Dataset Distillation using Neural Feature Regression

Yongchao Zhou
Department of Computer Science
University of Toronto
yongchao.zhou@mail.utoronto.ca

Ehsan Nezhadarya
Toronto AI Lab
LG Electronics Canada
ehsan.nezhadarya@lge.com

Jimmy Ba
Department of Computer Science
University of Toronto
jba@cs.toronto.edu

Abstract

Dataset distillation aims to learn a small synthetic dataset that preserves most of the information from the original dataset. Dataset distillation can be formulated as a bi-level meta-learning problem where the outer loop optimizes the meta-dataset and the inner loop trains a model on the distilled data. Meta-gradient computation is one of the key challenges in this formulation, as differentiating through the inner loop learning procedure introduces significant computation and memory costs. In this paper, we address these challenges using neural Feature Regression with Pooling (FRePo), achieving the state-of-the-art performance with an order of magnitude less memory requirement and two orders of magnitude faster training than previous methods. The proposed algorithm is analogous to truncated backpropagation through time with a pool of models to alleviate various types of overfitting in dataset distillation. FRePo significantly outperforms the previous methods on CIFAR100, Tiny ImageNet, and ImageNet-1K. Furthermore, we show that high-quality distilled data can greatly improve various downstream applications, such as continual learning and membership inference defense. Please check out our webpage at https://sites.google.com/view/frepo.

1 Introduction

Knowledge distillation [1] is a technique in deep learning to compress knowledge for easy deployment. Most previous works focus on model distillation [2, 3] where the knowledge acquired by a large teacher model is transferred to a small student model. In contrast, dataset distillation [4, 5] aims to learn a small set of synthetic examples preserving most of the information from a large dataset such that a model trained on it can achieve similar test performance as one trained on the original dataset. Distilled data can accelerate model training and reduce the cost of storing and sharing a dataset. Moreover, its highly condensed and synthetic nature can also benefit various applications, such as continual learning [5–8], neural architecture search [5, 7], and privacy-preserving tasks [9, 10].

Dataset distillation was first studied by Maclaurin et al. [11] in the context of gradient-based hyperparameter optimization and subsequently Wang et al. [4] formally proposed dataset distillation as a new task. Dataset distillation can be naturally formulated as a bi-level meta-learning problem. The inner loop optimizes the model parameters on the distilled data (meta-parameters), while the outer loop refines the distilled data with meta-gradient updates.

One key challenge in dataset distillation is computing the meta-gradient. Several methods [4, 11–13] compute it by back-propagating through the unrolled computation graph, but they often suffer from

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

(a) CIFAR100 (b) Tiny ImageNet (c) ImageNette and ImageWoof

Figure 1: Example distilled images from 32x32 CIFAR100, 64x64 Tiny ImageNet, and 128x128 ImageNet Subset. The images look real and transfer well to different architectures. They can be used for various downstream applications, such as continual learning and membership inference defense.

huge compute and memory requirement [14], training instability [15, 16], and truncation bias [17]. To avoid unrolled optimization, surrogate objectives are used to derive the meta-gradient, such as gradient matching [5, 7, 18], feature alignment [8, 19], and training trajectory matching [20]. Nevertheless, a surrogate objective may introduce its own bias [19], and thus, may not accurately reflect the true objective. An alternative is using kernel methods, such as Neural Tangent Kernel (NTK) [21], to approximate the inner optimization [22, 23]. However, computing analytical NTK for modern neural network can be extremely expensive [22, 23].

Even with an accurate meta-gradient, dataset distillation still suffers from various types of overfitting. For instance, the distilled data can easily overfit to a particular learning algorithm [4, 13, 20], a certain stage of optimization [13, 19], or a certain network architecture [5, 7, 20, 22, 23]. Meanwhile, the model can also overfit the distilled data during training, which is the most common cause of overfitting when we train on a small dataset. All these kinds of overfitting impose difficulties on the training and general-purpose use of the distilled data.

We propose an efficient meta-gradient computation method and a "model pool" to address the overfitting problems. The bottleneck in meta-gradient computation arises due to the complexity of inner optimization, as we need to know how the inner parameters vary with the outer parameters [24]. However, the inner optimization can be pretty simple if we only train the last layer of a neural network to convergence while keeping the feature extractor fixed. In this case, computing the prediction on the real data using the model trained on the distilled data can be expressed as a kernel ridge regression (KRR) with respect to the conjugate kernel [25]. Hence, computing the meta-gradient is simply back-propagating through the kernel and a fixed feature extractor. To alleviate overfitting, we propose to maintain a diverse pool of models instead of periodically training and resetting a single model as in prior work [7, 13, 18]. Intuitively, our algorithm targets the following question: what is the best data to train the linear classifier given the current feature extractor? Due to the diverse feature extractors we use, the distilled data generalize well to a wide range of model distributions.

Summary of Contributions:

- We propose an effective method for dataset distillation. Our method, named neural Feature Regression with Pooling (FRePo), achieves state-of-the-art results on various benchmark datasets with a 100x reduction in training time and a 10x reduction in GPU memory requirement. Our distilled data looks real (Figure 1) and transfers well to different architectures.
- We show that FRePo scales well to datasets with high-resolution images or complex label space. We achieve 7.5% top1 accuracy on ImageNet-1K [26] using only one image per class. The same classifier obtains only 1.1% accuracy from a random subset of real images. The previous methods struggle in this task due to large memory and compute requirements.
- We demonstrate that high-quality distilled data can significantly improve various downstream applications, such as continual learning and membership inference defense.

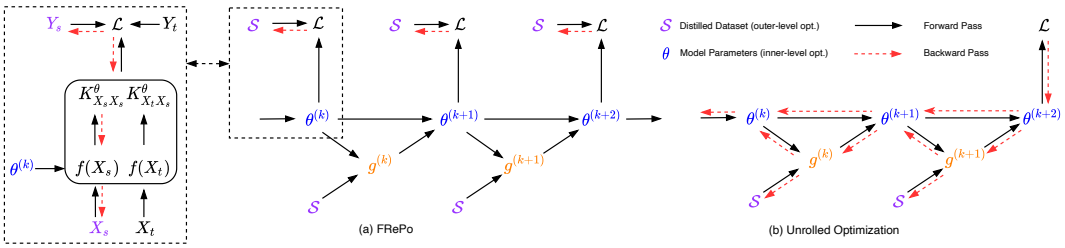


Figure 2: Comparison of FRePo and Unrolled Optimization. S, X_s, Y_s are the distilled dataset, images and labels. $\mathcal{L}$ is the meta-training loss and $\theta^{(k)}$, $g^{(k)}$ are the model parameter and gradient at step k. $f(X)$ is the feature for input X and $K^{\theta}_{X_t X_s}$ is the Gram matrix of X_t and X_s. FRePo is analogous to 1-step TBPTT as it computes the meta-gradient at each step while performing the online model update. However, instead of backpropagating through the inner optimization, FRePo computes the meta-gradient through a kernel and feature extractor.

2 Method

2.1 Dataset Distillation as Bi-level Optimization

Suppose we have a large labeled dataset $\mathcal{T} = \left\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_{|\mathcal{T}|}, \mathbf{y}_{|\mathcal{T}|})\right\}$ with $|\mathcal{T}|$ image and label pairs. Dataset distillation aims to learn a small synthetic dataset $\mathcal{S} = \left\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_{|\mathcal{S}|}, \mathbf{y}_{|\mathcal{S}|})\right\}$ that preserves most of the information in $\mathcal{T}$. We train several neural networks parameterized by θ on the dataset $\mathcal{S}$ and then compute the validation loss $\mathcal{L}(\mathcal{A}lg\,(\theta, \mathcal{S}), \mathcal{T})$ on the real dataset $\mathcal{T}$, where $\mathcal{A}lg\,(\theta, \mathcal{S})$ is the neural network parameters optimized by a learning algorithm $\mathcal{A}lg$ with the model initialization θ and distilled dataset $\mathcal{S}$ as its inputs. The validation loss $\mathcal{L}(\mathcal{A}lg\,(\theta, \mathcal{S}), \mathcal{T})$ is a noisy objective with the stochasticity coming from random model initialization and inner learning algorithm. Thus, we are interested in minimizing the expected value of this loss, which we denote it as $F(\mathcal{S})$. We formulate the dataset distillation as the following bi-level optimization problem.

$$\overbrace{\mathcal{S}^* := \operatorname*{argmin}_{\mathcal{S}} F(\mathcal{S})}^{\textit{outer-level}}, \quad \text{where } F(\mathcal{S}) = \mathbb{E}_{\theta \sim P_\theta}\left[\mathcal{L}\Big(\overbrace{\mathcal{A}lg\,(\theta, \mathcal{S})}^{\textit{inner-level}}, \mathcal{T}\Big)\right]. \tag{1}$$

In this bi-level setup, the outer loop optimizes the distilled data to minimize $F(\mathcal{S})$, while the inner loop trains a neural network using the learning algorithm, $\mathcal{A}lg$, to minimize the training loss on the distilled data $\mathcal{S}$. From the meta-learning perspective, the task is defined by the model initialization θ, and we want to learn a meta-parameter $\mathcal{S}$ that generalizes well to different models sampled from the model distributions P_θ. During learning, we optimize the meta-parameter $\mathcal{S}$ by minimizing the meta-training loss $F(\mathcal{S})$. In contrast, at meta-test time, we train a new model from scratch on $\mathcal{S}$ and evaluate the trained model on a held-out real dataset. This meta-test performance reflects the quality of the distilled data.

2.2 Dataset Distillation using Neural Feature Regression with Pooling (FRePo)

The outer-level problem can be solved using gradient-based methods of the form $\mathcal{S} \leftarrow \mathcal{S} - \alpha\, \nabla_{\mathcal{S}} F(\mathcal{S})$, where α is the learning rate for the distilled data and $\nabla_{\mathcal{S}} F(\mathcal{S})$ is the meta-gradient [27]. For a particular model θ, the meta-gradient can be expressed as $\nabla_{\mathcal{S}} \mathcal{L}\,(\mathcal{A}lg\,(\theta, \mathcal{S}), \mathcal{T})$. Computing this meta-gradient requires differentiating through inner optimization. If $\mathcal{A}lg$ is an iterative algorithm like gradient descent, then backpropagating through the unrolled computation graph [14] can be a solution. However, this type of unrolled optimization introduces significant computation and memory overhead, as the whole training trajectory needs to be stored in memory (Figure 2(b)).

Traditionally, these issues are alleviated with truncated backpropagation through time (TBPTT) [28–30]. Instead of backpropagating through an entire unrolled sequence, TBPTT performs backpropagation for each subsequence separately. It is efficient because its time and memory complexity scale linearly with respect to the truncation steps. However, truncation may yield highly biased gradients that severely impact training. To mitigate this truncation bias [14], we consider *training only the top layer of a network to convergence*. The key insight is that the data helpful for training the output layer can also help train the whole network. Thus, we decompose the neural network into a feature

Algorithm 1 Dataset Distillation using Neural Feature Regression with Pooling (FRePo)

Require: $\mathcal{T}$: a labeled dataset; α: the learning rate for the distilled data
Initialization: Initialize a labeled distilled dataset $\mathcal{S} = (X_s, Y_s)$.
Initialization: Initialize a model pool $\mathcal{M}$ with m models $\{\theta_i\}_{i=1}^{m}$ randomly initialized from P_θ.

while not converged **do**
  ▷ Sample a model uniformly from the model pool: $\theta_i \sim \mathcal{M}$.
  ▷ Sample a target batch uniformly from the labeled dataset: $(X_t, Y_t) \sim \mathcal{T}$.
  ▷ Compute the meta-training loss $\mathcal{L}$ using Eq. 2
  ▷ Update the distilled data $\mathcal{S}$: $X_s \leftarrow X_s - \alpha \nabla_{X_s} \mathcal{L}$, and $Y_s \leftarrow Y_s - \alpha \nabla_{Y_s} \mathcal{L}$
  ▷ Train the model θ_i on the current distilled data $\mathcal{S}$ for one step.
  ▷ Reinitialize the model $\theta_i \sim P_\theta$ if θ_i has been updated more than K steps.
end while

Output: Learned distilled dataset $\mathcal{S} = (X_s, Y_s)$

extractor and a linear classifier. We fix the feature extractor at each meta-gradient computation and train the linear classifier to convergence before updating $\mathcal{S}$. After that, we adjust the feature extractor by training the whole network on the updated distilled data. We note that similar two-phase procedure has been studied in the context of representation learning [31].

Meta-Gradient Computation: If we consider the mean square error loss, then the optimal weights for the linear classifier have a closed-form solution. Moreover, since the feature dimension is typically larger than the number of distilled data, we can use kernel ridge regression (KRR) with a conjugate kernel [25] rather than solving the weights explicitly [12]. The resulting meta-training loss (Eq. 2) is similar to that used in KIP [22, 23], but we use a more flexible kernel rather than NTK.

$$\mathcal{L}\left(\mathcal{A}lg\left(\theta, \mathcal{S}\right), \mathcal{T}\right) = \frac{1}{2}||Y_t - K^{\theta}_{X_t X_s}(K^{\theta}_{X_s X_s} + \lambda I)^{-1} Y_s||_2^2, \tag{2}$$

where (X_t, Y_t) and (X_s, Y_s) are the inputs and labels of the real data $\mathcal{T}$ and distilled data $\mathcal{S}$ respectively. The Gram matrix between real inputs and distilled inputs is denoted as $K^{\theta}_{X_t X_s} \in \mathbb{R}^{|\mathcal{T}| \times |\mathcal{S}|}$, while the Gram matrix between distilled inputs is denoted as $K^{\theta}_{X_s X_s} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$. λ controls the regularization strength for KRR. Let us denote the neural network feature for a given input X and model parameter θ as $f(X, \theta) \in \mathbb{R}^{N \times d}$, where N is the number of input and d is the feature dimension [1]. The conjugate kernel is defined by the inner product of the neural network features. Thus, the two Gram matrices are computed as follows:

$$K^{\theta}_{X_t X_s} = f(X_t, \theta) f(X_s, \theta)^{\top}, \quad K^{\theta}_{X_s X_s} = f(X_s, \theta) f(X_s, \theta)^{\top}, \tag{3}$$

Now, computing the meta-gradient $\nabla_{\mathcal{S}} \mathcal{L}\left(\mathcal{A}lg\left(\theta, \mathcal{S}\right), \mathcal{T}\right)$ is just back-propagating through the conjugate kernel and a fixed feature extractor, which is very efficient and takes even fewer operations than computing the gradient for the network's weights. Moreover, we decouple the meta-gradient computation from the model online update. Hence, we can train the online model using any optimizer, and the distilled data will be agnostic to the specific learning algorithm choice. Our proposed method is similar to 1-step TBPTT in that we compute the meta-gradient at each step while performing the online model update. Unlike the conventional 1-step TBPTT, we compute the meta-gradient using a KRR output layer to mitigate truncation bias, illustrated in Figure 2(a).

Model Pool: As discussed in Section 1, there are various types of overfitting in dataset distillation. Several techniques have been proposed to alleviate such problem, such as random initialization [4], periodic reset [13, 5, 7], and dynamic bi-level optimization [19]. These techniques share the same underlying principle: the model diversity matters. Thus, we propose to *maintain a "model pool" filled with diverse set of parameters* obtained from different number of training steps and different random initializations. Unlike the previous methods that periodically training and resetting a single model, FRePo randomly sample a model from the pool at each meta-gradient computation and update it using the current distilled data. However, if a model has been updated more than K steps, we reinitialize it with a new random seed. From the meta-learning perspective, we maintain a diverse

[1] In practice, we use all the synthetic data and sample a minibatch from the real dataset to compute the meta-gradient (Algorithm 1).

set of meta-tasks to sample from and avoid sampling very similar tasks at each consecutive gradient computation to avoid overfitting to a particular setup.

Pool Diversity: We can increase the regularization strength by increasing the diversity of the model pool by setting a larger K, using data augmentation when training the model on the distilled data, or using models with different architectures. To keep our method simple, we use the same architecture for all models in the pool and do not use any data augmentation when training the model on the distilled data. Thus, our model pool only contains models with different initialization, at different optimization stages, and trained at different time-step of the distilled data.

3 Related Work

Unrolling in Bi-Level Optimization: One way to compute the meta-gradient is to differentiate through the unrolled inner optimization [4, 11–13]. However, this approach inherits several difficulties of the unrolled optimization, such as: 1) large computation and memory cost [14]; 2) truncation bias with short unrolls [17]; 3) exploding or vanishing gradients with long unrolls [15]; 4) chaotic and poorly conditioned loss landscapes with long unrolls [16]. In contrast, our method considers approximating the inner optimization with kernel ridge regression instead of unrolled optimization.

Surrogate Objective: To avoid unrolled optimization, several works turn to surrogate objectives. DC [5], DSA [7], and DCC [18] formulate the dataset distillation as a gradient matching problem between the gradients of neural network weights computed on the real and distilled data. In contrast, DM [8] and CAFE [19] consider the feature distribution alignment between the real and distilled data. Moreover, MTT [20] shows that knowledge from many expert training trajectories can be distilled to a dataset by using a training trajectory matching objective. Nevertheless, surrogate objectives may introduce new biases and thus, may not accurately reflect the true objective. For example, gradient matching approaches [5, 7, 18] only focus on short-range behavior and may easily overfit to a biased set of samples that produce dominant gradients [19, 20].

Closed-form Approximation: An alternative way to circumvent unrolled optimization is to find a closed-form approximation to the inner optimization. Based on the correspondence between infinitely-wide neural networks and kernel methods, KIP [22, 23] approximates the inner optimization with NTK [21]. In this case, the meta-gradient can be computed by back-propagating through the NTK. However, computing NTK for modern neural networks is extremely expensive. Thus, using NTK for dataset distillation requires thousands of GPU hours and sophisticated implementation of the distributed kernel computation framework [23]. Similar to ours, Bohdal et al. [12] also decomposes the neural network as a feature extractor and a linear classifier. However, they only learn the label and explicitly solve for the optimal classifier weights rather than perform KRR.

4 Dataset Distillation

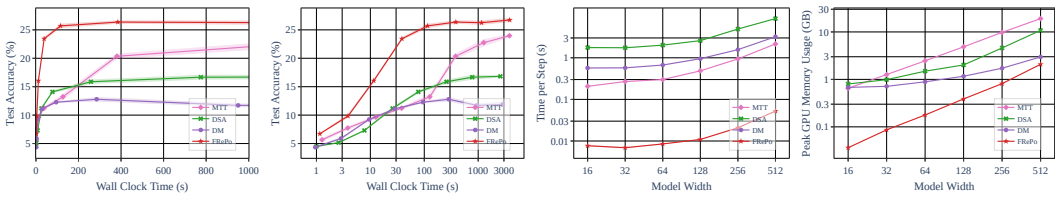
4.1 Implementation Details

We compare our method to four state-of-the-art dataset distillation methods [7, 8, 20, 23] on various benchmark datasets [26, 32–37]. We train the distilled data using Algorithm 1 with the same set of hyperparameters for all experiments except stated otherwise. Unlike prior work [7, 8, 20], we do not apply data augmentation during training. However, we apply the same data augmentation [7, 20] during evaluation for a fair comparison. We preprocess the data in a similar way as in previous works [20, 23] but use a wider architecture than previous works [7, 8, 20] because the KRR component does not behave well when the feature dimension is low, resulting in a significant performance drop for our method. Results on the original architecture are included in Appendix C.6. We evaluate each distilled data using five random neural networks and report the mean and standard deviation. For the baseline method, we report the best of the reported value in the original paper and our reproducing results.

For the sake of brevity, we provide implementation details about data preprocessing, distilled data initialization, and hyperparameters in Appendix A.1 and various ablation studies regarding the model pool, batch size, distilled data initialization, label learning, and model architectures in Appendix C. More distilled image visualizations can be found in Appendix E. Our code is available at `https://github.com/yongchao97/FRePo`.

Table 1: Test accuracies of models trained on the distilled data from scratch. † denotes performance better than the original reported performance. KRR preformance is shown in bracket. FRePo performs extremely well for one image per class setting on CIFAR100, Tiny ImageNet and CUB-200.

	Img/Cls	DSA [7]	DM [8]	KIP [23]	MTT [20]	FRePo
MNIST	1	88.7 ± 0.6	$89.9 \pm 0.8^{\dagger}$	90.1 ± 0.1	$91.4 \pm 0.9^{\dagger}$	$\mathbf{93.0 \pm 0.4}$ (92.6 ± 0.4)
	10	$97.9 \pm 0.1^{\dagger}$	$97.6 \pm 0.1^{\dagger}$	97.5 ± 0.0	$97.3 \pm 0.1^{\dagger}$	$\mathbf{98.6 \pm 0.1}$ (98.6 ± 0.1)
	50	$\mathbf{99.2 \pm 0.1}$	98.6 ± 0.1	98.3 ± 0.1	$98.5 \pm 0.1^{\dagger}$	$\mathbf{99.2 \pm 0.0}$ (99.2 ± 0.1)
F-MNIST	1	70.6 ± 0.6	$71.5 \pm 0.5^{\dagger}$	73.5 ± 0.5	$75.1 \pm 0.9^{\dagger}$	$\mathbf{75.6 \pm 0.3}$ (77.1 ± 0.2)
	10	$84.8 \pm 0.3^{\dagger}$	$83.6 \pm 0.2^{\dagger}$	86.8 ± 0.1	$\mathbf{87.2 \pm 0.3}^{\dagger}$	86.2 ± 0.2 (86.8 ± 0.1)
	50	$88.8 \pm 0.2^{\dagger}$	$88.2 \pm 0.1^{\dagger}$	88.0 ± 0.1	$88.3 \pm 0.1^{\dagger}$	$\mathbf{89.6 \pm 0.1}$ (89.9 ± 0.1)
CIFAR10	1	$36.7 \pm 0.8^{\dagger}$	$31.0 \pm 0.6^{\dagger}$	$\mathbf{49.9 \pm 0.2}$	46.3 ± 0.8	46.8 ± 0.7 (47.9 ± 0.6)
	10	$53.2 \pm 0.8^{\dagger}$	$49.2 \pm 0.8^{\dagger}$	62.7 ± 0.3	65.3 ± 0.7	$\mathbf{65.5 \pm 0.4}$ (68.0 ± 0.2)
	50	$66.8 \pm 0.4^{\dagger}$	$63.7 \pm 0.5^{\dagger}$	68.6 ± 0.2	71.6 ± 0.2	$\mathbf{71.7 \pm 0.2}$ (74.4 ± 0.1)
CIFAR100	1	$16.8 \pm 0.2^{\dagger}$	$12.2 \pm 0.4^{\dagger}$	15.7 ± 0.2	24.3 ± 0.3	$\mathbf{28.7 \pm 0.1}$ (32.3 ± 0.1)
	10	32.3 ± 0.3	29.7 ± 0.3	28.3 ± 0.1	40.1 ± 0.4	$\mathbf{42.5 \pm 0.2}$ (44.9 ± 0.2)
	50	42.8 ± 0.4	43.6 ± 0.4	–	$\mathbf{47.7 \pm 0.2}$	44.3 ± 0.2 (43.0 ± 0.3)
T-ImageNet	1	$6.6 \pm 0.2^{\dagger}$	3.9 ± 0.2	–	8.8 ± 0.3	$\mathbf{15.4 \pm 0.3}$ (19.1 ± 0.3)
	10	–	12.9 ± 0.4	–	23.2 ± 0.2	$\mathbf{25.4 \pm 0.2}$ (26.5 ± 0.1)
CUB-200	1	$1.3 \pm 0.1^{\dagger}$	$1.6 \pm 0.1^{\dagger}$	–	$2.2 \pm 0.1^{\dagger}$	$\mathbf{12.4 \pm 0.2}$ (13.7 ± 0.2)
	10	$4.5 \pm 0.3^{\dagger}$	$4.4 \pm 0.2^{\dagger}$	–	–	$\mathbf{16.8 \pm 0.1}$ (16.1 ± 0.3)



(a) Test Acc (b) Test Acc (Log Scale) (c) Time Per Step (d) GPU Memory Usage

Figure 3: (a,b) Training efficiency comparison when learning 1 Img/Cls on CIFAR100. (c,d) Time per iteration and peak memory usage as we increase the model size. FRePo is significantly more efficient than the previous methods, almost two orders of magnitude faster than the second-best method (i.e., MTT), with only 1/10 of the GPU memory requirement.

4.2 Standard Benchmarks

Distillation Performance: We first evaluate our method on six standard benchmark datasets. We learn 1, 10, and 50 images per class for datasets with only ten classes, while we learn 1 and 10 images per class for CIFAR100 [34] with 100 classes, Tiny ImageNet [35] with 200 classes, and CUB-200 [37] with 200 fine-grained classes. As shown in Table in 1, we achieve the state-of-the-art performance in most settings despite the hyperparameter may be suboptimal. Our method performs exceptionally well on datasets with a complex label space when learning few images per class. For example, we improve the CIFAR100, Tiny ImageNet, and CUB-200 in one image per class setting from 24.3%, 8.8%, and 2.2% to 28.7%, 15.4%, and 12.4%, respectively. Figure 4 shows that our distilled images look real and natural though we do not directly optimize for this objective. We observe a strong correlation between the test accuracy and image quality: the better the image quality, the higher the test accuracy. Our results suggest that a highly condensed dataset does not need to be very different from the real dataset as it may just reflect the most common pattern in a dataset. We also report the KRR predictor's test accuracy using the feature extractor trained on the distilled data. When the dataset is as simple as MNIST [32], the KRR predictor achieves similar performance as the neural network predictor. In contrast, for more complex datasets, the KRR predictor consistently outperforms the neural network predictor, with the most significant gap being 3.7% for Tiny ImageNet in the one image per class setting.

Table 2: Cross-architecture transfer performance on CIFAR10 with 10 Img/Cls. Despite being trained for a specific architecture, our distilled data transfer well to various architectures unseen during training. Conv is the default evaluation model used for each method. NN, DN, IN, and BN stand for no normalization, default normalization, Instance Normalization, Batch Normalization respectively.

	Train Arch	Evaluation Architecture					
		Conv	Conv-NN	ResNet-DN	ResNet-BN	VGG-BN	AlexNet
DSA [7]	Conv-IN	53.2 ± 0.8	36.4 ± 1.5	42.1 ± 0.7	34.1 ± 1.4	46.3 ± 1.3	34.0 ± 2.3
DM [8]	Conv-IN	49.2 ± 0.8	35.2 ± 0.5	36.8 ± 1.2	35.5 ± 1.3	41.2 ± 1.8	34.9 ± 1.1
MTT [20]	Conv-IN	64.4 ± 0.9	41.6 ± 1.3	49.2 ± 1.1	42.9 ± 1.5	46.6 ± 2.0	34.2 ± 2.6
KIP [23]	Conv-NTK	62.7 ± 0.3	58.2 ± 0.4	49.0 ± 1.2	45.8 ± 1.4	30.1 ± 1.5	57.2 ± 0.4
FRePo	Conv-BN	$\mathbf{65.5 \pm 0.4}$	$\mathbf{65.5 \pm 0.4}$	$\mathbf{58.1 \pm 0.6}$	$\mathbf{57.7 \pm 0.7}$	$\mathbf{59.4 \pm 0.7}$	$\mathbf{61.9 \pm 0.7}$

(a) FRePo (b) MTT [20] (c) DSA [7] (d) Label Visualization

Figure 4: (a,b,c) Distilled 1 img/cls from CIFAR100 using FRePo, MTT, and DSA. High quality images also produce high test accuracy. (d) Three categories of learned labels. (Top) High confidence, large margin; (Middle) High confidence, small margin; (Bottom) Low confidence, small margin.

Label Learning: A similar trend can also be observed for label learning. When the dataset is simple and has only a few classes, label learning may not be necessary. However, it becomes crucial for complex datasets with many labels, such as CIFAR100 and Tiny-ImageNet (See more details in Appendix C.4). Similar to the teacher label in the knowledge distillation [1], we observe that the distilled label also encodes the class similarity. We identify three typical cases in Figure 4d. The first group consists of highly confident labels with a much higher value for one class than other classes (large margin), such as sunflower, bicycle, and chair. In contrast, the distilled labels in the second group are confident but may get confused with some closely-related classes (small margin). For instance, the learned label for "girl" has almost equally high values for the girl, woman, man, boy, and baby, suggesting that these classes are very similar and may be difficult for the model to distinguish them apart. The last group contains distilled labels with low values for all classes, such as bear, beaver, and squirrel. It is often hard for humans to recognize the distilled images in such a group, suggesting that they may be the challenging classes in a dataset.

Training Cost Analysis: Figure 3a, 3b shows that our method is significantly more time-efficient than the previous methods. When learning one image per class on CIFAR100, FRePo reaches a similar test accuracy (23.4%) to the second-best method (24.0%) in 38 seconds, compared to 3805 seconds for MTT, which is roughly two orders of magnitude faster. Moreover, FRePo achieves 92% of its final test accuracy (26.4% out of 28.7%) in only 385 seconds. As shown in Figure 3c, our algorithm takes much less time to perform one gradient step on the distilled data. Thus, we can perform more gradient steps in a fixed time. Furthermore, Figure 3d suggests that our algorithm has much less GPU memory requirement. Therefore, we can potentially use a much larger and more complex model to take advantage of the advancement in neural network architecture.

Cross-Architecture Generalization: One desired property of our distilled data is that it generalizes well to architecture it has not seen during the training. Similar to previous works [5, 20], we evaluate the distilled data from CIFAR10 on a wide range of architectures which it has not seen during training, including AlexNet [38], VGG [39], and ResNet [40]. Table 2 shows that our method outperforms previous methods on all unseen architectures. Instance Normalization (IN) [41], as the vital ingredient in several methods (DSA, DM, MTT), seems to hurt the cross-architecture transfer. The performance

Table 3: Distillation performance on higher resolution (128x128) dataset (i.e. ImageNette, ImageWoof) and medium resolution (64x64) dataset with a complex label space (i.e. ImageNet-1K). FRePo scales to high-resolution images and learns the discriminate feature of complex datasets.

	ImageNette (128x128)		ImageWoof (128x128)		ImageNet (64x64)	
Img/Cls	1	10	1	10	1	2
Random Subset	23.5 ± 4.8	47.7 ± 2.4	14.2 ± 0.9	27.0 ± 1.9	1.1 ± 0.1	1.4 ± 0.1
MTT [20]	47.7 ± 0.9	63.0 ± 1.3	28.6 ± 0.8	35.8 ± 1.8	–	–
FRePo	$\mathbf{48.1 \pm 0.7}$	$\mathbf{66.5 \pm 0.8}$	$\mathbf{29.7 \pm 0.6}$	$\mathbf{42.2 \pm 0.9}$	$\mathbf{7.5 \pm 0.3}$	$\mathbf{9.7 \pm 0.2}$

degrades a lot when no normalization (NN) is applied (Conv-NN, AlexNet) or using a different normalization, like Batch Normalization (BN) [42]. It suggests that the distilled data generated by those methods encode the inductive bias of a particular training architecture. In contrast, our distilled data generalize well to various architectures, including those without normalization (Conv-NN, AlexNet). Note that Figure 1, 4 also indicate that our distilled data encode less architectural bias as the distilled images look natural and authentic. A simple idea to further alleviate the overfitting of a particular architecture is to include more architectures in the model pool. However, the training may not be stable as the meta-gradient computed by different architectures can be very different.

4.3 ImageNet

High Resolution ImageNet Subset To understand how well our method performs on high-resolution images, we evaluate it on ImageNette and ImageWoof datasets [36] with a resolution of 128x128. We learn 1 and 10 images per class on both datasets and report the performance in Table 3 and visualize some distilled images in Figure 1. As shown in Table 3, we outperform MTT on all settings and achieve much better performance when we distill ten images per class on a more difficult dataset ImageWoof. It suggests that our distilled data is better at capturing the discriminative features for each class. Figure 1 shows that our distilled images look real and capture the distinguishable feature of different classes. For the easy dataset (i.e., ImageNette), all images have clear different structures, while for ImageWoof, the texture of each dog seems to be crucial.

Resized ImageNet-1K: We also evaluate our method on a resized version of ILSVRC2012 [26] with a resolution of 64x64 to see how it performs on a complex label space. Surprisingly, we can achieve 7.5% and 9.7% Top1 accuracy using only 1k and 2k training examples, compared to 1.1% and 1.4% using an equally-sized real subset.

5 Application

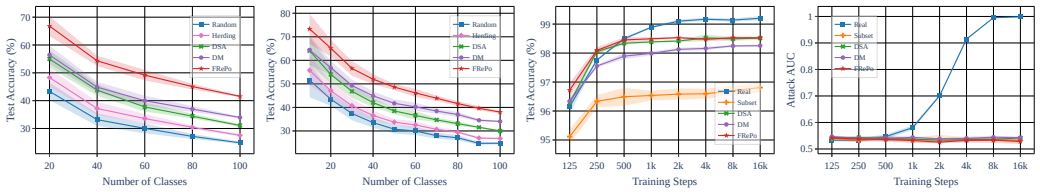
5.1 Continual Learning

Continual learning (CL) [43] aims to address the catastrophic forgetting problem [43–45] when a model learns sequentially from a stream of tasks. A commonly used strategy to recall past knowledge is based on a replay buffer, which stores representative samples from previous tasks [46–49]. Since sample selection is an important component of constructing an effective buffer [48–51], we believe distilled data can be a key ingredient for a continual learning algorithm due to its highly condensed nature. Several works [6–8, 52] have successfully applied the dataset distillation to the continual learning scenario. Our work shows that we can achieve much better results by using a better dataset distillation technique.

We follow Zhao and Bilen [8] that sets up the baseline based on GDumb [49] which greedily stores class-balanced training examples in memory and train model from scratch on the latest memory only. In that case, the continual learning performance only depends on the quality of the replay buffer. We perform 5 and 10 step class-incremental learning [53] on CIFAR100 with an increasing buffer size of 20 images per class. Specifically, we distill 400 and 200 images at each step and put them into the replay buffer. We follow the same class split as Zhao and Bilen [8] and compare our method to random [49], herding [54, 55], DSA [7], and DM [8]. We use the default data preprocessing and default model for each method in this experiment as we find it gives the best performance for each method. We use the test accuracy on all observed classes as the performance measure [8, 48].

Table 4: AUC of five attackers on models trained on the real and distilled MNIST data. The model trained on the real data is vulnerable to MIAs, while the model trained on the distilled data is robust to MIAs. Training on distilled data allows privacy preservation while retaining model performance.

	Test Acc (%)	Attack AUC				
		Threshold	LR	MLP	RF	KNN
Real	**99.2 ± 0.1**	0.99 ± 0.01	0.99 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.97 ± 0.00
Subset	96.8 ± 0.2	0.52 ± 0.00	**0.50 ± 0.01**	**0.53 ± 0.01**	0.55 ± 0.00	0.54 ± 0.00
DSA	98.5 ± 0.1	**0.50 ± 0.00**	0.51 ± 0.00	0.54 ± 0.00	0.54 ± 0.01	0.54 ± 0.01
DM	98.3 ± 0.0	**0.50 ± 0.00**	0.51 ± 0.01	0.54 ± 0.01	0.54 ± 0.01	0.53 ± 0.01
FRePo	98.5 ± 0.1	0.52 ± 0.00	0.51 ± 0.00	**0.53 ± 0.01**	**0.52 ± 0.01**	**0.51 ± 0.01**



(a) 5-step CL (b) 10-step CL (c) MIA: ACC (d) MIA: AUC

Figure 5: (a,b) Multi-class accuracies across all classes observed up to a certain time point. We perform significantly better than other methods in both 5 and 10 step class-incremental continual learning. (c,d) Test accuracy and attack AUC as we increase the number of training steps. AUC keeps increasing when training a model on the real data for more steps. In contrast, AUC keeps low when training on distilled data.

Figure 5 shows that our method performs significantly better than all previous methods. The final test accuracy for all classes for our method (FRePo) and the second-best method (DM) are 41.6%, 33.9% in 5-step learning, and 38.0%, 34.0% in 10-step learning. However, we notice that for FRePo, distilling 2000 images in a continual learning setup achieves a similar test accuracy (41.6%) as distilling only 1000 images from the whole dataset (41.3% from Table 1). In addition, performance drops as we perform more steps. It suggests that FRePo considers all available classes to derive the most condensed dataset. Splitting the data into multiple groups and performing independent distillation may generate redundant information or fail to capture the distinguishable features.

5.2 Membership Inference Defense

Membership inference attacks (MIA) aim to infer whether a given data point has been used to train the model or not [56–58]. Ideally, we want a model to learn from the data but not memorize it to preserve privacy. However, deep neural networks are well-known for their ability to memorize all the training examples, even on large and randomly labeled datasets [59]. Several methods have been proposed to defend against such attacks by either modifying the training procedure [60] or changing the inference workflow [61]. This section shows that the distilled data contain little information regarding sample presence in the original dataset. Thus, instead of training on the original datasets, training on distilled data allows privacy preservation while retaining model performance.

We consider three distilled data generated by DSA [7], DM [8] and FRePo. We perform five popular "black box" MIA provided by Tensorflow Privacy [62] on models trained on the real data or the data distilled from it. The attack methods include a threshold attack and four model-based attacks using logistic regression (LR), multi-layer perceptron (MLP), random forest (RF) and K-nearest neighbor (KNN). The inputs to those attack methods are ground-truth labels, model predictions, and losses. To measure the privacy vulnerability of the trained model, we compute the area under the ROC curve (AUC) of an attack classifier. Following prior work, [56, 63], we keep a balanced set of training examples (member) and test examples (non-member) with 10K each to maximize the uncertainty of MIA. Thus, the random guessing strategy results in a 50% MIA accuracy. We conduct experiments on MNIST and FashionMNIST with a distillation size of 500. For space reasons, we provide more implementation details and results in appendix.

As shown in Table 4, all models trained on the distilled data preserve privacy as their attack AUCs are closed to random guessing. However, we observe a small drop in test accuracy compared to the model trained on the full dataset, which is expected as we only distill 500 examples instead of 10,000 examples. Compared to the model trained on an equally sized subset of the original data, the model trained on distilled data results in much better test performance. Figure 5c, 5d demonstrate the trade-off between test accuracy and attack effectiveness as measured by ROC AUC. It shows that early stopping can be an effective technique to preserve privacy. However, we will still be under high MIA risk if we perform early stopping by monitoring the validation loss. In contrast, training a model on the distilled data does not have this problem as the attack AUCs keep at a very low level regardless of training steps.

6 Conclusion

We propose neural Feature Regression with Pooling (FRePo) to overcome two challenges in dataset distillation: meta-gradient computation and various types of overfitting in dataset distillation. We obtain state-of-the-art performance on various datasets with a 100x reduction in training time and a 10x reduction in GPU memory requirement. The distilled data generated by FRePo looks real and natural and generalizes well to a wide range of architectures. Furthermore, we demonstrate two applications that take advantage of the high-quality distilled data, namely, continual learning and membership inference defense.

Broader Impact "Synthetic data", in the broader sense of artificial data created by generative models, can help researchers understand how an otherwise opaque learning machine "sees" the world. There have been concerns regarding the risk of fake data. This paper explores a new research direction in generating synthetic data only for downstream classification tasks. We believe this work can provide additional interpretability and potentially address the common concerns in machine learning regarding training data privacy.

Acknowledgments and Disclosure of Funding

We would like to thank Harris Chan, Andrew Jung, Michael Zhang, Philip Fradkin, Denny Wu, Chong Shao, Leo Lee, Alice Gao, Keiran Paster, and Lazar Atanackovic for their valuable feedback. Jimmy Ba was supported by NSERC Grant [2020-06904], CIFAR AI Chairs program, Google Research Scholar Program and Amazon Research Award. This project was supported by LG Electronics Canada. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence.

References

[1] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL `http://arxiv.org/abs/1503.02531`.

[2] Takashi Fukuda, Masayuki Suzuki, Gakuto Kurata, Samuel Thomas, Jia Cui, and Bhuvana Ramabhadran. Efficient knowledge distillation from an ensemble of teachers. In Francisco Lacerda, editor, *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, pages 3697–3701. ISCA, 2017. URL `http://www.isca-speech.org/archive/Interspeech_2017/abstracts/0614.html`.

[3] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=S1XolQbRW`.

[4] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *CoRR*, abs/1811.10959, 2018. URL `http://arxiv.org/abs/1811.10959`.

[5] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event,*

Austria, May 3-7, 2021. OpenReview.net, 2021. URL https://openreview.net/forum?id=mSAKhLYLSsl.

[6] Andrea Rosasco, Antonio Carta, Andrea Cossu, Vincenzo Lomonaco, and Davide Bacciu. Distilled replay: Overcoming forgetting through synthetic samples. *CoRR*, abs/2103.15851, 2021. URL https://arxiv.org/abs/2103.15851.

[7] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 12674–12685. PMLR, 2021. URL http://proceedings.mlr.press/v139/zhao21a.html.

[8] Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. *CoRR*, abs/2110.04181, 2021. URL https://arxiv.org/abs/2110.04181.

[9] Guang Li, Ren Togo, Takahiro Ogawa, and Miki Haseyama. Soft-label anonymous gastric x-ray image distillation. *CoRR*, abs/2104.02857, 2021. URL https://arxiv.org/abs/2104.02857.

[10] Jack Goetz and Ambuj Tewari. Federated learning via synthetic data. *CoRR*, abs/2008.04489, 2020. URL https://arxiv.org/abs/2008.04489.

[11] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2113–2122. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/maclaurin15.html.

[12] Ondrej Bohdal, Yongxin Yang, and Timothy M. Hospedales. Flexible dataset distillation: Learn labels instead of images. *CoRR*, abs/2006.08572, 2020. URL https://arxiv.org/abs/2006.08572.

[13] Ilia Sucholutsky and Matthias Schonlau. Improving dataset distillation. *CoRR*, abs/1910.02551, 2019. URL http://arxiv.org/abs/1910.02551.

[14] Paul Vicol, Luke Metz, and Jascha Sohl-Dickstein. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 10553–10563. PMLR, 2021. URL http://proceedings.mlr.press/v139/vicol21a.html.

[15] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org, 2013. URL http://proceedings.mlr.press/v28/pascanu13.html.

[16] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, C. Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4556–4565. PMLR, 2019. URL http://proceedings.mlr.press/v97/metz19a.html.

[17] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger B. Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1MczcgR-.

[18] Saehyung Lee, Sanghyuk Chun, Sangwon Jung, Sangdoo Yun, and Sungroh Yoon. Dataset condensation with contrastive signals. *CoRR*, abs/2202.02916, 2022. URL https://arxiv.org/abs/2202.02916.

[19] Kai Wang, Bo Zhao, Xiangyu Peng, Zheng Zhu, Shuo Yang, Shuo Wang, Guan Huang, Hakan Bilen, Xinchao Wang, and Yang You. CAFE: learning to condense dataset by aligning features. *CoRR*, abs/2203.01531, 2022. doi: 10.48550/arXiv.2203.01531. URL https://doi.org/10.48550/arXiv.2203.01531.

[20] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A. Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. *CoRR*, abs/2203.11932, 2022. doi: 10.48550/arXiv.2203.11932. URL https://doi.org/10.48550/arXiv.2203.11932.

[21] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8570–8581, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/0d1a9651497a38d8b1c3871c84528bd4-Abstract.html.

[22] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=l-PrrQrK0QR.

[23] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 5186–5198, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/299a23a2291e2126b91d54f3601ec162-Abstract.html.

[24] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 1540–1552. PMLR, 2020. URL http://proceedings.mlr.press/v108/lorraine20a.html.

[25] Radford M. Neal. Bayesian learning for neural networks. 1995.

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[27] Aravind Rajeswaran, Chelsea Finn, Sham M. Kakade, and Sergey Levine. Meta-learning with implicit gradients. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 113–124, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/072b030ba126b2f4b2374f342be9ed44-Abstract.html.

[28] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

[29] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.

[30] Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *CoRR*, abs/1705.08209, 2017. URL http://arxiv.org/abs/1705.08209.

[31] Jimmy Ba, Murat A Erdogdu, Taiji Suzuki, Zhichao Wang, Denny Wu, and Greg Yang. High-dimensional asymptotics of feature learning: How one gradient step improves the representation. *arXiv preprint arXiv:2205.01445*, 2022.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://doi.org/10.1109/5.726791.

[33] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

[34] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[35] Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge. 2015.

[36] Jeremy Howard. A smaller subset of 10 easily classified classes from imagenet, and a little more french. URL https://github.com/fastai/imagenette/.

[37] Technical report.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

[41] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. URL http://arxiv.org/abs/1607.08022.

[42] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

[43] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. URL http://arxiv.org/abs/1612.00796.

[44] Robert French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3:128–135, 05 1999. doi: 10.1016/S1364-6613(99)01294-2.

[45] Anthony V. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connect. Sci.*, 7 (2):123–146, 1995. doi: 10.1080/09540099550039318. URL https://doi.org/10.1080/09540099550039318.

[46] Pietro Buzzega, Matteo Boschini, Angelo Porrello, and Simone Calderara. Rethinking experience replay: a bag of tricks for continual learning. In *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021*, pages 2180–2187. IEEE, 2020. doi: 10.1109/ICPR48806.2021.9412614. URL https://doi.org/10.1109/ICPR48806.2021.9412614.

[47] Yaoyao Liu, Bernt Schiele, and Qianru Sun. Adaptive aggregation networks for class-incremental learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 2544–2553. Computer Vision Foundation / IEEE, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Liu_Adaptive_Aggregation_Networks_for_Class-Incremental_Learning_CVPR_2021_paper.html.

[48] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016. URL http://arxiv.org/abs/1611.07725.

[49] Ameya Prabhu, Philip H. S. Torr, and Puneet K. Dokania. Gdumb: A simple approach that questions our progress in continual learning. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II*, volume 12347 of *Lecture Notes in Computer Science*, pages 524–540. Springer, 2020. doi: 10.1007/978-3-030-58536-5_31. URL https://doi.org/10.1007/978-3-030-58536-5_31.

[50] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11816–11825, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/e562cd9c0768d5464b64cf61da7fc6bb-Abstract.html.

[51] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11849–11860, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/15825aee15eb335cc13f9b559f166ee8-Abstract.html.

[52] Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. Mnemonics training: Multi-class incremental learning without forgetting. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 12242–12251. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.01226. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Liu_Mnemonics_Training_Multi-Class_Incremental_Learning_Without_Forgetting_CVPR_2020_paper.html.

[53] Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *CoRR*, abs/1904.07734, 2019. URL http://arxiv.org/abs/1904.07734.

[54] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XII*, volume 11216 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2018. doi: 10.1007/978-3-030-01258-8_15. URL https://doi.org/10.1007/978-3-030-01258-8_15.

[55] Yutian Chen, Max Welling, and Alexander J. Smola. Super-samples from kernel herding. In Peter Grünwald and Peter Spirtes, editors, *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*, pages 109–116. AUAI Press, 2010. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2148&proceeding_id=26.

[56] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 3–18. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.41. URL https://doi.org/10.1109/SP.2017.41.

[57] Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A. Gunter, and Kai Chen. Understanding membership inferences on well-generalized learning models. *CoRR*, abs/1802.04889, 2018. URL http://arxiv.org/abs/1802.04889.

[58] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[59] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

[60] Milad Nasr, Reza Shokri, and Amir Houmansadr. Machine learning with membership privacy using adversarial regularization. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 634–646. ACM, 2018. doi: 10.1145/3243734.3243855. URL https://doi.org/10.1145/3243734.3243855.

[61] Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. Memguard: Defending against black-box membership inference attacks via adversarial examples. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 259–274. ACM, 2019. doi: 10.1145/3319535.3363201. URL https://doi.org/10.1145/3319535.3363201.

[62] tensorflow/privacy: library for training machine learning models with privacy for training data, 2022. URL https://github.com/tensorflow/privacy.

[63] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 268–282. IEEE Computer Society, 2018. doi: 10.1109/CSF.2018.00027. URL https://doi.org/10.1109/CSF.2018.00027.

[64] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL http://github.com/google/flax.

[65] Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=Syx4wnEtvH.

[66] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[67] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[68] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

[69] Yuxin Wu and Kaiming He. Group normalization. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, volume 11217 of *Lecture*

Notes in Computer Science, pages 3–19. Springer, 2018. doi: 10.1007/978-3-030-01261-8_1. URL https://doi.org/10.1007/978-3-030-01261-8_1.

[70] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs V: tuning large neural networks via zero-shot hyperparameter transfer. *CoRR*, abs/2203.03466, 2022. doi: 10.48550/arXiv.2203.03466. URL https://doi.org/10.48550/arXiv.2203.03466.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Experimental Details

A.1 Implementation Details

Datasets: We evaluate our methods on the following datasets: i) **MNIST** [32]: A standard image dataset consists of 10 classes of 28x28 grey-scale images of handwritten digits, including 60,000 training examples and 10,000 test examples. ii) **FashionMNIST** [33]: A direct drop-in replacement for MNIST [32] consisting of 10 classes of 28x28 grey-scale images of clothing, including 60,000 training examples and 10,000 test examples. We denote FashionMNIST as F-MNIST for simplicity. iii) **CIFAR** [34]: A standard image dataset with two tasks: one coarse-grained over 10 classes (CIFAR10) and one fine-grained over 100 classes (CIFAR100). Both CIFAR10 and CIFAR100 have 50,000 training examples and 10,000 test examples with a resolution of 32x32. iv) **Tiny ImageNet** [35]: A higher resolution (64x64) dataset with 200 classes, including 100,000 training examples and 10,000 test examples. We denote Tiny ImageNet as T-ImageNet for simplicity. v) **ImageNette** and **ImageWoof** [36]: ImageNette (assorted objects) and ImageWoof (dog breeds) are two 10-class subsets from ILSVRC2012 [26] designed to be easy and hard to learn, respectively. ImageNette consists of 9,469 training examples and 3,925 testing examples, while ImageWoof contains 9025 training examples and 3929 testing examples. We resize all examples to a resolution of 128x128. vi) **ImageNet** [26]: A standard image benchmark dataset consists of 1000 classes from the Large Scale Visual Recognition Challenge 2012 [26], including 1,281,167 training examples and 50,000 testing examples. We resize all examples to a resolution of 64x64. vii) **CUB-200** [37]: A fine-grained classification task, consisting of 200 subcategories belonging to birds. It has 5,994 data points for training and 5,794 data points for testing. We resize all examples to a resolution of 32x32.

Data Preprocessing: We use the standard preprocessing for all datasets but add regularized ZCA transformation for RGB datasets as described in KIP [22, 23]. However, unlike KIP, we do not apply layer normalization to all examples. Moreover, for simplicity, we apply the same regularization strength $\lambda = 0.1$ to all datasets rather than tune each dataset. This regularization strength is tuned for CIFAR in KIP [23]. Besides, for ImageNette and ImageWoof, performing the full-size ZCA transformation is extremely expensive due to the high resolution, so we use a checkboard ZCA instead. It is the reason why our distilled images in Figure 1(c) have checkboard artifacts. As for the previous method, DSA [7], DM [8] only use standard preprocessing, while MTT [20] and KIP [23] consider both standard preprocessing and ZCA preprocessing, but they implement ZCA preprocessing differently. To account for the difference in data preprocessing, we reproduce each method using our data preprocessing and report the best of the reported value in the original paper and our reproducing results. As shown in Table 1, it turns out that our data preprocessing can improve DSA [7] and DM [8] a lot on RGB datasets but achieves a comparable performance for MTT [20]. When visualizing the distilled images, we directly apply the reverse transformation according to the corresponding data preprocessing without any other modifications.

Models: We use a simple convolutional neural network for all experiments based on the architecture used in previous works [7, 20, 23]. It consists of several blocks of 3x3 convolution, normalization, RELU, and 2×2 average pooling layer with stride 2. We use 3, 4, and 5 blocks for datasets with resolutions 32x32, 64x64, and 128x128, respectively. Unlike previous works, which use the same number of filters at every layer, we double the number of filters if the feature map size is halved, following the modern neural network designs that preserve the time complexity per layer[39, 40]. We observe this to be crucial for our method when we distill thousands of data because we need the feature dimension to be much larger than the distilled size to make the KRR work properly. Furthermore, we replace the Instance Normalization [41] with Batch Normalization [42] during training and do not use any normalization during evaluation. For DSA, DM, and MTT, the default normalization for both training and evaluation is the Instance Normalization [41]. In contrast, KIP [23] uses analytical NTK during training and uses a 1024-width network without normalization for evaluation. Moreover, KIP [23] adds an extra convolution layer at the first layer of the neural network. We initialize our model using Lecun Initialization, which is the default in Flax library [64]. In contrast, DSA, DM, and MTT use Kaiming Initialization, which is the default in PyTorch, while KIP [23] initializes the model using random Gaussian with standard deviations $\sqrt{2}$ and 0.1 for weights and biases, respectively. We denote the default model used in each method as Conv for simplicity. We reproduce each method using our model and report the best of the reported value in the original paper and our reproducing results. We find our model achieves comparable or worse performance than each method's default methods, so we use their default model in most settings.

Algorithm 2 Dataset Distillation using Neural Feature Regression with Pooling (FRePo)

Require: $\mathcal{T}$: a labeled dataset; α: the learning rate for the distilled data
Initialization: Initialize a labeled distilled dataset $\mathcal{S} = (X_s, Y_s)$.
Initialization: Initialize a model pool $\mathcal{M}$ with m models $\{\theta_i\}_{i=1}^{m}$ randomly initialized from P_θ.

while not converged **do**
  ▷ Sample a model uniformly from the model pool: $\theta_i \sim \mathcal{M}$.
  ▷ v1: Train the model θ_i on the current distilled data $\mathcal{S}$ for one step.
  ▷ Sample a target batch uniformly from the labeled dataset: $(X_t, Y_t) \sim \mathcal{T}$.
  ▷ Compute the meta-training loss $\mathcal{L}$ using Eq. 2
  ▷ Update the distilled data $\mathcal{S}$: $X_s \leftarrow X_s - \alpha \nabla_{X_s} \mathcal{L}$, and $Y_s \leftarrow Y_s - \alpha \nabla_{Y_s} \mathcal{L}$
  ▷ v2: Train the model θ_i on the current distilled data $\mathcal{S}$ for one step.
  ▷ Reinitialize the model $\theta_i \sim P_\theta$ if θ_i has been updated more than K steps.
end while

Output: Learned distilled dataset $\mathcal{S} = (X_s, Y_s)$

Besides, we notice that the distilled data can encode the architecture's inductive bias, so we provide an ablation study on how the model architecture affects the distilled data in Section C.6.

Initialization: Staying consistent with previous works, [7, 20, 23], we initialize the distilled image with randomly sampled real images. We also investigate initializing the distilled image with random Gaussian noise and observe that initializing does not affect the final performance too much but affects the convergence speed. We find the real initialization gives a decent convergence speed, so we choose the real initialization for simplicity rather than fine-tune the scale of random Gaussian initialization. As for labels, we initialize them using a scaled mean-centered one-hot vector for the corresponding image, where the scaling factor (i.e., $1/(\sqrt{C/10})$) depends on the number of classes C . We find that this label initialization scheme speeds up the convergence but has little impact on the final performance. We provide an ablation study regarding the initialization scheme in Section C.3.

Label Learning: Whether to learn the label is a Boolean hyperparameter in our experiments. When true, we optimize it using Algorithm 1. When false, we stop the gradient so that they remain fixed at its initialization. We provide an ablation study on label learning in Section C.4.

Training and Evaluation: The proposed algorithm has two versions depending on the order of meta-gradient computation and online model update. We present the two versions in Algorithm 2 and highlight the difference in red. We denote computing the meta-gradient after the online model update as v1 and denote computing the meta-gradient before the online model update as v2. The first implementation (v1) is more similar to the standard back-propagation through time as we first do the forward pass (online model update) and then the backward pass. However, instead of backpropagating the gradient through the inner optimization, we backpropagate through the kernel. In contrast, the insight behind the second implementation (v2) is that we want to find good data to train the last layer first, then we use it to train the whole network. Empirically, we do not observe any difference between these two implementations, and we choose v2 as the default for all experiments. Besides, we use the distilled data and its flipped version in meta-gradient computation to mitigate the mirroring effect for RGB datasets. For evaluation, we follow the standard protocol [7, 20]: training a randomly initialized neural network from scratch on distilled data and evaluating on a held-out test dataset. We apply the same data augmentation as in previous work [7, 20] during evaluation for a fair comparison.

Hyperparameters: We aim to keep our method as simple and efficient as possible. As a result, our method requires very few hyperparameter tuning efforts than all the previous methods. We use the same set of hyperparameters for all experiments, except stated otherwise. Specifically, we use LAMB optimizer [65] with a cosine learning rate schedule starting from 0.0003 for both images and labels. We use a batch size of 1024 and train the distilled data up to 2 million steps to see its long-run behavior. In practice, we observe that most of the convergence (more than 95% of its final test accuracy) is achieved after a few thousand steps with a slow, logarithmic increase with more iterations, as shown in Figure 3b. The model pool contains ten models trained up to $K = 100$ steps. Each model is trained using Adam optimizer [66] with a constant learning rate of 0.0003, and no weight decay is applied. We use the same kernel regularizer λ as KIP [23]. Instead of being a fixed constant, the regularizer is adapted to the scale of $K^{\theta}_{X_s X_s}$, $\lambda = \lambda_0 Tr(K^{\theta}_{X_s X_s})$, where $\lambda_0 = 10^{-6}$. We note that our hyperparameter choice may be sub-optimal, but it is a good starting point. We conduct some ablation study regarding the hyperparameters in Section C. Moreover, we provide a

hyperparameter tuning guideline for practitioners in Section D accompanied by a list of additional tricks that we find to improve the performance but do not include in the current algorithm.

Summary: We implement our method in JAX [67] and reproduce previous methods using their released code. To take into account the differences in data processing and architectures, we try our best to reproduce previous results by varying different data preprocessing and models. Experiments show that our data preprocessing can sometimes improve performance, but our model does not give better performance than previous methods. We report the best of the reported value in the original paper and our reproducing results. We evaluate each distilled data using five random neural networks and report the mean and standard deviation.

A.2 Experimental Setups

Figure 1: Selected images from (a) 1 Img/Cls CIFAR100 (ZCA, learn label=True), (b) 1 Img/Cls Tiny ImageNet (ZCA, learn label=True), (c) 10 Img/Cls ImageNette (Top 2 rows) (Checkboard ZCA, learn label=True), 10 Img/Cls ImageWoof (Bottom 2 rows) (Checkboard ZCA, learn label=True).

Figure 3a, 3b: We measure the time per step by measuring the average wall clock time ten times on ten steps. We take the first measurement after 50 steps when the statistics become stable and report the mean and standard deviation of 10 runs. To generate Figure 3a, 3b, we use the default model and default hyperparameter for each method. We first record the time per step of each method and run another program to evaluate the checkpoints at different time steps to get the test accuracy. Thus, the wall clock time is computed by multiplying the time per step and the number of steps taken at each checkpoint. All models are trained on Nvidia Quadro RTX 6000 with 22.17GB memory, and 7.5 compute capability.

Figure 3c, 3d, 14a, 14b: Similar to how we generate Figure 3a, 3b, we measure the time per step by measuring the average wall clock time ten times on ten steps. We take the first measurement after 50 steps when the statistics become stable and report the mean and standard deviation of 10 runs. We use jax.profiler and torch.profiler for the JAX program and PyTorch program to measure the peak GPU memory usage. Different from Figure 3a, 3b, we use the same model (i.e., Conv used by DSA [7]) and same batch size=256 in this section. We take an optimistic estimation for the previous methods (i.e., DSA, MTT) by choosing a smaller inner loop or outer loop number to generate more data points before encountering out of memory errors. Specifically, we use outer_loop=1, inner_loop=1 for DSA and use syn_steps=15 for MTT. All models are trained on Nvidia Quadro RTX 6000 with 22.17GB memory, and 7.5 compute capability.

Figure 4a: Distilled image visualization when distilling 1 Img/Cls from CIFAR100 using FRePo.

Figure 4b: Distilled image visualization when distilling 1 Img/Cls from CIFAR100 using MTT [20]. To give the best image quality, we use the distilled images provided by the original paper. (Url:`https://georgecazenavette.github.io/mtt-distillation/tensors/index.html#tensors`)

Figure 4c: Distilled image visualization when distilling 1 Img/Cls from CIFAR100 using DSA [7]. To give the best image quality, we use the distilled images provided by the original paper. (Url: `https://drive.google.com/drive/folders/1Dp6V6RvhJQPsB-2uZCwdlHXf1iJ9Wb_g`). Besides, we adjust the contrasts to give the best visualization.

Figure 4d: Distilled label visualization when distilling 1 Img/Cls from CIFAR100 using FRePo. The distilled images are sunflower, girl, and bear, respectively. All labels are unnormalized logits.

Figure 5a, 5b: Multi-class accuracies across all classes observed up to a specific time point. For a fair comparison, we follow the same setup in DM [8], including the class split of five different runs. We use the same set of hyperparameters as other experiments and train the distilled data up to 500K steps. We report the mean and standard deviation of the five different runs. We observe that the primary source of variance comes from the class split.

Figure 5c, 5d, 6a, 6b: Test accuracies and attack AUCs as we increase the number of training steps. We use the same set of hyperparameters as other experiments except for data augmentation. We do not apply any data augmentation when we train models on the distilled data since any type of regularization can alleviate the MIA risks, making it hard to see the effects of distilled data.

Table 1, 3, 7, 5: We try to reproduce the previous methods based on their official codebase and vary the data preprocessing and model architecture. We report the best of the reported value in the original

Table 5: Distillation performance on higher resolution (128x128) dataset (i.e. ImageNette, ImageWoof) and medium resolution (64x64) dataset with a complex label space (i.e. ImageNet-1K). FRePo scales well to high-resolution images and learns the discriminate feature of complex datasets well.

	ImageNette (128x128)		ImageWoof (128x128)		ImageNet (64x64)	
Img/Cls	1	10	1	10	1	2
Random Subset	23.5 ± 4.8	47.7 ± 2.4	14.2 ± 0.9	27.0 ± 1.9	1.1 ± 0.1	1.4 ± 0.1
MTT [20]	47.7 ± 0.9	63.0 ± 1.3	28.6 ± 0.8	35.8 ± 1.8	$-$	$-$
FRePo	$\mathbf{48.1 \pm 0.7}$	$\mathbf{66.5 \pm 0.8}$	$\mathbf{29.7 \pm 0.6}$	$\mathbf{42.2 \pm 0.9}$	$\mathbf{7.5 \pm 0.3}$	$\mathbf{9.7 \pm 0.2}$
Full Dataset	87.9 ± 1.0		74.4 ± 1.6		19.8 ± 0.6	

paper and our reproducing results. As for our method, Table 1, 3 report the best value of learning label or not learning label with our default architecture and data preprocessing. We use the same set of hyperparameters for all experiments. The complete results can be found in Table 7. Besides, we also include the results when training the model on the full dataset using mean square error loss in Table 5.

Table 2: We generate the distilled data for each method using their default model and default hyperparameter. For KIP, since the training is too expensive, we use the checkpoint provided by the original author. We perform a sweep on the checkpoints in "gs://kip-datasets/kip/cifar10/" and find "ConvNet_ssize100_zca_nol_noaug_ckpt1000.npz" gives the best performance that matches the reported value in the original paper. We evaluate DSA, DM, and MTT in PyTorch and KIP and FRePo in JAX. We notice that our reproducing results for KIP are much better than the reproducing results reported in MTT [20]. It may be due to the differences in initialization or hyperparameter choices, such as learning rate.

Table 4, 6: Table 4 and Table 6 present the test accuracies and MIA results on MNIST and Fashion-MNIST, respectively. We random sample 10K data points to generate the distilled data and sample another 10K non-overlapped data as non-member data. We use the same set of hyperparameters as other experiments except for data augmentation. We do not apply any data augmentation when we train models on the distilled data since any regularization can alleviate the MIA risks, making it hard to see the effects of distilled data.

B Additional Results

Resized ImageNet-1K: We also evaluate our method on a resized version of ILSVRC2012 [26] with a resolution of 64x64 to see how it performs on a complex label space. As shown in Table 5, we can achieve 7.5% and 9.7% Top1 accuracy using only 1k and 2k training examples, compared to 1.1% and 1.4% using an equally-sized real subset or 19.8% on the full dataset using 1281167 training examples. We notice that Mean Square Error (MSE) loss may not be suitable for a complex dataset like ImageNet as the same model trained with Cross-Entropy loss achieve a Top1 accuracy of 32.2%. However, since we use MSE loss to evaluate the distilled data, we report the model trained using MSE loss for a fair comparison.

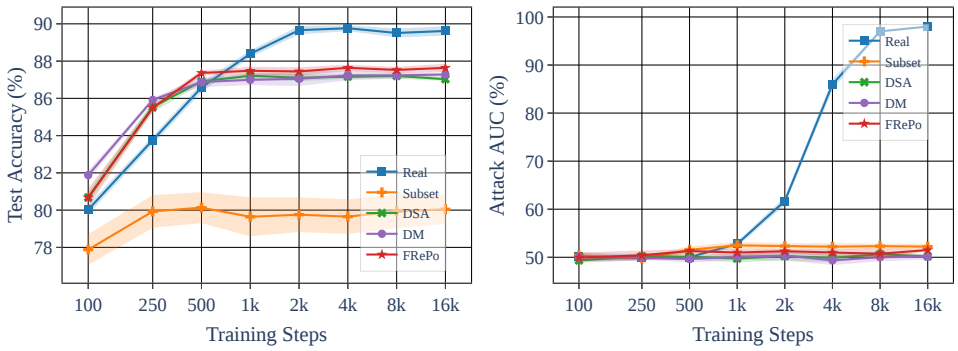
Membership Inference Defense: We only show MNIST results in Section 5.2 due to space reasons. We provide the same experimental results on FashionMNIST in Figure 6 and Table 6. We observe a similar trend on the two different datasets: training a model on the distilled data can preserve the privacy while achieving a good performance. We notice that there is a gap between training on the whole data set, which may be closed by distilling more data points and adding more noise when perform the distillation.

C Ablation Study

We conduct several ablation studies to understand the key components of the proposed method, including the role of kernel approximation (Section C.1), model pool (Section C.2), initialization (Section C.3), label learning (Section C.4), scalability (Section C.5), and model architecture C.6.

Table 6: AUC of attacker classifier trained on real data and distilled data on FashionMNSIT. We highlight the best attacker performance in bold for each model. The model trained on real data is vulnerable to membership inference attacks. In contrast, the model trained on distilled data is robust to membership inference attacks.

	Test Acc (%)	Attack AUC				
		Threshold	LR	MLP	RF	KNN
Real	89.7 ± 0.2	0.99 ± 0.01	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.98 ± 0.00
Subset	81.1 ± 0.7	0.53 ± 0.01	0.51 ± 0.01	0.52 ± 0.01	0.52 ± 0.01	0.53 ± 0.00
DSA	87.0 ± 0.1	0.51 ± 0.00	0.51 ± 0.01	0.51 ± 0.01	0.52 ± 0.01	0.51 ± 0.01
DM	87.3 ± 0.1	0.52 ± 0.00	0.51 ± 0.01	0.50 ± 0.01	0.52 ± 0.01	0.51 ± 0.01
FRePo	87.6 ± 0.2	0.52 ± 0.00	0.53 ± 0.01	0.53 ± 0.01	0.53 ± 0.01	0.52 ± 0.00



(a) MIA: ACC

(b) MIA: AUC

Figure 6: (a,b) Test accuracy and attack AUC on Fashion MNIST as we increase the number of training steps. AUC keeps increasing when training a model on the real data for more steps. In contrast, AUC keeps low when training on distilled data.

All experiments are conducted on CIFAR100 when learning 1 Img/Cls using Algorithm 1 with the default hyperparameters except stated otherwise.

C.1 FRePo vs TBPTT

Figure 7 illustrates the computation graph of FRePoand 1-step TBPTT, which is different from Figure 2 where we compare FRePo with unrolled optimization. FRePo differs from 1-step TBPTT in meta-gradient computation. As shown in Figure 7, 1-step TBPTT computes the meta-gradient by backpropagating through the inner optimization, while FRePo uses backpropagating through a kernel and a feature extractor. Figure 8 compares the training loss, training accuracy, and test accuracy of FRePo and TBPTT when using the same training and evaluation protocol. Note that FRePo computes the training loss and accuracy using the KRR head during training, while TBPTT uses the neural network head. We use the same optimizer to perform the online model update and use the same optimizer to train the same neural network for the same amount of steps on the distilled data during evaluation.

Figure 8c shows that the distilled data generated by TBPTT keeps getting worse test accuracy as the training goes on. It suggests that the meta-gradient computed by TBPTT is highly biased and does not help learn a generalizable distilled dataset. As a result, the distilled data is overfitted to a k-step learning setup, where the k is the truncation step. This learning scenario is very similar to the dataset distillation setup in DD [4] and MTT [20], where a specific optimizer is learned to take advantage of the distilled data. Though using more truncation steps can alleviate this problem, eliminating the truncation bias needs an infinite unrolled optimization, making it intractable. In contrast, FRePo alleviates the truncation bias by training a subset of a neural network to convergence. Moreover, FRePo decouples the meta-gradient computation and online model update such that the distilled data will not overfit the inner-level optimization. Conversely, TBPTT needs to fine-tune the inner optimization to get the best performance, which we do not explore further here.

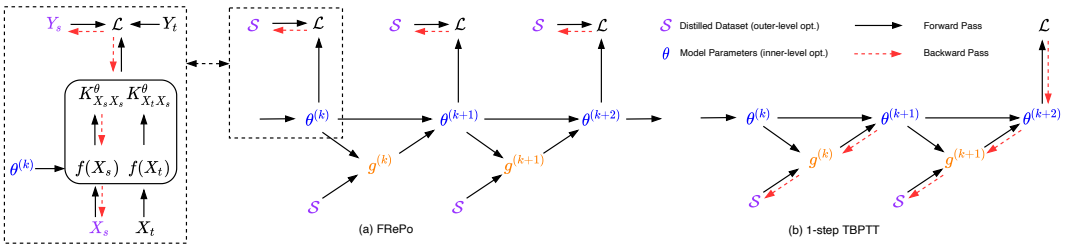
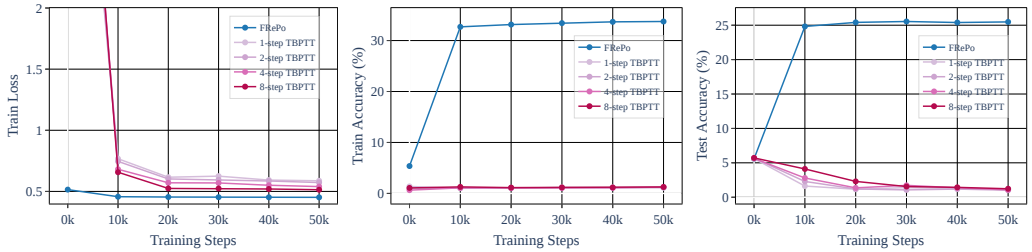


Figure 7: Comparison of FRePo and 1-step TBPTT. S, X_s, Y_s are the distilled dataset, images and labels. $\mathcal{L}$ is the meta-training loss and $\theta^{(k)}$, $g^{(k)}$ are the model parameter and gradient at step k. $f(X)$ is the feature for input X and $K^{\theta}_{X_t X_s}$ is the Gram matrix of X_t and X_s. FRePo is analogous to 1-step TBPTT as it computes the meta-gradient at each step while performing the online model update. However, instead of backpropagating through the inner optimization, FRePo computes the meta-gradient through a kernel and feature extractor.



(a) Train Loss (b) Train Acc (c) Test Acc

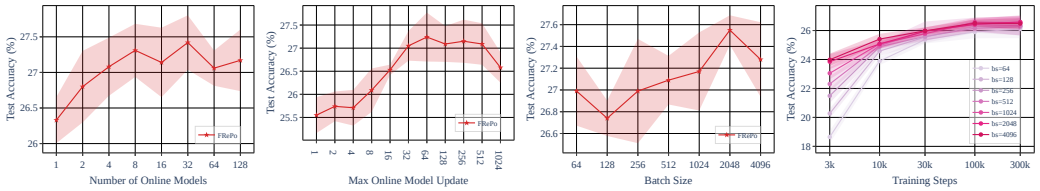
Figure 8: Ablation Study - Truncated Backpropagation through Time (TBPTT). Due to truncation bias, k-step TBPTT can easily overfit to the k-step training scheme. In contrast, FRePo alleviates the truncation bias of TBPTT by training a subset of a neural network to convergence.

C.2 Model pool and Batchsize

We investigate how the number of online models, the maximum online updates, and batch size affect the test accuracy. When one hyperparameter is modified, all the other hyperparameters are unchanged at their default value. As shown in Figure 9a, using more than one model is better than using only one model, which is equivalent to the training and resetting strategy used in the previous methods [5, 7, 13]. Our default choice of using ten models seems not to be the best but gives reasonable performance. Figure 9b shows that both using too few updates and using too many updates hurt the performance, and our default choice of 100 gives a good starting point. If we want to squeeze the performance, it is worth tuning these two hyperparameters. Intuitively, a small regularization strength is needed when we distill a small number of distilled data. On the contrary, when we distill more distilled data, we may want to use a stronger regularization. Figure 9c, 9d show that using a large batch size may give slightly better results and converge faster in terms of number of training steps. However, time per step is also an increasing function of batch size, so a large batch size may not give the best test accuracy and efficiency trade-off. Our default choice of 1024 may not be optimal, but it is a good starting point.

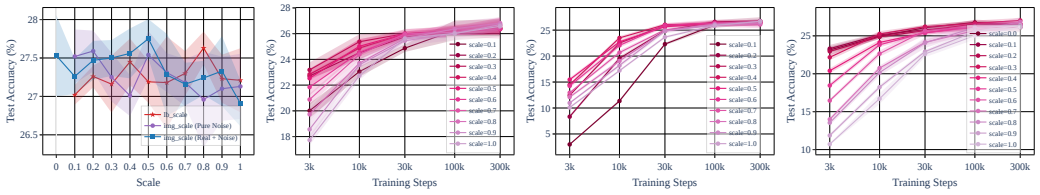
C.3 Initialization

We initialize the distilled image using real images and initialize the distilled label using a mean-centered one-hot vector scaled by $1/(\sqrt{C/10})$ as default. Is this real initialization that explains why our distilled images look real and natural? Does random initialization give a similar result? How does the label initialization affect the performance? We answer these questions by trying different initialization schemes. We investigate initializing the distilled image from random Gaussian noise or a combination of real images and random noises. We also vary the scaling factor of the label initialization and the noise scale to figure out the best setting for dataset distillation. Figure 10a shows that initializing from random noise is not a problem and the best initialization scheme is the



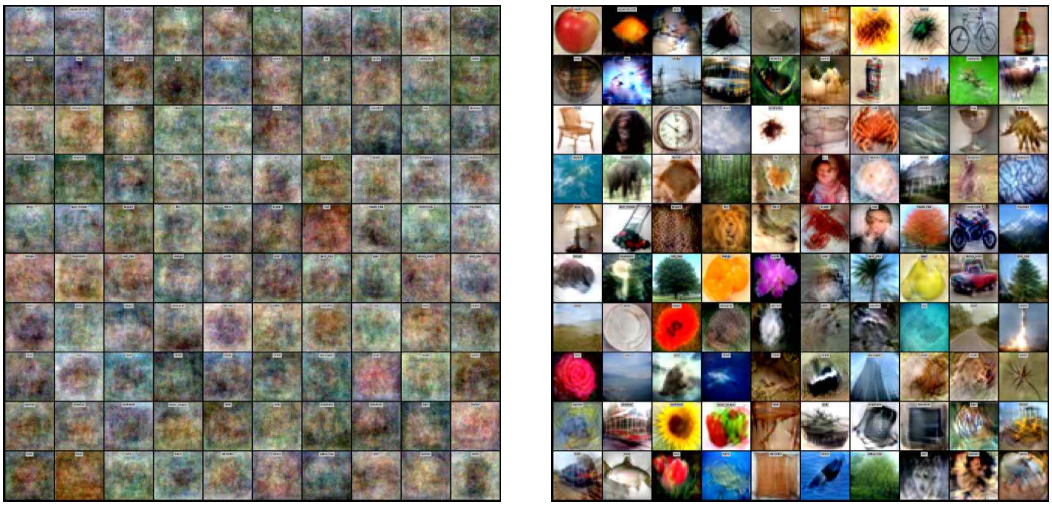
(a) Number of Models (b) Max Online Updates (c) Batch Size (d) Batch Size

Figure 9: Ablation Study - Model Pool and Batch Size. Our default value for number of online models, max online updates, and batch size may not yield the best performance for all datasets and settings. It provides a good starting point for further investigation.



(a) Overview (b) Label Scale (c) Random Image (d) Real Image

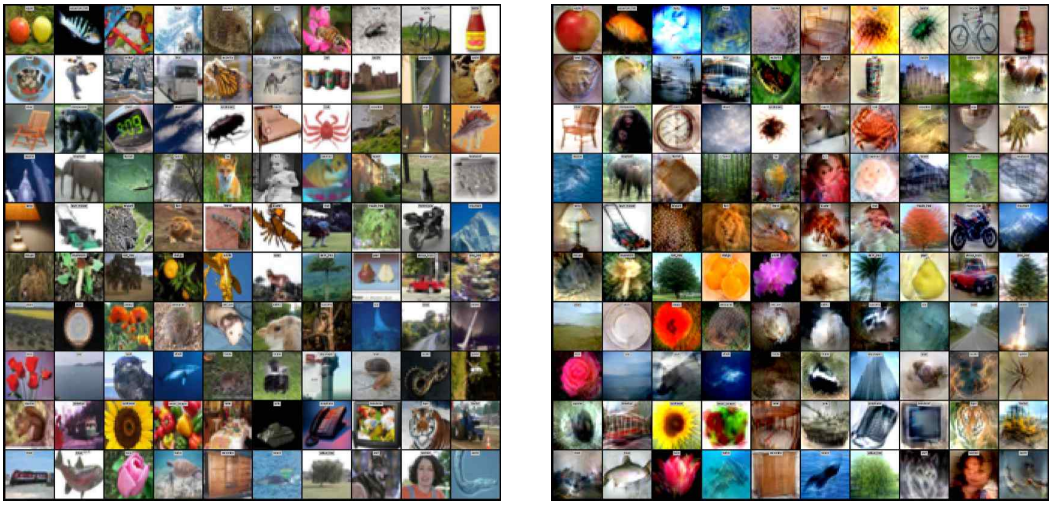
Figure 10: Ablation Study - Initialization. Initializing the distilled image using real images does not explain the effectiveness of our algorithm. Indeed, initializing using the combination of real image and a properly chosen random Gaussian noise gives the best performance. The scale of the label and random Gaussian noise is very crucial for convergence speed.



(a) Before Optimization (b) After Optimization

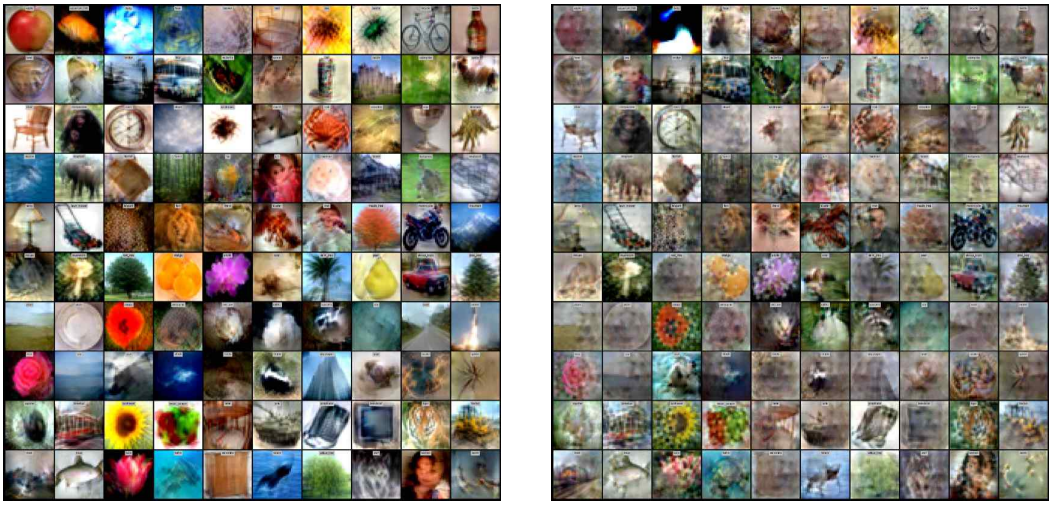
Figure 11: Ablation Study - Initialize the distilled image using random Gaussian noise.

combination of real images and random noises. With a properly chosen standard deviation of 0.5, we can achieve 27.8% test accuracy compared to 27.2% using the default hyperparameter. Figure 10b, 10c,10d show that a right scale of noise or label can significantly improve the convergence speed. Besides, Figure 10b shows that scaling the mean-centered one-hot vector by 0.3 is a good choice for CIFAR100, and we also find that 1.0 works well for datasets with ten classes. Therefore, we decide to use the mean-centered one-hot vector scaled by $1/(\sqrt{C/10})$ as our default label initialization. As shown in Figure 11, 12, though the distilled images look very different at initialization, they look quite similar after optimization. We also provide four videos to visualize the evolution of the distilled images in the supplementary.



(a) Before Optimization (b) After Optimization

Figure 12: Ablation Study - Initialize the distilled image using Real images.



(a) Learn Label=True (b) Learn Label=False

Figure 13: Ablation Study - Label Learning. Learning label can generate more natural and real-looking images.

C.4 Label Learning

As discussed in Section 4.2, label learning is an essential component of our algorithm. We provide a detailed comparison of learning and not learning labels in Table 7. We observe that when the label space is simple, such as MNIST, F-MNIST, and CIFAR10, label learning may not be necessary (Figure 21, 22, 23). However, it becomes crucial for complex datasets with many classes, such as CIFAR100 (Figure 13), Tiny-ImageNet (Figure 26), and ImageNet (Figure 27, 28). For ImageNet, we can achieve 7.5% test accuracy when we learn the label, compared to 1.6% when we fix the label. Besides the fact that the distilled label encodes rich information regarding the class similarity, we find that learning labels can make the optimization easier and distill more natural and real-looking images, as shown in Figure 13, 26 and supplementary videos.

Table 7: Test accuracies of models trained on the distilled data from scratch. We highlight the best test accuracy using neural network predictor either learn the label or not.

		Learn Label = True		Learn Label = False	
	Img/Cls	NN Acc	KRR Acc	NN Acc	KRR Acc
MNIST	1	92.5 ± 0.2	92.6 ± 0.3	$\mathbf{93.0 \pm 0.4}$	92.6 ± 0.4
	10	$\mathbf{98.6 \pm 0.1}$	98.6 ± 0.1	$\mathbf{98.6 \pm 0.1}$	98.6 ± 0.1
	50	$\mathbf{99.2 \pm 0.0}$	99.2 ± 0.1	$\mathbf{99.2 \pm 0.0}$	99.2 ± 0.0
F-MNIST	1	74.2 ± 0.5	76.4 ± 0.3	$\mathbf{75.6 \pm 0.2}$	77.1 ± 0.2
	10	$\mathbf{86.2 \pm 0.1}$	86.8 ± 0.1	86.0 ± 0.1	86.6 ± 0.1
	50	89.4 ± 0.1	89.9 ± 0.1	$\mathbf{89.6 \pm 0.1}$	89.9 ± 0.1
CIFAR10	1	45.5 ± 0.9	46.3 ± 0.7	$\mathbf{46.8 \pm 0.7}$	47.9 ± 0.6
	10	$\mathbf{65.5 \pm 0.6}$	68.0 ± 0.2	65.4 ± 0.6	66.9 ± 0.4
	50	$\mathbf{71.7 \pm 0.2}$	74.4 ± 0.1	$\mathbf{71.7 \pm 0.2}$	73.8 ± 0.2
CIFAR100	1	$\mathbf{28.7 \pm 0.1}$	32.3 ± 0.1	25.4 ± 0.1	27.3 ± 0.1
	10	$\mathbf{42.5 \pm 0.2}$	44.9 ± 0.2	39.6 ± 0.3	41.5 ± 0.1
	50	$\mathbf{44.3 \pm 0.2}$	43.0 ± 0.3	40.1 ± 0.2	37.0 ± 0.2
T-ImageNet	1	$\mathbf{15.4 \pm 0.1}$	19.1 ± 0.3	12.4 ± 0.8	15.8 ± 0.3
	10	$\mathbf{25.4 \pm 0.2}$	26.5 ± 0.1	21.9 ± 0.2	22.5 ± 0.2
CUB-200	1	$\mathbf{12.4 \pm 0.2}$	13.7 ± 0.2	7.8 ± 0.1	9.0 ± 0.3
	10	$\mathbf{16.8 \pm 0.1}$	16.1 ± 0.3	4.8 ± 0.4	1.0 ± 0.2
ImageNette	1	$\mathbf{48.1 \pm 0.7}$	50.6 ± 0.6	43.7 ± 0.9	46.8 ± 0.7
	10	$\mathbf{66.5 \pm 0.8}$	67.1 ± 0.7	64.1 ± 0.8	65.1 ± 0.8
ImageWoof	1	26.7 ± 0.6	31.3 ± 0.9	$\mathbf{29.7 \pm 0.6}$	28.2 ± 0.9
	10	$\mathbf{42.2 \pm 0.9}$	43.5 ± 0.8	41.7 ± 0.8	43.3 ± 0.7
ImageNet	1	$\mathbf{7.5 \pm 0.3}$	7.2 ± 0.2	1.6 ± 0.3	1.1 ± 0.3
	2	$\mathbf{9.7 \pm 0.2}$	9.5 ± 0.2	2.0 ± 0.3	1.7 ± 0.2

C.5 Training Cost Analysis

Similar to Figure 3, we also investigate how time per step and GPU memory usage vary when we increase the number of distilled data. As shown in Figure 14, our method becomes more expensive as we increase the number of distilled data. It is because 1) we always use all the distilled data for gradient computation rather than sample a batch as in other methods; 2) Matrix inversion with time complexity of $O(N^3)$ in KRR becomes more and more expensive as we distill more data. Similar to other kernel methods, distilling tens of thousands of data can be difficult for our method. We can circumvent this problem by 1) sampling a batch at each gradient computation, 2) performing subset distillation, or 3) distillation by class as in Section 5.1. However, the performance is expected to drop because the redundant information can be generated in different groups, and the distilled data may not be able to capture all the distinguishable features when learning from a subset. We leave it for future work to address this scaling challenge. We provide the numerical values for Figure 3c, 3d, 14a, and 14b in Table 8, 9, 10, and 11.

Table 8: Time per step measure in milliseconds (ms). Corresponds to Figure 3c.

Width	DSA	DM	MTT	FRePo
16	1764.3 ± 71.2	569.1 ± 13.2	206.2 ± 2.5	7.7 ± 0.3
32	1742.2 ± 17.1	573.5 ± 11.2	268.6 ± 2.0	6.9 ± 0.2
64	2014.6 ± 15.0	669.8 ± 10.1	299.9 ± 2.1	8.5 ± 0.2
128	2583.5 ± 14.7	950.6 ± 13.7	485.6 ± 3.0	10.9 ± 0.1
256	4909.8 ± 16.9	1569.2 ± 8.5	939.6 ± 10.5	20.8 ± 0.1
512	8764.6 ± 13.7	3209.5 ± 8.6	2153.6 ± 9.0	52.5 ± 0.1

Table 9: Peak GPU memory usage measured in gigabytes (GB). Corresponds to Figure 3d.

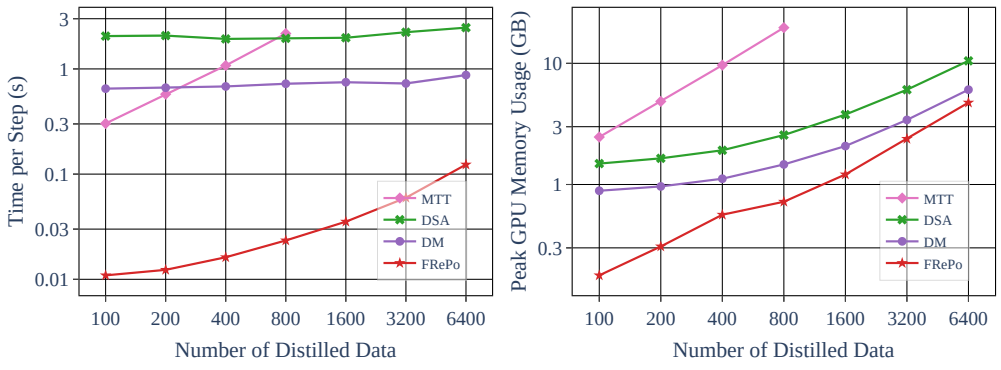
Width	DSA	DM	MTT	FRePo
16	0.800	0.678	0.650	0.036
32	0.986	0.714	1.260	0.086
64	1.488	0.888	2.462	0.178
128	2.014	1.164	4.858	0.387
256	4.594	1.718	9.790	0.814
512	10.768	2.970	19.110	2.080

Table 10: Time per step measure in millisecond (ms). Corresponds to Figure 14a.

Number of Distilled Data	DSA	DM	MTT	FRePo
100	2051.8 $\pm$ 52.2	649.4 $\pm$ 11.7	302.8 $\pm$ 1.8	10.9 $\pm$ 0.1
200	2073.4 $\pm$ 55.4	662.9 $\pm$ 15.7	575.0 $\pm$ 7.3	9.6 $\pm$ 0.1
400	1928.6 $\pm$ 13.6	681.4 $\pm$ 10.7	1077.5 $\pm$ 8.8	16.2 $\pm$ 0.2
800	1952.5 $\pm$ 12.3	722.7 $\pm$ 18.2	2169.1 $\pm$ 12.5	23.4 $\pm$ 0.2
1600	1977.4 $\pm$ 16.4	747.2 $\pm$ 13.7	-	35.3 $\pm$ 0.2
3200	2233.8 $\pm$ 8.8	727.3 $\pm$ 18.8	-	59.6 $\pm$ 0.1
6400	2467.8 $\pm$ 8.8	874.2 $\pm$ 25.7	-	123.2 $\pm$ 0.1

Table 11: Peak GPU memory usage measured in gigabytes (GB). Corresponds to Figure 14b.

Number of Distilled Data	DSA	DM	MTT	FRePo
100	1.488	0.888	2.464	0.178
200	1.644	0.964	4.848	0.308
400	1.916	1.116	9.662	0.563
800	2.558	1.464	19.700	0.721
1600	3.776	2.076	-	1.210
3200	6.056	3.412	-	2.380
6400	10.482	6.060	-	4.740



(a) Time Per Step

(b) GPU Memory Usage

Figure 14: Time per iteration and peak memory usage as we increase the number of distilled data.

C.6 Model Architectures

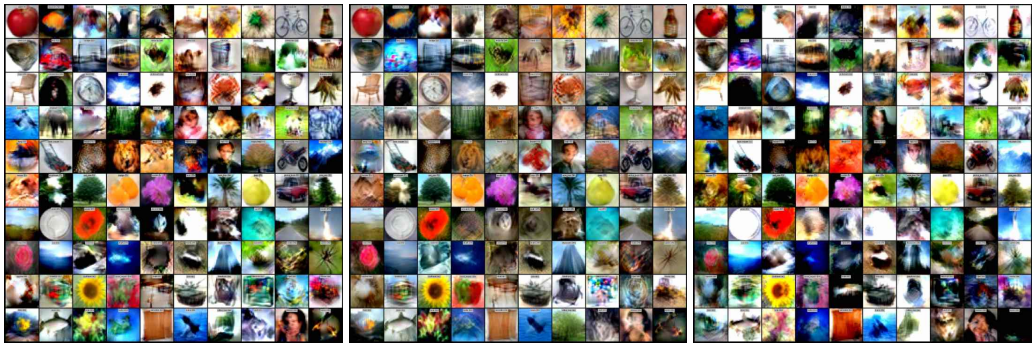
From the Cross-Architecture Generalization experiments in Section 4.2, we observe that the distilled data can encode the architecture's inductive bias. Thus, we perform a qualitative and quantitative comparison of the distilled data generated by different architectures to understand how architecture affects the distilled data. We evaluate across various architectures, including Conv (Our default model), DCConv (the default model of DSA [7], DM [8], MTT [20]), AlexNet [38], VGG [39], and ResNet [40]. We also consider a wide range range of normalization layers, such as no normalization (NN), Instance Normalization (IN) [41], Batch Normalization (BN) [42], Layer Normalization (LN) [68], and Group Normalization (GN) [69]. Besides, we also vary the depth of Conv and denote two-layer, three-layer, and four-layer Conv as Conv-BN-D2, Conv-BN-D3, and Conv-BN-D4, respectively. We do not rescale any image for better visualization, so the over-saturated images indicate that the images have different statistics from the real images.

Qualitative Results: Figure 15, 16, 17, 18, and 19 show that 1) the simplest architecture (i.e., Conv) gives the images that look almost like real images; 2) Different normalization layers have different effects on the distilled images, resulting in images with very different brightness and contrasts. No Normalization or Batch Normalization seems to generate the most natural-looking images; 3) The distilled images generated by modern architectures like ResNet and VGG are very different from the natural images. Thus additional attention is needed to transfer the distilled images to a different architecture. On the other hand, additional tricks such as image regularization or projection may help make the distilled images more similar to natural ones. 4) Number of average pooling layers (or size of the final activation map) can affect the distilled image quality. Using a fewer pooling (i.e., Conv-BN-D2) will generate blur images or repeated objects in the images, and we find it is more obvious in high resolution such as Tiny-ImageNet. 5) Distilled images may reflect the similarity of the architecture. For instance, the images generated by Conv and DCConv are almost identical since the only architectural difference is the filter width. Besides, we observe that the images generated by AlexNet are very similar to those generated by Conv, which suggests that the inductive bias of those two architectures is very similar. It may be one reason why Conv's distilled images work extremely well for AlexNet (Table 2).

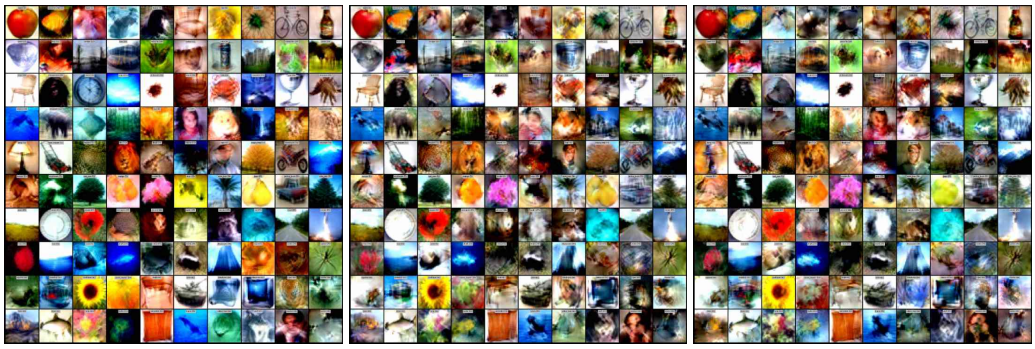
(a) Conv-NN (b) Conv-BN (c) Conv-LN

(d) Conv-IN (e) Conv-GN (GroupNum=32) (f) Conv-GN (GroupNum=1)

Figure 15: Ablation Study - Default Model with different Normalization Layer

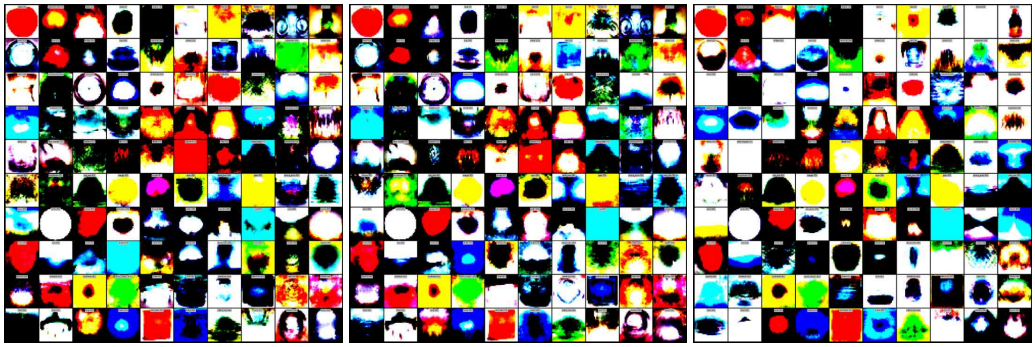


(a) DCConv-NN (b) DCConv-BN (c) DCConv-LN

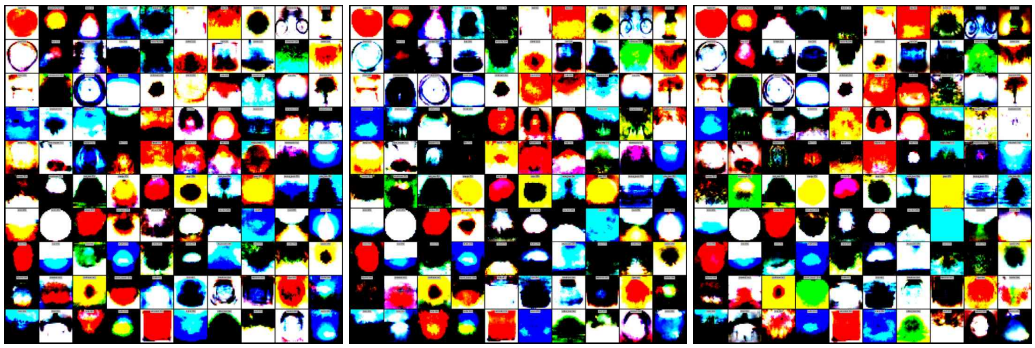


(d) DCConv-IN (e) DCConv-GN (GroupNum=32) (f) DCConv-GN (GroupNum=1)

Figure 16: Ablation Study - DCConv with different Normalization Layer

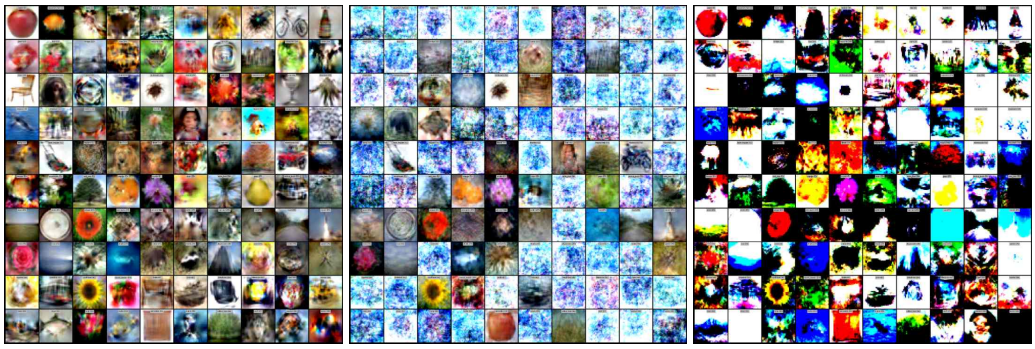


(a) ResNet-NN (b) ResNet-BN (c) ResNet-LN

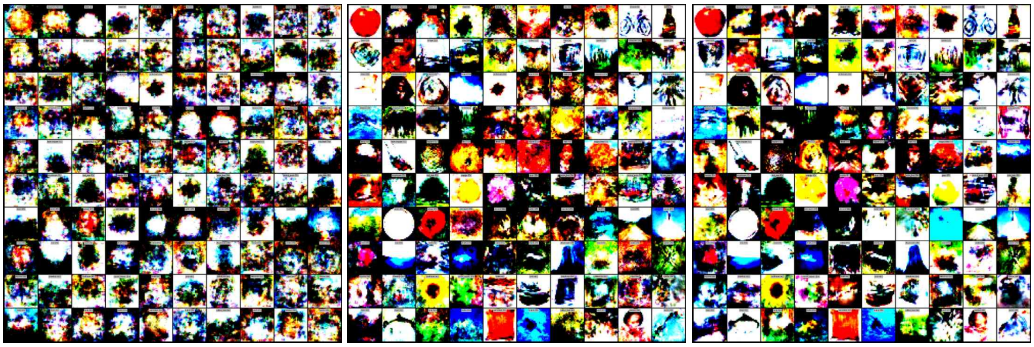


(d) ResNet-IN (e) ResNet-GN (GroupNum=32) (f) ResNet-GN (GroupNum=1)

Figure 17: Ablation Study - ResNet with different Normalization Layer

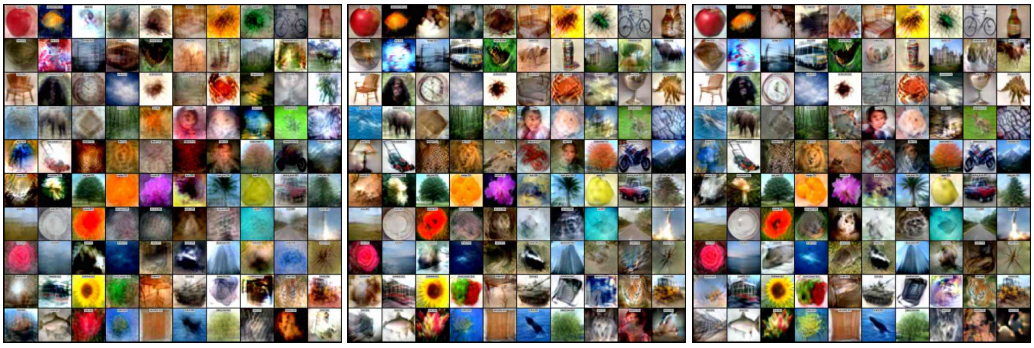


(a) VGG-NN (b) VGG-BN (c) VGG-LN



(d) VGG-IN (e) VGG-GN (GroupNum=32) (f) VGG-GN (GroupNum=1)

Figure 18: Ablation Study - VGG with different Normalization Layer.



(a) Conv-BN-D2 (b) Conv-BN-D3 (c) Conv-BN-D4

Figure 19: Ablation Study - Conv with different Depth.

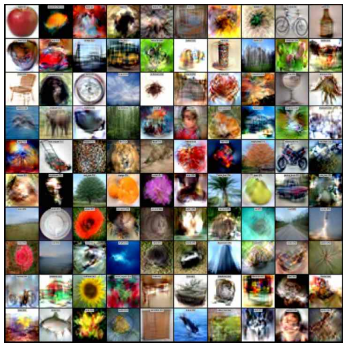


Figure 20: Ablation Study - AlexNet.

Quantitative Results: We evaluate the distilled data trained using different architectures (Figure 15 and 16) to study the role of normalization and width in training and evaluation. Table 12, 13, and 14 show that 1) Training with Conv-BN and evaluating using Conv-NN yields the best performance, which is our default choice; 2) Training with Conv-BN yields most generalizable and transferable images as it performs well for other architectures; 3) Evaluating using Conv-GN seems to be the best choice if the training architecture is unknown. 4) Evaluating using Conv-NN is a good way to see whether the inductive bias of architecture has been distilled to the dataset since it is very sensitive to the training architecture. There, it is not a good idea to use this architecture when the training architecture is unknown; 5) Training the distilled data using a wider network Conv achieves slightly better performance than a narrower network DCConv.

We also perform the same experiments in Table 1 using Conv-IN, and DCConv-IN and summarize the results in Table 16 and 17. Moreover, we also evaluate the cross-architecture transfer performance of Conv-IN in Table 15. We observe that DCConv-IN works reasonably well when we distill a small number of images (100). The performance degrades a lot when distilling 1000 images from CIFAR100 because the KRR component needs a larger feature dimension to perform well. Besides, Conv-IN performs slightly worse than the default Conv-BN. However, Table 15 suggests that the drawback of the instance norm is the transferability. The distilled data trained using instance normalization transfer less well to other architecture, especially those without normalization.

Table 12: Test accuracies of models trained on the distilled data from scratch. Each row has the same training architecture, while each column has the same evaluation architecture. We report the average performance of four random seeds. C denotes Conv and D denotes DCConv. The standard deviation are shown in Table 13, and 14. We visualize the distilled images in Figure 15 and 16.

T\E	C-NN	C-BN	C-GN	C-GN1	C-LN	C-IN	D-NN	D-BN	D-GN	D-GN1	D-LN	D-IN
C-NN	25.7	22.5	26.5	26.2	25.3	23.6	19.1	21.5	23.8	24.3	22.0	22.2
C-BN	28.8	26.0	28.1	27.6	27.4	27.0	22.7	24.6	24.3	25.4	23.4	24.1
C-GN	26.1	24.1	28.4	27.4	26.0	25.6	20.0	23.8	25.9	26.3	24.1	24.7
C-GN1	25.9	23.3	27.3	26.8	25.8	24.6	19.8	23.5	25.3	25.8	23.4	24.1
C-LN	21.7	20.6	23.1	23.2	23.8	21.0	14.8	20.4	22.0	22.8	23.5	20.4
C-IN	25.1	22.8	26.1	25.7	25.2	27.0	20.6	21.5	24.6	24.2	23.1	25.7
D-NN	25.1	22.4	27.3	27.3	26.3	22.0	17.2	21.7	25.6	26.3	24.1	23.8
D-BN	27.4	23.9	27.8	27.4	26.7	25.9	19.8	22.1	24.9	25.4	23.2	24.0
D-GN	24.8	21.1	26.4	27.0	25.3	22.1	15.7	21.1	25.5	25.6	22.3	23.1
D-GN1	24.8	21.5	26.4	27.2	25.7	21.5	16.3	21.4	24.4	25.7	22.9	22.0
D-LN	24.0	21.7	24.6	25.1	25.5	20.7	13.4	21.3	21.6	23.1	25.3	19.8
D-IN	23.3	20.2	24.2	24.0	23.0	23.4	16.9	20.1	22.8	22.5	21.0	24.8

Table 13: Test accuracies of Conv with different normalizations evaluated on the distilled data trained using Conv and DCConv with different normalizations. We report the mean and standard deviation of four random seeds. The distilled images are shown in Figure 15 and 16.

T\E	Conv-NN	Conv-BN	Conv-GN	Conv-GN1	Conv-LN	Conv-IN
Conv-NN	25.7 ± 0.3	22.5 ± 0.4	26.5 ± 0.2	26.2 ± 0.1	25.3 ± 0.2	23.6 ± 0.2
Conv-BN	28.8 ± 0.2	26.0 ± 0.3	28.1 ± 0.5	27.6 ± 0.3	27.4 ± 0.1	27.0 ± 0.3
Conv-GN	26.1 ± 0.3	24.1 ± 0.1	28.4 ± 0.3	27.4 ± 0.1	26.0 ± 0.1	25.6 ± 0.3
Conv-GN1	25.9 ± 0.1	23.3 ± 0.1	27.3 ± 0.3	26.8 ± 0.3	25.8 ± 0.1	24.6 ± 0.2
Conv-LN	21.7 ± 0.2	20.6 ± 0.1	23.1 ± 0.1	23.2 ± 0.3	23.8 ± 0.3	21.0 ± 0.1
Conv-IN	25.1 ± 0.3	22.8 ± 0.4	26.1 ± 0.4	25.7 ± 0.2	25.2 ± 0.3	27.0 ± 0.3
DCConv-NN	25.1 ± 0.2	22.4 ± 0.2	27.3 ± 0.3	27.3 ± 0.5	26.3 ± 0.2	22.0 ± 0.2
DCConv-BN	$27.4 + 0.2$	23.9 ± 0.1	27.8 ± 0.4	27.4 ± 0.3	26.7 ± 0.2	25.9 ± 0.3
DCConv-GN	24.8 ± 0.2	21.1 ± 0.1	26.4 ± 0.3	27.0 ± 0.3	25.3 ± 0.2	22.1 ± 0.3
DCConv-GN1	24.8 ± 0.4	21.5 ± 0.2	26.4 ± 0.1	27.2 ± 0.1	25.7 ± 0.4	21.5 ± 0.3
DCConv-LN	24.0 ± 0.1	21.7 ± 0.3	24.6 ± 0.4	25.1 ± 0.1	25.5 ± 0.2	20.7 ± 0.2
DCConv-IN	23.3 ± 0.2	20.2 ± 0.3	24.2 ± 0.3	24.0 ± 0.2	23.0 ± 0.4	23.4 ± 0.3

Table 14: Test accuracies of DCConv with different normalizations evaluated on the distilled data trained using Conv and DCConv with different normalizations. We report the mean and standard deviation of four random seeds. The distilled images are shown in Figure 15 and 16.

T\E	DCConv-NN	DCConv-BN	DCConv-GN	DCConv-GN1	DCConv-LN	DCConv-IN
Conv-NN	19.1 ± 0.1	21.5 ± 0.2	23.8 ± 0.1	24.3 ± 0.3	22.0 ± 0.2	22.2 ± 0.1
Conv-BN	22.7 ± 0.1	24.6 ± 0.1	24.3 ± 0.2	25.4 ± 0.2	23.4 ± 0.1	24.1 ± 0.2
Conv-GN	20.0 ± 0.7	23.8 ± 0.3	25.9 ± 0.1	26.3 ± 0.1	24.1 ± 0.2	24.7 ± 0.2
Conv-GN1	19.8 ± 0.7	23.5 ± 0.4	25.3 ± 0.1	25.8 ± 0.2	23.4 ± 0.1	24.1 ± 0.1
Conv-LN	14.8 ± 0.8	20.4 ± 0.3	22.0 ± 0.3	22.8 ± 0.3	23.5 ± 0.4	20.4 ± 0.2
Conv-IN	20.6 ± 0.4	21.5 ± 0.2	24.6 ± 0.1	24.2 ± 0.2	23.1 ± 0.2	25.7 ± 0.2
DCConv-NN	17.2 ± 0.9	21.7 ± 0.1	25.6 ± 0.2	26.3 ± 0.3	24.1 ± 0.1	23.8 ± 0.2
DCConv-BN	19.8 ± 0.5	22.1 ± 0.3	24.9 ± 0.3	25.4 ± 0.2	23.2 ± 0.2	24.0 ± 0.3
DCConv-GN	15.7 ± 0.7	21.1 ± 0.2	25.5 ± 0.3	25.6 ± 0.3	22.3 ± 0.2	23.1 ± 0.1
DCConv-GN1	16.3 ± 0.8	21.4 ± 0.2	24.4 ± 0.2	25.7 ± 0.4	22.9 ± 0.3	22.0 ± 0.2
DCConv-LN	13.4 ± 1.1	21.3 ± 0.4	21.6 ± 0.5	23.1 ± 0.3	25.3 ± 0.3	19.8 ± 0.3
DCConv-IN	16.9 ± 0.8	20.1 ± 0.3	22.8 ± 0.1	22.5 ± 0.2	21.0 ± 0.4	24.8 ± 0.3

Table 15: Cross-architecture transfer performance on CIFAR10 with 10 Img/Cls. Despite being trained for a specific architecture, our distilled data transfer well to various architectures unseen during training. Conv is the default evaluation model used for each method. NN, DN, IN, and BN stand for no normalization, default normalization, Instance Normalization, Batch Normalization, respectively.

	Train Arch	Evaluation Architecture						
		Conv	Conv-NN	ResNet-DN	ResNet-BN	VGG-DN	VGG-BN	AlexNet
DSA [7]	Conv-IN	53.2 ± 0.8	36.4 ± 1.5	42.1 ± 0.7	34.1 ± 1.4	48.3 ± 0.7	46.3 ± 1.3	34.0 ± 2.3
DM [8]	Conv-IN	49.2 ± 0.8	35.2 ± 0.5	36.8 ± 1.2	35.5 ± 1.3	45.5 ± 1.0	41.2 ± 1.8	34.9 ± 1.1
MTT [20]	Conv-IN	64.4 ± 0.9	41.6 ± 1.3	49.2 ± 1.1	42.9 ± 1.5	35.7 ± 3.35	46.6 ± 2.0	34.2 ± 2.6
KIP [23]	Conv-NTK	62.7 ± 0.3	58.2 ± 0.4	49.0 ± 1.2	45.8 ± 1.4	32.0 ± 0.4	30.1 ± 1.5	57.2 ± 0.4
FRePo	Conv-IN	59.2 ± 0.3	56.2 ± 0.2	51.1 ± 0.8	50.8 ± 0.2	**57.5 ± 0.7**	51.8 ± 0.3	55.3 ± 0.8
FRePo	Conv-BN	**65.5 ± 0.4**	**65.5 ± 0.4**	**58.1 ± 0.6**	**57.7 ± 0.7**	49.1 ± 0.5	**59.4 ± 0.7**	**61.9 ± 0.7**

Table 16: NN test accuracies of models trained on the distilled data from scratch using Conv, Conv-IN, and DCConv-IN. † denotes performance better than the original reported performance.

	Img/Cls	Previous SOTA				FRePo		
		DSA [7]	DM [8]	KIP [23]	MTT [20]	Conv-BN	Conv-IN	DCConv-IN
MNIST	1	88.7 ± 0.6	$89.9 \pm 0.8^\dagger$	90.1 ± 0.1	$91.4 \pm 0.9^\dagger$	**93.0 ± 0.4**	92.9 ± 0.5	92.4 ± 0.5
	10	$97.9 \pm 0.1^\dagger$	$97.6 \pm 0.1^\dagger$	97.5 ± 0.0	$97.3 \pm 0.1^\dagger$	**98.6 ± 0.1**	98.9 ± 0.1	98.4 ± 0.1
	50	99.2 ± 0.1	98.6 ± 0.1	98.3 ± 0.1	$98.5 \pm 0.1^\dagger$	99.2 ± 0.0	**99.4 ± 0.1**	98.8 ± 0.1
F-MNIST	1	70.6 ± 0.6	$71.5 \pm 0.5^\dagger$	73.5 ± 0.5	$75.1 \pm 0.9^\dagger$	75.6 ± 0.3	75.3 ± 0.8	**76.4 ± 1.2**
	10	$84.8 \pm 0.3^\dagger$	$83.6 \pm 0.2^\dagger$	86.8 ± 0.1	**$87.2 \pm 0.3^\dagger$**	86.2 ± 0.2	86.0 ± 0.3	85.7 ± 0.2
	50	$88.8 \pm 0.2^\dagger$	$88.2 \pm 0.1^\dagger$	88.0 ± 0.1	$88.3 \pm 0.1^\dagger$	**89.6 ± 0.1**	89.4 ± 0.1	87.2 ± 0.2
CIFAR10	1	$36.7 \pm 0.8^\dagger$	$31.0 \pm 0.6^\dagger$	**49.9 ± 0.2**	46.3 ± 0.8	46.8 ± 0.7	45.1 ± 0.5	41.3 ± 0.5
	10	$53.2 \pm 0.8^\dagger$	$49.2 \pm 0.8^\dagger$	62.7 ± 0.3	65.3 ± 0.7	**65.5 ± 0.4**	59.1 ± 0.3	59.6 ± 0.3
	50	$66.8 \pm 0.4^\dagger$	$63.7 \pm 0.5^\dagger$	68.6 ± 0.2	71.6 ± 0.2	**71.7 ± 0.2**	69.6 ± 0.4	63.6 ± 0.2
CIFAR100	1	$16.8 \pm 0.2^\dagger$	$12.2 \pm 0.4^\dagger$	15.7 ± 0.2	24.3 ± 0.3	**28.7 ± 0.1**	25.9 ± 0.1	24.8 ± 0.2
	10	32.3 ± 0.3	29.7 ± 0.3	28.3 ± 0.1	40.1 ± 0.4	**42.5 ± 0.2**	40.9 ± 0.1	31.2 ± 0.1
	50	42.8 ± 0.4	43.6 ± 0.4	–	**47.7 ± 0.2**	44.3 ± 0.2	–	–
T-ImageNet	1	$6.6 \pm 0.2^\dagger$	3.9 ± 0.2	–	8.8 ± 0.3	**15.4 ± 0.1**	13.5 ± 0.1	–
	10	–	12.9 ± 0.4	–	23.2 ± 0.2	**25.4 ± 0.2**	20.4 ± 0.1	–

Table 17: KRR test accuracies of FRePo trained on the distilled data from scratch using Conv, Conv-IN, and DCConv-IN. † denotes performance better than the original reported performance.

	Img/Cls	Previous SOTA				FRePo		
		DSA [7]	DM [8]	KIP [23]	MTT [20]	Conv-BN	Conv-IN	DCConv-IN
MNIST	1	88.7 ± 0.6	$89.9 \pm 0.8^{\dagger}$	90.1 ± 0.1	$91.4 \pm 0.9^{\dagger}$	92.6 ± 0.4	**92.7 ± 0.3**	91.1 ± 0.5
	10	$97.9 \pm 0.1^{\dagger}$	$97.6 \pm 0.1^{\dagger}$	97.5 ± 0.0	$97.3 \pm 0.1^{\dagger}$	98.6 ± 0.1	**98.8 ± 0.1**	98.4 ± 0.1
	50	**99.2 ± 0.1**	98.6 ± 0.1	98.3 ± 0.1	$98.5 \pm 0.1^{\dagger}$	99.2 ± 0.1	**99.3 ± 0.1**	98.9 ± 0.1
F-MNIST	1	70.6 ± 0.6	$71.5 \pm 0.5^{\dagger}$	73.5 ± 0.5	$75.1 \pm 0.9^{\dagger}$	77.1 ± 0.2	71.7 ± 1.2	**78.5 ± 0.2**
	10	$84.8 \pm 0.3^{\dagger}$	$83.6 \pm 0.2^{\dagger}$	86.8 ± 0.1	**$87.2 \pm 0.3^{\dagger}$**	86.8 ± 0.1	86.9 ± 0.2	86.2 ± 0.1
	50	$88.8 \pm 0.2^{\dagger}$	$88.2 \pm 0.1^{\dagger}$	88.0 ± 0.1	$88.3 \pm 0.1^{\dagger}$	**89.9 ± 0.1**	**89.9 ± 0.1**	87.4 ± 0.2
CIFAR10	1	$36.7 \pm 0.8^{\dagger}$	$31.0 \pm 0.6^{\dagger}$	**49.9 ± 0.2**	46.3 ± 0.8	47.9 ± 0.6	46.8 ± 0.3	43.3 ± 0.5
	10	$53.2 \pm 0.8^{\dagger}$	$49.2 \pm 0.8^{\dagger}$	62.7 ± 0.3	65.3 ± 0.7	**68.0 ± 0.2**	61.9 ± 0.4	61.8 ± 0.3
	50	$66.8 \pm 0.4^{\dagger}$	$63.7 \pm 0.5^{\dagger}$	68.6 ± 0.2	71.6 ± 0.2	**74.4 ± 0.1**	71.4 ± 0.3	64.3 ± 0.1
CIFAR100	1	$16.8 \pm 0.2^{\dagger}$	$12.2 \pm 0.4^{\dagger}$	15.7 ± 0.2	24.3 ± 0.3	**32.3 ± 0.1**	25.7 ± 0.2	26.9 ± 0.1
	10	32.3 ± 0.3	29.7 ± 0.3	28.3 ± 0.1	40.1 ± 0.4	**44.9 ± 0.2**	42.0 ± 0.3	30.9 ± 0.1
	50	42.8 ± 0.4	43.6 ± 0.4	–	**47.7 ± 0.2**	43.0 ± 0.3	–	–
T-ImageNet	1	$6.6 \pm 0.2^{\dagger}$	3.9 ± 0.2	–	8.8 ± 0.3	**19.1 ± 0.3**	15.8 ± 0.3	–
	10	–	12.9 ± 0.4	–	23.2 ± 0.2	**26.5 ± 0.1**	20.8 ± 0.1	–

D Hyperparameter Tuning Guideline

We find that several modifications to the current method can improve the test accuracy of the model trained on the distilled data. We do not include them in our current algorithm for simplicity or fair comparison, and we guess it may be helpful for practitioners.

- **Dropout:** We find dropout is very effective at alleviating the overfitting when training on a small set of examples.
- **Learning Rate Schedule for the online model:** Though we use a constant schedule in our experiments, we find that the learning rate schedule, especially the warm-up phase, may be crucial for some architectures. When NAN is in gradient, adding a learning rate schedule to the online model may solve the problem.
- **Data Augmentation during Training:** There are two ways to add data augmentation during training. One is to add to X_s during the online model update. The other is to add to X_t, which can be thought of as distilling data augmentation to the data.
- **Tune Max Online Update:** We train the online model up to 100 steps in our experiments. When the distillation size is tiny (e.g., ten examples in total), 100 may be too large. Setting to a lower value (less regularization) turns out to be better. On the contrary, if the distillation size is large, setting it to a higher value (more regularization) can give better results.
- **Exponential Moving Average (EMA):** We find that evaluating on the EMA version of the distilled data or using the EMA version of the model parameters can improve the test accuracy.
- **Soft Cross Entropy Loss:** We train models on the distilled dataset using MSE loss in all our experiments to take advantage of the distilled label. An alternative way is to use the soft cross-entropy loss with a fine-tuned temperature, which usually outperforms the MSE loss.
- **Image Regularization:** Though our method does not have any image regularization, we find a regularization term on the image norm is necessary for some architectures. Otherwise, the image norm will keep increasing or decreasing. We find that using an L2 penalty between the distilled image norm and the real image norm is enough in some cases. An alternative way is to project the distilled image to a norm ball every few iterations.
- **Label Regularization:** Our method has a small regularization to ensure the class-balanced distillation where we force the margin between the target label and any other label is greater than 1/C. We believe a better label regularization incorporating prior knowledge of class similarity can improve the performance.
- **Maximal Update Parametrization (μP) [70]:** We observe that using a model parameterization proposed by Yang et al. [70] can give additional test accuracy improvement.

Based on our experience, we provide the following hyperparameter tuning guideline for those who want to squeeze the performance of our method or apply our method to a different dataset or use a different model. Besides validation loss and accuracy, we suggest monitoring the norm and gradient norm of the distilled images and labels, which can be good indicators for the final performance.

- **Step1 - Online Model:** Choose an online model architecture and tune its hyperparameter (e.g., optimizer, weight decay) in the standard way on the whole real dataset or a subset of real data. The same hyperparameters can be used for the online model update and final evaluation.
- **Step2 - Distilled Data Optimization and Initialization:** Use the default setting for the model pool (i.e., ten models with max online update $K = 100$) and tune the learning rate and batch size for distilled data optimization and the scale of initialization.
- **Step3 - Pool Diversity:** Tune the model pool diversity by adjusting the max online update, adding models with different architectures, or applying data augmentation.

E Additional Visualization

E.1 Distilled Image Visualization

We provide some additional distilled images visualization for MNIST (Figure 21), FashionMNIST (Figure 22), CIFAR10 (Figure 23), CIFAR100 (Figure 24), CUB-200 (Figure 25), Tiny ImageNet (Figure 26), ImageNet (Figure 27, 28), ImageNette (Figure 29), and ImageWoof (Figure 30).

(a) 1 Img/Cls, Learn Label=True

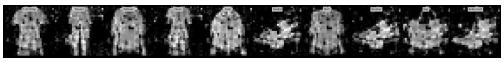
(b) 1 Img/Cls, Learn Label=False

(c) 10 Img/Cls, Learn Label=True

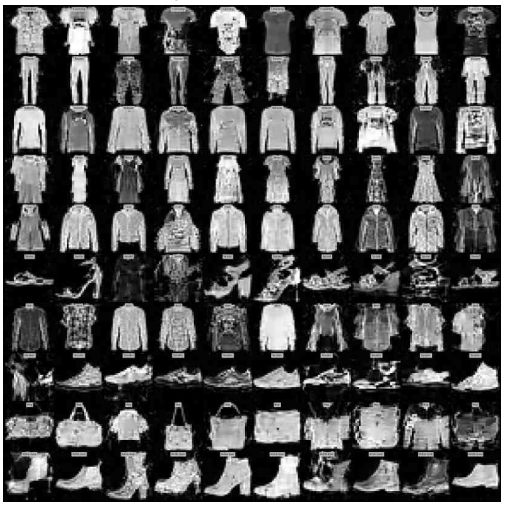
(d) 10 Img/Cls, Learn Label=False

Figure 21: Distilled Image Visualization - MNIST.

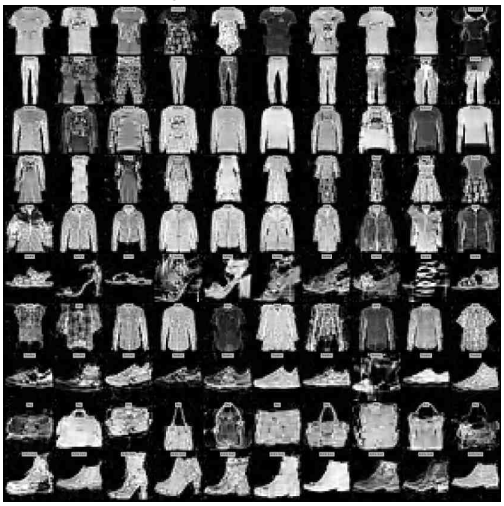
(a) 1 Img/Cls, Learn Label=True



(b) 1 Img/Cls, Learn Label=False



(c) 10 Img/Cls, Learn Label=True

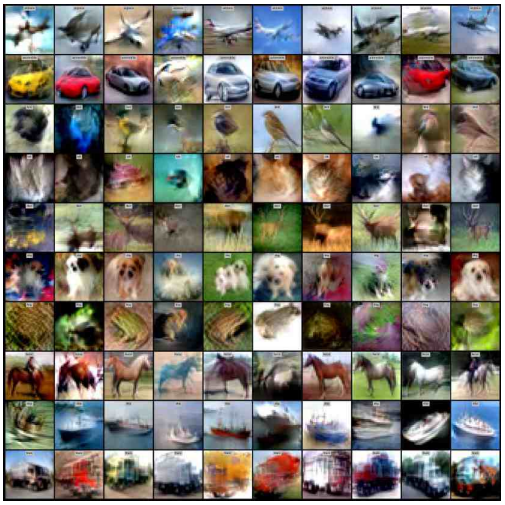


(d) 10 Img/Cls, Learn Label=False

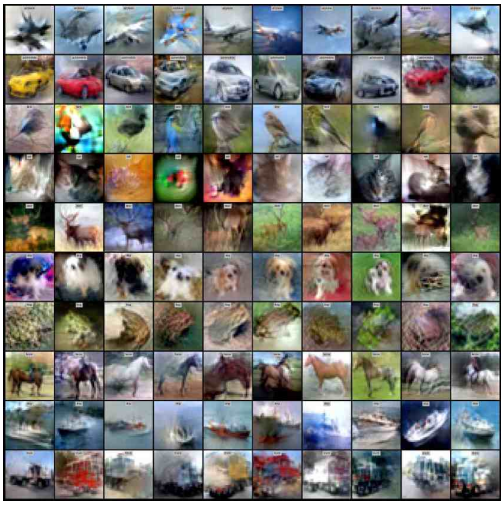
Figure 22: Distilled Image Visualization - Fashion MNIST.

(a) 1 Img/Cls, Learn Label=True

(b) 1 Img/Cls, Learn Label=False



(c) 10 Img/Cls, Learn Label=True



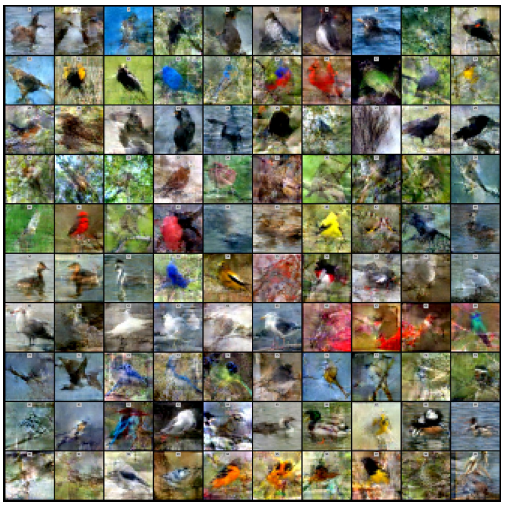
(d) 10 Img/Cls, Learn Label=False

Figure 23: Distilled Image Visualization - CIFAR10

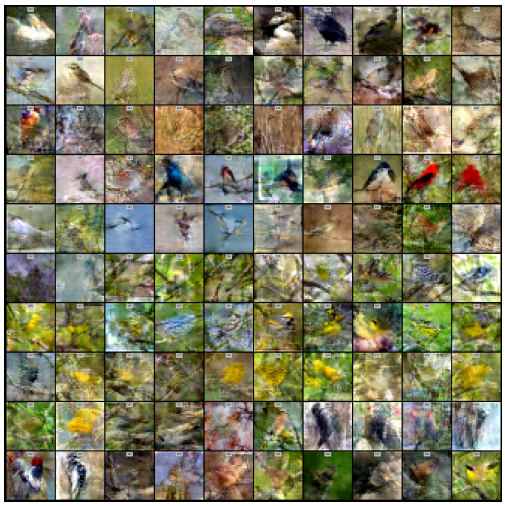
(a) 1 Img/Cls, Learn Label=True

(b) 1 Img/Cls, Learn Label=False

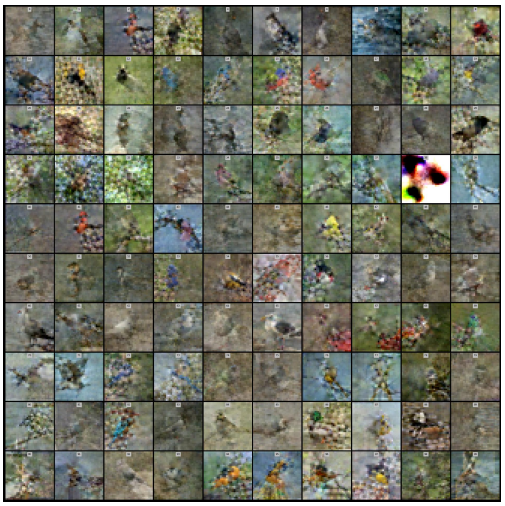
Figure 24: Distilled Image Visualization - CIFAR100



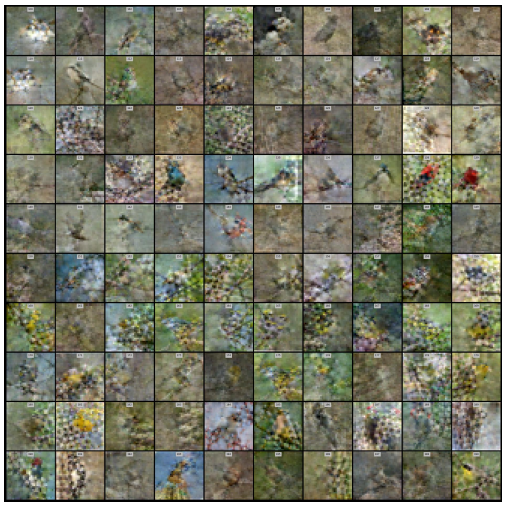
(a) Learn Label=True, Class ID = [0-99]



(b) Learn Label=True, Class ID = [100-199]

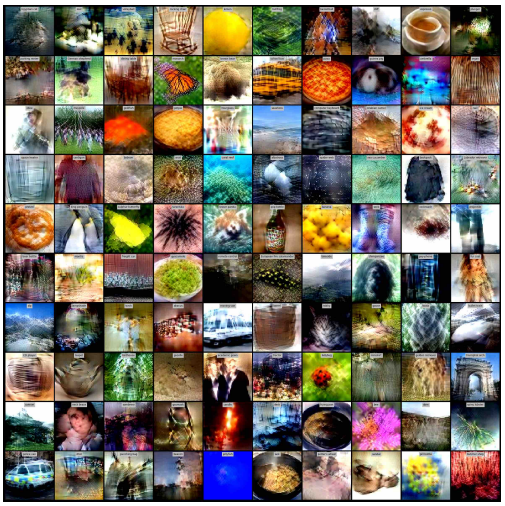


(c) Learn Label=False, Class ID = [0-99]



(d) Learn Label=False, Class ID = [100-199]

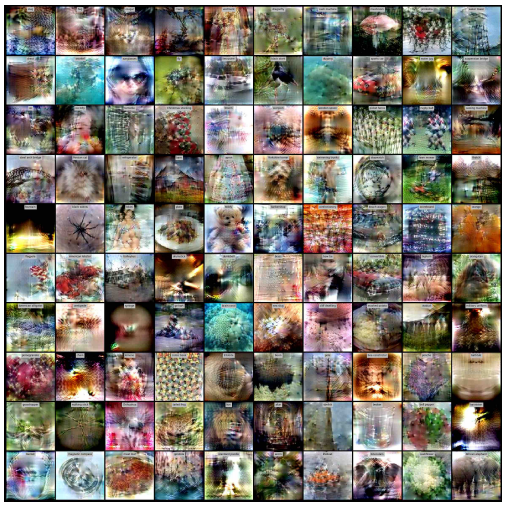
Figure 25: Distilled Image Visualization - CUB-200 (1 Img/Cls)



(a) Learn Label=True, Class ID = [0-99]

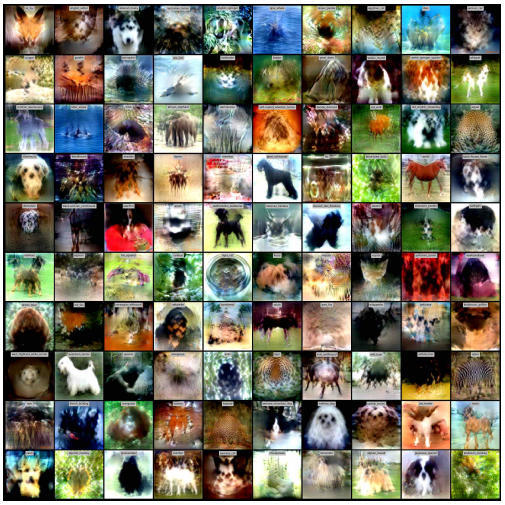
(b) Learn Label=True, Class ID = [100-199]

(c) Learn Label=False, Class ID = [0-99]

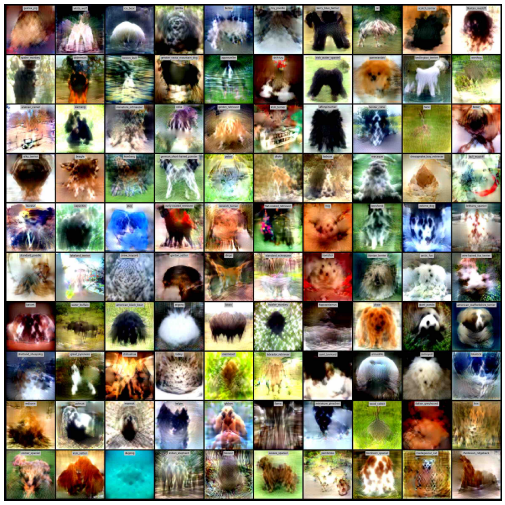


(d) Learn Label=False, Class ID = [100-199]

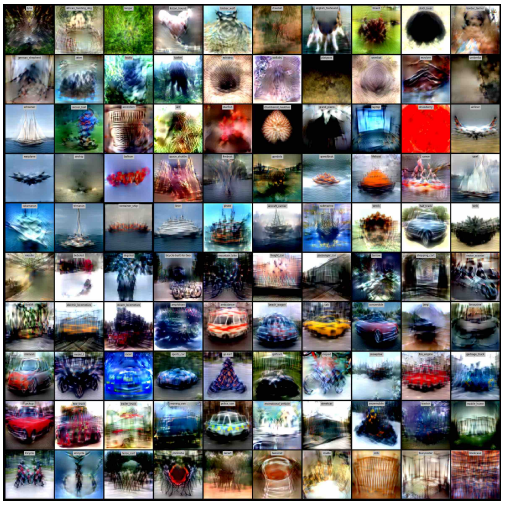
Figure 26: Distilled Image Visualization - Tiny ImageNet (1 Img/Cls)



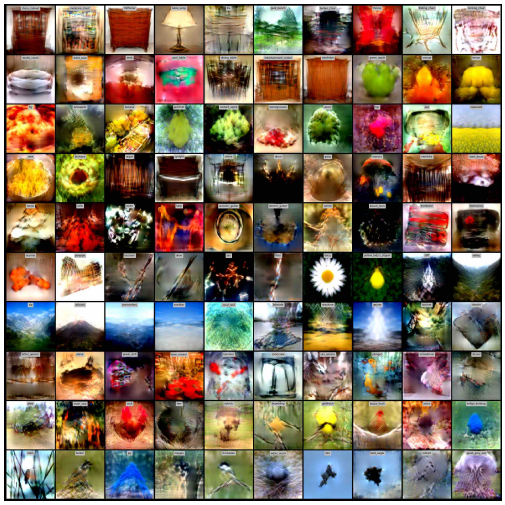
(a) Class ID = [0-99]



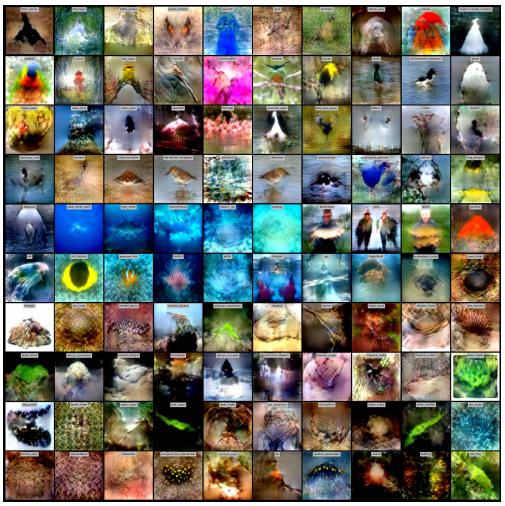
(b) Class ID = [100-199]



(c) Class ID = [200-299]



(d) Class ID = [300-399]

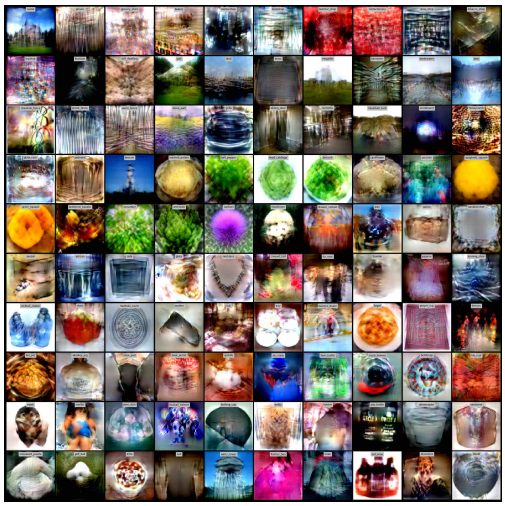


(e) Class ID = [400-499]

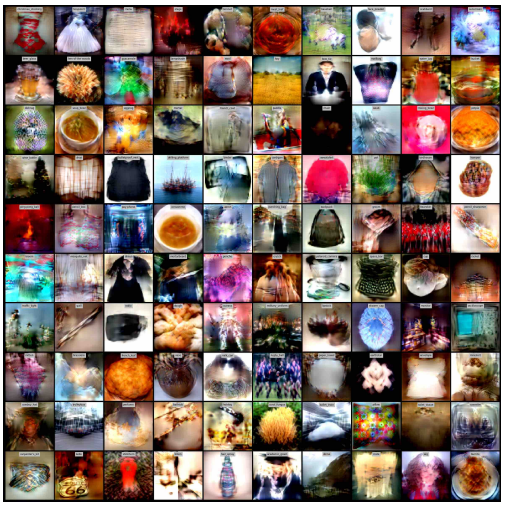
(f) Class ID = [500-599]

Figure 27: Distilled Image Visualization - ImageNet (1 Img/Cls), Learn Label=True

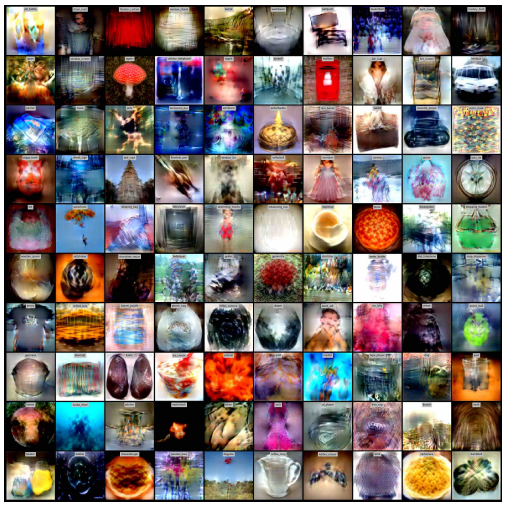
(a) Class ID = [600-699]



(b) Class ID = [700-799]



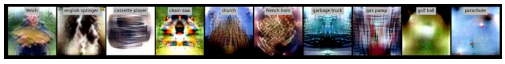
(c) Class ID = [800-899]



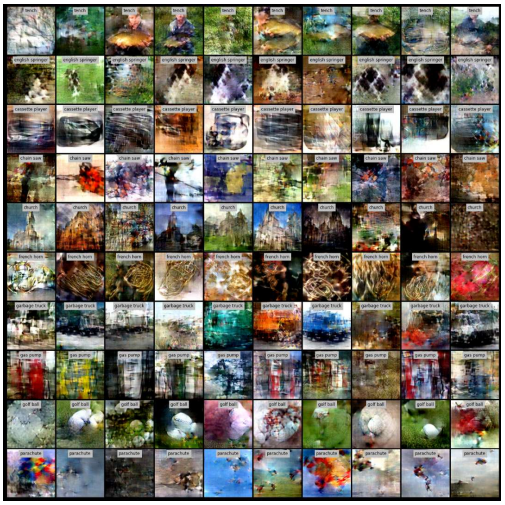
(d) Class ID = [900-999]

Figure 28: Distilled Image Visualization - ImageNet (1 Img/Cls), Learn Label=True

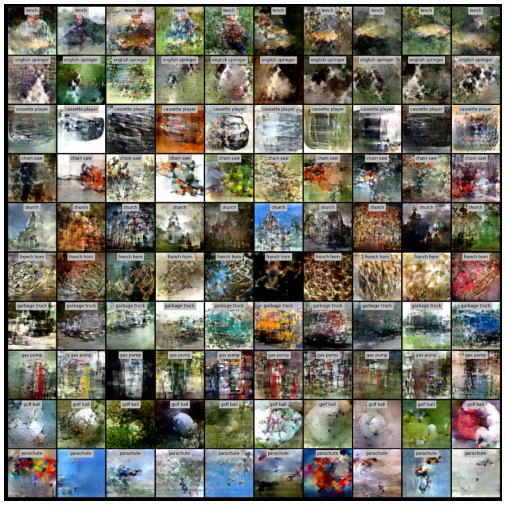
(a) 1 Img/Cls, Learn Label=True



(b) 1 Img/Cls, Learn Label=False

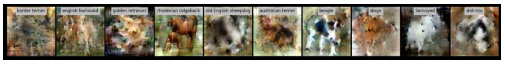


(c) 10 Img/Cls, Learn Label=True



(d) 10 Img/Cls, Learn Label=False

Figure 29: Distilled Image Visualization - ImageNette.



(a) 1 Img/Cls, Learn Label=True

(b) 1 Img/Cls, Learn Label=False

(c) 10 Img/Cls, Learn Label=True

(d) 10 Img/Cls, Learn Label=False

Figure 30: Distilled Image Visualization - ImageWoof.