>
<td>55.1±0.6</td>
<td>61.1±0.3</td>
<td>57.9±0.6</td>
<td>42.1±0.3</td>
</tr>
</tbody>
</table>

ImageNet: Birds-10, Fruits-10, Food-10, Marine-10, Herptiles-10, Mammals-10, Cats-10, Earth-10, Sports-10, and ImageNette. For each method and dataset, a 1 image-per-class synthetic set is distilled using a Depth-5 ConvNet as the “backbone” architecture. To evaluate cross-architecture generalization, we use the distilled set to train a ResNet-18 [24], VGG-11 [42], and ViT-b/16 [17] from scratch and record the validation accuracy. We record the average validation accuracy on these 3 architectures as “Cross-Arch” in Table 4.1.

We also include 2 non-distillation baselines. “Avg. W” refers to the average output of 1000 random Gaussian fed through the class-conditioned “mapping” network of StyleGAN-XL [41]. This average W vector is then fed through the “synthesis” network to obtain the synthetic images. “Rand. Real” simply refers to random images selected from the training set.

Our method matches MTT’s ability to fit to the backbone architecture (the current state of the art). Furthermore, our method shows significant improvement over both MTT and DM on transfer to ResNet and VGG for all our datasets. However, we do note that our new method has an advantage of using a model pre-trained on a large dataset. On the other hand, our method using a randomly initialized generator still shows better cross-architecture generalization than MTT, indicating that the StyleGAN architecture itself also applies a level of prior to our distilled data.

In Table 4.2, we also include cross-architecture results for CIFAR-10. Our method generalizes just as well as MTT, out-performing all other previous methods. Yet, we do not see the same jump in cross-architecture performance as with the 128×128 data in Table 4.1. This shows that the generative prior is not as imperative for low-resolution data, perhaps because the geometry of a 32×32 images is already quite constrained.
4. Generative Latent Distillation

Figure 4.4: ImageNet Birds-10 distilled using just one expert trajectory. Our standard F20 space (a) handily outperforms MTT (b) on both backbone and cross-architecture accuracy. Using a stronger generative prior of F8 space (c) yields even better backbone and cross-architecture performance in the single-expert setting.

4.3.3 Better Performance with Less Supervision

In MTT [10] (the previous state of the art), training supervision comes in the form of expert trajectories (i.e., model weight trajectories when training on real data), specifying how models should be updated. MTT distilled images are directly trained to match these trajectories.

However, achieving good performance requires a lot of supervision from a large number of expert trajectories [10]. The huge amount of time and disk space needed to train and store the many expert trajectories is one of MTT’s largest limitations. Intuitively, adding a prior should help reduce the amount of supervision needed. Therefore, we hypothesize that by using our generative prior, good performance can be attained with fewer expert trajectories.

The Single-Expert Setting. To test this hypothesis, we explore the single-expert scenario, where the distilled dataset is optimized to only match a single expert training trajectory starting from a single initialization. Distilled with respect to a fixed set of initial weights and testing on random unseen initial weights is extremely challenging. The original Dataset Distillation work [51] argues that many sets of initial weights is necessary for the dis-
4. Generative Latent Distillation

Table 4.3: Single-expert distillation results. With our generative prior, we significantly improve performance of prior state of the art (MTT) in the challenging single-expert setting (where distillation is performed w.r.t. a fixed initialization but tested on random unseen ones) and thus greatly reduce computation and memory cost required to reach good performance. Compared to the many-expert setting (Table 4.1), this single-expert setting has far less supervision and thus a relatively stronger prior (F8 space) becomes optimal.

<table>
<thead>
<tr>
<th>Method</th>
<th>Birds</th>
<th>Fruits</th>
<th>Food</th>
<th>Marine</th>
<th>Herptiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLD (F8)</td>
<td>35.0 ± 0.9</td>
<td>23.2 ± 1.2</td>
<td>19.8 ± 0.9</td>
<td>20.6 ± 1.6</td>
<td>22.0 ± 0.9</td>
</tr>
<tr>
<td>GLD (F20)</td>
<td>30.9 ± 1.2</td>
<td>17.6 ± 1.1</td>
<td>15.5 ± 1.6</td>
<td>20.5 ± 1.5</td>
<td>23.2 ± 1.0</td>
</tr>
<tr>
<td>MTT</td>
<td>16.1 ± 2.0</td>
<td>11.9 ± 1.5</td>
<td>15.1 ± 0.8</td>
<td>16.5 ± 1.4</td>
<td>20.6 ± 2.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Mammals</th>
<th>Cats</th>
<th>Earth</th>
<th>Sports</th>
<th>ImageNette</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLD (F8)</td>
<td>36.7 ± 2.3</td>
<td>22.9 ± 1.2</td>
<td>23.0 ± 1.2</td>
<td>18.0 ± 0.8</td>
<td>26.6 ± 0.8</td>
</tr>
<tr>
<td>GLD (F20)</td>
<td>29.1 ± 2.3</td>
<td>18.3 ± 1.1</td>
<td>19.2 ± 1.2</td>
<td>13.8 ± 0.9</td>
<td>20.2 ± 1.7</td>
</tr>
<tr>
<td>MTT</td>
<td>19.1 ± 3.1</td>
<td>12.1 ± 1.0</td>
<td>13.4 ± 1.1</td>
<td>11.1 ± 1.1</td>
<td>13.1 ± 1.0</td>
</tr>
</tbody>
</table>

tiled images to work. In MTT, using a single expert is shown to yield severely degraded performance [10].

**Generative Prior Reduces the Need for Many Experts.** By applying our deep generative prior, we significantly improve single-expert performance (Figure 4.4, Table 4.3) and, in many datasets, come very close to the many-expert results (Table 4.1). Given the small amount of supervision (only from a single trajectory), a relatively stronger prior (F8) is needed to achieve optimal performance, compared to the many-expert setting (Table 4.1) where the weaker prior (F20) is optimal. However, even the weaker prior significantly improves over MTT (i.e., pixel space). Indeed, our generative prior significantly reduces the amount of supervision required along with the associated computation and memory costs.

### 4.3.4 Extra High Resolution

Finally, we conclude with some brief results on even higher resolution data. When we move beyond 128 × 128 resolution to 512 × 512, the problem of high-frequency noise in MTT is even further exacerbated. Yet, our new method, thanks to the generative prior, still produces smooth, coherent images that also greatly improve cross-architecture generalization (Figure 4.5). Practitioners using distilled dataset for orthogonal tasks (pattern generation [9], etc.) may desire high-resolution, high-fidelity images that MTT (and all other distillation
4. Generative Latent Distillation

Ours (F16)
Backbone validation accuracy: 33.1%
Cross-arch. validation accuracy: 24.6%

MTT [10]
Backbone validation accuracy: 35.7%
Cross-arch. validation accuracy: 14.3%

Figure 4.5: “Macaw” distilled from 512×512 ImageNet Birds-10. While MTT can still form somewhat coherent images at 128×128 resolution, 512×512 MTT images are completely littered with high-frequency noise. Conversely, our weak generative prior allows our method to still produce high-fidelity images even at this much higher resolution. (See the rest of the classes in the supplementary material.)

4.4 Discussion and Limitations

In this work, we propose applying a generative prior to the task of dataset distillation. By applying our deep generative prior to the previous state-of-the-art method, we introduce a new method that greatly improves the generalization of the distilled images while also requiring fewer supervising expert trajectories. This trend extends all the way to (any likely beyond) 512×512 images, allowing us to generate high-quality distilled images at higher resolutions than ever before.

Societal Impact. Dataset Distillation creates a small summarized form of a large dataset. Because this small dataset is synthetic and only trained to contain information useful for classification, it likely incurs less privacy issues than the full dataset (e.g., when released
4. Generative Latent Distillation

to the public), with ideally little to none performance costs. On the other hand, if the real dataset is severely biased [46], the distilled dataset may also exhibit the same biases (inherited via the distillation process and the generative prior). Thus, Dataset Distillation cannot be used to remove dataset biases, but can potentially be a tool to discover and analyze them.

**Limitations.** Although our method significantly reduces storage costs of the previous state of the art (MTT [10]), it remains extremely memory-hungry (for the distillation step). On top of the bilevel optimization, we must also fit StyleGAN-XL’s parameters, activations, and gradients into (GPU) memory, and, as the name alludes, StyleGAN-XL is quite large. Our image regeneration trick (Section 4.2.3) solves this new memory sink at the cost of a second pass through the generator. Since a large portion of our method’s computational time is spent on the forward pass through the generator, using the regeneration trick greatly increases our distillation training time. Conversely, if a user has the VRAM resources, they can choose to simply not do the regeneration trick and keep the whole graph in memory, cutting the time to distill our datasets by over a third. At $\sim 1.6$ seconds per iteration, our method (with regeneration) takes $\sim 2.2$ hours to perform 5k iterations distilling a $128 \times 128$ dataset into F20 space with a single RTX6000 while the same number of MTT iterations would take $\sim 0.7$ hours.
Bibliography


[3] Cenk Baykal, Murad Tukan, Dan Feldman, and Daniela Rus. Small coresets to represent large training data for support vector machines. 2018. 33


47
Bibliography


[19] Fastai. Fastai/imagenette: A smaller subset of 10 easily classified classes from imagenet, and a little more french. URL https://github.com/fastai/imagenette. xvi, xvii, 21, 38, 75, 81, 92, 97


[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desma-


Appendix A

Matching Training Trajectories

A.1 Additional Visualizations

We first include some additional visualizations here. CIFAR-100 (1 image per class) can be seen in Figure A.4. All of Tiny ImageNet (1 image per class) is broken up into Figures A.11 and A.12. We specifically show the 10 best and worst-performing distilled classes in Figures A.5 and A.6 respectively. We include 10 image per class visualizations of all our 128×128 ImageNet subsets in Figures A.13-A.18.

A.2 Additional Quantitative Results

Analysis of learned learning rates $\alpha$. In Figure A.3, we explore the effect of our learnable synthetic step size $\alpha$. The left plot confirms that we learn different values of $\alpha$ for different combinations of $M$ and $N$. The logic here is that different numbers of synthetic steps $N$ require a different step size $\alpha$ to cover the same distance as $M$ real steps. The right plot illustrates the practical benefits of our adaptive learning rate; instead of yet another hyper-parameter to tune, our adaptive learning rate works from a wide range of initializations.

Effects of ZCA Whitening    Note that DC, DSA, and DM do not use ZCA normalization, while KIP started using ZCA as it was a “crucial ingredient for [their] strong results.” We report our results w/o ZCA in Figure A.1 (Left). We find that ZCA normalization is not
A. Matching Training Trajectories

critical to our performance. However, the expert models trained without ZCA normalization take significantly longer to converge. Thus, when distilling using these models as experts, we must use a larger value of $T^+$ (and therefore save more model snapshots). When we use a larger value of $T^+$ for non-ZCA distillations, we get results comparable to or even better than those of the ZCA distillations. In short, ZCA helps expert convergence but does not notably improve distillation performance.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Img/Cls</th>
<th>Yes-ZCA</th>
<th>No-ZCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>1</td>
<td>46.3</td>
<td>45.2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>65.3</td>
<td>62.8</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>71.5</td>
<td>71.6</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>1</td>
<td>24.3</td>
<td>22.7</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>39.4</td>
<td>40.1</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>47.7</td>
<td>47.2</td>
</tr>
</tbody>
</table>

Figure A.1: **Left:** ZCA Ablation. **Right:** Distillation Time.

**Additional Ablation Studies**

**Initialization, normalization, and augmentation.** In the main paper, we show ablations over several hyper-parameters. Here, we study the role of initialization, data normalization, and data augmentation for CIFAR-100 (1 image per class) in Table A.1. For initialization in particular, recall that we typically initialize our synthetic images with real samples. Here, we evaluate initializing with Gaussian noise instead. Visualizations of these distilled sets can be seen in Figures A.7-A.9. We also include a visualization of a set distilled with only one expert trajectory in Figure A.10.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Ours</th>
<th>Gauss. Init.</th>
<th>w/o ZCA</th>
<th>w/o Diff. Aug.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc.</td>
<td>24.3 ± 0.3</td>
<td>22.9 ± 0.4</td>
<td>19.1 ± 0.6</td>
<td>20.7 ± 0.7</td>
</tr>
</tbody>
</table>

Table A.1: As we ablate our categorical hyper-parameters, we still achieve state-of-the-art performance (compared to **DSA:** 13.9%). This is evaluated on CIFAR-100 with 1 image per class. Each design choice in our final method improves the performance of distilled images. Here we use the default set of hyper-parameters for these ablations.
A. Matching Training Trajectories

Figure A.2: **Left:** We see logarithmic performance improvement with respect to the number of expert trajectories used, quickly saturating near 200. **Right:** The upper bound on the expert epoch at which the synthetic data starts working cannot be too high or low to ensure quality learning signal.

Figure A.3: **Left:** Our learned synthetic step size $\alpha$ seems to scale inversely with the number of synthetic steps $N$ to cover the same distance in parameter space as the expert steps $M$. **Right:** Having a learnable step size $\alpha$ saves us from having to search for an appropriate fixed $\alpha_0$. 
A. Matching Training Trajectories

**Performance w.r.t. the number of expert trajectories.** Since they effectively make up our method’s “training set,” it is reasonable to assume that having more expert trajectories would lead to better performance. We see that this is indeed the case for the CIFAR-100, 1 image per class setting in Figure A.2 (left). However, what’s most interesting is the sharp, logarithmic increase in validation accuracy w.r.t. the number of experts. We note the most amount of improvement when increasing from 1 to 20 experts but see almost complete saturation by the time we reach 200. Given how high-dimensional the parameter space of a neural network is, it is remarkable that we can achieve such high performance with so few expert trajectories.

**Performance w.r.t. expert time-step range.** When we initialize our student networks, we do so at a randomly selected time-step from an expert trajectory. We find that it is important to put an upper bound on this starting time-step (Figure A.2, right). If the upper bound is too high, the synthetic data receives gradients from points where the experts movements are small and uninformative. If it is too low, the synthetic data is never exposed to mid and later points in the trajectories, missing out on a significant portion of the training dynamics.

A.3 Experiment Details

**Hyper-Parameters.** In Table A.2, we enumerate the hyper-parameters used for our results reported in the main text. Limited compute forced us to batch our synthetic data for some of the larger sets. The “ConvNet” architectures are as explained in the main text.

**Compute resources.** We had a relatively limited compute budget for our experiments, using any GPU we could access. As such, our experiments were run on a mixture of RTX2080ti, RTX3090, and RTX6000 GPUs. The largest amount of VRAM we used for a single experiment was 144GB over 6xRTX6000 GPUs.

**Training Time.** Distillation time varied based on dataset and type and number of GPUs used. Regardless of dataset or compute resources, time per distillation iteration scaled linearly with the number of synthetic steps $N$. For CIFAR-100, 1 image per class with $N = 20$, we had an average time of 0.6 seconds per distillation step when using a single RTX3090. We ran our experiments for 10000 distillation steps but saw the most improvement within the first 1000.

Our distillation time, in general, is comparable to DC/DSA, as they also utilize a bi-level
A. Matching Training Trajectories

Figure A.4: CIFAR-100, 1 Image Per Class
A. Matching Training Trajectories

| Dataset       | Model   | Img/Cls | Synthetic Steps (N) | Expert Epochs (M)† | Max Start Epoch (T+) | Synthetic Batch Size (|b|) | ZCA |
|---------------|---------|---------|---------------------|-------------------|----------------------|--------------------------|-----|
| CIFAR-10      | ConvNetD3 | 1       | 50                  | 2                 | 2                    | -                        | Y   |
|               |         | 10      | 30                  | 2                 | 20                   | -                        | Y   |
|               |         | 50      | 30                  | 2                 | 40                   | -                        | N   |
| CIFAR-100     | ConvNetD3 | 1       | 20                  | 3                 | 20                   | -                        | Y   |
|               |         | 10      | 20                  | 2                 | 20                   | -                        | N   |
|               |         | 50      | 80                  | 2                 | 40                   | -                        | Y   |
| Tiny ImageNet | ConvNetD4 | 1       | 10                  | 2                 | 10                   | -                        | -   |
|               |         | 10      | 20                  | 2                 | 40                   | 200                     | -   |
|               |         | 50      | 20                  | 2                 | 40                   | 300                     | -   |
| ImageNet (All)| ConvNetD5 | 1       | 20                  | 2                 | 10                   | -                        | -   |
|               |         | 10      | 20                  | 2                 | 10                   | 20                      | -   |

Table A.2: Hyper-parameters used for our best-performing distillation experiments. A synthetic batch size of “-” indicates that we used the full support set at each synthetic step. Note: instead of number of expert updates (M), here we list number of expert epochs (M†) for simplicity across datasets.

optimization. In the 10 img/class setting (for example), DC/DSA trains on the synthetic data for 50 epochs on the between each update. We include a sample distillation curve in Figure A.1 (Right). Both experiments were run on RTX3090. Note that KIP requires over 1,000 GPU hours.

Regarding the distillation time for learning different sets on CIFAR10/100 and TinyImageNet, we report them in Table A.3. Note that most improvement occurs within the first 1k iterations, but we continue training for 10k.

**KIP to NN** In the KIP paper, results are presented for images distilled using the neural tangent kernel method and then evaluated by training a modified width-1024 ConvNetD3. Aside from the increased width of the finite model, the ConvNet architecture used in the KIP paper also has an additional 1-layer convolutional stem.

Using the training notebook provided with the KIP paper, we perform an exhaustive search over a reasonable set of hyper-parameters for the KIP to width-128 NN problem: checkpoint ∈ {112, 335, 1000}, weight_decay ∈ {0, 0.0001, 0.001, 0.01}, aug ∈ {True, False}, zca ∈ {True, False}, label_learn ∈ {True, False}, and norm ∈ {none, instance}. The architecture originally used for KIP to NN in the KIP
### A. Matching Training Trajectories

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Img/Cls</th>
<th>1 Iter. (sec)</th>
<th>1k Iter. (min)</th>
<th>5k Iter. (min)</th>
<th>10k Iter. (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>1</td>
<td>0.5</td>
<td>8</td>
<td>42</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.6</td>
<td>10</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.8</td>
<td>13</td>
<td>67</td>
<td>133</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>1</td>
<td>0.6</td>
<td>10</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.8</td>
<td>13</td>
<td>67</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1.9</td>
<td>32</td>
<td>158</td>
<td>317</td>
</tr>
<tr>
<td>Tiny ImageNet</td>
<td>1</td>
<td>1.1</td>
<td>18</td>
<td>92</td>
<td>183</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2.3</td>
<td>38</td>
<td>192</td>
<td>383</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>2.6</td>
<td>43</td>
<td>217</td>
<td>433</td>
</tr>
</tbody>
</table>

Table A.3: Distillation time for each dataset and support size.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Img/Cls</th>
<th>Learn Labels</th>
<th>Aug.</th>
<th>ZCA</th>
<th>Norm</th>
<th>Weight Decay</th>
<th>Ckpt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>1</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>0.001</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>0.001</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>I</td>
<td>0.01</td>
<td>112</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>I</td>
<td>0.001</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>I</td>
<td>0.001</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table A.4: Optimal hyper-parameters for our reported width-128 KIP to NN results. These were obtained via grid search using the notebook provided by the KIP authors.

The paper contained no normalization layers. However, we found that with the smaller width, this model could not even converge on the synthetic training data for CIFAR-100, so we added instance normalization layers as found in the ConvNets we and DC, DSA, and DM use.

In Table A.4, we include the optimal hyper-parameters from this search that were used to obtain the KIP to NN (128-width) values reported in the main text.
A. Matching Training Trajectories

Figure A.5: Most-correct distilled images for Tiny ImageNet (≥ 30%)

Figure A.6: Least-correct distilled images for Tiny ImageNet (≤ 4%)
Figure A.7: CIFAR-100, Initialized as Random Noise
Figure A.8: CIFAR-100, No ZCA Whitening
A. Matching Training Trajectories

Figure A.9: CIFAR-100, No Differentiable Augmentation
A. Matching Training Trajectories

Figure A.10: CIFAR-100, Only 1 Expert Trajectory
Figure A.11: Tiny ImageNet, 1 Image Per Class (Classes 1-100)
Figure A.12: Tiny ImageNet, 1 Image Per Class (Classes 101-200)
Figure A.13: ImageNette, 10 Images Per Class
A. Matching Training Trajectories

Figure A.14: ImageWoof, 10 Images Per Class
A. Matching Training Trajectories

Figure A.15: ImageSquawk, 10 Images Per Class
Figure A.16: ImageMeow, 10 Images Per Class
Figure A.17: ImageFruit, 10 Images Per Class
A. Matching Training Trajectories

Figure A.18: ImageYellow, 10 Images Per Class
Appendix B

Texture Synthesis

B.1 Performance as Training Data

Despite not explicitly following the dataset distillation objective, our distilled textures still achieve surprising classification results. By training on only random patches of our ImageSquawk textures, we achieve 39.6% test accuracy. Likewise, we achieve 26.8% test accuracy by only training on random patches of ImageFruit.
B. Texture Synthesis

B.2 More Visualizations

Figure B.1: ImageFruit Distilled Textures Tiled 3x3
Appendix C

Generative Latent Distillation

C.1 Hyper-Parameters and Experimental Details

For the experiments on MTT and our new method, we base our experiments on the open-source MTT code [10]. To optimize the distilled images/latents and learnable synthetic step-size (α), we use the same optimizer as MTT: SGD with 0.5 momentum. For the images (MTT) and F-latents (ours), we use a learning rate of $10^4$. For the $W^+$ latents (ours), we use a learning rate of $10^2$. For the learnable step-size, we use a learning rate of $10^{-6}$ and initialize at $10^{-2}$.

For our $128 \times 128$ and $512 \times 512$ resolution experiments (for both MTT and our method), we set the number of synthetic steps per iteration ($N$) as 10, the number of real epochs to match ($M$) as 2, and the maximum starting epoch ($T^+$) set to 5. For CIFAR-10, the 50 image/class experiments (for both MTT and our method) use $N = 30$, $M = 2$, and $T^+ = 5$. In the many-expert setting, we use 50 total expert trajectories. In the single-expert settings, we use just 1 expert trajectory. All experiments on MTT and our new method are run for 5k iterations and then evaluated via the protocol described in the body of the paper.

For the DM experiments, we use all the default parameters: image learning rate of 1.0, 20k distillation iterations, and a “real batch size” (see DM [52]) of 256.

All $32 \times 32$, $128 \times 128$, and $512 \times 512$ experiments are distilled using ConvNetD3, ConvNetD5, and ConvNetD7 respectively as the backbone.

The same suite of differentiable augmentations (originally from the DSA codebase
C. Generative Latent Distillation

[53]) is used for all experiments: color, crop, cutout, flip, scale, and rotate with the default parameters.

To obtain the expert trajectories used by MTT and our method, we train a model from scratch on the real dataset for 15 epochs of SGD with a learning rate of $10^{-2}$, a batch size of 256, and NO momentum or regularization.

Our experiments were run on a combination of RTX2080ti, RTX3090, and RTX6000 GPUs depending on availability.

C.2 Licensing

We use and adapt pre-existing code from StyleGAN-XL [41], DC [54], DSA [53], DM [52], and MTT [10], all of which are under the MIT License. ImageNette [19] is under the Apache 2.0 License. CIFAR-10 [30] is under the MIT license. The license of ImageNet is unknown.

C.3 Dataset Specifications

<table>
<thead>
<tr>
<th>Dataset</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Birds</td>
<td>Peacock</td>
<td>Flamingo</td>
<td>Macaw</td>
<td>Pelican</td>
<td>King Penguin</td>
<td>Bald Eagle</td>
<td>Toucan</td>
<td>Ostrich</td>
<td>Black Swan</td>
<td>Cockatoo</td>
</tr>
<tr>
<td>Fruits</td>
<td>Pineapple</td>
<td>Banana</td>
<td>Strawberry</td>
<td>Orange</td>
<td>Lemon</td>
<td>Pomegranate</td>
<td>Fig</td>
<td>Bell Pepper</td>
<td>Cucumber</td>
<td>Guava Smaller</td>
</tr>
<tr>
<td>Food</td>
<td>Cheeseburger</td>
<td>Hotdog</td>
<td>Pretzel</td>
<td>Pizza</td>
<td>French Loaf</td>
<td>Ice Cream</td>
<td>Guacamole</td>
<td>Carbonara</td>
<td>Bagel</td>
<td>Trifle</td>
</tr>
<tr>
<td>Marine</td>
<td>Rock Beauty</td>
<td>Clownfish</td>
<td>Loggerhead</td>
<td>Puffer</td>
<td>Stingray</td>
<td>Jellyfish</td>
<td>Starfish</td>
<td>Eel</td>
<td>Anemone</td>
<td>American Lobster</td>
</tr>
<tr>
<td>Herptiles</td>
<td>Axolotl</td>
<td>Tree Frog</td>
<td>King Snake</td>
<td>American Chameleon</td>
<td>Iguana</td>
<td>Eft</td>
<td>Fire Salamander</td>
<td>Box Turtle</td>
<td>American Alligator</td>
<td>Agama</td>
</tr>
<tr>
<td>Mammals</td>
<td>African Elephant</td>
<td>Red Panda</td>
<td>Camel</td>
<td>Zebra</td>
<td>Guinea Pig</td>
<td>Kangaroo</td>
<td>Platypus</td>
<td>Arctic Fox</td>
<td>Porcupine</td>
<td>Gorilla</td>
</tr>
<tr>
<td>Cats</td>
<td>Tabby Cat</td>
<td>Bengal Cat</td>
<td>Persian Cat</td>
<td>Siamese Cat</td>
<td>Egyptian Cat</td>
<td>Lion</td>
<td>Tiger</td>
<td>Jaguar</td>
<td>Snow Leopard</td>
<td>Lynx</td>
</tr>
<tr>
<td>Earth</td>
<td>Volcano</td>
<td>Alp</td>
<td>Lakeside</td>
<td>Geyser</td>
<td>Coral Reef</td>
<td>Sandbar</td>
<td>Promontory</td>
<td>Seashore</td>
<td>Cliff</td>
<td>Valley</td>
</tr>
<tr>
<td>Sports</td>
<td>Baseball</td>
<td>Basketball</td>
<td>Croquet</td>
<td>Golf Ball</td>
<td>Ping-Pong Ball</td>
<td>Rugby Ball</td>
<td>Soccer Ball</td>
<td>Tennis Ball</td>
<td>Volleyball</td>
<td>Hockey Puck</td>
</tr>
<tr>
<td>ImageNette</td>
<td>Tench</td>
<td>English Springer</td>
<td>Cassette Player</td>
<td>Chainsaw</td>
<td>Church</td>
<td>French Horn</td>
<td>Garbage Truck</td>
<td>Gas Pump</td>
<td>Golf Ball</td>
<td>Parachute</td>
</tr>
</tbody>
</table>

Our high-resolution data is taken directly from the ImageNet 1k dataset [15] using PyTorch’s built-in ImageNet loader [38]. To train our expert trajectories, we use data from the ImageNet training set. To compile our training set for the expert trajectories, we select the classes from the given subset, resize the short side of the image to the given
Table C.2: Whole-dataset validation accuracy for 128×128 ImageNet subsets. These values represent the upper-bound of the dataset distillation problem given our evaluation training protocol. For consistency, we include “Cross-Arch” here as the average of “ResNet-18,” “VGG-11,” and “ViT-b/16.”

<table>
<thead>
<tr>
<th>Model</th>
<th>Birds</th>
<th>Fruits</th>
<th>Food</th>
<th>Marine</th>
<th>Herptiles</th>
<th>Mammals</th>
<th>Cats</th>
<th>Earth</th>
<th>Sports</th>
<th>ImageNette</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNetD5</td>
<td>87.3±0.8</td>
<td>63.9±1.2</td>
<td>64.4±1.3</td>
<td>74.0±0.5</td>
<td>73.8±0.4</td>
<td>81.0±0.7</td>
<td>66.6±0.9</td>
<td>61.5±1.0</td>
<td>67.2±1.4</td>
<td>86.8±0.5</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>83.0±0.8</td>
<td>63.2±0.6</td>
<td>60.6±0.9</td>
<td>62.4±1.2</td>
<td>66.6±0.9</td>
<td>75.0±0.1</td>
<td>61.4±1.5</td>
<td>62.6±2.0</td>
<td>63.6±1.0</td>
<td>84.4±1.1</td>
</tr>
<tr>
<td>VGG-11</td>
<td>71.2±0.2</td>
<td>54.9±1.5</td>
<td>48.5±1.0</td>
<td>56.2±0.9</td>
<td>57.0±0.6</td>
<td>67.4±0.8</td>
<td>51.1±0.6</td>
<td>52.8±1.6</td>
<td>55.5±1.5</td>
<td>74.9±1.5</td>
</tr>
<tr>
<td>ViT-b/16</td>
<td>66.2±0.7</td>
<td>51.2±0.2</td>
<td>46.7±1.3</td>
<td>54.4±1.3</td>
<td>51.3±0.8</td>
<td>61.2±1.3</td>
<td>42.6±1.1</td>
<td>52.6±1.6</td>
<td>54.4±0.5</td>
<td>66.4±0.6</td>
</tr>
<tr>
<td>Cross-Arch</td>
<td>73.5±0.6</td>
<td>56.4±0.8</td>
<td>51.9±1.1</td>
<td>57.9±1.1</td>
<td>58.3±0.8</td>
<td>67.9±0.7</td>
<td>51.7±1.1</td>
<td>56.0±1.7</td>
<td>57.8±1.0</td>
<td>75.2±1.1</td>
</tr>
</tbody>
</table>

resolution, and take a center crop according to the given resolution (as done by MTT [10]). The validation set is obtained in the same way from the ImageNet validation set.

For an enumeration of which ImageNet classes are in each of our datasets, please see Table C.1.

In Table C.2, we report the validation accuracy obtained by training each model on the full, real dataset. We use the same training protocol as we use for evaluating our synthetic sets, except we train for 100 total epochs and decay the learning rate after the first 50. These values represent the upper bound of our distilled image performance.

C.4 More Visualizations

In Figures C.1-C.9, we include comparisons of the 3 distillation methods capable of distilling high-resolution images (DM [52], MTT [10], and ours), analogous to Figure 4.2 in the main text.

In Figures C.21-C.29, we include comparisons of the 3 distillation spaces in the single-expert setting (Pixel, F20, and F8), analogous to Figure 4.4 in the main text.

In Figure C.10, we visualize 1 image per class of CIFAR-10 distilled into different spaces.

In Figures C.11-C.20, we include visualizations of all our ImageNet subsets distilled into various spaces.

In Figures C.30-C.32, we include the rest of our distilled 512×512 Birds-10 synthetic images, extending Figure 4.5 from the main text.
C. Generative Latent Distillation

Figure C.1: High-resolution distillation methods applied to the Fruits-10 subset of ImageNet.

Figure C.2: High-resolution distillation methods applied to the Food-10 subset of ImageNet.
C. Generative Latent Distillation

Figure C.3: High-resolution distillation methods applied to the Marine-10 subset of ImageNet.

Figure C.4: High-resolution distillation methods applied to the Herptiles-10 subset of ImageNet.
C. Generative Latent Distillation

(a) Distribution Matching (DM) [52]. Cross-arch. validation accuracy: 17.0%

(b) Matching Training Trajectories (MTT) [10]. Cross-arch. validation accuracy: 15.7%

(c) Our Method: Generative Latent Distillation. Cross-arch. validation accuracy: 27.7%

Figure C.5: High-resolution distillation methods applied to the Mammals-10 subset of ImageNet.

(a) Distribution Matching (DM) [52]. Cross-arch. validation accuracy: 17.2%

(b) Matching Training Trajectories (MTT) [10]. Cross-arch. validation accuracy: 14.8%

(c) Our Method: Generative Latent Distillation. Cross-arch. validation accuracy: 21.3%

Figure C.6: High-resolution distillation methods applied to the Cats-10 subset of ImageNet.
C. Generative Latent Distillation

Figure C.7: High-resolution distillation methods applied to the Earth-10 subset of ImageNet.

Figure C.8: High-resolution distillation methods applied to the Sports-10 subset of ImageNet.
C. Generative Latent Distillation

Figure C.9: High-resolution distillation methods applied to the ImageNette [19] subset of ImageNet.

(a) Distribution Matching (DM) [52]. Cross-arch. validation accuracy: 23.5%

(b) Matching Training Trajectories (MTT) [10]. Cross-arch. validation accuracy: 18.9%

(c) Our Method: Generative Latent Distillation. Cross-arch. validation accuracy: **28.3%**

81
C. Generative Latent Distillation

Figure C.10: CIFAR-10 distilled into different spaces with varying levels of prior. Unlike in the 50 images/class setting (Table 4.2), choice of distillation space seems to have a significant impact on CIFAR-10 cross-architecture generalization in the 1 image/class setting.
C. Generative Latent Distillation

<table>
<thead>
<tr>
<th>Peacock</th>
<th>Flamingo</th>
<th>Macaw</th>
<th>Pelican</th>
<th>King</th>
<th>Bald Eagle</th>
<th>Toucan</th>
<th>Ostrich</th>
<th>Black Swan</th>
<th>Cockatoo</th>
</tr>
</thead>
</table>

- **W⁺**: Backbone accuracy: 35.4%. Cross-arch. validation accuracy: 23.2%
- **F4**: Backbone accuracy: 35.7%. Cross-arch. validation accuracy: 21.8%
- **F8**: Backbone accuracy: 35.8%. Cross-arch. validation accuracy: 23.4%
- **F12**: Backbone accuracy: 35.0%. Cross-arch. validation accuracy: 23.9%
- **F16**: Backbone accuracy: 38.1%. Cross-arch. validation accuracy: 22.4%
- **F20**: Backbone accuracy: 38.7%. Cross-arch. validation accuracy: 27.2%
- **F24**: Backbone accuracy: 38.6%. Cross-arch. validation accuracy: 20.4%

**Pixel (MTT [10])**. Backbone accuracy: 38.9%. Cross-arch. validation accuracy: 19.0%

Figure C.11: Birds-10 subset of ImageNet distilled into different spaces with varying levels of prior.
Figure C.12: Fruits-10 subset of ImageNet distilled into different spaces with varying levels of prior.
C. Generative Latent Distillation

<table>
<thead>
<tr>
<th>Cheeseburger</th>
<th>Hotdog</th>
<th>Pretzel</th>
<th>Pizza</th>
<th>French Loaf</th>
<th>Ice Cream</th>
<th>Guacamole</th>
<th>Carbonara</th>
<th>Bagel</th>
<th>Trifle</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Images" /></td>
<td><img src="image2.png" alt="Images" /></td>
<td><img src="image3.png" alt="Images" /></td>
<td><img src="image4.png" alt="Images" /></td>
<td><img src="image5.png" alt="Images" /></td>
<td><img src="image6.png" alt="Images" /></td>
<td><img src="image7.png" alt="Images" /></td>
<td><img src="image8.png" alt="Images" /></td>
<td><img src="image9.png" alt="Images" /></td>
<td><img src="image10.png" alt="Images" /></td>
</tr>
</tbody>
</table>

- **W⁺**: Backbone accuracy: 25.0%. Cross-arch. validation accuracy: 15.9%
- **F4**: Backbone accuracy: 26.6%. Cross-arch. validation accuracy: 15.0%
- **F8**: Backbone accuracy: 25.1%. Cross-arch. validation accuracy: 17.7%
- **F12**: Backbone accuracy: 27.1%. Cross-arch. validation accuracy: 18.7%
- **F16**: Backbone accuracy: 26.9%. Cross-arch. validation accuracy: 17.8%
- **F20**: Backbone accuracy: 28.0%. Cross-arch. validation accuracy: **20.2%**
- **F24**: Backbone accuracy: 29.0%. Cross-arch. validation accuracy: 16.2%

Pixel (MTT [10]). Backbone accuracy: 29.2%. Cross-arch. validation accuracy: 13.8%

Figure C.13: Food-10 subset of ImageNet distilled into different spaces with varying levels of prior.
C. Generative Latent Distillation

<table>
<thead>
<tr>
<th>Rock Beauty</th>
<th>Clownfish</th>
<th>Loggerhead</th>
<th>Puffer</th>
<th>Stingray</th>
<th>Jellyfish</th>
<th>Starfish</th>
<th>Eel</th>
<th>Anemone</th>
<th>American Lobster</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="figures/wplus.png" alt="Images" /></td>
<td>Backbone accuracy: 34.3%. Cross-arch. validation accuracy: 21.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="figures/f4.png" alt="Images" /></td>
<td>Backbone accuracy: 30.1%. Cross-arch. validation accuracy: 21.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="figures/f8.png" alt="Images" /></td>
<td>Backbone accuracy: 30.9%. Cross-arch. validation accuracy: 23.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="figures/f12.png" alt="Images" /></td>
<td>Backbone accuracy: 31.4%. Cross-arch. validation accuracy: 22.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="figures/f16.png" alt="Images" /></td>
<td>Backbone accuracy: 32.2%. Cross-arch. validation accuracy: 23.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="figures/f20.png" alt="Images" /></td>
<td>Backbone accuracy: 33.0%. Cross-arch. validation accuracy: <strong>26.7%</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="figures/f24.png" alt="Images" /></td>
<td>Backbone accuracy: 33.3%. Cross-arch. validation accuracy: 24.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pixel (MTT [10]). Backbone accuracy: 34.3%. Cross-arch. validation accuracy: 21.4%

Figure C.14: Marine-10 subset of ImageNet distilled into different spaces with varying levels of prior.
### C. Generative Latent Distillation

<table>
<thead>
<tr>
<th>Axolotl</th>
<th>Tree Frog</th>
<th>King Snake</th>
<th>American Chameleon</th>
<th>Iguana</th>
<th>Eft</th>
<th>Fire Salamander</th>
<th>Box Turtle</th>
<th>American Alligator</th>
<th>Agama</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
</tr>
</tbody>
</table>

**W⁺.** Backbone accuracy: 29.9%. Cross-arch. validation accuracy: 18.2%

| ![Image](image11.png) | ![Image](image12.png) | ![Image](image13.png) | ![Image](image14.png) | ![Image](image15.png) | ![Image](image16.png) | ![Image](image17.png) | ![Image](image18.png) | ![Image](image19.png) | ![Image](image20.png) |

**F4.** Backbone accuracy: 29.4%. Cross-arch. validation accuracy: 17.8%

| ![Image](image21.png) | ![Image](image22.png) | ![Image](image23.png) | ![Image](image24.png) | ![Image](image25.png) | ![Image](image26.png) | ![Image](image27.png) | ![Image](image28.png) | ![Image](image29.png) | ![Image](image30.png) |

**F8.** Backbone accuracy: 29.7%. Cross-arch. validation accuracy: 17.9%

| ![Image](image31.png) | ![Image](image32.png) | ![Image](image33.png) | ![Image](image34.png) | ![Image](image35.png) | ![Image](image36.png) | ![Image](image37.png) | ![Image](image38.png) | ![Image](image39.png) | ![Image](image40.png) |

**F12.** Backbone accuracy: 30.4%. Cross-arch. validation accuracy: 18.2%

| ![Image](image41.png) | ![Image](image42.png) | ![Image](image43.png) | ![Image](image44.png) | ![Image](image45.png) | ![Image](image46.png) | ![Image](image47.png) | ![Image](image48.png) | ![Image](image49.png) | ![Image](image50.png) |

**F16.** Backbone accuracy: 32.0%. Cross-arch. validation accuracy: 19.5%

| ![Image](image51.png) | ![Image](image52.png) | ![Image](image53.png) | ![Image](image54.png) | ![Image](image55.png) | ![Image](image56.png) | ![Image](image57.png) | ![Image](image58.png) | ![Image](image59.png) | ![Image](image60.png) |

**F20.** Backbone accuracy: 32.9%. Cross-arch. validation accuracy: **21.1%**

| ![Image](image61.png) | ![Image](image62.png) | ![Image](image63.png) | ![Image](image64.png) | ![Image](image65.png) | ![Image](image66.png) | ![Image](image67.png) | ![Image](image68.png) | ![Image](image69.png) | ![Image](image70.png) |

**F24.** Backbone accuracy: 33.9%. Cross-arch. validation accuracy: 16.5%

| ![Image](image71.png) | ![Image](image72.png) | ![Image](image73.png) | ![Image](image74.png) | ![Image](image75.png) | ![Image](image76.png) | ![Image](image77.png) | ![Image](image78.png) | ![Image](image79.png) | ![Image](image80.png) |

Pixel (MTT [10]). Backbone accuracy: 34.5%. Cross-arch. validation accuracy: 12.9%

Figure C.15: Herptiles-10 subset of ImageNet distilled into different spaces with varying levels of prior.
### C. Generative Latent Distillation

The following table and images illustrate the distillation process into different spaces with varying levels of prior:

<table>
<thead>
<tr>
<th>African Elephant</th>
<th>Red Panda</th>
<th>Camel</th>
<th>Zebra</th>
<th>Guinea Pig</th>
<th>Kangaroo</th>
<th>Platypus</th>
<th>Arctic Fox</th>
<th>Porcupine</th>
<th>Gorilla</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
<td><img src="image3" alt="Image" /></td>
<td><img src="image4" alt="Image" /></td>
<td><img src="image5" alt="Image" /></td>
<td><img src="image6" alt="Image" /></td>
<td><img src="image7" alt="Image" /></td>
<td><img src="image8" alt="Image" /></td>
<td><img src="image9" alt="Image" /></td>
<td><img src="image10" alt="Image" /></td>
</tr>
</tbody>
</table>

- **W⁺.** Backbone accuracy: 38.6%. Cross-arch. validation accuracy: 23.9%
- **F4.** Backbone accuracy: 38.3%. Cross-arch. validation accuracy: 24.0%
- **F8.** Backbone accuracy: 38.8%. Cross-arch. validation accuracy: 25.6%
- **F12.** Backbone accuracy: 38.9%. Cross-arch. validation accuracy: 24.1%
- **F16.** Backbone accuracy: 39.2%. Cross-arch. validation accuracy: 24.7%
- **F20.** Backbone accuracy: 40.6%. Cross-arch. validation accuracy: **27.7%**
- **F24.** Backbone accuracy: 40.8%. Cross-arch. validation accuracy: 17.1%

Pixel (MTT [10]). Backbone accuracy: 40.0%. Cross-arch. validation accuracy: 15.7%

**Figure C.16:** Mammals-10 subset of ImageNet distilled into different spaces with varying levels of prior.
C. Generative Latent Distillation

![Image](image.png)

**Figure C.17:** Cats-10 subset of ImageNet distilled into different spaces with varying levels of prior.
C. Generative Latent Distillation

![Earth-10 subset of ImageNet distilled into different spaces with varying levels of prior.](image)

- **W⁺.** Backbone accuracy: 27.8%. Cross-arch. validation accuracy: 21.9%
- **F4.** Backbone accuracy: 29.6%. Cross-arch. validation accuracy: 20.5%
- **F8.** Backbone accuracy: 29.2%. Cross-arch. validation accuracy: 20.5%
- **F12.** Backbone accuracy: 29.2%. Cross-arch. validation accuracy: 20.0%
- **F16.** Backbone accuracy: 28.4%. Cross-arch. validation accuracy: 22.1%
- **F20.** Backbone accuracy: 31.1%. Cross-arch. validation accuracy: **21.8%**
- **F24.** Backbone accuracy: 29.2%. Cross-arch. validation accuracy: 14.5%

Pixel (MTT [10]). Backbone accuracy: 28.6%. Cross-arch. validation accuracy: 13.8%

Figure C.18: Earth-10 subset of ImageNet distilled into different spaces with varying levels of prior.
Figure C.19: Sports-10 subset of ImageNet distilled into different spaces with varying levels of prior.
C. Generative Latent Distillation

<table>
<thead>
<tr>
<th>Location</th>
<th>Backbone accuracy</th>
<th>Cross-arch. validation accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>W+</td>
<td>38.4%</td>
<td>25.7%</td>
</tr>
<tr>
<td>F4</td>
<td>40.1%</td>
<td>26.3%</td>
</tr>
<tr>
<td>F8</td>
<td>38.8%</td>
<td>25.4%</td>
</tr>
<tr>
<td>F12</td>
<td>39.0%</td>
<td>24.1%</td>
</tr>
<tr>
<td>F16</td>
<td>41.9%</td>
<td>29.2%</td>
</tr>
<tr>
<td>F20</td>
<td>44.9%</td>
<td>28.5%</td>
</tr>
<tr>
<td>F24</td>
<td>46.4%</td>
<td>21.2%</td>
</tr>
<tr>
<td>Pixel (MTT [10])</td>
<td>45.3%</td>
<td>18.9%</td>
</tr>
</tbody>
</table>

Figure C.20: ImageNette [19] subset of ImageNet distilled into different spaces with varying levels of prior.
C. Generative Latent Distillation

Figure C.21: Different distillation spaces in the single-expert setting on the Fruits-10 subset of ImageNet.

Figure C.22: Different distillation spaces in the single-expert setting on the Food-10 subset of ImageNet.
C. Generative Latent Distillation

Figure C.23: Different distillation spaces in the single-expert setting on the Marine-10 subset of ImageNet.

Figure C.24: Different distillation spaces in the single-expert setting on the Herptiles-10 subset of ImageNet.
C. Generative Latent Distillation

<table>
<thead>
<tr>
<th>African Elephant</th>
<th>Red Panda</th>
<th>Camel</th>
<th>Zebra</th>
<th>Guinea Pig</th>
<th>Kangaroo</th>
<th>Platypus</th>
<th>Arctic Fox</th>
<th>Porcupine</th>
<th>Gorilla</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pixel Space (MTT [10]). Backbone accuracy: 19.1%. Cross-arch. validation accuracy: 12.1%

|                  |          |       |       |            |          |          |            |           |         |

F20 (Ours). Backbone accuracy: 29.1%. Cross-arch. validation accuracy: 18.3%

|                  |          |       |       |            |          |          |            |           |         |

F8 (Ours). Backbone accuracy: 36.7%. Cross-arch. validation accuracy: 22.9%

Figure C.25: Different distillation spaces in the single-expert setting on the Mammals-10 subset of ImageNet.

<table>
<thead>
<tr>
<th>Tabby Cat</th>
<th>Bengal Cat</th>
<th>Persian Cat</th>
<th>Siamese Cat</th>
<th>Egyptian Cat</th>
<th>Lion</th>
<th>Tiger</th>
<th>Jaguar</th>
<th>Snow Leopard</th>
<th>Lynx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pixel Space (MTT [10]). Backbone accuracy: 13.4%. Cross-arch. validation accuracy: 11.1%

|           |            |             |             |              |      |       |         |             |      |

F20 (Ours). Backbone accuracy: 19.2%. Cross-arch. validation accuracy: 13.8%

|           |            |             |             |              |      |       |         |             |      |

F8 (Ours). Backbone accuracy: 23.0%. Cross-arch. validation accuracy: 18.0%

Figure C.26: Different distillation spaces in the single-expert setting on the Cats-10 subset of ImageNet.
C. Generative Latent Distillation

Pixel Space ($\text{MTT}$ [10]). Backbone accuracy: 13.1%. Cross-arch. validation accuracy: 12.6%

F20 (Ours). Backbone accuracy: 20.2%. Cross-arch. validation accuracy: 11.9%

F8 (Ours). Backbone accuracy: 26.6%. Cross-arch. validation accuracy: 16.0%

Figure C.27: Different distillation spaces in the single-expert setting on the Earth-10 subset of ImageNet.

Pixel Space ($\text{MTT}$ [10]). Backbone accuracy: 19.5%. Cross-arch. validation accuracy: 12.6%

F20 (Ours). Backbone accuracy: 24.7%. Cross-arch. validation accuracy: 15.7%

F8 (Ours). Backbone accuracy: 26.4%. Cross-arch. validation accuracy: 18.5%

Figure C.28: Different distillation spaces in the single-expert setting on the Sports-10 subset of ImageNet.
C. Generative Latent Distillation

Figure C.29: Different distillation spaces in the single-expert setting on the ImageNette [19] subset of ImageNet.

Pixel Space (MTT [10]). Backbone accuracy: 15.8%. Cross-arch. validation accuracy: 15.3%

F20 (Ours). Backbone accuracy: 34.3%. Cross-arch. validation accuracy: 23.0%

F8 (Ours). Backbone accuracy: 38.8%. Cross-arch. validation accuracy: 22.9%
C. Generative Latent Distillation

<table>
<thead>
<tr>
<th>Ours (F16)</th>
<th>MTT [10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backbone validation accuracy: 33.1%</td>
<td>Backbone validation accuracy: 35.7%</td>
</tr>
<tr>
<td>Cross-arch. validation accuracy: 24.6%</td>
<td>Cross-arch. validation accuracy: 14.3%</td>
</tr>
</tbody>
</table>

Figure C.30: 512×512: Peacock, Flamingo, Pelican

98
C. Generative Latent Distillation

Backbone validation accuracy: 33.1%  
Cross-arch. validation accuracy: 24.6%

Backbone validation accuracy: 35.7%  
Cross-arch. validation accuracy: 14.3%

Figure C.31: 512×512: King Penguin, Bald Eagle, Toucan
C. Generative Latent Distillation

Figure C.32: 512×512: Ostrich, Black Swan, Cockatoo