Dataset Distillation by Matching Training Trajectories

George Cazenavette[1] Tongzhou Wang[2] Antonio Torralba[2] Alexei A. Efros[3] Jun-Yan Zhu[1]

[1]Carnegie Mellon University [2]Massachusetts Institute of Technology [3]UC Berkeley

Figure 1. Example distilled images from 32x32 CIFAR-100 (top), 64x64 Tiny ImageNet (middle), and 128x128 ImageNet subsets (bottom). Training a standard CNN using only such distilled images (as few as one per category) yields a trained model capable of test accuracy significantly better than previous methods of dataset distillation. Please see more results at https://georgecazenavette.github.io/mtt-distillation/.

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 paper, we propose a new formulation that optimizes our distilled data to guide networks to a similar state as those trained on real data across many training steps. 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.

1. Introduction

In the seminal 2015 paper, Hinton et al. [**?**] proposed *model distillation*, which aims to distill the knowledge of a complex model into a simpler one. *Dataset distillation*, proposed by Wang et al. [**?**], is a related but orthogonal task: rather than distilling the model, the idea is to distill the dataset. As shown in Figure **??**, 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 [**? ? ? ? ? ? ? **] 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 **??**.

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 **??**. Thus, the distilling algorithm must strike a delicate balance by heavily compressing information without completely obliterating the

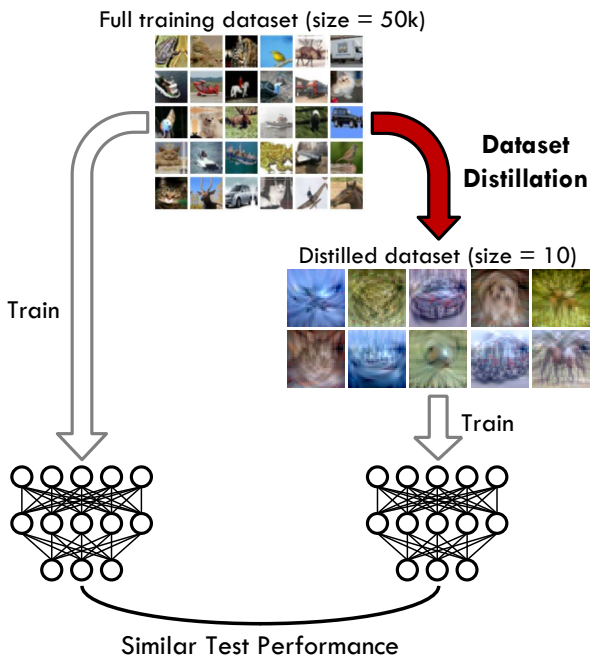

Figure 2. Dataset distillation aims to generate a small synthetic dataset for which a model trained on it can achieve a similar test performance as a model trained on the whole real train set.

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 [**? ? ?**] but often require huge compute and memory and suffer from inexact relaxation [**? ?**] or training instability of unrolling many iterations [**? ?**]. To reduce the optimization difficulty, other methods [**? ?**] 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 **??**.

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 (**??**). 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 backpropagate 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 [**? ?**] and 36.1% / 46.5% from [**?**]). Furthermore, our method also generalizes well to larger data, allowing us to see high 128×128-resolution images distilled from ImageNet [**?**] for the first time. Finally, we analyze our method through additional ablation studies and visualizations. Code and models are also available on our webpage.

2. Related Work

Dataset Distillation. Dataset distillation was first introduced by Wang et al. [**?**], who proposed expressing the model weights as a function of distilled images and optimized them using gradient-based hyperparameter optimization [**?**], which is also widely used in meta-learning research [**? ?**]. Subsequently, several works significantly improved the results by learning soft labels [**? ?**], amplifying learning signal via gradient matching [**?**], adopting augmentations [**?**], and optimizing with respect to the infinite-width kernel limit [**? ?**]. Dataset distillation has enabled various applications including continual learning [**? ? ?**], efficient neural architecture search [**? ?**], federated learning [**? ? ?**], and privacy-preserving ML [**? ?**] for images, text, and medical imaging data. As mentioned in the introduction, our method does not rely on single-step behavior matching [**? ?**], costly unrolling of full optimization trajectories [**? ?**], or large-scale Neural Tangent Kernel computation [**? ?**]. Instead, our method achieves long-range trajectory matching by transferring the knowledge from pre-trained experts.

Concurrent with our work, the method of Zhao and Bilen [**?**] completely disregards optimization steps, instead focusing on distribution matching between synthetic and real data. While this method is applicable to higher-resolution datasets (e.g., Tiny ImageNet) due to reduced memory requirements, it attains inferior performance in most cases (e.g., when compared to previous works [**? ?**]). In contrast, our method simultaneously reduces memory costs while outper-

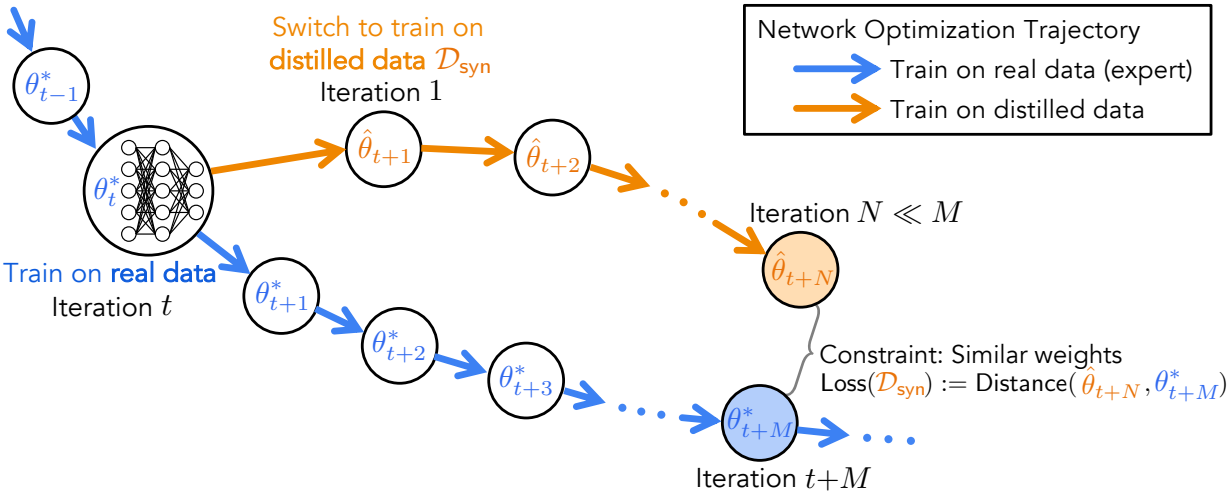

Figure 3. We perform long-range parameter matching between training on distilled synthetic data and training on real data. Starting from the same initial parameters, we train distilled data $\mathcal{D}_{\text{syn}}$ such that N training steps on them match the same result (in parameter space) from much more M steps on real data.

forming existing works [**?** **?**] and the concurrent method [**?**] on both standard benchmarks and higher-resolution datasets.

A related line of research learns a generative model to synthesize training data [**?** **?**]. However, such methods do not generate a small-size dataset, and are thus not directly comparable with dataset distillation methods.

Imitation Learning. Imitation learning attempts to learn a good policy by observing a collection of expert demonstrations [**?** **?** **?**]. Behavior cloning trains the learning policy to act the same as expert demonstrations. Some more sophisticated formulations involve on-policy learning with labeling from the expert [**?**], while other approaches avoid any label at all, e.g., via distribution matching [**?**]. Such methods (behavior cloning in particular) have been shown to work well in offline settings [**?** **?**]. Our method can be viewed as imitating a collection of expert network training trajectories, which are obtained via training on real datasets. Therefore, it can be considered as doing imitation learning over optimization trajectories.

Coreset and Instance Selection. Similar to dataset distillation, coreset [**?** **?** **?** **?** **?**] and instance selection [**?**] aim to select a subset of the entire training dataset, where training on this small subset achieves good performance. Most of such methods do not apply to modern deep learning, but new formulations based on bi-level optimization have shown promising results on applications like continual learning [**?**]. Related to coreset, other lines of research aim to understand which training samples are "valuable" for modern machine learning, including measuring single-example accuracy [**?**] and counting misclassification rates [**?**]. In fact, dataset distillation is a generalization of such ideas, as the distilled data do not need to be realistic or come from the training set.

3. Method

Dataset Distillation refers to the curation of a small, synthetic training set $\mathcal{D}_{\text{syn}}$ such that a model trained on this synthetic data will have similar performance on the real test set as a model trained on the large, real training set $\mathcal{D}_{\text{real}}$. In this section, we describe our method that directly mimics the long-range behavior of real-data training, matching multiple training steps on distilled data to *many* more steps on the real data.

In Section **??**, we discuss how we obtain expert trajectories of networks trained on real datasets. In Section **??**, we describe a new dataset distillation method that explicitly encourages the distilled dataset to induce similar long-range network parameter trajectories as the real dataset, resulting in a synthetically-trained network that performs similarly to a network trained on real data. Finally, Section **??** describes our techniques to reduce memory consumption.

3.1. Expert Trajectories

The core of our method involves using *expert trajectories* τ^* 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

Algorithm 1 Dataset Distillation via Trajectory Matching

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

Initialize distilled data $\mathcal{D}_{\text{syn}} \sim \mathcal{D}_{\text{real}}$
Initialize trainable learning rate $\alpha := \alpha_0$ for apply $\mathcal{D}_{\text{syn}}$
for each distillation step... **do**
  ▷ Sample expert trajectory: $\tau^* \sim \{\tau_i^*\}$ with $\tau^* = \{\theta_t^*\}_0^T$
  ▷ Choose random start epoch, $t \leq T^+$
  ▷ Initialize student network with expert params:
    $\hat{\theta}_t := \theta_t^*$
  for $n = 0 \rightarrow N-1$ **do**
    ▷ Sample a mini-batch of distilled images:
      $b_{t+n} \sim \mathcal{D}_{\text{syn}}$
    ▷ Update student network w.r.t. classification loss:
      $\hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha \nabla \ell(\mathcal{A}(b_{t+n}); \hat{\theta}_{t+n})$
  end for
  ▷ Compute loss between ending student and expert params:
    $\mathcal{L} = \|\hat{\theta}_{t+N} - \theta_{t+M}^*\|_2^2 \;/\; \|\theta_t^* - \theta_{t+M}^*\|_2^2$
  ▷ Update $\mathcal{D}_{\text{syn}}$ and α with respect to $\mathcal{L}$
end for

Output: distilled data $\mathcal{D}_{\text{syn}}$ and learning rate α

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.

3.2. 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 θ_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 the student parameters with respect to the classification loss of the *synthetic* data:

$$\hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha \nabla \ell(\mathcal{A}(\mathcal{D}_{\text{syn}}); \hat{\theta}_{t+n}), \tag{1}$$

where $\mathcal{A}$ is the differentiable augmentation technique [**? ? ? ?**] used in previous work [**?**], and α 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 θ_{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 parameters θ_{t+M}^*:

$$\mathcal{L} = \frac{\|\hat{\theta}_{t+N} - \theta_{t+M}^*\|_2^2}{\|\theta_t^* - \theta_{t+M}^*\|_2^2}, \tag{2}$$

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 α, by back-propagating through all N updates to the student network. The optimization of trainable learning rate α serves as automatic adjusting for the number of student and expert updates (hyperparameters M and N). We use SGD with momentum to optimize $\mathcal{D}_{\text{syn}}$ and α with respect to the above objective. Algorithm **??** illustrates our main algorithm.

3.3. 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 [**? ? ?**], 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

	Img/Cls	Ratio %	Coreset Selection			Training Set Synthesis							Full Dataset
			Random	Herding	Forgetting	DD† [?]	LD† [?]	DC [?]	DSA [?]	DM [?]	CAFE [?]	CAFE+DSA [?]	Ours	
CIFAR-10	1	0.02	14.4 ± 2.0	21.5 ± 1.2	13.5 ± 1.2	-	25.7 ± 0.7	28.3 ± 0.5	28.8 ± 0.7	26.0 ± 0.8	30.3 ± 1.1	31.6 ± 0.8	**46.3 ± 0.8***	84.8 ± 0.1
	10	0.2	26.0 ± 1.2	31.6 ± 0.7	23.3 ± 1.0	36.8 ± 1.2	38.3 ± 0.4	44.9 ± 0.5	52.1 ± 0.5	48.9 ± 0.6	46.3 ± 0.6	50.9 ± 0.5	**65.3 ± 0.7***	
	50	1	43.4 ± 1.0	40.4 ± 0.6	23.3 ± 1.1	-	42.5 ± 0.4	53.9 ± 0.5	60.6 ± 0.5	63.0 ± 0.4	55.5 ± 0.6	62.3 ± 0.4	**71.6 ± 0.2**	
CIFAR-100	1	0.2	4.2 ± 0.3	8.4 ± 0.3	4.5 ± 0.2	-	11.5 ± 0.4	12.8 ± 0.3	13.9 ± 0.3	11.4 ± 0.3	12.9 ± 0.3	14.0 ± 0.3	**24.3 ± 0.3***	56.2 ± 0.3
	10	2	14.6 ± 0.5	17.3 ± 0.3	15.1 ± 0.3	-	-	25.2 ± 0.3	32.3 ± 0.3	29.7 ± 0.3	27.8 ± 0.3	31.5 ± 0.2	**40.1 ± 0.4**	
	50	10	30.0 ± 0.4	33.7 ± 0.5	30.5 ± 0.3	-	-	-	42.8 ± 0.4	43.6 ± 0.4	37.9 ± 0.3	42.9 ± 0.2	**47.7 ± 0.2***	
Tiny ImageNet	1	0.2	1.4 ± 0.1	2.8 ± 0.2	1.6 ± 0.1	-	-	-	-	3.9 ± 0.2	-	-	**8.8 ± 0.3**	37.6 ± 0.4
	10	2	5.0 ± 0.2	6.3 ± 0.2	5.1 ± 0.2	-	-	-	-	12.9 ± 0.4	-	-	**23.2 ± 0.2**	
	50	10	15.0 ± 0.4	16.7 ± 0.3	15.0 ± 0.3	-	-	-	-	24.1 ± 0.3	-	-	**28.0 ± 0.3**	

Table 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 [**?**] for MNIST and AlexNet [**?**] for CIFAR-10. All others use a 128-width ConvNet. CIFAR values marked by $(*)$ signify best results were obtained with ZCA whitening.

outer loop in Line **??** of Algorithm **??**). 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 **??** of Algorithm **??**) such that all distilled images will have been seen by the time the final weight matching loss (Eqn. **??**) 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

$$\begin{aligned} b_{t+n} &\sim \mathcal{D}_{\mathsf{syn}} \\ \hat{\theta}_{t+n+1} &= \hat{\theta}_{t+n} - \alpha \nabla \ell(\mathcal{A}(b_{t+n}); \hat{\theta}_{t+n}). \end{aligned} \tag{3}$$

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.

4. Experiments

We evaluate our method on various datasets, including

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

We provide additional visualizations and ablation studies in the supplementary material.

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 **??**, using the same suite of differentiable augmentations as done in previous work [**? ?**]. 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 [**?**] (DD), Flexible Dataset Distillation [**?**] (LD), Dataset Condensation [**?**] (DC), and Differentiable Siamese Augmentation [**?**] (DSA), along with a method based on the infnite-width kernel limit [**? ?**] (KIP) and concurrent works Distribution Matching [**?**] (DM) and Aligning Features [**?**] (CAFE). We also compare our methods with instance selection algorithms including random selection (random), herding methods [**?**] (herding), and example forgetting [**?**] (forgetting).

Network Architectures. Staying with precedent [**? ? ? ?**], we mainly employ a simple ConvNet architecture designed by Gidaris and Komodakis [**?**] for our distillation tasks. The architecture consists of several convolutional blocks, each containing a 3×3 convolution layer with 128 filters, Instance normalization [**?**], RELU, and 2×2 average pooling with stride 2. After the convoluation 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.

4.1. Low-Resolution Data (32×32)

For low-resolution tasks, we begin with the 32×32 CIFAR-10 and CIFAR-100 datasets [**?**]. For these datasets, we employ ZCA whitening as done in previous work [**? ?**], using the Kornia [**?**] implementation with default parameters. Staying with precedent, we use a depth-3 ConvNet taken directly from the open-source code [**? ?**].

As seen in Table **??**, 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 [**?**]) to almost twice test accuracy, on both datasets. For CIFAR-10, these distilled images can be seen in Figure **??**. CIFAR-100 images are visualized in the supplementary material

In Table **??**, we also compare with a recent method KIP [**? ?**], 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

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

Table 2. Kernel Inducing Point (KIP) [**?**] 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.

		Evaluation Model			
		ConvNet	ResNet	VGG	AlexNet
Method	Ours	**64.3 ± 0.7**	**46.4 ± 0.6**	**50.3 ± 0.8**	**34.2 ± 2.6**
	DSA	52.1 ± 0.4	42.8 ± 1.0	43.2 ± 0.5	**35.9 ± 1.3**
	KIP	47.6 ± 0.9	36.8 ± 1.0	42.1 ± 0.4	24.4 ± 3.9

Table 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.

paper [**?**]). 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 [**?**], 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.

If we look at the one image per class visualizations in Figure **??** (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 **??** (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 **??**, we show our baseline ConvNet performance and evaluate on ResNet [**?**], VGG [**?**], and AlexNet [**?**].

For KIP, instead of the Kornia [**?**] 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

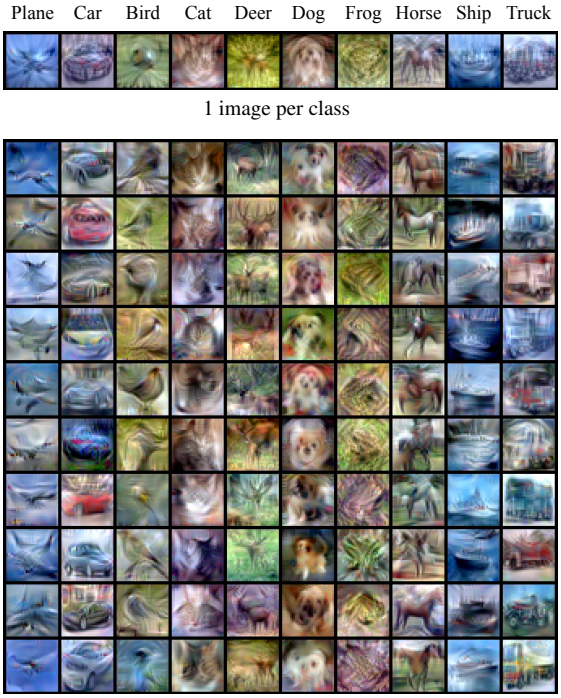

Figure 4. CIFAR-10: The 1 image per class images are more abstract but also more information-dense while the 10 images per class images are more expressive and contain more structure.

we lie within one standard deviation of DSA. This could be attributed to our higher baseline performance, but it still shows that our method is robust to changes in architecture.

4.2. Short-Range vs. Long-Range Matching

Unlike some prior works (DC and 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., DD and KIP) can be viewed as even longer range matching. However, their performances fall short of our method (e.g., in Table **??**), 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 DSA).

In Figure **??** (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 **??** (right), we evaluate our method and a short-range matching work (DSA) on their abilities to approximate

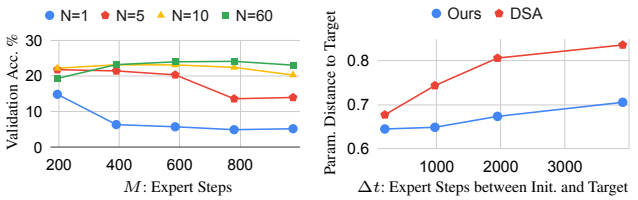

Figure 5. CIFAR-100 (1 image / class). **Left:** Smaller M and N match shorter-range behavior, which performs worse than longer-range matching. **Right:** Over 1000 training steps on distilled data, we track the closest distance in parameter space (normalized MSE in Eqn. **??**) to a target set of parameters, obtained with Δ_t training steps on real data. Matching long-range behavior, our method better approximate real data training for longer ranges (large Δ_t).

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

Table 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.

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 Δ_t training steps on real data (i.e., the long-range behavior that distilled data should mimic). A small (or large) Δ_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. DSA is only optimized to match short-range behavior, and thus errors may accumulate during longer training. Indeed, as Δ_t grows larger, 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.

4.3. Tiny ImageNet (64×64)

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

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 [**?**]. In Table **??**, 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 **??**. 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 6. 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![1])

4.4. 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 [**?**]. 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 [**?**] 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 ImageNet subset settings. As the code of DM [**?**] 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 **??**.

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 **??**), 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 **??**), 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 **??**). 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

[1] Answers for Figure **??**: **First Row:** African Elephant, Jellyfish, Kimono, Lampshade, Monarch **Second Row:** Organ, Pizza, Pretzel, Teapot, Teddy

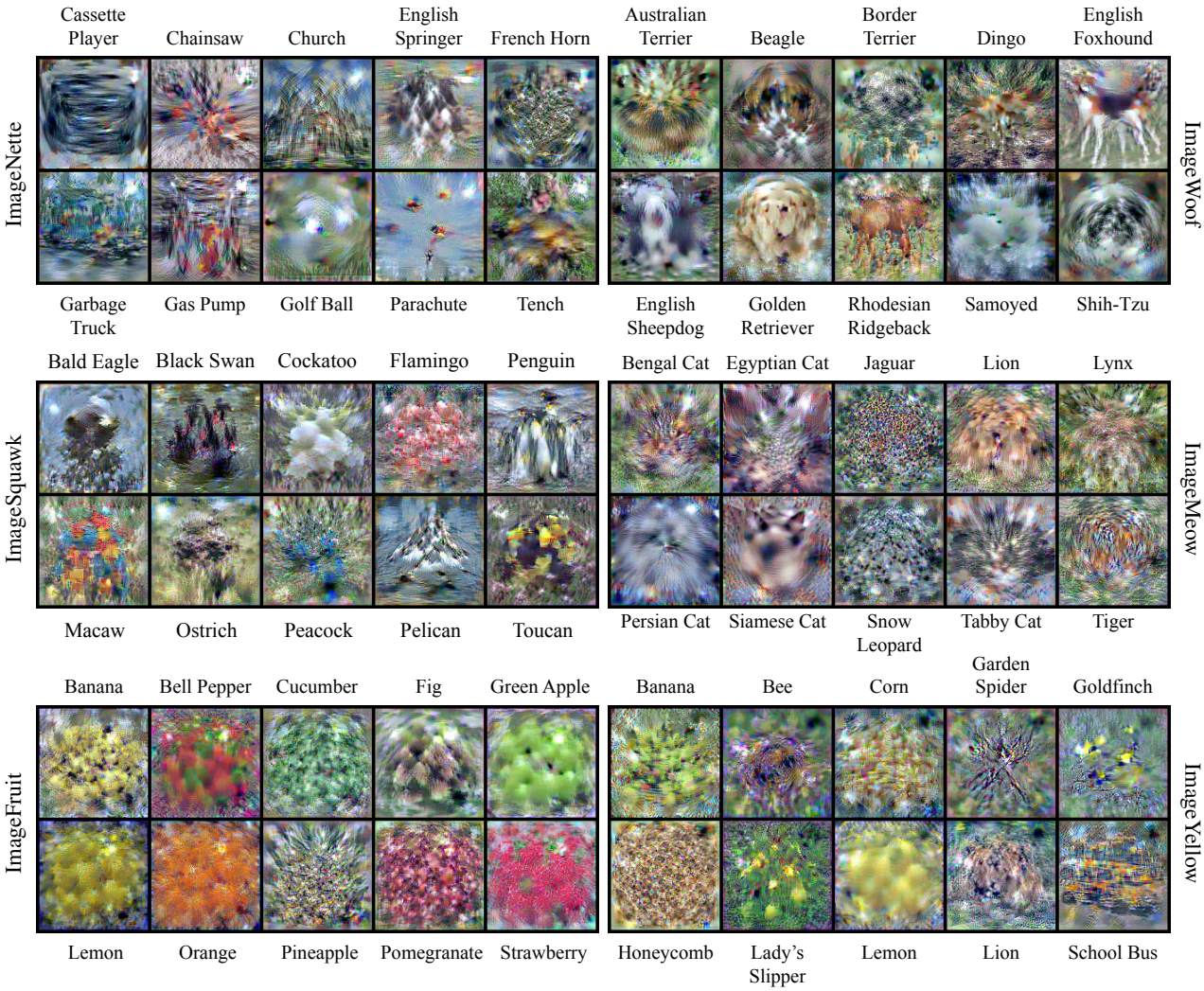

Figure 7. Our method is the first capable of distilling higher-resolution (128×128) images, allowing us to explore the ImageNet [**?**] dataset.

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.

5. 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 128×128 ImageNet images, which not only allows us to gain interesting insights of the dataset (e.g., in Section **??**) 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. The computational overhead of training and storing expert trajectories is quite high. For example, CIFAR experts took $\sim$3 seconds per epoch (8 GPU hours total for all 200 CIFAR experts) while each ImageNet (subset) expert took $\sim$11 seconds per epoch (15 GPU hours total for all 100 ImageNet experts). Storage-wise, each CIFAR expert took up $\sim$60MB of storage while each ImageNet expert took up $\sim$120MB.

Acknowledgements. We would like to thank Alexander Li, Assaf Shocher, Gokul Swamy, Kangle Deng, Ruihan Gao, Nupur Kumari, Muyang Li, Garuav Parmar, Chonghyuk Song, Sheng-Yu Wang, and Bingliang Zhang as well as Simon Lucey's Vision Group at the University of Adelaide for their valuable feedback. This work is supported, in part, by the NSF Graduate Research Fellowship under Grant No. DGE1745016 and grants from J.P. Morgan Chase, IBM, and SAP.

A. Appendix

A.1. Additional Visualizations

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

A.2. Additional Quantitative Results

Analysis of learned learning rates α. In Figure **??**, we explore the effect of our learnable synthetic step size α. The left plot confirms that we learn different values of α for different combinations of M and N. The logic here is that different numbers of synthetic steps N require a different step size α 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 **??** (Left). We find that ZCA normalization is *not* 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.

Dataset	Img/Cls	Yes-ZCA	No-ZCA
CIFAR-10	1	46.3	45.2
	10	65.3	62.8
	50	71.5	71.6
CIFAR-100	1	24.3	22.7
	10	39.4	40.1
	50	47.7	47.2

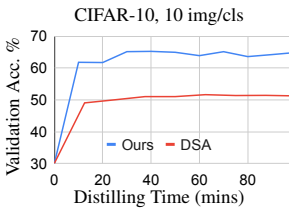

Figure 8. **Left**: ZCA Ablation. **Right**: Distillation Time.

A.2.1 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 **??**. 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 **??**-**??**. We also include a visualization of a set distilled with only one expert trajectory in Figure **??**.

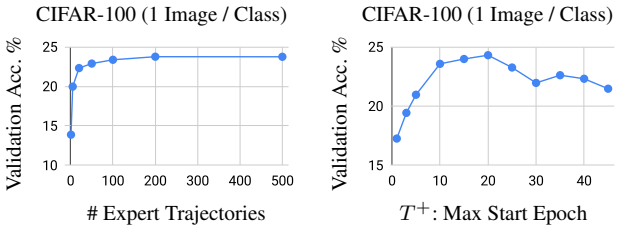

Figure 9. **Left:** We see logarithmic performance improvement with respect to the number of expert trajectories used, quickly saturating near 200. **Right:** The upper bound on the expert epoch at which the synthetic data starts working cannot be too high or low to ensure quality learning signal.

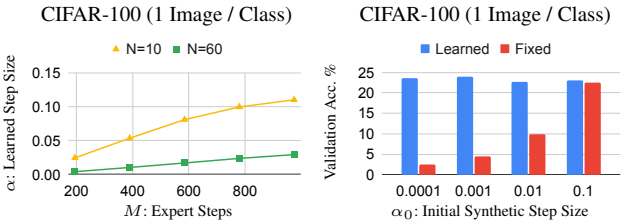

Figure 10. **Left:** Our learned synthetic step size α seems to scale inversely with the number of synthetic steps N to cover the same distance in parameter space as the expert steps M. **Right:** Having a learnable step size α saves us from having to search for an appropriate fixed α_0.

Setting	Ours	Gauss. Init.	w/o ZCA	w/o Diff. Aug.
Acc.	24.3 ± 0.3	22.9 ± 0.4	19.1 ± 0.6	20.7 ± 0.7

Table 5. 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.

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 **??** (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

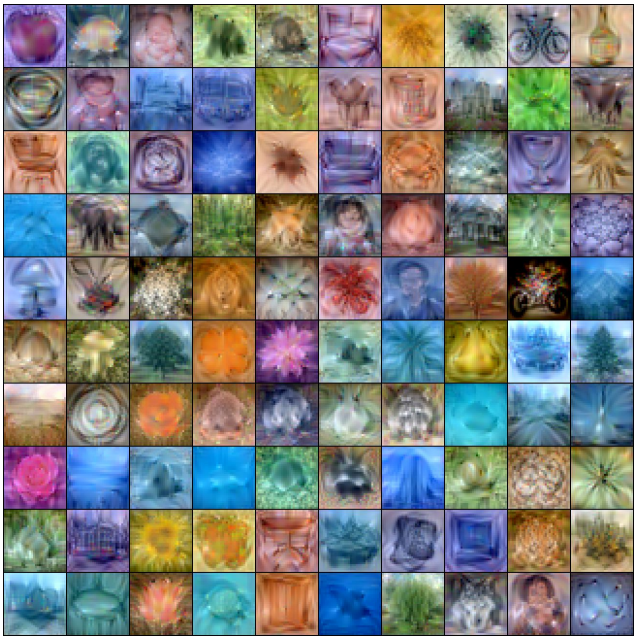

Figure 11. CIFAR-100, 1 Image Per Class

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 **??**, 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 **??**, 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.

Dataset	Model	Img/Cls	Synthetic Steps (N)	Expert Epochs ($M^\dagger$)	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 6. 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* ($M^\dagger$) for simplicity across datasets.

Our distillation time, in general, is comparable to `DC/DSA`, as they also utilize a bi-level 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 **??** (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 **??**. Note that most improvement occurs within the first 1k iterations, but we continue training for 10k.

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

Table 7. Distillation time for each dataset and support size.

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` $\in \{112, 335, 1000\}$, `weight_decay` $\in \{0, 0.0001, 0.001, 0.01\}$, `aug` $\in \{$`True`, `False`$\}$, `zca` $\in \{$`True`, `False`$\}$, `label_learn` $\in \{$`True`, `False`$\}$, and `norm` $\in \{$`none`, `instance`$\}$. The architecture originally used for KIP to NN in the `KIP` paper contained no normaliza-

	Img/Cls	Learn Labels	Aug.	ZCA	Norm	Weight Decay	Ckpt.
CIFAR-10	1	N	Y	Y	N	0.001	1000
	10	N	N	Y	N	0.001	112
	50	N	N	Y	I	0.01	112
CIFAR-100	1	N	N	Y	I	0.001	1000
	10	N	N	N	I	0.001	1000

Table 8. 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.

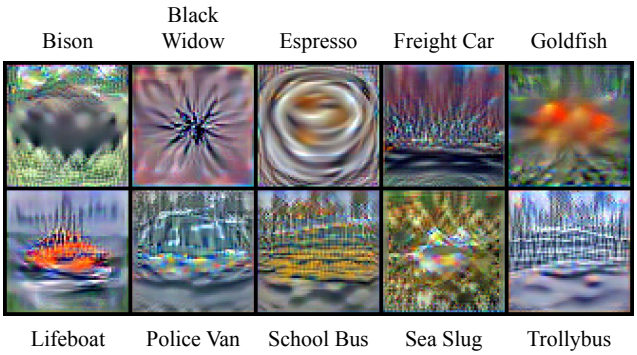

Figure 12. Most-correct distilled images for Tiny ImageNet ($\geq 30\%$)

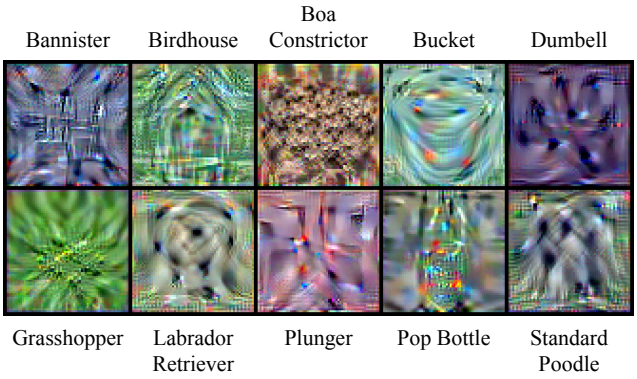

Figure 13. Least-correct distilled images for Tiny ImageNet ($\leq 4\%$)

tion 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 **??**, 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.

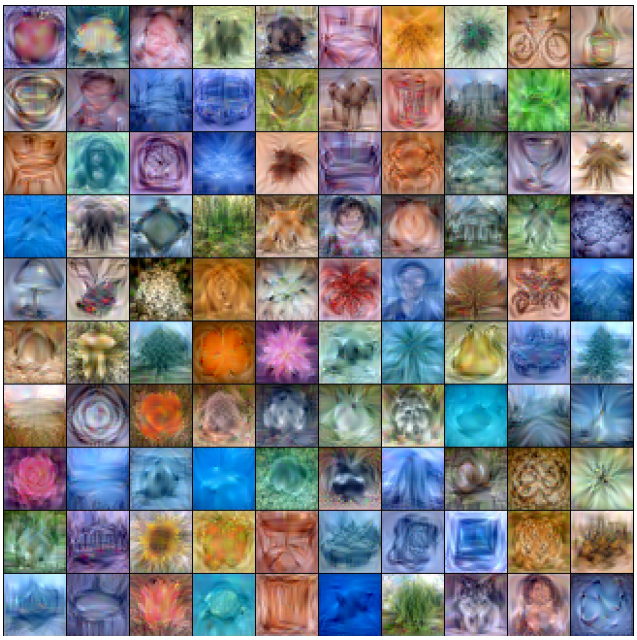

Figure 14. CIFAR-100, Initialized as Random Noise

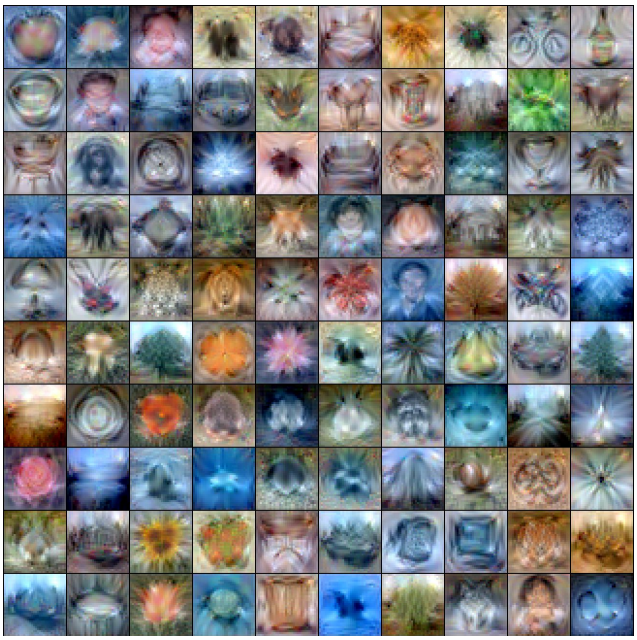

Figure 16. CIFAR-100, No Differentiable Augmentation

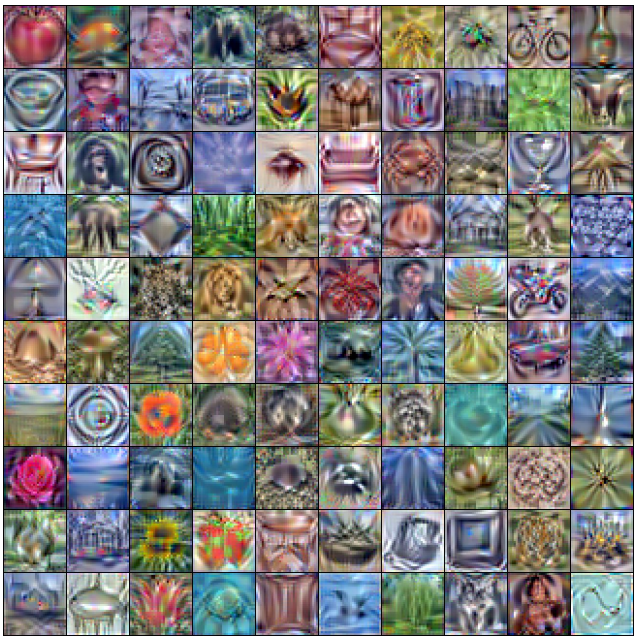

Figure 15. CIFAR-100, No ZCA Whitening

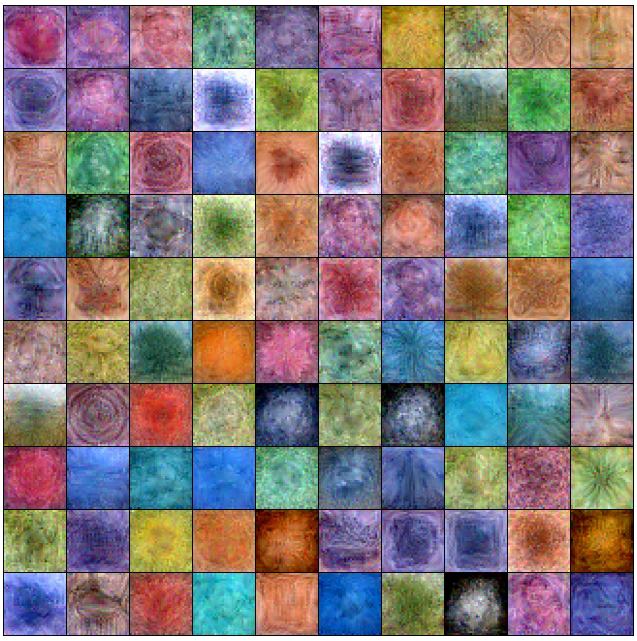

Figure 17. CIFAR-100, Only 1 Expert Trajectory

Figure 18. Tiny ImageNet, 1 Image Per Class (Classes 1-100)

Figure 19. Tiny ImageNet, 1 Image Per Class (Classes 101-200)

Figure 20. ImageNette, 10 Images Per Class

Figure 21. ImageWoof, 10 Images Per Class

Figure 22. ImageSquawk, 10 Images Per Class

Figure 23. ImageMeow, 10 Images Per Class

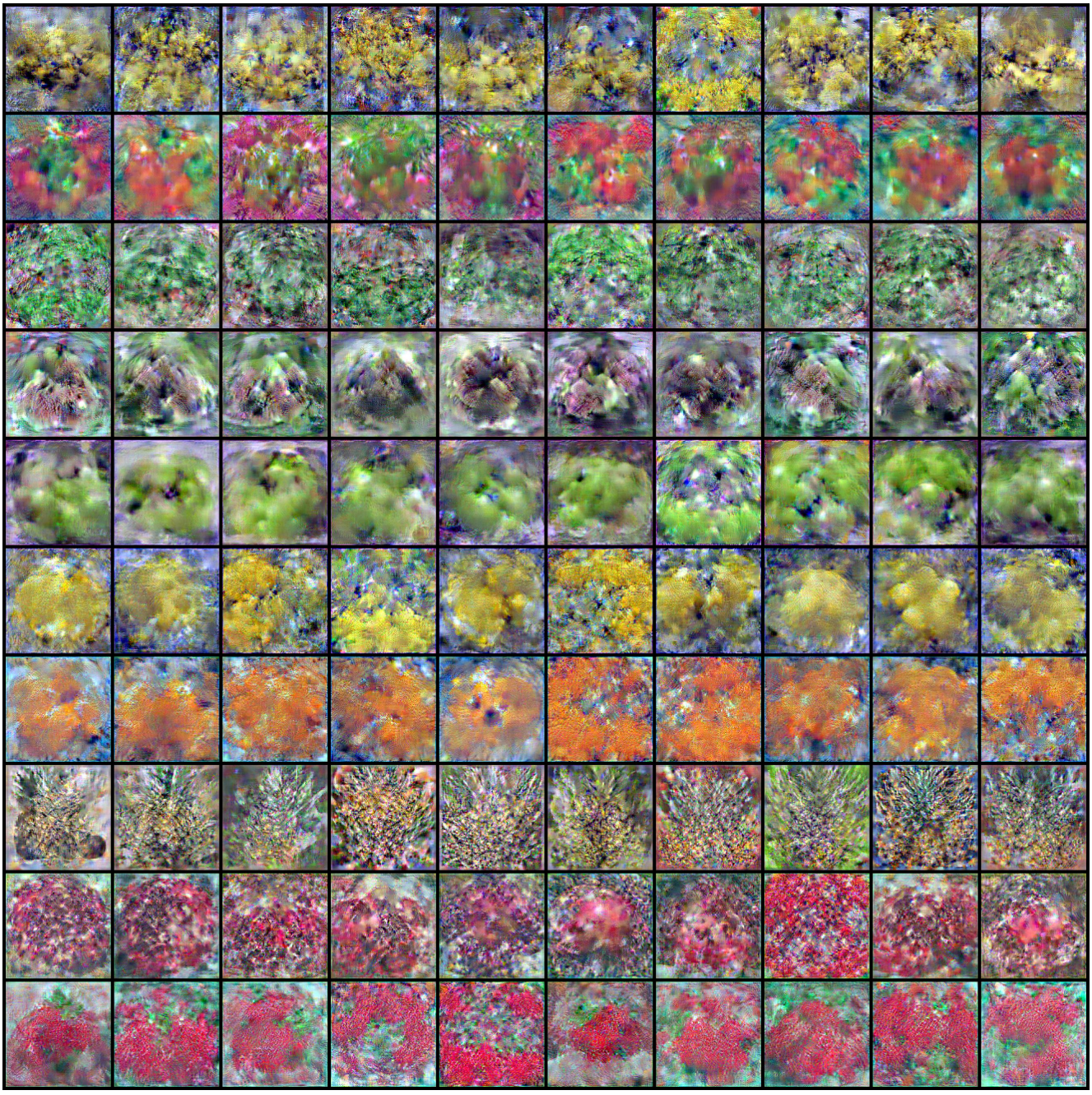

Figure 24. ImageFruit, 10 Images Per Class

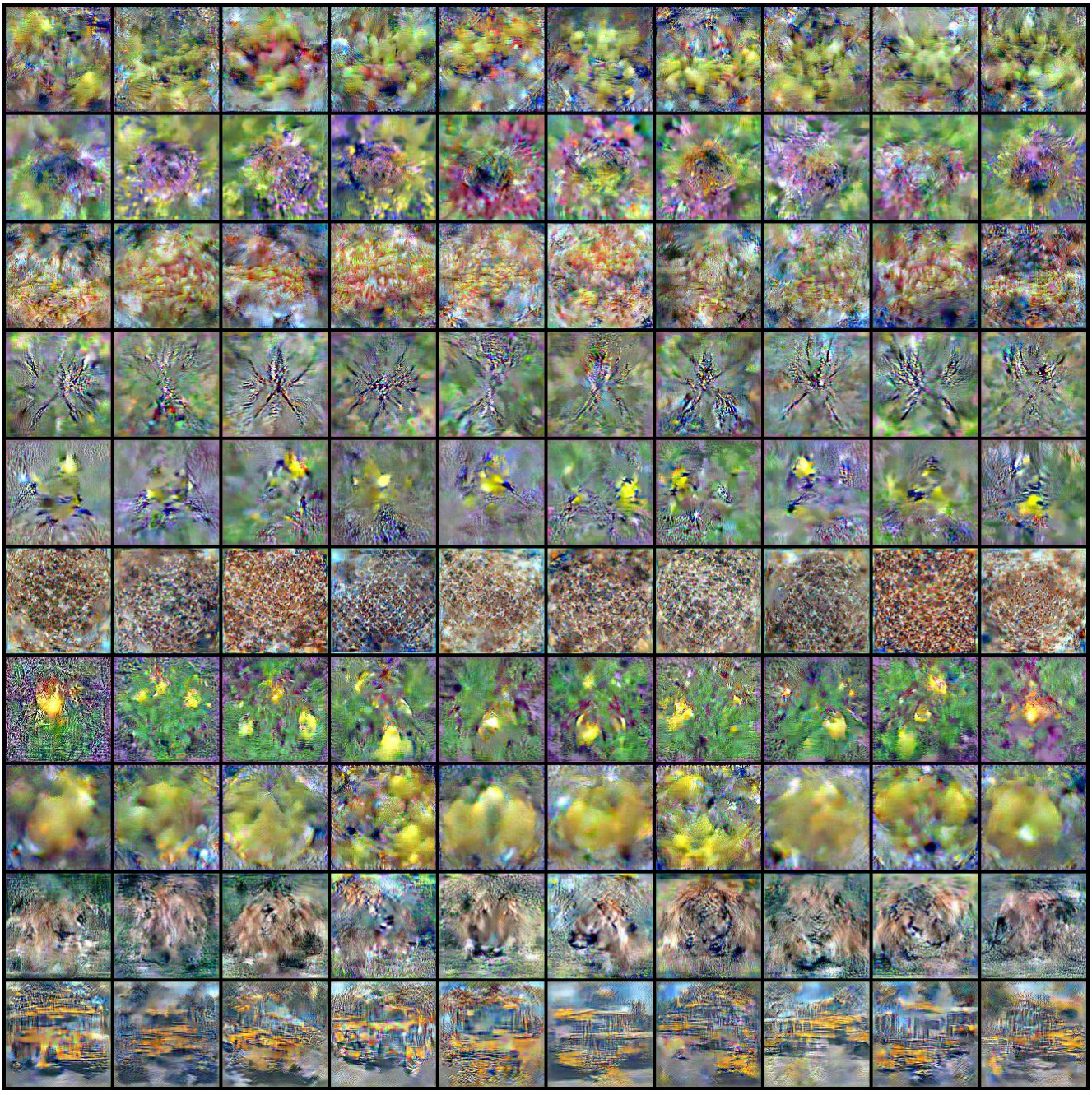

Figure 25. ImageYellow, 10 Images Per Class