Dataset Distillation with Convexified Implicit Gradients

Noel Loo [1] **Ramin Hasani** [1] **Mathias Lechner** [1] **Daniela Rus** [1]

Abstract

We propose a new dataset distillation algorithm using reparameterization and convexification of implicit gradients (RCIG), that substantially improves the state-of-the-art. To this end, we first formulate dataset distillation as a bi-level optimization problem. Then, we show how implicit gradients can be effectively used to compute meta-gradient updates. We further equip the algorithm with a convexified approximation that corresponds to learning on top of a frozen finite-width neural tangent kernel. Finally, we improve bias in implicit gradients by parameterizing the neural network to enable analytical computation of final-layer parameters given the body parameters. RCIG establishes the new state-of-the-art on a diverse series of dataset distillation tasks. Notably, with one image per class, on resized ImageNet, RCIG sees on average a 108% improvement over the previous state-of-the-art distillation algorithm. Similarly, we observed a 66% gain over SOTA on Tiny-ImageNet and 37% on CIFAR-100. [1]

1. Introduction

Dataset distillation aims to condense a given dataset into a synthetic version that ideally preserves the information of the original set (Wang et al., 2018). Training on this synthetic set should result in similar performance compared to training on the original dataset.

Dataset distillation can be formulated as a bi-level optimization problem with an inner objective to update model parameters on the support/synthetic/distilled set, and an outer (meta) objective to refine the distilled sets via meta-gradient updates (Wang et al., 2018). Evaluating the meta-optimization loop is difficult, as we have to solve the inner optimization loop and then back-propagate errors through it to obtain the meta gradients. This could be done via backpropagation through time (Werbos, 1988), evolution strategies (Salimans et al., 2017), and equilibrium propagation (Scellier & Bengio, 2016). Numerous research works have improved dataset distillation by introducing surrogate objectives for computing the meta gradients through gradient matching (Zhao & Bilen, 2021a), training trajectory matching (Cazenavette et al., 2022a), feature alignment (Wang et al., 2022), by regressing on neural features (Zhou et al., 2022), and by leveraging the neural tangent kernel (NTK) theory (Jacot et al., 2018; Arora et al., 2019a) in exact (Nguyen et al., 2021a;b) and approximate forms (Loo et al., 2022a).

What does it take to significantly increase the accuracy and performance of dataset distillation? In this paper, we construct a novel dataset distillation algorithm that substantially outperforms state-of-the-art methods, by re-parameterizing and convexifying implicit gradients. Implicit gradients (IGs) leverage implicit function theorem (Bengio, 2000; Chen & Hagan, 1999; Rajeswaran et al., 2019), that defines a meta-gradient update for a bi-level optimization problem (meta-learning with inner and outer loops). IG, off the bat, can serve as a dataset distillation algorithm, but it could perform significantly better via linearized training. Linearized training, which corresponds to learning on a frozen finite width neural tangent kernel (NTK) (Jacot et al., 2018), convexifies the inner model, and as a result refines the implicit gradients. We show that such convexified implicit gradients dataset distillation algorithm equipped with a reparameterization technique for reducing the bias in IG considerably supersedes the state-of-the-art performance on 17 out of 22 reported benchmarks.

In summary, we make the following **new contributions:**

I. We step-by-step construct a new dataset distillation algorithm called reparameterized convexified implicit gradients (RCIG), that establishes the new state-of-the-art. **II.** We show how to effectively design and improve implicit gradients to obtain a bi-level dataset distillation algorithm. **III.** We conduct a large experimental evaluation of our method in a diverse set of dataset distillation tasks and benchmarks and compare its performance to other advanced baselines.

[*]Equal contribution [1]Computer Science and Artificial Intelligence Lab (CSAIL), Massachusetts Institute of Technology (MIT). Correspondence to: Noel Loo <loo@mit.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

[1]Code available at https://github.com/yolky/RCIG

2. Background

Coresets and Dataset Distillation. Coresets are weighted subsets of the training data such that training on them results in the similar performance to training on the full dataset (Munteanu et al., 2018; Mirzasoleiman et al., 2020; Pooladzandi et al., 2022). Existing coreset selection methods employ clustering (Feldman & Langberg, 2011; Lucic et al., 2016; Bachem et al., 2016), bilevel optimization (Borsos et al., 2020), and sensitivity analysis (Munteanu et al., 2018; Huggins et al., 2016; Tukan et al., 2020).

Dataset distillation shares many characteristics with coresets, however, instead of selecting subsets of the training data distillation generates synthetic samples. Similar to coresets, training on the synthetic samples should be faster and result in a better performing model (Wang et al., 2018; Zhao et al., 2021; Zhao & Bilen, 2021b; Nguyen et al., 2021a;b; Zhou et al., 2022; Loo et al., 2022a). Dataset distillation algorithms range from directly unrolling the model training computation graph (Wang et al., 2018), or approximately matching training trajectories with the full dataset (Cazenavette et al., 2022b) As unrolling of the training comes with high memory and computation requirements, more recent works try to approximate the unrolled computation (Zhou et al., 2022; Loo et al., 2022a; Nguyen et al., 2021a;b). Dataset distillation has also shown promise in applications such as continual learning (Zhou et al., 2022; Sangermano et al., 2022), and neural architecture search (Such et al., 2019).

Bilevel Optimization and Implicit gradients. Bilevel optimization problems are a class of problems where one optimization problem is nested inside a second optimization problem. Formally, define the inner objective as $\mathcal{L}_i$ and the outer (meta) objective as $\mathcal{L}_o$. Bilevel optimization problems aim to solve:

$$\underset{\psi}{\operatorname{argmin}}\, \mathcal{L}_o(\theta^*, \psi), \quad s.t. \quad \theta^* \in \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}_i(\theta, \psi)$$

With θ is a set of inner parameters, and ψ is a set of outer/hyperparameters that we aim to solve for. This type of problem arises in many deep learning fields such as hyperparameters optimization (Domke, 2012; Maclaurin et al., 2015; MacKay et al., 2019), meta-learning (Finn et al., 2017; Rajeswaran et al., 2019), and adversarial training (Szegedy et al., 2013; Madry et al., 2017). Similarly, dataset distillation can also be framed as a bilevel optimization problem, with θ the set of network parameters, and ψ our distilled dataset parameters, given by the coreset images and labels (Wang et al., 2018; Nguyen et al., 2021a; Zhou et al., 2022).

Evaluating $\mathcal{L}_o(\theta^*, \psi)$, and evaluating meta-gradients $\frac{dL_o}{d\psi}$ is difficult, as it generally not only requires solving the inner optimization problem to evaluate the outer loss but even worse, to backpropagate through inner optimization to compute meta-gradients. The most standard method is Backpropagation-through-time (BPTT) (Werbos, 1988; Rumelhart & McClelland, 1987; Werbos, 1990), which is what Wang et al. (2018) uses for dataset distillation. Other methods exist such as evolution strategies(Salimans et al., 2017; Vicol et al., 2021) and equilibrium propagation (Scellier & Bengio, 2016; Zucchet et al., 2021). One technique is *implicit differentiation* (implicit gradients/IG methods), which leverages the implicit function theorem (Bengio, 2000; Chen & Hagan, 1999; Rajeswaran et al., 2019). This theorem states that if the inner object admits a unique minimum and the outer objective is continuously differentiable, then we have

$$\frac{\partial \theta^*}{\partial \psi} = \left(\frac{\partial^2 \mathcal{L}_i}{\partial \theta \partial \theta^T}\right)^{-1} \Bigg|_{\theta=\theta^*} \frac{\partial^2 \mathcal{L}_i}{\partial \theta \partial \psi} \Bigg|_{\theta=\theta^*}$$

And our full meta-gradient is given by:

$$\frac{d\mathcal{L}_o}{d\psi} = \underbrace{\frac{\partial \mathcal{L}_o}{\partial \psi}}_{\text{direct grad}} + \underbrace{\frac{\partial}{\partial \psi}\left(\frac{\partial \mathcal{L}_i}{\partial \theta}^{\intercal} v\right)}_{\text{implicit grad}} \tag{1}$$

With v given by $\left(\frac{\partial^2 \mathcal{L}_i}{\partial \theta \partial \theta^T}\right)^{-1} \frac{\partial \mathcal{L}_o}{\partial \theta}$.

Neural Tangent Kernel and Network Linearization. The Neural Tangent Kernel (NTK) is a method of analyzing the behavior of neural networks in the infinite width limit (Jacot et al., 2018; Arora et al., 2019b). It states that as network width approaches infinity, with appropriate network initialization, neural networks behave as first-order Taylor expansion about their initialization, and are thus linear in their weights. The corresponding kernel formulation of the linear classifier results in the NTK, and the finite-width NTK converges to a deterministic architecture-dependent NTK in the infinite width limit. Networks behaving in this regime are said to be in the *kernel regime*, and when the first-order Taylor expansion is accurate, networks are said to be in *lazy training* (Chizat et al., 2019). While typical neural networks have highly non-convex loss landscapes, networks in the NTK regime have **convex** loss landscapes. This convexity has been successfully exploited for tasks such as dataset distillation (Nguyen et al., 2021a;b; Loo et al., 2022a), and federated learning (Yu et al., 2022).

3. Method: Reparameterized Convexified Implicit Gradient

In this section, we step-by-step motivate and explain how we build our reparameterized convexified Implicit Gradient (RCIG) dataset distillation algorithm.

3.1. Implicit Gradients with Dataset Distillation. Dataset distillation can directly be framed as bilevel optimization by letting $\mathcal{L}_o = \mathcal{L}_T(\theta)$, $\mathcal{L}_i = \mathcal{L}_{S(\psi)}(\theta)$, with $\mathcal{L}_T$ and

$\mathcal{L}_{S(\psi)}$ being training losses of the full training set and coreset/support set, with $\psi = \{X_S, y_S\}$ our set of coreset images/labels.

This simple formulation leads directly to a straightforward dataset distillation algorithm, provided that we can compute $v = H_S^{-1} g_T$, with $H_S = \frac{\partial^2 \mathcal{L}_i}{\partial \theta \partial \theta^T}$ and $g_T = \frac{\partial \mathcal{L}_o}{\partial \theta}$. This can be done using a Neumann series, conjugate gradient methods, or the methods we discuss in Section 3, but assume that this computation can be done for the time being. We call this naive implicit gradient algorithm VIG (Vanilla implicit gradients). We evaluate in this algorithm on MNIST, CIFAR-10, and CIFAR-100 distilling 1, 10, and 50 images per class (except on CIFAR-100) on a three-layer convolutional network, with results shown in Table 1 (Lecun et al., 1998; Krizhevsky, 2009). We additionally use small L_2 regularization so that the Hessian inverse is properly defined.

Immediately we see that implicit gradients perform poorly, sometimes performing *worse* than random images, but why? As discussed in Vicol et al. (2022), there existing a unique minimizer is a necessary condition for the implicit gradient theorem to hold. In contrast, deep neural networks are highly non-convex, so we should not expect vanilla implicit gradients to work out of the box. Furthermore, the implicit gradient method relies heavily on computing $g_S^T H_S^{-1} g_T$, with $g_S = \frac{\partial \mathcal{L}_i}{\partial \theta}$ (i.e. the support set gradient). This expression has a striking similarity to *influence functions* (Hampel, 1974; Koh & Liang, 2017; Basu et al., 2020), which leads to a second interpretation of implicit-gradient based dataset distillation as maximizing the mutual influence between our distilled dataset and the full dataset.

While this interpretation of dataset distillation as influence maximization is appealing, it also suggests that the success of our algorithm is heavily dependent on how well $g_S H_S^{-1} g_T$ actually approximates influence. Recent work (Bae et al., 2022) has shown that for deep models, these influence functions are brittle and do not accurately estimate leave-one-out retraining (which influence functions claim to do). Bae et al. (2022) shows that this discrepancy is partially caused by the non-convexity of deep models, as deep models undergo a period of highly erratic optimization behavior before only settling in approximately convex loss region. These two findings suggest that using implicit gradients to perform dataset distillation for highly non-convex deep networks is challenging.

3.2. Convexification. The literature and our simple experiments strongly suggest that our naive implicit gradients will not work unless we are able to make our inner model exhibit more convex behavior. One method of this is by considering the tangent space of the neural network parameters. Specifically, we define:

$$f_\theta(x) \approx f_{lin,\theta}(x) = f_{\theta_0}(x) + (\theta - \theta_0)^\intercal \nabla_\theta f_{\theta_0}(x). \quad (2)$$

We call this 1st-order Taylor approximation of learning dynamics *linearized dynamics* (Fort et al., 2020; Loo et al., 2022b; Lee et al., 2019), as opposed to standard dynamics which corresponds to no Taylor expansion. If we fix θ_0 and consider optimizing θ, this new formulation is now strongly convex in θ, provided that some L_2 regularization is added. Seeing that at initialization, $f_{\theta_0}(x) \approx 0$, we then have *centered* linearized training. This can be efficiently calculated using forward-mode auto-differentiation.

This convex approximation, while fixing our non-convexity issue, is only as useful as it is accurate. This linearization technique corresponds to learning on top of a frozen finite-width neural tangent kernel (NTK) (Jacot et al., 2018; Lee et al., 2019; Aitken & Gur-Ari, 2020; Lee et al., 2020; Hanin & Nica, 2020). For very wide neural nets, it has been shown that neural networks approximately behave as lazy/linear learners, with the correspondence increasing with wider networks (Lee et al., 2019). For narrow networks, it has been shown that networks undergo a brief period of rapid NTK evolution before behaving approximately as lazy learners (Fort et al., 2020; Loo et al., 2022b; Hanin & Nica, 2020).

This leads to a second implicit-gradient-based dataset distillation algorithm, which we call CIG (convexified implicit gradients). Now we replace both the inner and outer objectives from VIG with their convexified counterparts. We show CIG's performance in Table 1 and note that we evaluate using networks under standard dynamics (i.e. the evaluation network is unmodified). We see that convexification/linearization consistently improves distillation performance. While it is interesting that convexifying the problem improves distillation performance, the performance still falls short in state-of-the-art distillation algorithms, which can achieve over 65% on CIFAR-10 with 10 images per class (Zhou et al., 2022). Next, we close this gap by reparameterizing the problem to enable faster convergence.

3.3. Combine Analytic Solutions and Implicit Gradients. A key limitation of CIG is that in practice we are unable to find the true minima of the inner problem in any feasible amount of time. These truncated unrolls lead to bias in implicit gradients. Because the problem is convex, we can consider warm-starting the problem by reusing old intermediate values (Vicol et al., 2022), but this still biases our results as warm-starts are only unbiased if we achieve the minimum every time, which we cannot feasibly do. From an algorithmic standpoint, warm starting also leads to a trade-off in terms of model diversity, as we are forced to reuse the same model as opposed to instantiating new ones, which has shown to be vital in dataset distillation (Zhou et al., 2022; Cazenavette et al., 2022b).

To perform better optimization of the inner problem, we exploit the structure of the neural network. Specifically, we split the network parameters into the final layer parameters,

Table 1. Performance of Vanilla Implicit Gradients (VIG), Convexified Implicit Gradients (CIG), and Reparameterization Covexified Implicit Gradients (RCIG), on distilling MNIST, CIFAR-10, and CIFAR-100. Linearization/convexification improves performance on almost all datasets, and reparameterization further improves performance to achieve state-of-the-art. (n=15)

	MNIST			CIFAR-10			CIFAR-100	
Img/cls	1	10	50	1	10	50	1	10
Random Subset	62.73 ± 0.95	93.22 ± 0.16	97.79 ± 0.12	20.76 ± 0.49	38.43 ± 0.36	54.44 ± 0.34	6.24 ± 0.08	21.08 ± 0.11
VIG	77.42 ± 1.41	90.17 ± 0.61	91.43 ± 0.35	26.54 ± 1.20	54.61 ± 0.13	35.63 ± 0.59	17.79 ± 0.13	29.30 ± 0.13
CIG (+ Linearization)	69.23 ± 1.43	95.37 ± 0.26	95.78 ± 0.17	29.70 ± 0.95	56.48 ± 0.60	56.68 ± 0.57	19.72 ± 0.55	31.36 ± 0.23
RIG (+ Reparam)	**94.80** ± 0.43	98.55 ± 0.11	98.88 ± 0.08	44.48 ± 4.33	66.16 ± 0.78	62.07 ± 4.02	18.39 ± 2.84	**46.09** ± 0.23
RCIG (+ Lin + Reparam)	**94.79** ± 0.35	**98.93** ± 0.03	**99.23** ± 0.03	**52.75** ± 0.76	**69.24** ± 0.40	**73.34** ± 0.29	**39.55** ± 0.16	44.14 ± 0.25

θ_F and the body parameters θ_B, and note that for any setting of θ_B, we can **efficiently analytically compute the optimal $\theta_\mathbf{F}$** when trained under mean squared error (MSE) loss. Specifically, consider MSE loss with labels y_S, and let $h_{\theta_0}(X_S) \in R^{H \times |S|}$ be the hidden layer embeddings, with $|S|$ the distilled dataset size and H the final layer dimension. We know that $\nabla_\theta f_{lin,\theta} = h_{\theta_0}(X_S)$. Defining $k^{\theta_0}(x, x') = h_{\theta_0}(x)^T h_{\theta_0}(x')$, be the associated final-layer NTK, neural network gaussian process (NNGP) or conjugate kernel, we have the optimal set of final layer parameters for our centered linearized problem is given by

$$\theta_F^* = h_{\theta_0}(X_S)\left(K^{\theta_0}_{X_S,X_S} + \lambda I_{|S|}\right)^{-1} \hat{y}_S \tag{3}$$

$$\hat{y}_S = \left(y_S - \theta_B^\intercal \frac{\partial f_{lin,\theta}(X_S)}{\partial \theta_B}\right), \tag{4}$$

where $\theta_B^\intercal \frac{\partial f_{lin,\theta}(X_S)}{\partial \theta_B}$ could be considered an *offset* which changes the labels given by how much the body parameters already changed. Note that without this offset, this method of solving the optimal final layer parameters corresponds to training using the NNGP/Conjugate kernel to convergence, which has been used in Loo et al. (2022a) and Zhou et al. (2022). However, these methods ignore the contribution from the body parameters, which our method does not.

Given that we can solve for the optimal θ_F given θ_B, we now reparameterize our problem so that we **only learn $\theta_\mathbf{B}$**, and automatically compute θ_F. Specifically, our parameterized inner and outer objectives then become:

$$\mathcal{L}_i^{\text{rep}}(\theta_B) = \mathcal{L}_{S(\psi)}(\theta_B, \theta_F^*(\theta_B, \psi)), \text{ and} \tag{5}$$

$$\mathcal{L}_o^{\text{rep}}(\theta_B, \psi) = \mathcal{L}_{platt,T}(\theta_B, \theta_F^*(\theta_B, \psi), \tau). \tag{6}$$

We additionally add L_2 regularization to the inner objective $\frac{\lambda}{2}(\theta_B^\intercal \theta_B + \theta_F^*(\theta_B)^\intercal \theta_F^*(\theta_B))$. For the outer objective, we adopt the same Platt scaling loss with a learnable temperature parameter τ used in (Loo et al., 2022a), as it has shown to be highly effective in dataset distillation settings. At a high level, the Platt scaling loss replaces the MSE loss with $\mathcal{L}_{platt} = \text{xent}(\hat{y}/\tau, y)$, with $\hat{y}$ our predictions, τ the learnable temperature parameter and y the true labels and xent the cross entropy function.

Thus, our final meta-gradient is:

$$\frac{d\mathcal{L}_o^{\text{rep}}}{d\psi} = \underbrace{\frac{\partial \mathcal{L}_o^{\text{rep}}}{\partial \psi}}_{\text{direct grad}} + \underbrace{\frac{\partial}{\partial \psi}\left(\frac{\partial \mathcal{L}_i^{\text{rep}\,\intercal}}{\partial \theta_B} v\right)}_{\text{implicit grad}},$$

with $v = H_S^{\text{rep},-1} g_T^{\text{rep}}$, with $H_S^{\text{rep}} = \frac{\partial^2 \mathcal{L}_i^{\text{rep}}}{\partial \theta_B \partial \theta_B^T}$ and $g_T^{\text{rep}} = \frac{\partial \mathcal{L}_o^{\text{rep}}}{\partial \theta_B}$. Unlike CIG, this reparameterized version has a non-zero contribution from the direct gradient, as ψ is used to compute the optimal set of θ_F. When using an MSE-loss as opposed to the Platt-loss, this direct gradient corresponds to the FRePo (Zhou et al., 2022) loss, evaluated using the perturbed labels given in Equation (4), thus we could consider our algorithm to be a generalization of FRePo to consider the body parameters as well as the final layer parameters.

Finally, noting that neural networks undergo a period of rapid kernel evolution early in training (Fort et al., 2020; Hanin & Nica, 2020; Loo et al., 2022b), it is important to not only use the initialization finite-width NTK but also the evolved NTK. Thus we adopt the same technique used in Zhou et al. (2022), where we have a pool of partially trained models. We fix this pool to contain $m = 30$ models and set the max number of training steps of these models to be $K = 100$, in line with Zhou et al. (2022). Next, we discuss computing $v = H_S^{\text{rep},-1} g_T^{\text{rep}}$.

3.4. Hessian-inverse Vector Computation. To compute implicit gradients, we need to compute $v = H_S^{\text{rep},-1} g_T^{\text{rep}}$. As H has dimension $P \times P$, where P is the parameter dimension, this cannot be done exactly. Thus, it is typical to approximate v using methods such as conjugate gradients (Martens, 2010) or the Neumann series (Agarwal et al., 2016; Koh & Liang, 2017). The method we use is closely related to the Neumann series. We note that v is a minimizer of the following loss:

$$\mathcal{L}_{H^{-1}} = (Hv - g)^\intercal H^{-1}(Hv - g), \tag{7}$$

which has gradients w.r.t v as $Hv - g$. Thus we perform stochastic gradient descent (SGD) on this objective using an optimizer such as Adam for $n_{H^{-1}}$ gradient steps. Note that using SGD on this objective corresponds to the Neumann

Algorithm 1 Reparam Convexified Implicit Gradients

Input: Training set and labels $\mathcal{T}$, inner, Hessian-inverse and distilled dataset learning rates $\alpha_{\text{inner}}, \alpha_{H^{-1}}, \alpha_S$
Initialize: Initialize distilled dataset and labels $\mathcal{S}$ with parameters $\psi = \{X_S, y_S, \tau\}$
Initialize: Initialize a model pool $\mathcal{M}$ with m randomly initialized models $\{\theta_i\}_{i=1}^m$
while not converged **do**
  Sample a random model from the pool: $\theta_i \sim \mathcal{M}$
  Sample a training batch from the training set: $\{X_T, y_T\} \sim \mathcal{T}$
  Perform n_{inner} optimization steps on inner objective $\mathcal{L}_i^{\text{rep}}$ given by Equation (5) to obtain θ_i^* with **linearized** dynamics
  Perform $n_{H^{-1}}$ optimization steps on Equation (7) to obtain v
  Compute direct gradient $g_{\text{direct}} = \frac{\partial \mathcal{L}_o^{\text{rep}}}{\partial \psi}$, with $\mathcal{L}_o^{\text{rep}}$ given by Equation (6)
  Compute implicit gradient $g_{\text{implicit}} = \frac{\partial}{\partial \psi}\left(\frac{\partial \mathcal{L}_i^{\text{rep}}}{\partial \theta_B}^\intercal v\right)$
  Update the distilled dataset: $\psi \leftarrow \psi - \alpha_S(g_{\text{direct}} + g_{\text{implicit}})$
  Train the model θ_i on the current distilled dataset $\mathcal{S}$ for one step using **standard** dynamics
  Reinitialize the model θ_i if it has been updated for more than K steps.
end while

series method of computing Hessian-inverse vector products. Hessian-vector products can be efficiently computed using the Perlmutter trick (Pearlmutter, 1994). For the inner optimization objective, we perform n_{inner} optimization steps and then $n_{H^{-1}}$ optimization steps on Equation (7). This leads to the Reparameterized Convexified Implicit Gradients (RCIG) algorithm, with pseudo-code given in Algorithm 1.

Complexity Analysis. Let our coreset size be $|S|$, training batch size be $|B|$, and our network parameter have dimension P. One training iteration, per Algorithm 1 contains three main steps: optimizing the inner objective, computing the Hessian-inverse vector product, and computing the meta-gradient. Optimizing the inner objective takes $O(n_{\text{inner}}|S|)$ time, as it requires a full forward pass of the coreset at each training iteration to compute Equation (5). Likewise, computing the Hessian-inverse vector product takes $O(n_{H^{-1}}|S|)$ time, as we perform $n_{H^{-1}}$ optimization steps on Equation (7). Computing the direct gradient and implicit gradient also costs $O(|S|)$ time, giving a total time complexity of $O((n_{\text{inner}} + n_{H^{-1}})|S|)$. The memory requirements of inner optimization and Hessian inverse computation are constant in n_{inner} and $n_{H^{-1}}$, as we do not retain intermediate computations, so the total memory consumption is $O(P)$.

3.6. Bias-free subsampling. One limitation of RCIG compared to CIG is that when optimizing the inner objective, we need to compute $\nabla_{\theta_B}(\mathcal{L}_{S(\psi)}(\theta_B, \theta_F^*(\theta_B))$, which we know from Equation (3), depends on the entire training set X_S, y_S. Thus, we cannot use stochastic gradient descent as we can with CIG, as leaving out elements of X_S would lead to biased gradients. Likewise, when computing $H_S^{\text{rep},-1} g_T^{\text{rep}}$, we need to compute Hessian-vector products, which again rely on all of X_S, y_S. This, combined with the fact that linearization, in general, incurs a doubling of memory cost, makes direct implementation of RCIG difficult for very large support sets (for example CIFAR-100 with 50 images per class). Here, we present a simple technique for getting unbiased gradients without incurring the full memory cost.

When computing these gradients, we note that the gradient contribution from each of the X_S elements is interchangeable, implying that when performing the backward pass, the gradients computing through the nodes associated with $h_{\theta_0}(x_i)$, are all unbiased estimates of the gradient when computed through all nodes $h_{\theta_0}(X_S)$. This means that rather than computing the backward pass through all nodes, we can compute a backward pass through only some of the $h_{\theta_0}(X_S)$ nodes, provided that prior to those nodes the backward pass, the whole computation is stored. Note that from a memory perspective, computing $h_{\theta_0}(X_S)$ **is not expensive**, but requires storing the computation graph leading to that point for the backward pass **is expensive**. Thus, by randomly dropping parts of this computation graph we can save substantial memory. The important note is that during the backward pass, we still need accurate gradients leading into h_{θ_0}, meaning that we still need to full forward pass so that we can compute θ_F^*.

Thus, we propose running a full forward pass on the whole support set, but stop gradients for a random subset of $h_{\theta_0}(X_S)$, only allowing the backward pass through the complementary subset of size $n_{\text{subset}} < |S|$, with $|S|$ the distilled dataset size. A more detailed description and formal justification for this technique are present in Oktay et al. (2021). Schematically, this is shown in Figure 4. We use this technique whenever the coreset size exceeds 1000 images.

4. Experimental Results

In this section, we present our comprehensive experimental evaluation of our method, RCIG, compared to modern baselines using a diverse series of benchmarks and tasks.

4.1. Results on Standard Benchmarks. We first ran RCIG on six standard benchmarks tests including MNIST (10 classes) (Lecun et al., 1998), Fashion-MNIST (10 classes) (Xiao et al., 2017), CIFAR-10 (10 classes), CIFAR-100 (10 classes) (Krizhevsky, 2009), Tiny-ImageNet (200 classes) (Le & Yang, 2015), and Caltech Birds 2011 (200 classes)

Table 2. Distillation performance of RCIG and six baseline distillation algorithms on six benchmark datasets. RCIG attains the highest accuracy on 13/16 of these benchmarks, with the largest gains in the 1 Img/Cls category. (n=15)

	Img/Cls	DSA	DM	KIP	RFAD	MTT	FRePo	RCIG	Full Dataset
MNIST	1	88.7 ± 0.6	89.9 ± 0.8	90.1 ± 0.1	94.4 ± 1.5	91.4 ± 0.9	93.0 ± 0.4	$\mathbf{94.7 \pm 0.5}$	99.6 ± 0.0
	10	97.9 ± 0.1	97.6 ± 0.1	97.5 ± 0.0	98.5 ± 0.1	97.3 ± 0.1	98.6 ± 0.1	$\mathbf{98.9 \pm 0.0}$	
	50	99.2 ± 0.1	98.6 ± 0.1	98.3 ± 0.1	98.8 ± 0.1	98.5 ± 0.1	$\mathbf{99.2 \pm 0.0}$	$\mathbf{99.2 \pm 0.0}$	
F-MNIST	1	70.6 ± 0.6	71.5 ± 0.5	73.5 ± 0.5	78.6 ± 1.3	75.1 ± 0.9	75.6 ± 0.3	$\mathbf{79.8 \pm 1.1}$	93.5 ± 0.1
	10	84.8 ± 0.3	83.6 ± 0.2	86.8 ± 0.1	87.0 ± 0.5	87.2 ± 0.3	86.2 ± 0.2	$\mathbf{88.5 \pm 0.2}$	
	50	88.8 ± 0.2	88.2 ± 0.1	88.0 ± 0.1	88.8 ± 0.4	88.3 ± 0.1	89.6 ± 0.1	$\mathbf{90.2 \pm 0.2}$	
CIFAR-10	1	36.7 ± 0.8	31.0 ± 0.6	49.9 ± 0.2	53.6 ± 1.2	46.3 ± 0.8	46.8 ± 0.7	$\mathbf{53.9 \pm 1.0}$	84.8 ± 0.1
	10	53.2 ± 0.8	49.2 ± 0.8	62.7 ± 0.3	66.3 ± 0.5	65.3 ± 0.7	65.5 ± 0.4	$\mathbf{69.1 \pm 0.4}$	
	50	66.8 ± 0.4	63.7 ± 0.5	68.6 ± 0.2	71.1 ± 0.4	71.6 ± 0.2	71.7 ± 0.2	$\mathbf{73.5 \pm 0.3}$	
CIFAR-100	1	16.8 ± 0.2	12.2 ± 0.4	15.7 ± 0.2	26.3 ± 1.1	24.3 ± 0.3	28.7 ± 0.1	$\mathbf{39.3 \pm 0.4}$	56.2 ± 0.3
	10	32.3 ± 0.3	29.7 ± 0.3	28.3 ± 0.1	33.0 ± 0.3	40.1 ± 0.4	42.5 ± 0.2	$\mathbf{44.1 \pm 0.4}$	
	50	42.8 ± 0.4	43.6 ± 0.4	-	-	$\mathbf{47.7 \pm 0.2}$	44.3 ± 0.2	46.7 ± 0.3	
T-ImageNet	1	6.6 ± 0.2	3.9 ± 0.2	-	-	8.8 ± 0.3	15.4 ± 0.3	$\mathbf{25.6 \pm 0.3}$	37.6 ± 0.4
	10	-	12.9 ± 0.4	-	-	23.2 ± 0.2	25.4 ± 0.2	$\mathbf{29.4 \pm 0.2}$	
CUB-200	1	1.3 ± 0.1	1.6 ± 0.1	-	-	2.2 ± 0.1	$\mathbf{12.4 \pm 0.2}$	12.1 ± 0.2	20.5 ± 0.3
	10	4.5 ± 0.3	4.4 ± 0.2	-	-	-	$\mathbf{16.8 \pm 0.1}$	15.7 ± 0.3	

(Welinder et al., 2010), with performances reported in Table 2. We compare RCIG to six baseline dataset distillation algorithms including Differentiable Siamese Augmentation (DSA) (Zhao & Bilen, 2021b), Distribution Matching (DM) (Zhao & Bilen, 2021a), Kernel-Inducing-Points (KIP) (Nguyen et al., 2021b), Random Feature Approximation (RFAD) (Loo et al., 2022a), Matching Training Trajectories (MTT) (Cazenavette et al., 2022b), and neural Feature Regression with Pooling (FRePo) (Zhou et al., 2022).

Our method establishes the new state-of-the-art performance in 13 out of 16 of these benchmark tasks, sometimes with a significant margin. We see the greatest performance gains in datasets that have many classes, with few images per class. In CIFAR-100 with one image per class, we are able to achieve $39.3 \pm 0.4\%$, compared to the previous state of the art $28.7 \pm 0.4\%$, which is equivalent to **37% improvement over SOTA**. Similarly, in Tiny-ImageNet with one image per class, we achieve $25.6 \pm 0.3\%$ compared to the previous state-of-the-art $15.4 \pm 0.3\%$, which is an improvement of **66% over SOTA**. For CUB-200, we notice that we underperform compared to FRePo. Noting the relatively small training set size (5,994), compared to the number of classes (200), we observed that our algorithm tended to overfit to the training set, as with 10 images per class, we observed that we would achieve 100% training classification accuracy. A solution to this would have been to apply data augmentation during dataset distillation, however, to keep our method simple we applied no augmentation.

RCIG outperforms SOTA methods even with larger support sizes with a good margin. However, the performance gain is not as significant as in the case of smaller support sets. For example, we see a performance gain of 14% going from 1 to 10 classes in Tiny-ImageNet over SOTA. We hypothesize that this is because minimizing the inner objective is harder when the dataset is larger, and likely would require a lower learning rate and more inner steps to converge.

4.2. Cross-architecture generalization. A desirable property of distilled datasets is the ability to transfer well to unseen training architectures. Here we evaluate the transferability of RCIG's distilled datasets for CIFAR-10 10 images per class. Following prior work (Zhao & Bilen, 2021b;a; Cazenavette et al., 2022b; Zhou et al., 2022), we evaluate our models on the ResNet-18 (He et al., 2015), VGG11 (Simonyan & Zisserman, 2014), and AlexNet (Krizhevsky et al., 2012). Additionally, we consider various normalization layers such as using no normalization (NN), Batch Normalization (BN) (Ioffe & Szegedy, 2015), and Instance Normalization (IN). Default normalization (DN) refers to the evaluation architecture used in the respective paper, which is the same as the training architecture, except for FRePo, which trains with BN and evaluates with NN. RCIG Typically trains with NN and evaluates with NN, but we also consider training with BN. Table 3 summarizes the results. We see that RCIG can achieve high transferability, in particular when we use BN during training. We hypothesize that using BN during training helps ensure that magnitudes of distilled images remain similar to real images, allowing for wider generalizability, although future work should investigate the role of normalization during training further.

4.3. Experiments with ImageNet Datasets. We next considered higher-resolution subsets. Consistent with Zhou et al. (2022) and Cazenavette et al. (2022b), we consider two ImageNet subsets: ImageNette and ImageWoof, both subsets of ImageNet with 10 classes each with resolutions of 128×128 (Howard, 2020). On these two datasets, we see that RCIG outperforms the baselines substantially in the 1 image per class setting, but only performs similarly to the previous state-of-the-art with more images. To evaluate how well RCIG scales to more complex label spaces, we also consider the full ImageNet-1K dataset with 1000 classes,

Table 3. Performance of CIFAR-10 10 Img/Cls distilled datasets evaluated on different network architectures. Default Normalization (DN) refers to the test architecture used in reported results in the respective papers. RCIG-distilled datasets achieve high accuracy across a variety of datasets, particularly when BatchNorm (BN) is used during training. * - see footnote[3] (n=15)

	Training Architecture	Evaluation Architecture					
		Conv-DN	Conv-NN	ResNet-DN	ResNet-BN	VGG-BN	AlexNet
DSA	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	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	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	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	65.5 ± 0.4	65.5 ± 0.4	$\mathbf{54.4 \pm 0.8}^*$	$52.4 \pm 0.7^*$	$\mathbf{55.4 \pm 0.7}^*$	$61.6 \pm 0.2^*$
RCIG	Conv-NN	$\mathbf{69.1 \pm 0.4}$	$\mathbf{69.1 \pm 0.4}$	51.3 ± 1.7	49.7 ± 1.4	46.2 ± 1.8	60.8 ± 1.8
RCIG	Conv-BN	66.0 ± 0.6	66.0 ± 0.6	$\mathbf{54.4 \pm 1.1}$	$\mathbf{54.8 \pm 1.1}$	$\mathbf{55.4 \pm 1.1}$	$\mathbf{62.1 \pm 0.8}$

Table 4. Distillation performance for ImageNet subsets. RCIG attains the highest performance in the single image per class category on all benchmarks. In particular, RCIG **doubles** the performance of the state-of-the-art in the ImageNet 1 Img/Cls setting. (n=15)

	ImageNette (128x128)		ImageWoof (128x128)		ImageNet (64x64)	
Img/Cls	1	10	1	10	1	2
Random Subset	23.5 ± 4.8	47.4 ± 2.4	14.2 ± 0.9	27.0 ± 1.9	1.1 ± 0.1	1.4 ± 0.1
MTT	47.7 ± 0.9	63.0 ± 1.3	28.6 ± 0.8	35.8 ± 1.8	-	-
FRePo	48.1 ± 0.7	$\mathbf{66.5 \pm 0.8}$	29.7 ± 0.6	$\mathbf{42.2 \pm 0.9}$	7.5 ± 0.3	9.7 ± 0.2
RCIG	$\mathbf{53.0 \pm 0.9}$	65.0 ± 0.7	$\mathbf{33.9 \pm 0.6}$	$\mathbf{42.2 \pm 0.7}$	$\mathbf{15.6 \pm 0.2}$	$\mathbf{16.6 \pm 0.1}$

resized to 64×64 (Russakovsky et al., 2014). With one image per class, **our algorithm doubles the performance of the previous SOTA**, achieving $\mathbf{15.6 \pm 0.2}\%$.

4.4. Ablation: Number of Optimization Steps. To run RCIG, we require two key hyperparameters, the number of inner optimization steps, n_{inner}, and the number of steps used to compute the Hessian-inverse-vector product, $n_{H^{-1}}$. These two hyperparameters play a critical role in the fidelity of all implicit gradient methods. n_{inner} controls the quality of the inner optimization procedure, with larger n_{inner} resulting in better approximations of the true minimum. Likewise, $n_{H^{-1}}$ controls the accuracy of our approximation of $v = H^{-1}g_T$. In previous sections, we fixed $n_{\text{inner}} = n_{H^{-1}} = 20$ for all datasets/experiments for simplicity. In this section, we study the effect of both these critical parameters on the runtime and accuracy of the algorithm.

Specifically, for CIFAR-100 1 Img/Cls, CIFAR-10 1 Img/Cls, and CIFAR-10 10 Img/Cls, we rerun our algorithm with $n_{\text{inner}}, n_{H^{-1}} \in \{0, 2, 5, 10, 20, 30\}$, and report the resulting runtime per iteration in Figure 1 and accuracy in Figure 2, with the green stars, indicate our chosen hyperparameter configuration $n_{\text{inner}} = n_{H^{-1}} = 20$, and red circle in Figure 2 indicating the case where $n_{\text{inner}} = n_{H^{-1}} = 0$. From Figure 1, we see that runtime is linear in both n_{inner} and $n_{H^{-1}}$, in line with our expectations. This has a large impact on the runtime of the algorithm, as for CIFAR-100 with 1 Img/Cls, the runtime per iteration is from 150ms to 760ms per iteration with the largest hyperparameter configurations.

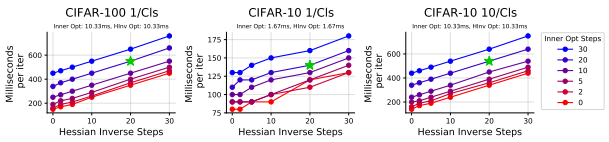


Figure 1. The effect of n_{inner} and $n_{H^{-1}}$ on the computation time for CIFAR-100 1 Img/Cls, and CIFAR-10 with 1 and 10 Img/Cls. The green star denotes the hyperparameter configuration used throughout the paper ($n_{\text{inner}} = 20, n_{H^{-1}} = 20$).

In terms of accuracy, we see from Figure 2, **running inner optimization provides a clear performance benefit, only if we take into account the implicit gradient**. The $n_{\text{inner}} = n_{H^{-1}} = 0$ (baseline) configuration achieves relatively high performance, and as discussed in Section 3, this corresponds to the same algorithm as FRePo (provided that we use an MSE rather than Platt outer loss) and uses information from only the last layer. This method only leverages the information in the final layer weights and only has the direct gradient component in Equation (1). If we set $n_{\text{inner}} > 0$ but $n_{H^{-1}} = 0$, we see a clear performance drop compared to the baseline configuration, as it corresponds to ignoring the implicit gradient. However, adding a small number of Hessian Inverse steps (as few as 2) allows methods with $n_{\text{inner}} > 0$, to exceed the base configuration's performance. When n_{inner} is small, larger $n_{H^{-1}}$ values hurt performance, as the implicit gradient formula only makes sense provided that we are at a minimum, which is not the case when n_{inner} is small. Finally, we observe that $n_{\text{inner}} = 2$ sees very poor performance in general.

4.5. Application: Privacy Preservation. Membership inference attacks (MIA) (Shokri et al., 2016) try to determine

Table 5. AUC of five MIA attack strategies for neural networks trained on distilled data. (n=25)

	Test Acc (%)	Attack AUC				
		Threshold	LR	MLP	RF	KNN
Real	$\mathbf{99.2 \pm 0.1}$	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	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.53 \pm 0.01}$	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	$\mathbf{0.53 \pm 0.01}$	$\mathbf{0.52 \pm 0.01}$	$\mathbf{0.51 \pm 0.01}$
RCIG	$\underline{98.9 \pm 0.0}$	$\mathbf{0.49 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.53 \pm 0.00}$	0.53 ± 0.00	0.52 ± 0.00

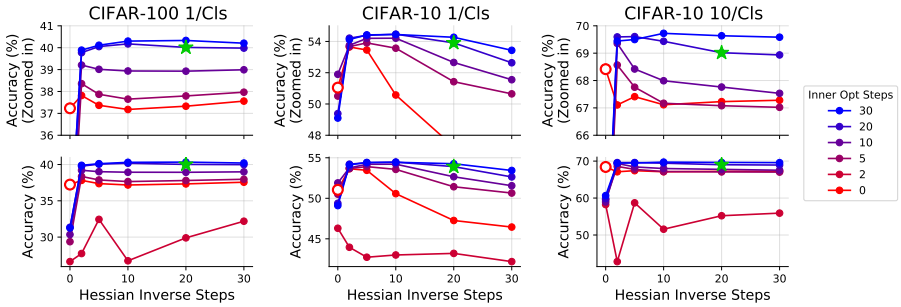


Figure 2. The effect of n_{inner} and $n_{H^{-1}}$ on the distillation accuracy for CIFAR-100 1 Img/Cls, and CIFAR-10 with 1 and 10 Img/Cls. The green star denotes the hyperparameter configuration used throughout the paper ($n_{\text{inner}} = 20, n_{H^{-1}} = 20$), while the red circle denotes using no implicit gradients and only direct gradients. The top row is the same as the bottom row except zoomed in. There is a clear advantage to using more inner optimization steps, provided that we account for the implicit gradient ($n_{H^{-1}} > 0$)

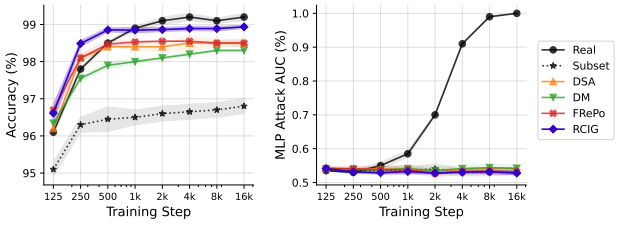


Figure 3. Test accuracy and MLP Attack AUC for models trained on distilled data. Training on distilled data is not vulnerable to MIA attacks while training on real data leaks information

whether a particular example was used to train a machine learning model. As training data can contain sensitive user information, it is critical that adversaries cannot infer what these training examples were. There exist many methods to defend against these attacks, such as adding gradient noise during training (Abadi et al., 2016), or strategic regularization (Nasr et al., 2018). Recently it has been shown that Dataset Distillation is a viable empirical defense strategy (Dong et al., 2022)[4], as training neural networks on distilled data in practice resists membership inference attacks queried on the original training set.

Here we verify that RCIG also works as a viable defense strategy. We repeat the experimental procedure of Zhou et al. (2022) and distill 10000 images of MNIST or Fashion-MNIST to 500 images. We run five (Threshold, Logistic-Regression (LR), Multi-Layered Perceptron (MLP), Random Forest (RF), and K-nearest-neighbor (KNN) attacks from Tensorflow-Privacy on neural networks trained on these 500 distilled images. We report the AUC of these attacks and resulting test accuracy in Table 5 after training for 16k iterations. More experimental details are in Appendix. We see that RCIG achieves the highest accuracy of all the dataset distillation approaches, but remains resistant to the five attack strategies.

[4]see Carlini et al. (2022) for a more careful discussion

5. Discussions, Limitations, and Conclusion

In this paper, we presented RCIG, a dataset distillation algorithm based on implicit gradients, which achieved state-of-the-art performance on a wide variety of benchmarks. To derive our method, we first considered implicit gradients, then showed that linearizing the inner objective to make the problem convex improves performance, owing to the non-convexity of deep neural networks. Then we show that exploiting the neural network structure lets us more quickly converge to near-optimal solutions. We verified the efficiency of our approach on a wide variety of datasets, obtaining $\mathbf{15.6}\%$ accuracy on ImageNet with 1 image per class, doubling the performance of prior art.

While our approach performs well, as discussed, our method can overfit on datasets with fewer training samples such as CUB-200. Future work could look at mitigating this

overfitting issue. As we presented in Section 3, and showed in practice in Appendix A, we can scale our algorithm to larger datasets without increasing the memory footprint by subsampling the backward pass. This method is still limited as it still requires a full forwards pass on the full distilled dataset. Future work can look at removing this limitation.

6. Acknowledgements

Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. The research was also funded in part by the AI2050 program at Schmidt Futures (Grant G-22-63172) and Capgemini SE.

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pp. 308–318, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394. doi: 10.1145/2976749.2978318. URL https://doi.org/10.1145/2976749.2978318.

Agarwal, N., Bullins, B., and Hazan, E. Second-order stochastic optimization for machine learning in linear time. 2016. doi: 10.48550/ARXIV.1602.03943. URL https://arxiv.org/abs/1602.03943.

Aitken, K. and Gur-Ari, G. On the asymptotics of wide networks with polynomial activations. *ArXiv*, abs/2006.06687, 2020.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R., and Wang, R. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pp. 8141–8150. Curran Associates, Inc., 2019a.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R., and Wang, R. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pp. 8141–8150. Curran Associates, Inc., 2019b.

Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., Kapturowski, S., Keck, T., Kemaev, I., King, M., Kunesch, M., Martens, L., Merzic, H., Mikulik, V., Norman, T., Quan, J., Papamakarios, G., Ring, R., Ruiz, F., Sanchez, A., Schneider, R., Sezener, E., Spencer, S., Srinivasan, S., Wang, L., Stokowiec, W., and Viola, F. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

Bachem, O., Lucic, M., Hassani, S. H., and Krause, A. Approximate k-means++ in sublinear time. In Schuurmans, D. and Wellman, M. P. (eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pp. 1459–1467. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12147.

Bae, J., Ng, N. H., Lo, A., Ghassemi, M., and Grosse, R. B. If influence functions are the answer, then what is the question? In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=hzbguA9zMJ.

Basu, S., Pope, P., and Feizi, S. Influence functions in deep learning are fragile, 2020. URL https://arxiv.org/abs/2006.14651.

Bengio, Y. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000. doi: 10.1162/089976600300015187.

Borsos, Z., Mutny, M., and Krause, A. Coresets via bilevel optimization for continual learning and streaming. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 14879–14890, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Carlini, N., Feldman, V., and Nasr, M. No free lunch in "privacy for free: How does dataset condensation help privacy", 2022. URL https://arxiv.org/abs/2209.14987.

Cazenavette, G., Wang, T., Torralba, A., Efros, A. A., and Zhu, J.-Y. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4750–4759, 2022a.

Cazenavette, G., Wang, T., Torralba, A., Efros, A. A., and Zhu, J.-Y. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022b.

Chen, D. and Hagan, M. Optimal use of regularization and cross-validation in neural network modeling. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, volume 2, pp. 1275–1280 vol.2, 1999. doi: 10.1109/IJCNN.1999.831145.

Chizat, L., Oyallon, E., and Bach, F. R. On lazy training in differentiable programming. In *NeurIPS*, 2019.

Domke, J. Generic methods for optimization-based modeling. In Lawrence, N. D. and Girolami, M. (eds.), *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pp. 318–326, La Palma, Canary Islands, 21–23 Apr 2012. PMLR. URL https://proceedings.mlr.press/v22/domke12.html.

Dong, T., Zhao, B., and Lyu, L. Privacy for free: How does dataset condensation help privacy? *ArXiv*, abs/2206.00240, 2022.

Feldman, D. and Langberg, M. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pp. 569–578, 2011.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

Fort, S., Dziugaite, G. K., Paul, M., Kharaghani, S., Roy, D. M., and Ganguli, S. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/405075699f065e43581f27d67bb68478-Abstract.html.

Hampel, F. R. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974. ISSN 01621459. URL http://www.jstor.org/stable/2285666.

Hanin, B. and Nica, M. Finite depth and width corrections to the neural tangent kernel. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgndT4KwB.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

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

Howard, J. A smaller subset of 10 easily classified classes from imagenet, and a little more french, 2020. URL https://github.com/fastai/imagenette/.

Huggins, J. H., Campbell, T., and Broderick, T. Coresets for scalable bayesian logistic regression. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4080–4088, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *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.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1885–1894. JMLR. org, 2017.

Krizhevsky, A. Learning multiple layers of features from tiny images. pp. 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Le, Y. and Yang, X. S. Tiny imagenet visual recognition challenge. 2015.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Lee, J., Schoenholz, S., Pennington, J., Adlam, B., Xiao, L., Novak, R., and Sohl-Dickstein, J. Finite versus infinite neural networks: an empirical study. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 15156–15172. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/ad086f59924fffe0773f8d0ca22ea712-Paper.pdf.

Loo, N., Hasani, R., Amini, A., and Rus, D. Efficient dataset distillation using random feature approximation. *Advances in Neural Information Processing Systems*, 2022a.

Loo, N., Hasani, R., Amini, A., and Rus, D. Evolution of neural tangent kernels under benign and adversarial training. In *Advances in Neural Information Processing Systems*, 2022b.

Lucic, M., Bachem, O., and Krause, A. Strong coresets for hard and soft bregman clustering with applications to exponential family mixtures. In *Artificial intelligence and statistics*, pp. 1–9. PMLR, 2016.

MacKay, M., Vicol, P., Lorraine, J., Duvenaud, D., and Grosse, R. B. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *CoRR*, abs/1903.03088, 2019. URL http://arxiv.org/abs/1903.03088.

Maclaurin, D., Duvenaud, D., and Adams, R. P. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 2113–2122. JMLR.org, 2015.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks, 2017. URL https://arxiv.org/abs/1706.06083.

Martens, J. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 735–742, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Mirzasoleiman, B., Bilmes, J. A., and Leskovec, J. Coresets for data-efficient training of machine learning models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6950–6960. PMLR, 2020. URL http://proceedings.mlr.press/v119/mirzasoleiman20a.html.

Munteanu, A., Schwiegelshohn, C., Sohler, C., and Woodruff, D. On coresets for logistic regression. *Advances in Neural Information Processing Systems*, 31, 2018.

Nasr, M., Shokri, R., and Houmansadr, A. Machine learning with membership privacy using adversarial regularization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pp. 634–646, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356930. doi: 10.1145/3243734.3243855. URL https://doi.org/10.1145/3243734.3243855.

Nguyen, T., Chen, Z., and Lee, J. Dataset meta-learning from kernel ridge-regression. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=l-PrrQrK0QR.

Nguyen, T., Novak, R., Xiao, L., and Lee, J. Dataset distillation with infinitely wide convolutional networks. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021b. URL https://openreview.net/forum?id=hXWPpJedrVP.

Oktay, D., McGreivy, N., Aduol, J., Beatson, A., and Adams, R. P. Randomized automatic differentiation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=xpx9zj7CUlY.

Pearlmutter, B. A. Fast Exact Multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 01 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.1.147. URL https://doi.org/10.1162/neco.1994.6.1.147.

Pooladzandi, O., Davini, D., and Mirzasoleiman, B. Adaptive second order coresets for data-efficient machine learning. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings*

of Machine Learning Research, pp. 17848–17869. PMLR, 2022. URL https://proceedings.mlr.press/v162/pooladzandi22a.html.

Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. Meta-learning with implicit gradients. *CoRR*, abs/1909.04630, 2019. URL http://arxiv.org/abs/1909.04630.

Rumelhart, D. E. and McClelland, J. L. *Learning Internal Representations by Error Propagation*, pp. 318–362. 1987.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Fei-Fei, L. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. URL http://arxiv.org/abs/1409.0575.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning, 2017. URL https://arxiv.org/abs/1703.03864.

Sangermano, M., Carta, A., Cossu, A., and Bacciu, D. Sample condensation in online continual learning, 2022. URL https://arxiv.org/abs/2206.11849.

Scellier, B. and Bengio, Y. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation, 2016. URL https://arxiv.org/abs/1602.05179.

Shokri, R., Stronati, M., and Shmatikov, V. Membership inference attacks against machine learning models. *CoRR*, abs/1610.05820, 2016. URL http://arxiv.org/abs/1610.05820.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Such, F. P., Rawal, A., Lehman, J., Stanley, K. O., and Clune, J. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *CoRR*, abs/1912.07768, 2019. URL http://arxiv.org/abs/1912.07768.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. –, 12 2013.

Tukan, M., Maalouf, A., and Feldman, D. Coresets for near-convex functions. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Vicol, P., Metz, L., and Sohl-Dickstein, J. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. *CoRR*, abs/2112.13835, 2021. URL https://arxiv.org/abs/2112.13835.

Vicol, P., Lorraine, J. P., Pedregosa, F., Duvenaud, D., and Grosse, R. B. On implicit bias in overparameterized bilevel optimization. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 22234–22259. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/vicol22a.html.

Wang, K., Zhao, B., Peng, X., Zhu, Z., Yang, S., Wang, S., Huang, G., Bilen, H., Wang, X., and You, Y. Cafe: Learning to condense dataset by aligning features. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12196–12205, 2022.

Wang, T., Zhu, J.-Y., Torralba, A., and Efros, A. A. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., and Perona, P. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

Werbos, P. 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.

Werbos, P. J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(88)90007-X. URL https://www.sciencedirect.com/science/article/pii/089360808890007X.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Yu, Y., Wei, A., Karimireddy, S. P., Ma, Y., and Jordan, M. I. Tct: Convexifying federated learning using bootstrapped neural tangent kernels, 2022. URL https://arxiv.org/abs/2207.06343.

Zhao, B. and Bilen, H. Dataset condensation with distribution matching. *arXiv preprint arXiv:2110.04181*, 2021a.

Zhao, B. and Bilen, H. Dataset condensation with differentiable siamese augmentation. *arXiv preprint arXiv:2102.08259*, 2021b.

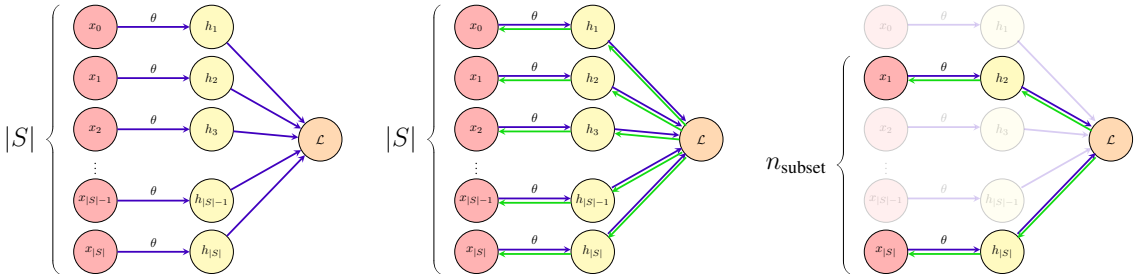
Zhao, B., Mopuri, K. R., and Bilen, H. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=mSAKhLYLSsl.

Zhou, Y., Nezhadarya, E., and Ba, J. Dataset distillation using neural feature regression. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Zhuang, J., Tang, T., Ding, Y., Tatikonda, S., Dvornek, N., Papademetris, X., and Duncan, J. S. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *arXiv preprint arXiv:2010.07468*, 2020.

Zucchet, N., Schug, S., von Oswald, J., Zhao, D., and Sacramento, J. A contrastive rule for meta-learning. *CoRR*, abs/2104.01677, 2021. URL https://arxiv.org/abs/2104.01677.

A. Memory Requirements and Subsampling Ablation



(a) Full forward pass

(b) Standard full backwards pass to all images

(c) Unbiased backwards pass to only subset of images

Figure 4. Our proposed subsampling scheme. First, we perform a full forwards pass using all distilled images in (a). Typically, one performs a backward pass to all distilled images (b), but this requires retaining the entire computation graph which is expensive for large coresets. Instead, we backpropagate only through to $n_{\text{subset}} < |S|$ distilled images (c), using values computed using the full forward pass. Due to the exchangeability of the coreset images, this results in unbiased gradients.

In Section 3, we claimed that we can obtain unbiased gradient estimates by performing a full forward pass on the entire distilled dataset, and only performing a backward pass on a subset. Here we verify this claim by varying the subset size, n_{subset} for the backward pass. We test this for CIFAR-100 with 50 images per class, which results in the largest number of distilled images at 5000 of all the experiments in Section 4. In Section 4, we set n_{subset} to be 2000, 40% of the full dataset, but here we let $n_{\text{subset}} \in \{200, 500, 1000, 2000, 5000\}$, and measure the resulting memory consumption, wall clock time, and resulting accuracy in Figure 5, distilling for 4000 iterations. We see that larger n_{subset} consumes significantly more memory, and that wall clock time is linear in n_{subset}. Despite this, accuracy is only slightly affected, moving from $45.9 \pm 0.4\%$ to 47.3 ± 0.3 for $n_{\text{subset}} = 200$, and $n_{\text{subset}} = 5000$, respectively, despite $n_{\text{subset}} = 5000$ requiring 26.7Gb compared to $n_{\text{subset}} = 200$'s 10.0Gb. The small performance drop could be due to increased gradient variance associated with the estimator, or that the effective number of dataset updates is fewer for small n_{subset}, as only a subset of distilled images is updated per iteration.

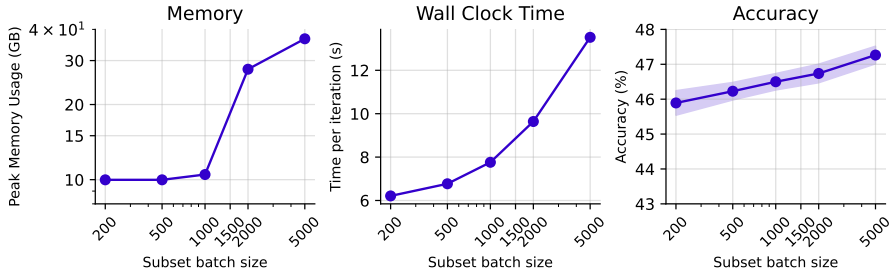


Figure 5. The effect of n_{subset} on memory consumption, wall clock time, and accuracy on CIFAR-100 with 50 Img/Cls. Memory consumption increases significantly with larger n_{subset} and time per iteration is approximately linear in the n_{subset}. In contrast, n_{subset}, has minimal effect on distillation accuracy.

B. Privacy Preservation: Membership Inference Attacks. More details.

For the non-member examples we use the test set of 10k images so that an AUC of 0.5 effectively means that the attack is a random guess. Additionally, we plot the MLP attack AUC and accuracy as a function of training iteration. We report baselines of using the original training set (real), a random subset of 500 images, and the DSA, DM, FRePo dataset distillation algorithms. We also report results on Fashion-MNIST in Figure 6 and Table 6. Results for baseline algortihms are taken from Zhou et al. (2022). We also show the resulting curves for CIFAR-10 and CIFAR-100 for RCIG compared to the real dataset baselines. In this two more complex datasets, there is a better advantage to training with distilled data at a given privacy budget.

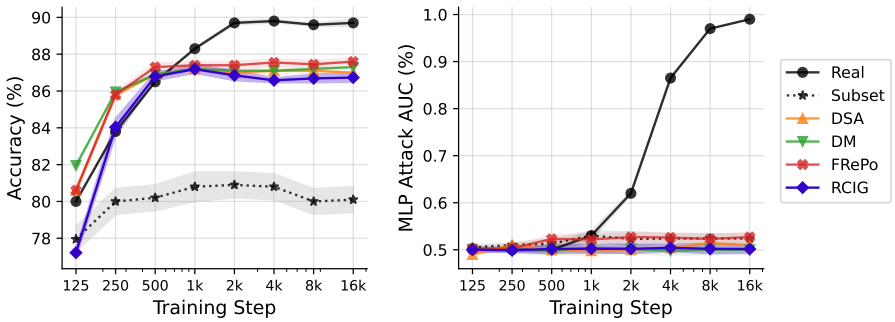


Figure 6. Test accuracy and MLP Attack AUC for models trained on distilled data for Fashion-MNIST. Training on distilled data is not vulnerable to MIA attacks while training on real data leaks information

Table 6. AUC of five MIA attack strategies for neural networks trained on distilled data on Fashion-MNIST. (n=25)

	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
RCIG	86.7 ± 0.3	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.00

C. Implementation details

Libraries and Hardware. Code is implemented using the libraries JAX, Optax, and Flax (Bradbury et al., 2018; Babuschkin et al., 2020; Heek et al., 2020). We use a mix of Nvidia Titan RTXs with 24Gb, RTX 4090s with 24Gb, and Quadro RTX A6000s with 48Gb VRAM. The training time per iteration plots in Figure 1 and Figure 5 are run on an RTX 4090.

Optimizers, Hyperparameter selection. For the λ L_2 regularization term, for depth 3 models we used $\lambda = 0.0005 \times |S|$, for depth 4 we use $\lambda = 0.005 \times |S|$ and depth 5 we use $\lambda = 0.05 \times |S|$. We did not find that this had a major impact in terms of performance, but Hessian-Inverse computation could be unstable if it is too low.

For the coreset optimizer, we use Adabelief (Zhuang et al., 2020) optimizer with learning rate 0.003 for the coreset images and labels, and a learning rate of 0.03 for $\log \tau$, the Platt scaling loss temperature. For inner optimization and Hessian inverse computation, we use Adam optimizer (Kingma & Ba, 2015), with learning rates α_{inner} and $\alpha_{H^{-1}}$, we perform a small tuning round before beginning coreset optimization. We increase both learning rates as high as possible such that the optimization of loss of the inner problem and Hessian ivnerse loss (Equation (7)) are monotonically decreasing (this is done automatically). During the course of training if we observe that either loss diverges, we reduce the corresponding learning rate by 10%. In general these learning rates should be as high as possible such that optimization is stable.

Dataset Preprocessing. We use the same ZCA regularization with regularization strength $\lambda = 0.1$ as Zhou et al. (2022) for RGB datasets and standard preprocessing for grayscale datasets.

Subsampling. As discussed in Section 3 and Appendix A, we can run out of memory for very large datasets. We use subsampling with batch sizes of 2000 for CIFAR-100 with 50 img/cls, and 500 for Tiny-ImageNet 10 img/cls and resized ImageNet. In general this should be as large as possible that fits in the memory.

Evaluation. During evaluation, we train neural networks for 1000 iterations if $|S| = 10$, otherwise for 2000 iterations. We used Adam optimizer with a learning rate of 0.0001. In line with prior work (Zhou et al., 2022), we use a learning rate schedule with 500 iterations of linear warm up following be a cosine decay. During testing evaluation we either use no data augmentation, or we use flip, color, crop, rotate, translate and cutout data augmentation for RGB datasets, or crop, rotate, translate, cutout for grayscale ones. This is the same procedure used in prior work.

Models. Unless otherwise stated, we used the same Convolutional network used in Zhou et al. (2022), which doubles the

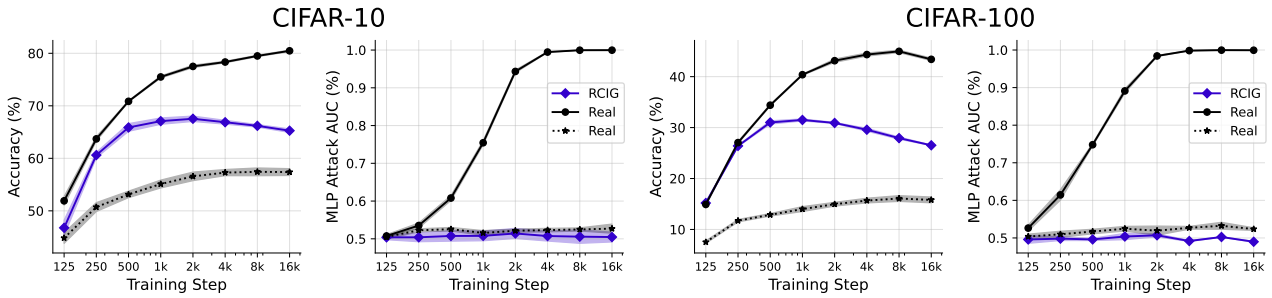


Figure 7. Test accuracy and MLP Attack AUC for models trained on distilled data for CIFAR-10 (left) and CIFAR-100 (right).

number of convolutional filters for every layer. For datasets of resolution 32×32 or 28×28, we use depth 3, for datasets of resolution 64×64 (Tiny-ImageNet, Resized ImageNet), we use depth 4, for 128×128 we use depth 5, in line with prior work. We do not use normalization during training, unless otherwise stated. For models of depth 3, during training with replaced with ReLUs with softplus activations with temperature 60, this did not make a major difference in the final result but sped up convergence, as it allows the gradients to be smoother as opposed to a step function. During evaluation we used standard ReLUs.

Initialization We initialize distilled datasets with samples from the original dataset. We note that for the high resolution datasets, the distilled images still resemble their initializations (see Appendix E). For low resolution datasets, we typically found that random noise initialization slowed convergence speed, but did not significantly affect the final result. However, for high resolution datasets, performance is harmed with noise initialization. For example, for ImageWoof on 10 ipc, we found that performance drops from $42.2 \pm 0.7\%$ with real image initialization to $38.5 \pm 0.6\%$ with random noise initialization. Better initialization strategies could be the subject of future work.

Miscellaneous. We use the same flipping strategy employed in Zhou et al. (2022) to avoid the mirroring effect for RGB datasets. We use this whenever $|S| \leq 1000$. For MNIST 50 img/cls and Fashion-MNIST 50 img/cls, we use 64-bit computation when computing the matrix inverse in Equation (3), as we found otherwise it would be unstable. Note that the network forward pass and everything else is still 32-bit. Every other dataset uses the default 32-bit. For Tables 1 to 4 we repeat distillation for 3 independent coreset initializations, and evaluate each on five networks (15 total evaluations). For MIA results, we perform distillation on 5 independents chunks of the dataset, and evaluate on 5 models for each (25 total evaluations).

C.1. FRePo Code Error

We noted in Table 3 that we achieved different results on FRePo (Zhou et al., 2022) using their public code repository. Additionally we found a bug where in line 202-203 in `https://github.com/yongchao97/FRePo/blob/master/script/eval.py`, where their code sets all evaluation normalization layers to no normalization. The corrected results are in Table 3.

D. Additional Results

D.1. Training Curves

In Figure 8, we report the test accuracy as a function of distillation wall clock time for RCIG and other algorithms at distilling CIFAR-100 with 1 IPC. RCIG quickly exceeds the performance of other algorithms within 100s. This experiment was run on a RTX 4090. In order to account for the stronger GPU and make our times comparable to those in Zhou et al. (2022) we increase all measured times by 40%. We also show the memory requirements of RCIG in Figure 8. RCIG typically requires more memory than other methods due to linearization, but memory requirements do not scale with the number of inner optimization steps or unroll steps.

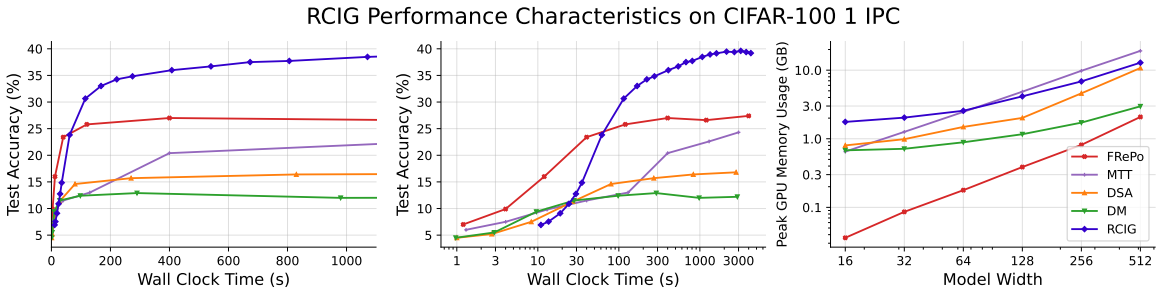


Figure 8. Test accuracy vs. wall clock time for CIFAR-100 distillation with 1 IPC, with linear (left) and log (center) axes, and the memory consumption of RCIG (right) at different model widths. We use $w = 128$ for all other experiments in the paper.

D.2. Total Training Time

In Table 7 we report the total training time for or algorithm on the benchmarks. These experiments were run on Quadro RTX A6000s with 48Gb memory. This memory requirement is not necessary as we can reduce the memory consumption using the techniques described in Section 3 and Appendix A. There is high variance associated with the training times because some of the GPUs were run with reduced power limits.

D.3. Evaluation with/without data augmentation

In Table 8 we report the evaluation accuracy of our algorithm when applying no data augment or data augmentation during test time. We observe that for large distilled datasets ($|S| > 200$), data augmentation seems to help, but hinders performance for smaller ones.

E. Distilled Dataset Visualization

Here we show the resulting distilled dataset made by RCIG.

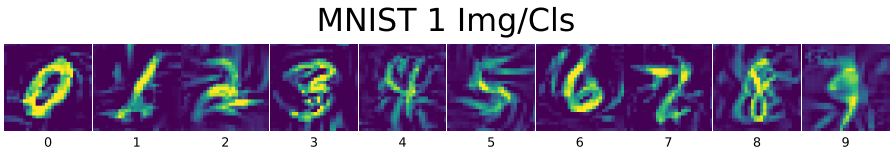


Figure 9. RCIG distilled dataset on MNIST with 1 Img/Cls

	Img/Cls	Total Training Time (h)	Time per iter (s)	Training iterations
MNIST	1	0.67 ± 0.01	0.16 ± 0.00	15000
	10	1.25 ± 0.03	0.45 ± 0.01	10000
	50	2.75 ± 0.00	1.98 ± 0.00	5000
F-MNIST	1	0.67 ± 0.01	0.16 ± 0.00	15000
	10	1.25 ± 0.03	0.45 ± 0.01	10000
	50	2.67 ± 0.06	1.92 ± 0.04	5000
CIFAR-10	1	0.74 ± 0.00	0.18 ± 0.00	15000
	10	1.82 ± 0.07	0.65 ± 0.02	10000
	50	4.09 ± 0.20	2.94 ± 0.14	5000
CIFAR-100	1	1.88 ± 0.07	0.68 ± 0.03	10000
	10	8.55 ± 0.08	6.15 ± 0.06	5000
	50	11.21 ± 0.65	10.09 ± 0.58	4000
T-ImageNet	1	7.78 ± 0.00	5.60 ± 0.00	5000
	10	6.43 ± 0.46	11.58 ± 0.82	2000
CUB-200	1	2.75 ± 0.69	0.99 ± 0.25	10000
	10	9.58 ± 3.22	17.24 ± 5.80	2000
ImageNette	1	5.05 ± 0.23	3.03 ± 0.14	6000
	10	12.62 ± 0.03	15.15 ± 0.04	3000
ImageWoof	1	4.78 ± 0.00	2.87 ± 0.00	6000
	10	11.02 ± 0.02	13.22 ± 0.02	3000
ImageNet	1	4.75 ± 0.34	8.55 ± 0.61	2000
	2	6.87 ± 0.38	12.37 ± 0.69	2000

Table 7. Total runtime of RCIG (n=3)

	Img/Cls	No Data Augmentation	With Data Augmentation
MNIST	1	$\mathbf{94.7 \pm 0.5}$	92.7 ± 0.8
	10	$\mathbf{98.9 \pm 0.0}$	98.7 ± 0.1
	50	99.1 ± 0.1	$\mathbf{99.2 \pm 0.0}$
F-MNIST	1	$\mathbf{79.8 \pm 1.1}$	70.5 ± 3.5
	10	$\mathbf{88.5 \pm 0.2}$	86.7 ± 0.2
	50	$\mathbf{90.2 \pm 0.2}$	88.7 ± 0.2
CIFAR-10	1	$\mathbf{53.9 \pm 1.0}$	50.9 ± 1.5
	10	$\mathbf{69.1 \pm 0.4}$	66.4 ± 1.0
	50	$\mathbf{73.5 \pm 0.3}$	72.6 ± 0.4
CIFAR-100	1	$\mathbf{39.3 \pm 0.4}$	36.7 ± 0.3
	10	$\mathbf{44.1 \pm 0.4}$	43.5 ± 0.3
	50	45.5 ± 0.4	$\mathbf{46.7 \pm 0.3}$
T-ImageNet	1	$\mathbf{25.6 \pm 0.3}$	24.2 ± 0.2
	10	27.4 ± 0.3	$\mathbf{29.4 \pm 0.2}$
CUB-200	1	11.2 ± 0.4	$\mathbf{12.1 \pm 0.2}$
	10	14.3 ± 0.3	$\mathbf{15.7 \pm 0.3}$
ImageNette	1	$\mathbf{53.0 \pm 0.9}$	47.2 ± 1.2
	10	$\mathbf{65.0 \pm 0.7}$	63.9 ± 0.8
ImageWoof	1	$\mathbf{33.9 \pm 0.6}$	27.0 ± 2.1
	10	$\mathbf{42.2 \pm 0.7}$	40.2 ± 0.6
ImageNet	1	14.6 ± 0.2	$\mathbf{15.6 \pm 0.2}$
	2	15.9 ± 0.1	$\mathbf{16.6 \pm 0.1}$

Table 8. Evaluation result of RCIG with or without using data augmentation. For large distilled datasets ($|S| > 200$), data augmentation seems to help, but hinders performance for smaller ones.

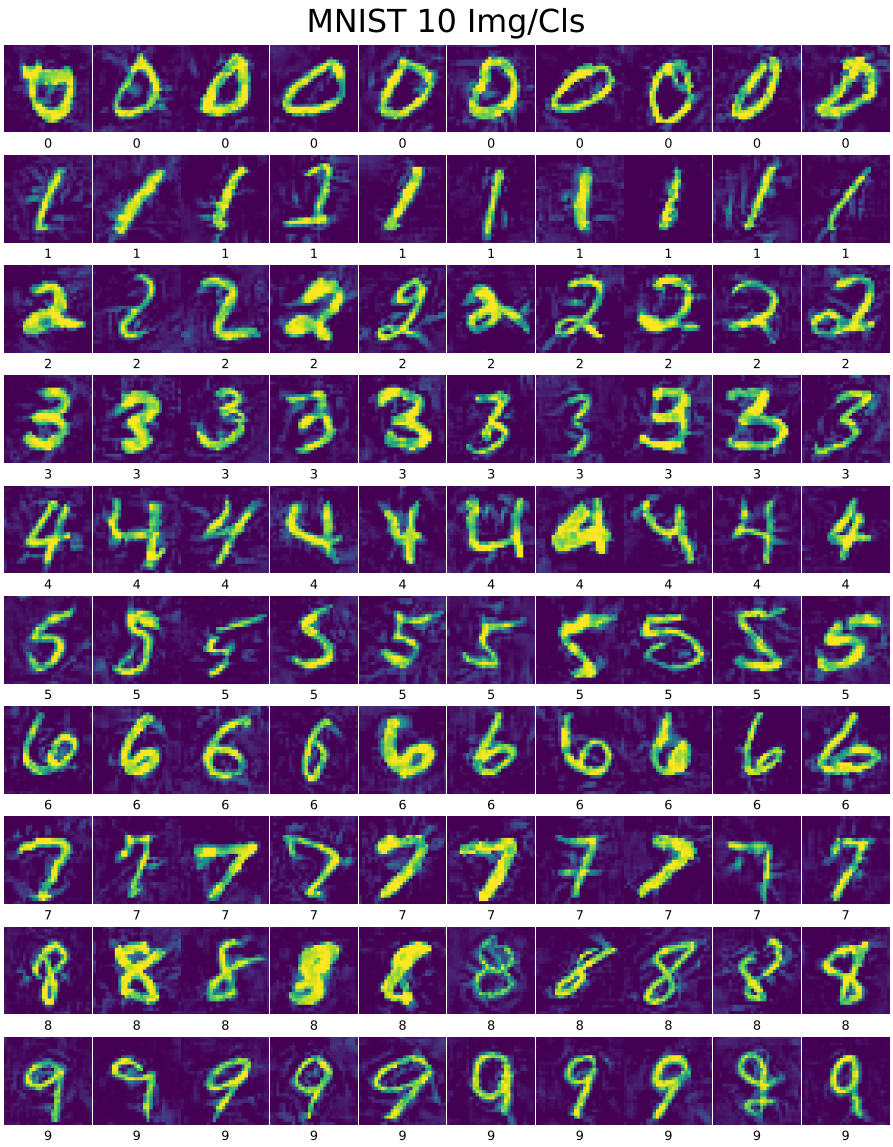


Figure 10. RCIG distilled dataset on MNIST with 10 Img/Cls

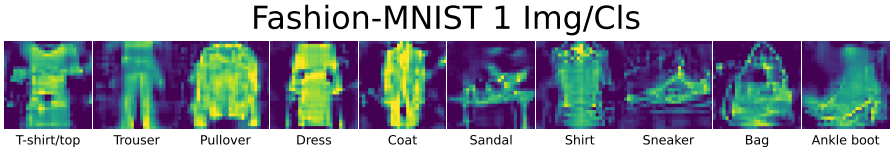


Figure 11. RCIG distilled dataset on Fashion-MNIST with 1 Img/Cls

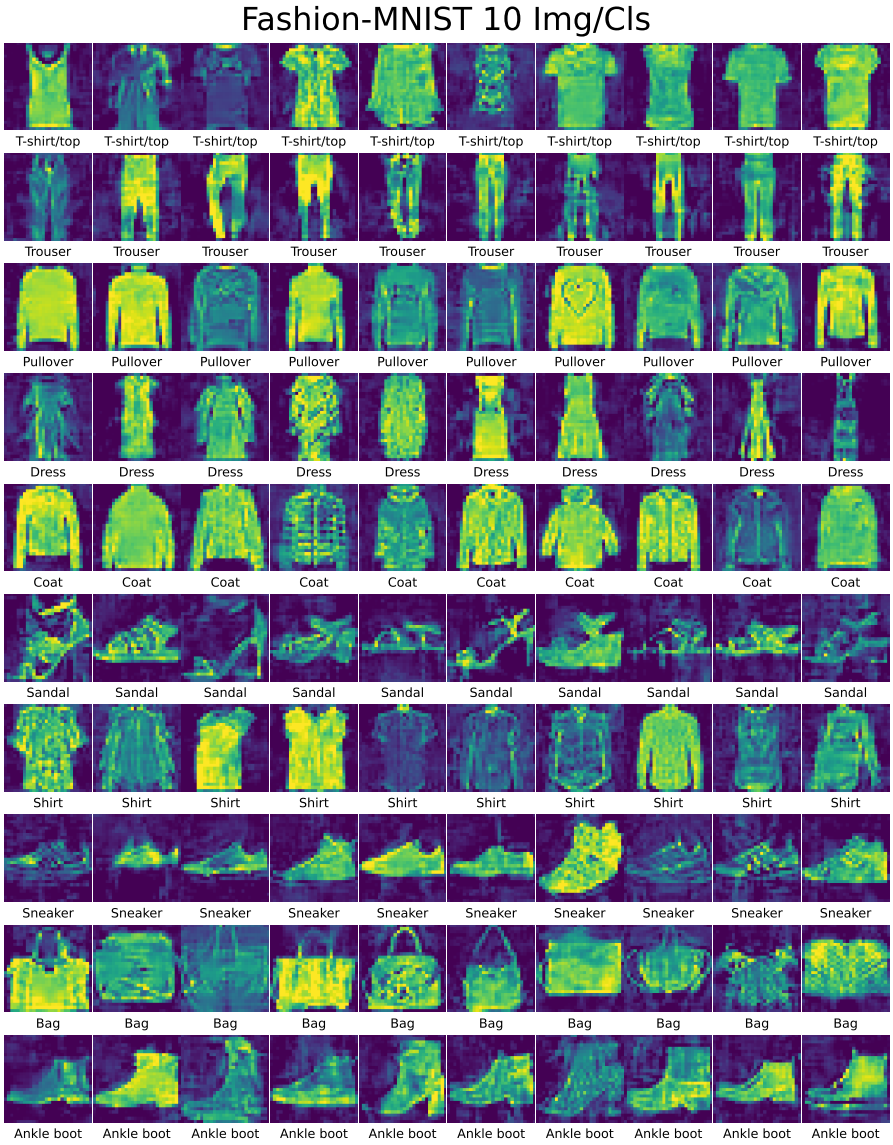


Figure 12. RCIG distilled dataset on Fashion-MNIST with 10 Img/Cls

Figure 13. RCIG distilled dataset on CIFAR-10 with 10 Img/Cls

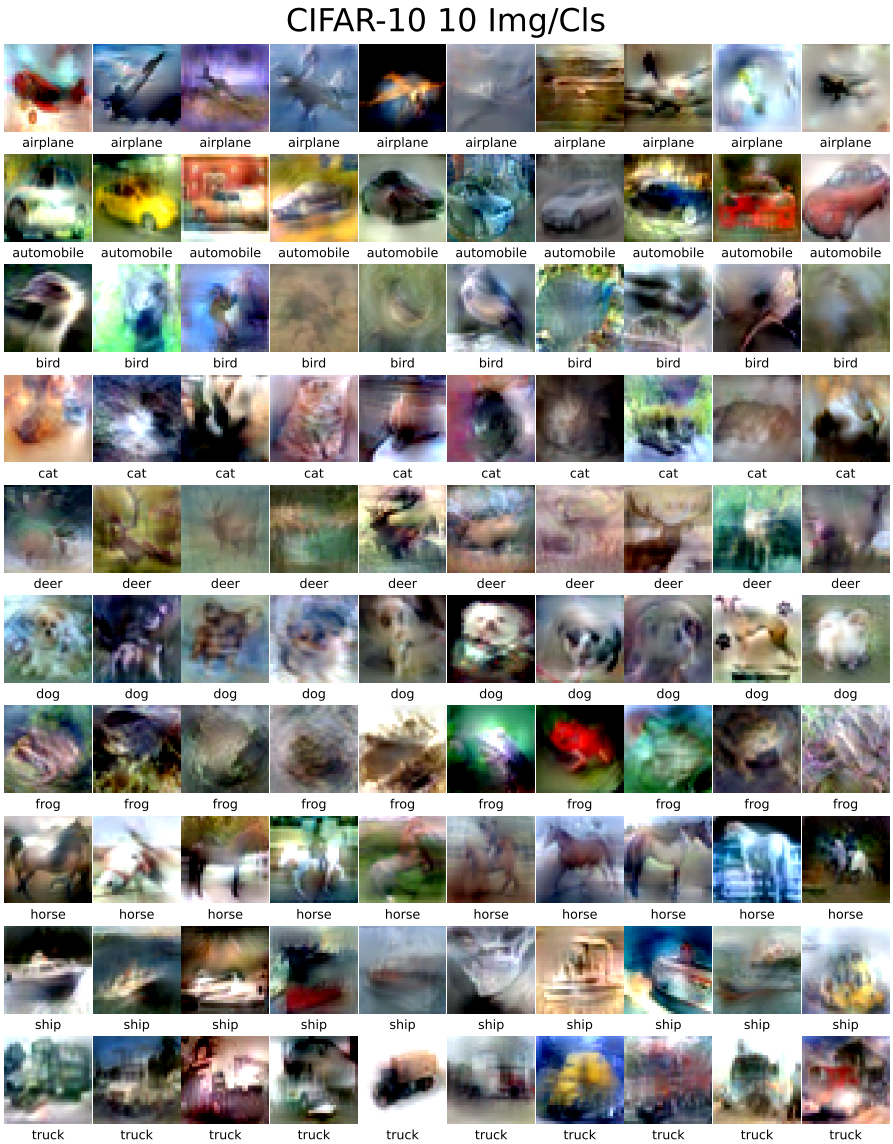


Figure 14. RCIG distilled dataset on CIFAR-10 with 10 Img/Cls

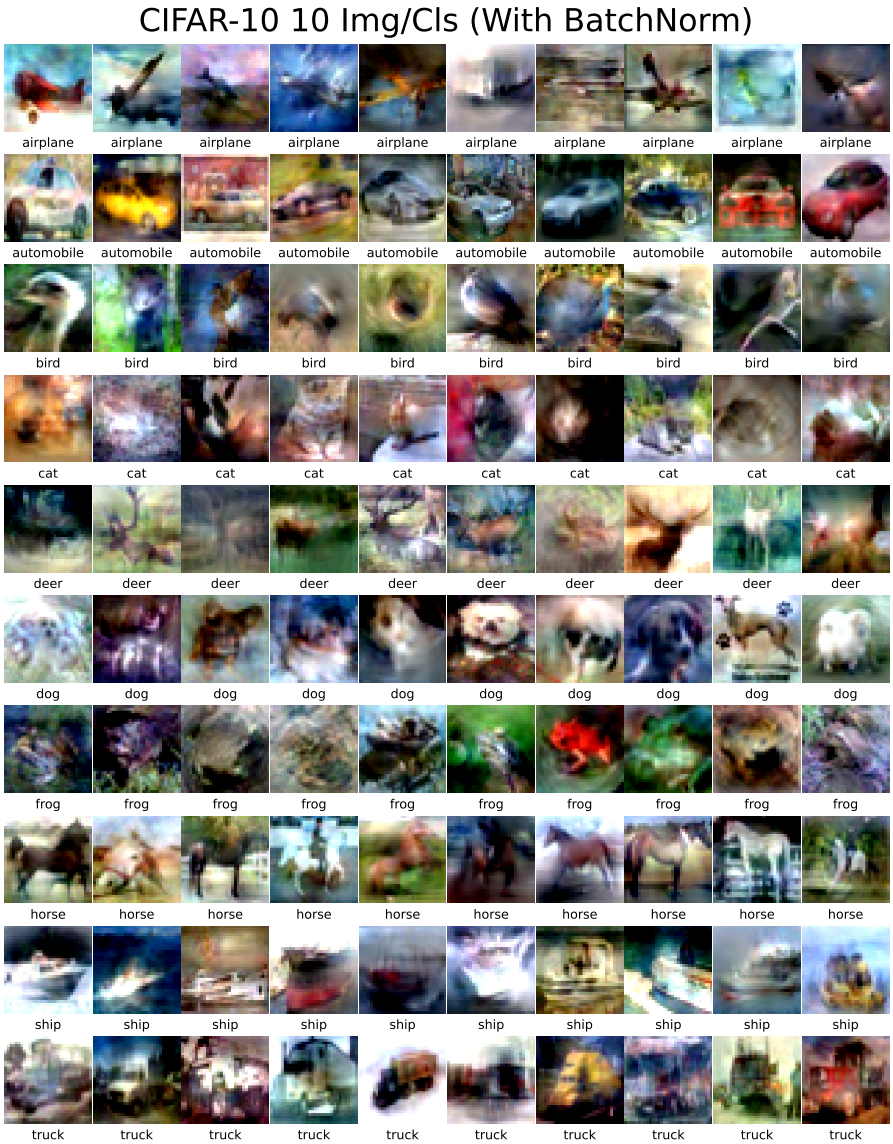


Figure 15. RCIG distilled dataset on CIFAR-10 with 10 Img/Cls (Trained using BatchNorm)

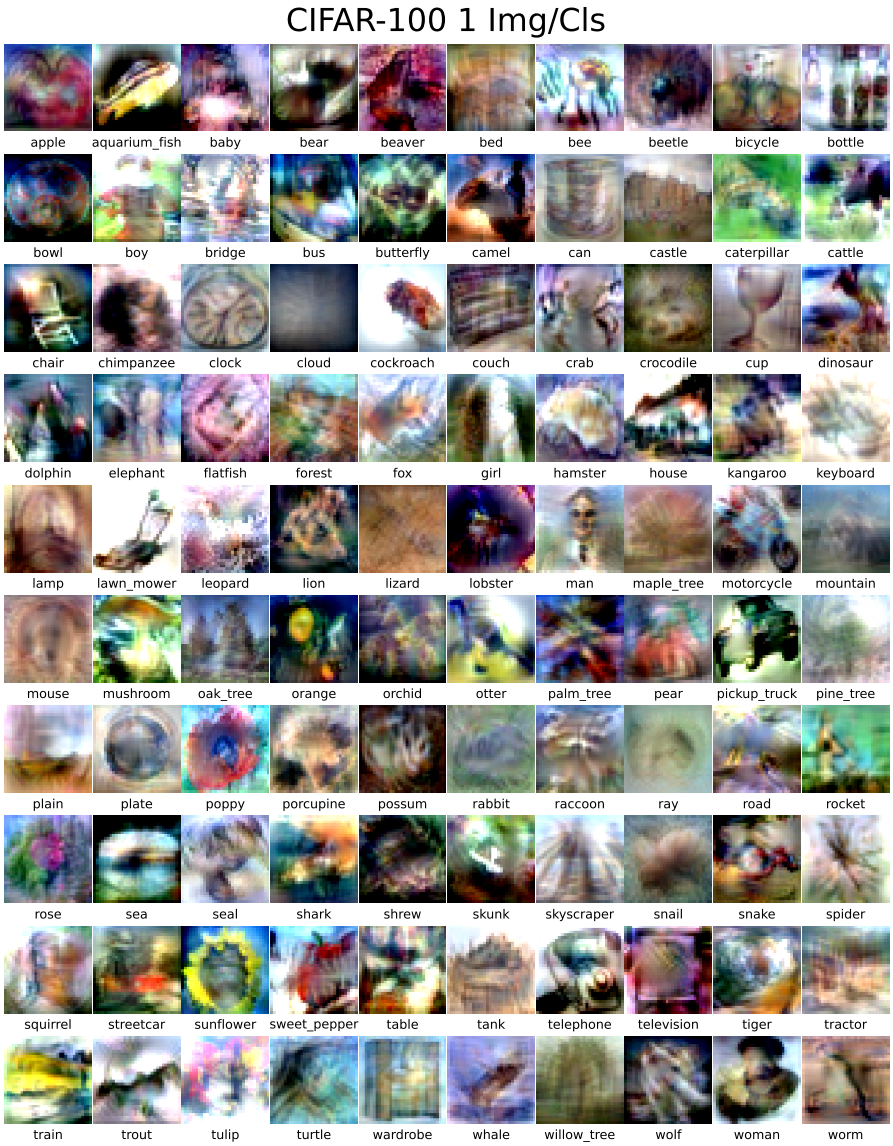


Figure 16. RCIG distilled dataset on CIFAR-100 with 1 Img/Cls

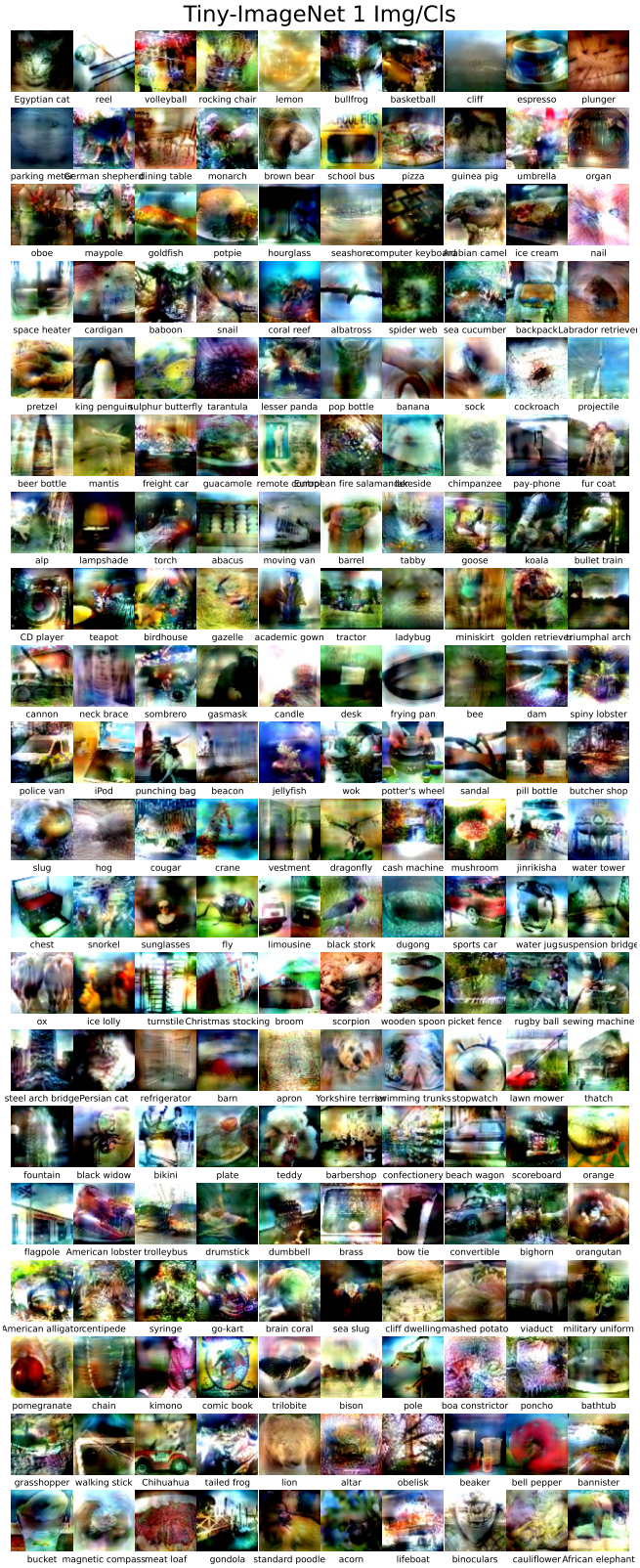


Figure 17. RCIG distilled dataset on Tiny-ImageNet with 1 Img/Cls

Figure 18. RCIG distilled dataset on ImageWoof with 1 Img/Cls

Figure 19. RCIG distilled dataset on ImageNette with 1 Img/Cls