DATASET CONDENSATION WITH GRADIENT MATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

As the state-of-the-art machine learning methods in many fields rely on larger datasets, storing them and training models on them becomes more expensive. This paper proposes a training set synthesis technique for *data-efficient* learning, called *Dataset Condensation*, that learns to condense a large dataset into a small set of informative samples for training deep neural networks from scratch. We formulate this goal as a gradient matching problem between the gradients of a deep neural network trained on the original data and our synthetic data. We rigorously evaluate its performance in several computer vision benchmarks and demonstrate that it significantly outperforms the state-of-the-art methods. Finally we explore the use of our method in continual learning and neural architecture search and show that it achieves promising gains on a tight budget of memory and computations.

1 INTRODUCTION

Large-scale datasets, comprising millions of samples, are becoming the norm to obtain state-of-the-art machine learning models in multiple fields including computer vision, natural language processing and speech recognition. At such scales, even storing and preprocessing the data becomes burdensome, and training machine learning models on them demands for specialized equipment and infrastructure. An effective way to deal with large data is data selection that improves *data efficiency* of machine learning by identifying the most representative training samples. While classical data selection methods, also known as coreset construction (Agarwal et al., 2004; Har-Peled & Mazumdar, 2004; Feldman et al., 2013), focus on clustering problems, recent work can be found in continual learning (Rebuffi et al., 2017; Toneva et al., 2019; Castro et al., 2018; Aljundi et al., 2019) and active learning (Sener & Savarese, 2018) where there is typically a fixed budget in storing and labeling training samples respectively. These methods commonly first define a criterion for representativeness (*e.g.* in terms of compactness (Rebuffi et al., 2017; Castro et al., 2018), diversity (Sener & Savarese, 2018; Aljundi et al., 2019), forgetfulness (Toneva et al., 2019)), and select the representative samples by solving an optimization problem based on the criterion, finally use the selected small set to train their model for a downstream task.

Unfortunately, these methods have two shortcomings: they typically rely on i) heuristics (*e.g.* picking cluster centers) that does not guarantee any optimal solution for the downstream task (*e.g.* image classification), ii) presence of representative samples, which is neither guaranteed. A recent method, Dataset Distillation (DD) (Wang et al., 2018) goes beyond these limitations by *learning* a small set of informative images from large training data. In particular, the authors model the network parameters as a function of the synthetic training data and learn them by minimizing the training loss over the original training data w.r.t. synthetic data. Unlike in the coreset methods, the synthesized data are directly optimized for the downstream task and thus the success of the method does not rely on the presence of representative samples.

Inspired from DD, we focus on learning to *synthesize new samples* that are optimized to train neural networks for downstream tasks and not limited to individual samples in original dataset. Like DD (Wang et al., 2018), our goal is to obtain the highest generalization performance with a model trained on a small set of synthetic images. Ideally, it can obtain comparable performance to that of a model trained on the original images (see Figure 1(a)). In particular, we investigate the following questions. Is it possible to i) compress a large image classification dataset into a small synthetic set, ii) train an image classification model on the synthetic set that can be further used to classify real images, iii) learn a single set of synthetic images that can be used to train different neural network architectures? To this end, we propose a *Dataset Condensation* method to learn a small set of

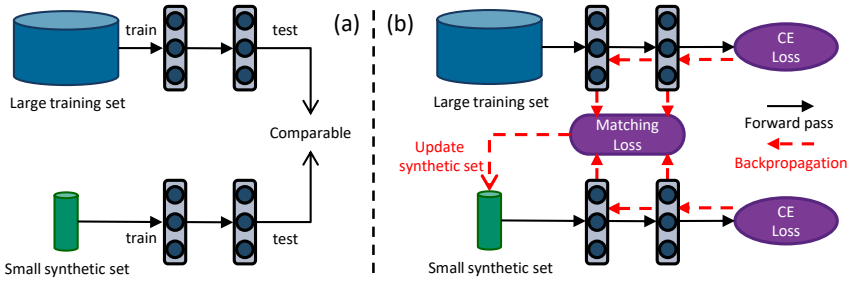


Figure 1: Dataset Condensation (left) aims to generate a small set of synthetic images that can match the performance of a network trained on a large image dataset. Our method (right) realizes this goal by learning a synthetic set such that a deep network trained on it and the large set produces similar gradients w.r.t. the parameters. The synthetic data can later be used to train a network from scratch in a fraction of the original computational load. CE denotes Cross-Entropy.

"condensed" synthetic samples such that a deep network trained on them obtains not only similar performance but also a close solution to a network trained on the large training data in the network parameter space. We formulate this goal as a minimization problem between two sets of gradients for the network parameters that are computed for a training loss over a large fixed training set and a learnable condensed set (see Figure 1(b)). We show that our method enables a more effective learning of synthetic images and neural networks trained on them outperforms (Wang et al., 2018) and coreset methods with a wide margin in multiple computer vision benchmarks. In addition, learning a compact set of synthetic samples also benefits other learning problems when there is a fixed budget on training images. We show that our method outperforms popular data selection methods by providing more informative training samples in continual learning. Finally, we explore a promising use case of our method in neural architecture search, and show that – once our condensed images are learned – they can be used to train numerous network architectures extremely efficiently.

Our method is related to knowledge distillation (KD) (Hinton et al., 2015) techniques (Buciluǎ et al., 2006; Ba & Caruana, 2014; Romero et al., 2014) that transfer the knowledge of an ensemble of models to a single one. Unlike KD, we distill knowledge of a large training set into a small synthetic set. Our method is also related to Generative Adversarial Networks (Goodfellow et al., 2014a; Mirza & Osindero, 2014; Radford et al., 2015) and Variational AutoEncoders (Kingma & Welling, 2013) that synthesize high fidelity samples by capturing the data distribution. In contrast, our goal is to generate informative samples for training deep neural networks rather than to produce "real-looking" samples. Finally our method is related to the methods that recover images by projecting the feature activations back to the input pixel space (Zeiler & Fergus, 2014), reconstruct the input image by matching the feature activations (Mahendran & Vedaldi, 2015), recover private training images for given training gradients (Zhu et al., 2019; Zhao et al., 2020), synthesize features from semantic embeddings for zero-shot learning (Sariyildiz & Cinbis, 2019). Our goal is however to synthesize a set of condensed training images not to recover the original or missing training images.

In the remainder of this paper, we first review the problem of dataset condensation and introduce our method in section 2, present and analyze our results in several image recognition benchmarks in section 3.1, showcase applications in continual learning and network architecture search in section 3.2, and conclude the paper with future directions in section 4.

2 Method

2.1 Dataset Condensation

Suppose we are given a large dataset consisting of m pairs of a training image and its class label $\mathcal{T} = \{(\boldsymbol{x}_i, y_i)\}|_{i=1}^{m}$ where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$, $y \in \{1, \dots, C\}$, $\mathcal{X}$ is a d-dimensional input space and C is the number of classes. We wish to learn a differentiable function ϕ (*i.e.* deep neural network) with parameters $\boldsymbol{\theta}$ that correctly predicts labels of previously unseen images, *i.e.* $y = \phi_{\boldsymbol{\theta}}(\boldsymbol{x})$. One can learn the parameters of this function by minimizing an empirical loss term over the training set:

$$\boldsymbol{\theta}^{\mathcal{T}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}) \tag{1}$$

where $\mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}) = \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \ell(\phi_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$, $\ell(\cdot,\cdot)$ is a task specific loss (*i.e.* cross-entropy) and $\boldsymbol{\theta}^{\mathcal{T}}$ is the minimizer of $\mathcal{L}^{\mathcal{T}}$. The generalization performance of the obtained model $\phi_{\boldsymbol{\theta}^{\mathcal{T}}}$ can be written as $\mathbb{E}_{\boldsymbol{x}\sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{T}}}(\boldsymbol{x}), y)]$ where $P_{\mathcal{D}}$ is the data distribution. Our goal is to generate a small set of condensed synthetic samples with their labels, $\mathcal{S} = \{(\boldsymbol{s}_i, y_i)\}|_{i=1}^{n}$ where $\boldsymbol{s} \in \mathbb{R}^d$ and $y \in \mathcal{Y}$, $n \ll m$. Similar to eq. (1), once the condensed set is learned, one can train ϕ on them as follows

$$\boldsymbol{\theta}^{\mathcal{S}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}) \tag{2}$$

where $\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}) = \sum_{(\boldsymbol{s},y)\in\mathcal{S}} \ell(\phi_{\boldsymbol{\theta}}(\boldsymbol{s}), y)$ and $\boldsymbol{\theta}^{\mathcal{S}}$ is the minimizer of $\mathcal{L}^{\mathcal{S}}$. As the synthetic set $\mathcal{S}$ is significantly smaller (2-3 orders of magnitude), we expect the optimization in eq. (2) to be significantly faster than that in eq. (1). We also wish that the generalization performance of $\phi_{\boldsymbol{\theta}^{\mathcal{S}}}$ as close as possible to $\phi_{\boldsymbol{\theta}^{\mathcal{T}}}$, *i.e.* $\mathbb{E}_{\boldsymbol{x}\sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{T}}}(\boldsymbol{x}), y)] \simeq \mathbb{E}_{\boldsymbol{x}\sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{S}}}(\boldsymbol{x}), y)]$ over the real data distribution $P_{\mathcal{D}}$.

Discussion. The goal of obtaining comparable generalization performance by training on the condensed data can be formulated in different ways. One approach, which is proposed in (Wang et al., 2018) and improved in (Sucholutsky & Schonlau, 2019), is to pose the parameters $\boldsymbol{\theta}^{\mathcal{S}}$ as a function of the synthetic data $\mathcal{S}$:

$$\mathcal{S}^* = \arg\min_{\mathcal{S}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S})) \qquad \text{subject to} \qquad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}). \tag{3}$$

The method aims to find the optimum set of synthetic images $\mathcal{S}^*$ such that the model $\phi_{\boldsymbol{\theta}^{\mathcal{S}}}$ trained on them minimizes the training loss over the original data. Optimizing eq. (3) involves a nested loop optimization and solving the inner loop for $\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S})$ at each iteration to recover the gradients for $\mathcal{S}$ which requires a computationally expensive procedure – unrolling the recursive computation graph for $\mathcal{S}$ over multiple optimization steps for $\boldsymbol{\theta}$ (see (Samuel & Tappen, 2009; Domke, 2012)). Hence, it does not scale to large models and/or accurate inner-loop optimizers with many steps. Next we propose an alternative formulation for dataset condensation.

2.2 Dataset condensation with parameter matching

Here we aim to learn $\mathcal{S}$ such that the model $\phi_{\boldsymbol{\theta}^{\mathcal{S}}}$ trained on them achieves not only comparable generalization performance to $\phi_{\boldsymbol{\theta}^{\mathcal{T}}}$ but also converges to a similar solution in the parameter space (*i.e.* $\boldsymbol{\theta}^{\mathcal{S}} \approx \boldsymbol{\theta}^{\mathcal{T}}$). Let $\phi_{\boldsymbol{\theta}}$ be a locally smooth function[1], similar weights ($\boldsymbol{\theta}^{\mathcal{S}} \approx \boldsymbol{\theta}^{\mathcal{T}}$) imply similar mappings in a local neighborhood and thus generalization performance, *i.e.* $\mathbb{E}_{\boldsymbol{x}\sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{T}}}(\boldsymbol{x}), y)] \simeq \mathbb{E}_{\boldsymbol{x}\sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{S}}}(\boldsymbol{x}), y)]$. Now we can formulate this goal as

$$\min_{\mathcal{S}} D(\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\theta}^{\mathcal{T}}) \quad \text{subject to} \quad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}) \tag{4}$$

where $\boldsymbol{\theta}^{\mathcal{T}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta})$ and $D(\cdot,\cdot)$ is a distance function. In a deep neural network, $\boldsymbol{\theta}^{\mathcal{T}}$ typically depends on its initial values $\boldsymbol{\theta}_0$. However, the optimization in eq. (4) aims to obtain an optimum set of synthetic images only for one model $\phi_{\boldsymbol{\theta}^{\mathcal{T}}}$ with the initialization $\boldsymbol{\theta}_0$, while our actual goal is to generate samples that can work with a distribution of random initializations $P_{\boldsymbol{\theta}_0}$. Thus we modify eq. (4) as follows:

$$\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0\sim P_{\boldsymbol{\theta}_0}}[D(\boldsymbol{\theta}^{\mathcal{S}}(\boldsymbol{\theta}_0), \boldsymbol{\theta}^{\mathcal{T}}(\boldsymbol{\theta}_0))] \quad \text{subject to} \quad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}(\boldsymbol{\theta}_0)) \tag{5}$$

where $\boldsymbol{\theta}^{\mathcal{T}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}(\boldsymbol{\theta}_0))$. For brevity, we use only $\boldsymbol{\theta}^{\mathcal{S}}$ and $\boldsymbol{\theta}^{\mathcal{T}}$ to indicate $\boldsymbol{\theta}^{\mathcal{S}}(\boldsymbol{\theta}_0)$ and $\boldsymbol{\theta}^{\mathcal{T}}(\boldsymbol{\theta}_0)$ respectively in the next sections. The standard approach to solving eq. (5) employs implicit differentiation (see (Domke, 2012) for details), which involves solving an inner loop optimization for $\boldsymbol{\theta}^{\mathcal{S}}$. As the inner loop optimization $\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta})$ can be computationally expensive in case of large-scale models, one can adopt the back-optimization approach in (Domke, 2012) which re-defines $\boldsymbol{\theta}^{\mathcal{S}}$ as the output of an incomplete optimization:

$$\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}), \varsigma) \tag{6}$$

where `opt-alg` is a specific optimization procedure with a fixed number of steps (ς).

[1]Local smoothness is frequently used to obtain explicit first-order local approximations in deep networks (*e.g.* see (Rifai et al., 2012; Goodfellow et al., 2014b; Koh & Liang, 2017)).

In practice, $\boldsymbol{\theta}^{\mathcal{T}}$ for different initializations can be trained first in an offline stage and then used as the target parameter vector in eq. (5). However, there are two potential issues by learning to regress $\boldsymbol{\theta}^{\mathcal{T}}$ as the target vector. First the distance between $\boldsymbol{\theta}^{\mathcal{T}}$ and intermediate values of $\boldsymbol{\theta}^{\mathcal{S}}$ can be too big in the parameter space with multiple local minima traps along the path and thus it can be too challenging to reach. Second `opt-alg` involves a limited number of optimization steps as a trade-off between speed and accuracy which may not be sufficient to take enough steps for reaching the optimal solution. These problems are similar to those of (Wang et al., 2018), as they both involve parameterizing $\boldsymbol{\theta}^{\mathcal{S}}$ with $\mathcal{S}$ and $\boldsymbol{\theta}_0$.

2.3 Dataset condensation with curriculum gradient matching

Here we propose a curriculum based approach to address the above mentioned challenges. The key idea is that we wish $\boldsymbol{\theta}^{\mathcal{S}}$ to be close to not only the final $\boldsymbol{\theta}^{\mathcal{T}}$ but also to follow a similar path to $\boldsymbol{\theta}^{\mathcal{T}}$ throughout the optimization. While this can restrict the optimization dynamics for $\boldsymbol{\theta}$, we argue that it also enables a more guided optimization and effective use of the incomplete optimizer. We can now rewrite eq. (5) as a sum of subproblems:

$$\begin{gathered}\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}}[\sum_{t=1}^{T} D(\boldsymbol{\theta}_t^{\mathcal{S}}, \boldsymbol{\theta}_t^{\mathcal{T}})] \quad \text{subject to} \\ \boldsymbol{\theta}_t^{\mathcal{S}}(\mathcal{S}) = \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_{t-1}^{\mathcal{S}}), \varsigma^{\mathcal{S}}) \quad \text{and} \quad \boldsymbol{\theta}_t^{\mathcal{T}} = \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_{t-1}^{\mathcal{T}}), \varsigma^{\mathcal{T}})\end{gathered} \tag{7}$$

where T is the number of iterations, $\varsigma^{\mathcal{S}}$ and $\varsigma^{\mathcal{T}}$ are the numbers of optimization steps for $\boldsymbol{\theta}^{\mathcal{S}}$ and $\boldsymbol{\theta}^{\mathcal{T}}$ respectively. In words, we wish to generate a set of condensed samples $\mathcal{S}$ such that the network parameters trained on them ($\boldsymbol{\theta}_t^{\mathcal{S}}$) are similar to the ones trained on the original training set ($\boldsymbol{\theta}_t^{\mathcal{T}}$) at each iteration t. In our preliminary experiments we observe that $\boldsymbol{\theta}_t^{\mathcal{S}}$, which is parameterized with $\mathcal{S}$, can successfully track $\boldsymbol{\theta}_t^{\mathcal{T}}$ by updating $\mathcal{S}$ and minimize $D(\boldsymbol{\theta}_{t-1}^{\mathcal{S}}, \boldsymbol{\theta}_{t-1}^{\mathcal{T}})$ close to zero.

In the case of one step gradient descent optimization for `opt-alg`, the update rule is:

$$\boldsymbol{\theta}_t^{\mathcal{S}} \leftarrow \boldsymbol{\theta}_{t-1}^{\mathcal{S}} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_{t-1}^{\mathcal{S}}) \quad \text{and} \quad \boldsymbol{\theta}_t^{\mathcal{T}} \leftarrow \boldsymbol{\theta}_{t-1}^{\mathcal{T}} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_{t-1}^{\mathcal{T}}), \tag{8}$$

where $\eta_{\boldsymbol{\theta}}$ is the learning rate. Based on our observation ($D(\boldsymbol{\theta}_{t-1}^{\mathcal{S}}, \boldsymbol{\theta}_{t-1}^{\mathcal{T}}) \approx 0$), we simplify the formulation in eq. (7) by replacing $\boldsymbol{\theta}_{t-1}^{\mathcal{T}}$ with $\boldsymbol{\theta}_{t-1}^{\mathcal{S}}$ and use a single symbol $\boldsymbol{\theta}$ to denote $\boldsymbol{\theta}^{\mathcal{S}}$ in the rest of the paper:

$$\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}}[\sum_{t=1}^{T} D(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_{t-1}), \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_{t-1}))]. \tag{9}$$

We now have a single deep network with parameters $\boldsymbol{\theta}$ trained on the synthetic set $\mathcal{S}$ which is optimized such that the distance between the gradients for the loss over the training samples $\mathcal{L}^{\mathcal{T}}$ w.r.t. $\boldsymbol{\theta}$ and the gradients for the loss over the condensed samples $\mathcal{L}^{\mathcal{S}}$ w.r.t. $\boldsymbol{\theta}$ is minimized. In words, our goal reduces to matching the gradients for the real and synthetic training loss w.r.t. $\boldsymbol{\theta}$ via updating the condensed samples. This approximation has the key advantage over (Wang et al., 2018) and eq. (5) that it does not require the expensive unrolling of the recursive computation graph over the previous parameters $\{\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_{t-1}\}$. The important consequence is that the optimization is significantly faster, memory efficient and thus scales up to the state-of-the-art deep neural networks (*e.g.* ResNet (He et al., 2016)).

Discussion. The synthetic data contains not only samples but also their labels $(\boldsymbol{s}, y)$ that can be jointly learned by optimizing eq. (9) in theory. However, their joint optimization is challenging, as the content of the samples depend on their label and vice-versa. Thus in our experiments we learn to synthesize images for fixed labels, *e.g.* one synthetic image per class.

Algorithm. We depict the optimization details in Alg. 1. At the outer level, it contains a loop over random weight initializations, as we want to obtain condensed images that can later be used to train previously unseen models. Once $\boldsymbol{\theta}$, which denotes $\boldsymbol{\theta}^{\mathcal{S}}$ for brevity, is randomly initialized, we use $\phi_{\boldsymbol{\theta}}$ to first compute the loss over both the training samples ($\mathcal{L}^{\mathcal{T}}$), synthetic samples ($\mathcal{L}^{\mathcal{S}}$) and their gradients w.r.t. $\boldsymbol{\theta}$, then optimize the synthetic samples to match these gradients $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}$ to $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}$ by applying $\varsigma_{\mathcal{S}}$ gradient descent steps with learning rate $\eta_{\mathcal{S}}$. We use the stochastic gradient descent optimization for both $\texttt{opt-alg}_{\boldsymbol{\theta}}$ and $\texttt{opt-alg}_{\mathcal{S}}$. Next we train $\boldsymbol{\theta}$ on the updated synthetic images by minimizing the loss $\mathcal{L}^{\mathcal{S}}$ with learning rate $\eta_{\boldsymbol{\theta}}$ for $\varsigma_{\boldsymbol{\theta}}$ steps. Note that each real and synthetic batch

sampled from $\mathcal{T}$ and $\mathcal{S}$ contains samples from a single class and the synthetic data for each class are separately updated at each iteration (t) for the following reasons: i) this reduces the memory use at train time, ii) imitating the mean gradients w.r.t. the data from single class is easier compared to that of multiple classes. This does not bring any extra computational cost.

Algorithm 1: Dataset condensation with gradient matching

Input: Training set $\mathcal{T}$

Required: Randomly initialized set of synthetic samples $\mathcal{S}$, probability distribution over randomly initialized weights $P_{\boldsymbol{\theta}_0}$, deep neural network $\phi_{\boldsymbol{\theta}}$, number of outer-loop steps K, number of inner-loop steps T, number of steps for updating weights $\varsigma_{\boldsymbol{\theta}}$ and synthetic samples $\varsigma_{\mathcal{S}}$ in each inner-loop step respectively, learning rates for updating weights $\eta_{\boldsymbol{\theta}}$ and synthetic samples $\eta_{\mathcal{S}}$.

for $k = 0, \cdots, K-1$ **do**

  Initialize $\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}$

  for $t = 0, \cdots, T-1$ **do**

    Sample a minibatch $B^{\mathcal{T}} \sim \mathcal{T}$ and $B^{\mathcal{S}} \sim \mathcal{S}$

    Compute $\mathcal{L}^{\mathcal{T}} = \sum_{(\boldsymbol{x},y)\in B^{\mathcal{T}}} \ell(\phi_{\boldsymbol{\theta}_t}(\boldsymbol{x}), y)$ and $\mathcal{L}^{\mathcal{S}} = \sum_{(\boldsymbol{s},y)\in B^{\mathcal{S}}} \ell(\phi_{\boldsymbol{\theta}_t}(\boldsymbol{s}), y)$

    Update synthetic samples: $\mathcal{S} \leftarrow \texttt{opt-alg}_{\mathcal{S}}(D(\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}}\mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_t)), \varsigma_{\mathcal{S}}, \eta_{\mathcal{S}})$

    Update model parameters: $\boldsymbol{\theta}_{t+1} \leftarrow \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \varsigma_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\theta}})$

Output: $\mathcal{S}$

Gradient matching loss. The matching loss $D(\cdot,\cdot)$ in eq. (9) measures the distance between the gradients for $\mathcal{L}^{\mathcal{S}}$ and $\mathcal{L}^{\mathcal{T}}$ w.r.t. $\boldsymbol{\theta}$. When $\phi_{\boldsymbol{\theta}}$ is a multi-layered neural network, the gradients correspond to a set of learnable 2D (`out`$\times$`in`) and 4D (`out`$\times$`in`$\times$`h`$\times$`w`) weights for each fully connected (FC) and convolutional layer resp where `out`, `in`, `h`, `w` are number of output and input channels, kernel height and width resp. The matching loss can be decomposed into a sum of layerwise losses as $D(\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\mathcal{S}}, \nabla_{\boldsymbol{\theta}}\mathcal{L}^{\mathcal{T}}) = \sum_{l=1}^{L} d(\nabla_{\boldsymbol{\theta}^{(l)}}\mathcal{L}^{\mathcal{S}}, \nabla_{\boldsymbol{\theta}^{(l)}}\mathcal{L}^{\mathcal{T}})$ where l is the layer index, L is the number of layers with weights and

$$d(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{\texttt{out}} \left(1 - \frac{\mathbf{A_{i\cdot}} \cdot \mathbf{B_{i\cdot}}}{\|\mathbf{A_{i\cdot}}\| \|\mathbf{B_{i\cdot}}\|}\right) \tag{10}$$

where $\mathbf{A}_{i\cdot}$ and $\mathbf{B}_{i\cdot}$ are flattened vectors of gradients corresponding to each output node *i.e.* `in` dimensional for FC weights and `in`$\times$`h`$\times$`w` dimensional for convolutional weights. In contrast to (Lopez-Paz et al., 2017; Aljundi et al., 2019; Zhu et al., 2019) that ignore the layer-wise structure by flattening tensors over all layers to one vector and then computing the distance between two vectors, we group them for each output node. We found that this is a better distance for gradient matching and enables using a single learning rate across all layers in our preliminary experiments.

3 EXPERIMENTS

3.1 DATASET CONDENSATION

First we evaluate classification performance with the condensed images on four standard benchmark datasets: digit recognition on MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011) and object classification on FashionMNIST (Xiao et al., 2017), CIFAR10 (Krizhevsky et al., 2009). We test our method using six standard deep network architectures: MLP, ConvNet (Gidaris & Komodakis, 2018), LeNet (LeCun et al., 1998), AlexNet (Krizhevsky et al., 2012), VGG-11 (Simonyan & Zisserman, 2014) and ResNet-18 (He et al., 2016). MLP is a multilayer perceptron with two nonlinear hidden layers, each has 128 units. ConvNet is a commonly used modular architecture in few-shot learning (Snell et al., 2017; Vinyals et al., 2016; Gidaris & Komodakis, 2018) with D duplicate blocks, and each block has a convolutional layer with W (3×3) filters, a normalization layer N, an activation layer A and a pooling layer P, denoted as $[W, N, A, P] \times D$. The default ConvNet (unless specified otherwise) includes 3 blocks, each with 128 filters, followed by InstanceNorm (Ulyanov et al., 2016), ReLU and AvgPooling modules. The final block is followed by a linear classifier. We use Kaiming initialization (He et al., 2015) for network weights. More details about the datasets and networks can be found in the supplementary. The code will be released after review period.

The pipeline for dataset condensation has two stages: learning the condensed images (denoted as `C`) and training classifiers from scratch on them (denoted as `T`). Note that the model architectures

	Img/Cls	Ratio %	Coreset Selection				Whole Dataset
			Random	Herding	K-Center	Forgetting	Ours	
MNIST	1	0.017	64.9±3.5	89.2±1.6	89.3±1.5	35.5±5.6	**91.7±0.5**	99.6±0.0
	10	0.17	95.1±0.9	93.7±0.3	84.4±1.7	68.1±3.3	**97.4±0.2**	
FashionMNIST	1	0.017	51.4±3.8	67.0±1.9	66.9±1.8	42.0±5.5	**70.5±0.6**	93.5±0.1
	10	0.17	73.8±0.7	71.1±0.7	54.7±1.5	53.9±2.0	**82.3±0.4**	
SVHN	1	0.014	14.6±1.6	20.9±1.3	21.0±1.5	12.1±1.7	**31.2±1.4**	95.4±0.1
	10	0.14	35.1±4.1	50.5±3.3	14.0±1.3	16.8±1.2	**76.1±0.6**	
CIFAR10	1	0.02	14.4±2.0	21.5±1.2	21.5±1.3	13.5±1.2	**28.3±0.5**	84.8±0.1
	10	0.2	26.0±1.2	31.6±0.7	14.7±0.9	23.3±1.0	**44.9±0.5**	

Table 1: The performance comparison to coreset methods. This table shows the testing accuracies (%) of different methods on four datasets. ConvNet is used for training and testing. Img/Cls: image(s) per class, Ratio (%): the ratio of condensed images to whole training set.

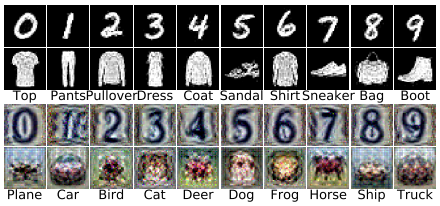


Figure 2: Visualization of condensed 1 image/class with ConvNet for MNIST, FashionMNIST, SVHN and CIFAR10.

C\T	MLP	ConvNet	LeNet	AlexNet	VGG	ResNet
MLP	70.5±1.2	63.9±6.5	77.3±5.8	70.9±11.6	53.2±7.0	80.9±3.6
ConvNet	69.6±1.6	**91.6±0.5**	85.3±1.8	85.1±3.0	**83.4±1.8**	**90.0±0.8**
LeNet	71.0±1.6	90.3±1.2	85.0±1.7	84.7±2.4	80.3±2.7	89.0±0.8
AlexNet	72.1±1.7	87.5±1.6	84.0±2.8	82.7±2.9	81.2±3.0	88.9±1.1
VGG	70.3±1.6	90.1±0.7	83.9±2.7	83.4±3.7	81.7±2.6	89.1±0.9
ResNet	**73.6±1.2**	**91.6±0.5**	**86.4±1.5**	**85.4±1.9**	**83.4±2.4**	89.4±0.9

Table 2: Cross-architecture performance in accuracy (%) for condensed 1 image/class in MNIST.

used in two stages might be different. For the coreset baselines, the coreset is selected in the first stage. We investigate two settings: 1 and 10 image/class learning, which means that the condensed set or coreset contains 1 and 10 images per class respectively. Each method is run for 5 times, and 5 synthetic sets are generated in the first stage; each generated synthetic set is used to train 20 randomly initialized models in the second stage and evaluated on the test set, which amounts to evaluating 100 models in the second stage. In all experiments, we report the mean and standard deviation of these 100 testing results.

Baselines. We compare our method to four coreset baselines (Random, Herding, K-Center and Forgetting) and also to DD (Wang et al., 2018). In Random, the training samples are randomly selected as the coreset. Herding baseline, which selects closest samples to the cluster center, is based on (Welling, 2009) and used in (Rebuffi et al., 2017; Castro et al., 2018; Wu et al., 2019; Belouadah & Popescu, 2020). K-Center (Wolf, 2011; Sener & Savarese, 2018) picks multiple center points such that the largest distance between a data point and its nearest center is minimized. For Herding and K-Center, we use models trained on the whole dataset to extract features, compute l_2 distance to centers. Forgetting method (Toneva et al., 2019) selects the training samples which are easy to forget during training. We do not compare to GSS-Greedy (Aljundi et al., 2019), because it is also a similarity based greedy algorithm like K-Center, but GSS-Greedy trains an online learning model to measure the similarity of samples, which is different from general image classification problem. More detailed comparisons can be found in the supplementary.

Comparison to coreset methods. We first compare our method to the coreset baselines on MNIST, FashionMNIST, SVHN and CIFAR10 in Table 1 using the default ConvNet in classification accuracy. Whole dataset indicates training on the whole original set which serves as an approximate upper-bound performance. First we observe that our method outperforms all the baselines significantly and achieves a comparable result (97.4%) in case of 10 images per class to the upper bound (99.6%) in MNIST which uses two orders of magnitude more training images per class (6000). We also obtain promising results in FashionMNIST, however, the gap between our method and upper bound is bigger in SVHN and CIFAR10 which contain more diverse images with varying foregrounds and backgrounds. We also observe that, (i) the random selection baseline is competitive to other coreset methods in case of 10 images per class and (ii) herding method is on average the best coreset technique. We visualize the synthetic images produced by our method under 1 image/class setting in Figure 2. Interestingly they are interpretable and look like "prototypes" of each class.

Comparison to DD (Wang et al., 2018). Unlike the setting in Table 1, DD (Wang et al., 2018) reports results only for 10 images per class on MNIST and CIFAR10 over LeNet and AlexCifarNet (a customized AlexNet). We strictly follow the experimental setting in (Wang et al., 2018), use

Dataset	Img/Cls	DD	Ours	Whole Dataset
MNIST	1	-	85.0±1.6	99.5±0.0
	10	79.5±8.1	**93.9±0.6**	
CIFAR10	1	-	24.2±0.9	83.1±0.2
	10	36.8±1.2	**39.1±1.2**	

Table 3: Comparison to DD (Wang et al., 2018) in terms of testing accuracy (%).

	Random	Herding	Ours	Early-stopping	Whole Dataset
Performance (%)	76.2	76.2	**84.5**	**84.5**	85.9
Correlation	-0.21	-0.20	**0.79**	0.42	1.00
Time cost (min)	**18.8**	**18.8**	**18.8**	**18.8**	8604.3
Storage (imgs)	$\mathbf{10^2}$	$\mathbf{10^2}$	$\mathbf{10^2}$	10^4	5×10^4

Table 4: Neural Architecture Search. Methods are compared in performance, ranking correlation, time and memory cost.

the same architectures and report our and their original results in Table 3 for a fair comparison. Our method achieves significantly better performance than DD on both benchmarks; obtains 5% higher accuracy with only 1 synthetic sample per class than DD with 10 samples per class. In addition, our method obtains consistent results over multiple runs with a standard deviation of only 0.6% on MNIST, while DD's performance significantly vary over different runs (8.1%). Finally our method trains 2 times faster than DD and requires 50% less memory on CIFAR10 experiments. More detailed runtime and qualitative comparison can be found in the supplementary.

Cross-architecture generalization. Another key advantage of our method is that the condensed images learned using one architecture can be used to train another unseen one. Here we learn to generate 1 condensed image per class for MNIST over a diverse set of networks including MLP, ConvNet (Gidaris & Komodakis, 2018), LeNet (LeCun et al., 1998), AlexNet (Krizhevsky et al., 2012), VGG-11 (Simonyan & Zisserman, 2014) and ResNet-18 (He et al., 2016) (see Table 2). Once the condensed sets are synthesized, we train every network on all the sets separately from scratch and evaluate their cross architecture performance in terms of classification accuracy on the MNIST test set. Table 2 shows that the condensed images, especially the ones that are trained with convolutional networks, perform well and are thus architecture generic. MLP generated images do not work well for training convolutional architectures which is possibly due to the mismatch between translation invariance properties of MLP and convolutional networks. Interestingly, MLP achieves better performance with convolutional network generated images than the MLP generated ones. The best results are obtained in most cases with ResNet generated images and ConvNet or ResNet as classifiers which is inline with their performances when trained on the original dataset.

Number of condensed images. We also study the relation between the number of condensed images per class and test performance of a ConvNet trained on them for MNIST, FashionMNIST, SVHN and CIFAR10 in Figure 3 in absolute and relative terms – normalized by its upper-bound. While increasing the number of condensed images improves the accuracies in all benchmarks and further closes the gap with the upper-bound performance especially in MNIST and FashionMNIST. Our method outperforms the compared Herding method by a large margin in all cases.

Activation, normalization & pooling. We also study the effect of various activation (sigmoid, ReLU (Nair & Hinton, 2010; Zeiler et al., 2013), leaky ReLU (Maas et al., 2013)), pooling (max, average) and normalization functions (batch (Ioffe & Szegedy, 2015), group (Wu & He, 2018), layer (Ba et al., 2016), instance norm (Ulyanov et al., 2016)) and have the following observations: i) leaky ReLU over ReLU and average pooling over max pooling enable learning better condensed images, as they allow for denser gradient flow; ii) instance normalization obtains better classification performance than its alternatives when used in the networks that are trained on a small set of condensed images. We refer to the supplementary for detailed results and discussion.

3.2 Applications

Continual Learning First we apply our method to a continual-learning scenario (Rebuffi et al., 2017; Castro et al., 2018) where new tasks are learned incrementally and the goal is to preserve the performance on the old tasks while learning the new ones. We build our model on E2E method in (Castro et al., 2018) that uses a limited budget rehearsal memory (we consider 10 images/class here) to keep representative samples from the old tasks and knowledge distillation (KD) to regularize the network's output w.r.t. to previous predictions. We replace its sample selection mechanism (herding) with ours such that a set of condensed images are generated and stored in the memory, keep the rest of the model same and evaluate this model on the task-incremental learning problem on the digit recognition datasets, SVHN (Netzer et al., 2011), MNIST (LeCun et al., 1998) and USPS (Hull, 1994) in the same order. MNIST and USPS images are reshaped to 32×32 RGB images.

We compare our method to E2E (Castro et al., 2018), depicted as herding in Figure 4, with and without KD regularization. The experiment contains 3 incremental training stages

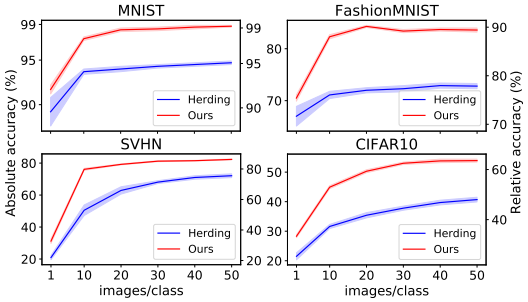


Figure 3: Absolute and relative testing accuracies for varying the number of condensed images/class for MNIST, FashionMNIST, SVHN and CIFAR10. The relative accuracy means the ratio compared to its upper-bound, *i.e.* training with the whole dataset.

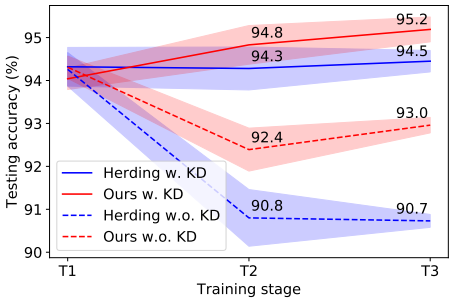


Figure 4: Continual learning performance in accuracy (%). Herding denotes the original E2E (Castro et al., 2018). T1, T2, T3 are three learning stages. The performance at each stage is the mean testing accuracy on all learned tasks.

(SVHN→MNIST→USPS) and testing accuracies are computed by averaging over the test sets of the previous and current tasks after each stage. The desired outcome is to obtain high mean classification accuracy at T3. The results indicate that the condensed images are more *data-efficient* than the ones sampled by herding and thus our method outperforms E2E in both settings, while by a large margin (2.3% at T3) when KD is not employed.

Neural Architecture Search. Here we explore the use of our method in a simple neural architecture search (NAS) experiment on CIFAR 10 which typically requires expensive training of numerous architectures multiple times on the whole training set and picking the best performing ones on a test set. Our goal is to verify that our condensed images can be used to efficiently train multiple networks to identify the best network. To this end, we construct the search space of 720 ConvNets as described in Section 3.1 by varying hyper-parameters W, N, A, P, D over an uniform grid (see supplementary for more details), train them for 100 epochs on three small proxy datasets (10 images/class) that are obtained with Random sampling, Herding and our method. Note that we train the condensed images for once only with the default ConvNet architecture and use them to train all kinds of architectures. We also compare to early-stopping (Li & Talwalkar, 2020) in which the model is trained on whole training set but with the same number of training iterations as the one required for the small proxy datasets, in other words, with the same time cost.

Table 4 depicts i) the average test performance of the best selected model over 5 runs when trained on the whole dataset, ii) Spearman's rank correlation coefficient between the validation accuracies obtained by training the selected top 10 models on the proxy dataset and whole dataset, iii) time for training 720 architectures on a NVIDIA GTX1080-Ti GPU, and iv) memory print of the training images. Our method achieves the highest testing performance (84.5%) and performance correlation (0.79), meanwhile significantly decreases the the searching time (from 8604.3 to 18.8 minutes) and storage space (from 5×10^4 to 1×10^2 images) compared to whole-dataset training. The competitive early-stopping baseline achieves on par performance for the best performing model with ours, however, the rank correlation (0.42) of top 10 models is significantly lower than ours (0.79) which indicates unreliable correlation of performances between early-stopping and whole-dataset training. Furthermore, early-stopping needs 100 times as many training images as ours needs. Note that the training time for synthetic images is around 50 minutes (if $K = 500$) which is one time off and negligible cost when training thousands even millions of candidate architectures in NAS.

4 Conclusion

In this paper, we propose a dataset condensation method that learns to synthesize a small set of informative images. We show that these images are significantly more data-efficient than the same number of original images and the ones produced by the previous method and they are not architecture dependent, can be used to train different deep networks. Once trained, they can be used to lower the memory print of datasets and efficiently train numerous networks which are crucial in continual learning and neural architecture search resp. As future work, we plan to explore the use of condensed images in more diverse and thus challenging datasets like ImageNet (Deng et al., 2009) that contain higher resolution images with larger variations in pose, appearance of objects, background.

REFERENCES

Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems*, pp. 11816–11825, 2019.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pp. 2654–2662, 2014.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Eden Belouadah and Adrian Popescu. Scail: Classifier weights scaling for class incremental learning. In *The IEEE Winter Conference on Applications of Computer Vision*, 2020.

Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Flexible dataset distillation: Learn labels instead of images. *arXiv preprint arXiv:2006.08572*, 2020.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 233–248, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326, 2012.

Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1434–1453. SIAM, 2013.

Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4367–4375, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014a.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 2004.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Pang Wei Koh and Percy Liang. 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.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pp. 367–377. PMLR, 2020.

Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. Data-free knowledge distillation for deep neural networks. In *LLD Workshop at Neural Information Processing Systems (NIPS)*, 2017.

David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International conference on machine learning (ICML)*, 2013.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5188–5196, 2015.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Gaurav Kumar Nayak, Konda Reddy Mopuri, Vaisakh Shaj, Venkatesh Babu Radhakrishnan, and Anirban Chakraborty. Zero-shot knowledge distillation in deep networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Salah Rifai, Yoshua Bengio, Yann Dauphin, and Pascal Vincent. A generative process for sampling contractive auto-encoders. *arXiv preprint arXiv:1206.6434*, 2012.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Kegan GG Samuel and Marshall F Tappen. Learning optimized map estimates in continuously-valued mrf models. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 477–484. IEEE, 2009.

Mert Bulent Sariyildiz and Ramazan Gokberk Cinbis. Gradient matching generative networks for zero-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2168–2178, 2019.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *ICLR*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pp. 4077–4087, 2017.

Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *arXiv preprint arXiv:1912.07768*, 2019.

Ilia Sucholutsky and Matthias Schonlau. Soft-label dataset distillation and text dataset distillation. *arXiv preprint arXiv:1910.02551*, 2019.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *ICLR*, 2019.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, 2016.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1121–1128. ACM, 2009.

G W Wolf. Facility location: concepts, models, algorithms and case studies. 2011.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 374–382, 2019.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Matthew D. Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Mark Z. Mao, Kyeongcheol Yang, Quoc V. Le, Patrick Nguyen, Andrew W. Senior, Vincent Vanhoucke, Jeffrey Dean, and Geoffrey E. Hinton. On rectified linear units for speech processing. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3517–3521, 2013.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pp. 14747–14756, 2019.

A IMPLEMENTATION DETAILS

In this part, we explain the implementation details for the dataset condensation, continual learning and neural architecture search experiments.

Dataset condensation. The presented experiments involve tuning of six hyperparameters – the number of outer-loop K and inner-loop steps T, learning rates $\eta_{\mathcal{S}}$ and number of optimization steps $\varsigma_{\mathcal{S}}$ for the condensed samples, learning rates $\eta_{\boldsymbol{\theta}}$ and number of optimization steps $\varsigma_{\boldsymbol{\theta}}$ for the model weights. In all experiments, we set $K = 1000$, $\eta_{\mathcal{S}} = 0.1$, $\eta_{\boldsymbol{\theta}} = 0.01$, $\varsigma_{\mathcal{S}} = 1$ and employ Stochastic Gradient Descent (SGD) as the optimizer. The only exception is that we set $\eta_{\mathcal{S}}$ to 0.01 for synthesizing data with MLP in cross-architecture experiments (Table 2), as MLP requires a slightly different treatment. Note that while K is the maximum number of outer-loop steps, the optimization early-stops automatically if it converges before K steps. For the remaining hyperparameters, we use two different sets of for each 1 and 10 image/class setting. We set $T = 1$, $\varsigma_{\boldsymbol{\theta}} = 1$ for 1 image/class and $T = 10$, $\varsigma_{\boldsymbol{\theta}} = 50$ for 10 images/class. Note that when $T = 1$, it is not required to update the model parameters (see the Step 8 in Algorithm 1), as this model is not further used. For those experiments where more than 10 images/class are synthesized, we set T to be the same number as the synthetic images per class and $\varsigma_{\boldsymbol{\theta}} = 500/T$, *e.g.* $T = 20$, $\varsigma_{\boldsymbol{\theta}} = 25$ for 20 images/class learning. The ablation study on hyper-parameters are given in Appendix B which shows that our method is not sensitive to varying hyper-parameters.

The minibatches (Step 5 in Algorithm 1) that are sampled at each inner iteration contain samples from a class at a time. Specifically, we randomly sample 256 real images of a class (c) as a batch to calculate the mean gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\mathcal{T}_c}(\boldsymbol{\theta}_t)$ and match it with the mean gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\mathcal{S}_c}(\boldsymbol{\theta}_t)$ that is averaged over the condensed samples with the corresponding class. We train the condensed samples class-by-class in every inner-loop.

In all experiments, we use the standard train/test splits of the datasets – the train/test statistics are shown in Table T1. We apply data augmentation (crop, scale and rotate) only for experiments (coreset methods and ours) on MNIST. The only exception is that we also use data augmentation when compared to DD (Wang et al., 2018) on CIFAR10 with AlexCifarNet, and data augmentation is also used in (Wang et al., 2018). For initialization of condensed images, we tried both Gaussian noise and randomly selected real training images, and obtained comparable performances.

	MNIST	FashionMNIST	SVHN	CIFAR10	USPS
Train	60,000	60,000	73,257	50,000	7,291
Test	10,000	10,000	26,032	10,000	2,007

Table T1: Train/test statistics for MNIST, FashionMNIST, SVHN, CIFAR10 and USPS datasets.

In the first stage – while training the condensed images–, we use Batch Normalization in the VGG and ResNet networks. For reliable estimation of the running mean and variance, we sample many real training data to estimate the running mean and variance and then freeze them ahead of Step 7. In the second stage – while training a deep network on the condensed set –, we replace Batch Normalization layers with Instance Normalization in VGG and ResNet, due to the fact that the batch statistics are not reliable when training networks with few condensed images. Another minor modification that we apply to the standard network ResNet architecture in the first stage is replacing the strided convolutions where $stride = 2$ with convolutional layers where $stride = 1$ coupled with an average pooling layer. We observe that this change enables more detailed (per pixel) gradients w.r.t. the condensed images and leads to better condensed images.

Continual learning. In this experiment, we focus on a task-incremental learning on SVHN, MNIST and USPS with the given order. The three tasks share the same label space, however have significantly different image statistics. The images of the three datasets are reshaped to 32×32 RGB size for standardization. We use the standard splits for training sets and randomly sample 2,000 test images for each datasets to obtain a balanced evaluation over three datasets. Thus each model is tested on a growing test set with 2,000, 4,000 and 6,000 images at the three stages respectively. We use the default ConvNet in this experiment and set the weight of distillation loss to 1.0 and the temperature to 2. We run 5,000 and 500 iterations for training and balanced finetuning as in (Castro et al., 2018) with the learning rates 0.01 and 0.001 respectively. We run 5 experiments and report the mean and standard variance in Figure 4.

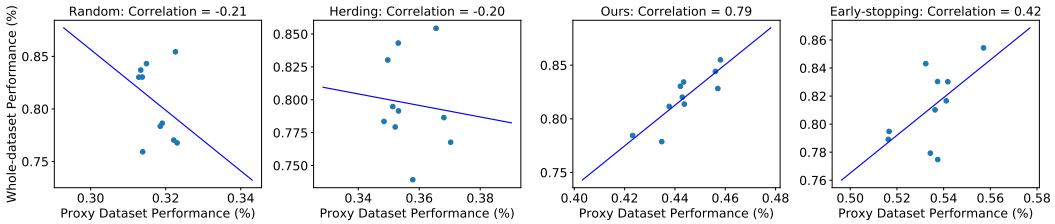


Figure F5: The performance correlation between the training on proxy dataset and whole-dataset. For each proxy dataset, the best 10 models are selected based on validation set performance. In the figure, each point represents an architecture. The straight line is to fit these points.

C\T	Sigmoid	ReLu	LeakyReLu
Sigmoid	86.7±0.7	91.2±0.6	91.2±0.6
ReLu	86.1±0.9	91.7±0.5	91.7±0.5
LeakyReLu	86.3±0.9	91.7±0.5	91.7±0.4

Table T2: Cross-activation experiments in accuracy (%) for 1 condensed image/class in MNIST.

C\T	None	MaxPooling	AvgPooling
None	78.7±3.0	80.8±3.5	88.3±1.0
MaxPooling	81.2±2.8	89.5±1.1	91.1±0.6
Avgpooing	81.8±2.9	90.2±0.8	91.7±0.5

Table T3: Cross-pooling experiments in accuracy (%) for 1 condensed image/class in MNIST.

Neural Architecture Search. To construct the searching space of 720 ConvNets, we vary hyper-parameters $W \in \{32, 64, 128, 256\}$, $D \in \{1, 2, 3, 4\}$, $N \in \{$None, BatchNorm, LayerNorm, InstanceNorm, GroupNorm$\}$, $A \in \{$Sigmoid, ReLu, LeakyReLu$\}$, $P \in \{$None, MaxPooling, AvgPooling$\}$. We randomly sample 5,000 images from the 50,000 training images in CIFAR10 as the validation set. Every candidate ConvNet is trained with the proxy dataset, and then evaluated on the validation set. These candidate ConvNets are ranked by the validation performance. 10 architectures with top validation accuracies are selected to calculate Spearman's rank correlation coefficient, because the best model that we want will come from the top 10 architectures. We train each ConvNet for 5 times to get averaged validation and testing accuracies.

We visualize the performance correlation for different proxy datasets in Figure F5. Obviously, the condensed proxy dataset produced by our method achieves the highest performance correlation (0.79) which significantly higher than early-stopping (0.42). It means our method can produce more reliable results for NAS.

B FURTHER ANALYSIS

Next we provide additional results on ablative studies over various deep network layers including activation, pooling and normalization functions and also over depth and width of deep network architecture. An additional qualitative analysis on the learned condensed images is also given.

Ablation study on activation functions. Here we study the use of three activation functions – Sigmoid, ReLU, LeakyReLu (negative slope is set to 0.01) – in two stages, when training condensed images (denoted as C) and when training a ConvNet from scratch on the learned condensed images (denoted as T). The experiments are conducted in MNIST dataset for 1 condensed image/class setting. Table T2 shows that all three activation functions are good for the first stage while generating good condensed images, however, Sigmoid performs poor in the second stage while learning a classifier on the condensed images – its testing accuracies are lower than ReLu and LeakyReLu by around 5%. This suggests that ReLU can provide sufficiently informative gradients for learning condensed images, though the gradient of ReLU w.r.t. to its input is typically sparse.

Ablation study on pooling functions. Next we investigate the performance of two pooling functions – average pooling and max pooling – also no pooling for 1 image/class dataset condensation with ConvNet architecture in MNIST in terms of classification accuracy. Table T3 shows that max and average pooling both perform significantly better than no pooling (None) when they are used in the second stage. When the condensed samples are trained and tested on models with average pooling, the best testing accuracy ($91.7 \pm 0.5\%$) is obtained, possibly, because average pooling provides more informative and smooth gradients for the whole image rather than only for its discriminative parts.

C\T	None	BatchNorm	LayerNorm	InstanceNorm	GroupNorm
None	79.0±2.2	80.8±2.0	85.8±1.7	90.7±0.7	85.9±1.7
BatchNorm	78.6±2.1	80.7±1.8	85.7±1.6	90.9±0.6	85.9±1.5
LayerNorm	81.2±1.8	78.6±3.0	87.4±1.3	90.7±0.7	87.3±1.4
InstanceNorm	72.9±7.1	56.7±6.5	82.7±5.3	91.7±0.5	84.3±4.2
GroupNorm	79.5±2.1	81.8±2.3	87.3±1.2	91.6±0.5	87.2±1.2

Table T4: Cross-normalization experiments in accuracy (%) for 1 condensed image/class in MNIST.

C\T	1	2	3	4
1	61.3±3.5	78.2±3.0	77.1±4.0	76.4±3.5
2	78.3±2.3	89.0±0.8	91.0±0.6	89.4±0.8
3	81.6±1.5	89.8±0.8	91.7±0.5	90.4±0.6
4	82.5±1.3	89.9±0.8	91.9±0.5	90.6±0.4

Table T5: Cross-depth performance in accuracy (%) for 1 condensed image/class in MNIST.

C\T	32	64	128	256
32	90.6±0.8	91.4±0.5	91.5±0.5	91.3±0.6
64	91.0±0.8	91.6±0.6	91.8±0.5	91.4±0.6
128	90.8±0.7	91.5±0.6	91.7±0.5	91.2±0.7
256	91.0±0.7	91.6±0.6	91.7±0.5	91.4±0.5

Table T6: Cross-width performance in accuracy (%) for 1 condensed image/class in MNIST.

Ablation study on normalization functions. Next we study the performance of four normalization options – No normalization, Batch (Ioffe & Szegedy, 2015), Layer (Ba et al., 2016), Instance (Ulyanov et al., 2016) and Group Normalization (Wu & He, 2018) (number of groups is set to be four) – for 1 image/class dataset condensation with ConvNet architecture in MNIST classification accuracy. Table T4 shows that the normalization layer has little influence for learning the condensed set, while the choice of normalization layer is important for training networks on the condensed set. LayerNorm and GroupNorm have similar performance, and InstanceNorm is the best choice for training a model on condensed images. BatchNorm obtains lower performance which is similar to None (no normalization), as it is known to perform poorly when training models on few condensed samples as also observed in (Wu & He, 2018). Note that Batch Normalization does not allow for a stable training in the first stage (C); thus we replace its running mean and variance for each batch with those of randomly sampled real training images.

Ablation study on network depth and width. Here we study the effect of network depth and width for 1 image/class dataset condensation with ConvNet architecture in MNIST in terms of classification accuracy. To this end we conduct multiple experiments by varying the depth and width of the networks that are used to train condensed synthetic images and that are trained to classify testing data in ConvNet architecture and report the results in Table T5 and Table T6. In Table T5, we observe that deeper ConvNets with more blocks generate better condensed images that results in better classification performance when a network is trained on them, while ConvNet with 3 blocks performs best as classifier. Interestingly, Table T6 shows that the best results are obtained with the classifier that has 128 filters at each block, while network width (number of filters at each block) in generation has little overall impact on the final classification performance.

Ablation study on hyper-parameters Our performance is not sensitive to hyper-parameter selection. The testing accuracy for various K and T is depicted in Figure F6. The results show that the optimum K and T are around similar values across all datasets. Thus we simply set K to 1000 and T to 10 for all datasets. Similarly, for the remaining ones including learning rate, weight decay, we use a single set of hyperparameters that are observed to work well for all datasets and architectures in our preliminary experiments.

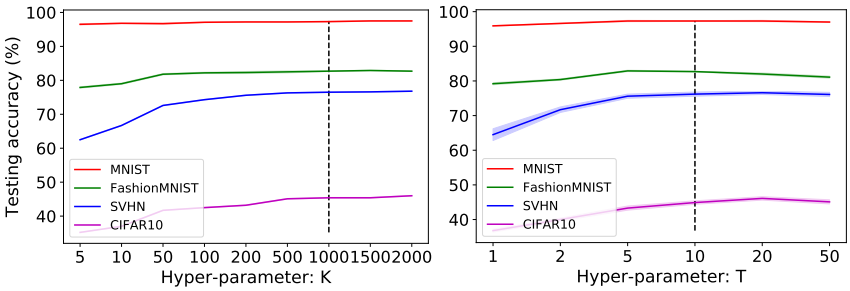
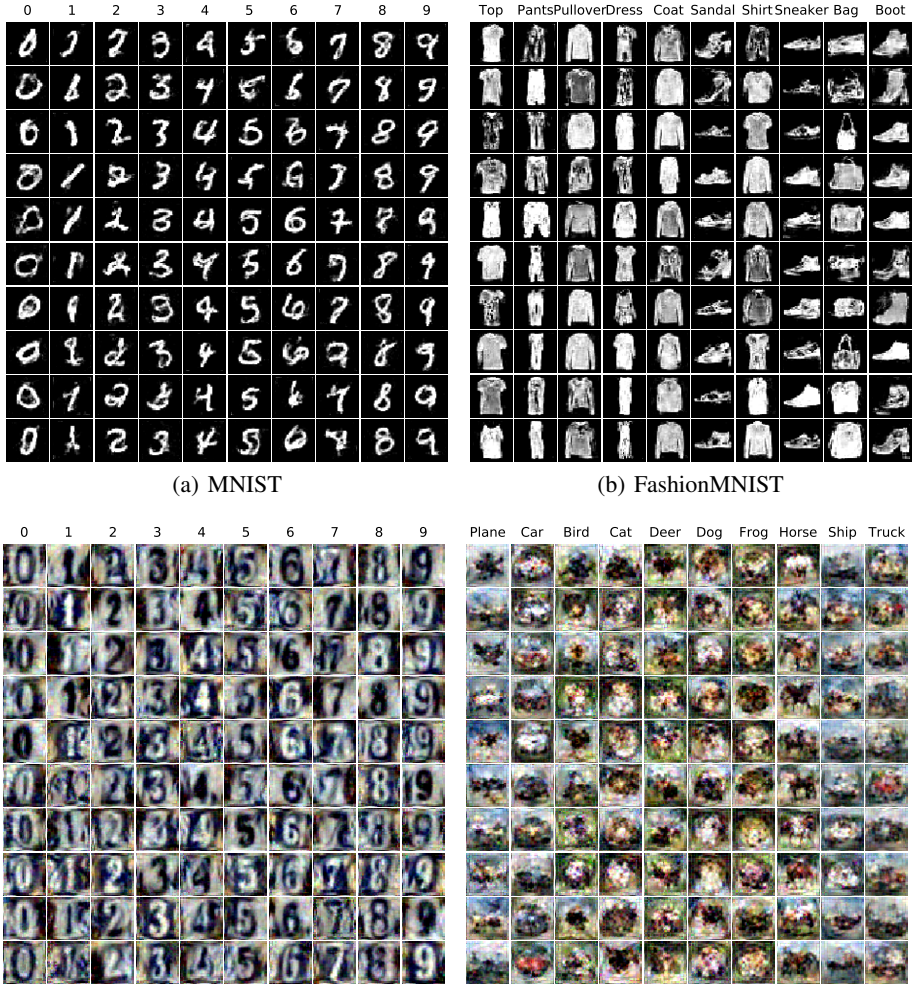


Figure F6: Ablation study on the hyper-parameters K and T.



(a) MNIST (b) FashionMNIST

(c) SVHN (d) CIFAR10

Figure F7: The synthetic images for MNIST, FashionMNIST, SVHN and CIFAR10 produced by our method with ConvNet under 10 images/class setting.

Further qualitative analysis We first depict the condensed images that are learned on MNIST, FashionMNIST, SVHN and CIFAR10 datasets in one experiment by using the default ConvNet in 10 images/class setting in Figure F7. It is interesting that the 10 images/class results in Figure F7 are diverse which cover the main variations, while the condensed images for 1 image/class setting (see Figure 2) look like the "prototype" of each class. For example, in Figure F7 (a), the ten images of "four" indicate ten different styles. The ten "bag" images in Figure F7 (b) are significantly different from each other, similarly "wallet" (1st row), "shopping bag" (3rd row), "handbag" (8th row) and "schoolbag" (10th row). Figure F7 (c) also shows the diverse house numbers with different shapes, colors and shadows. Besides, different poses of a "horse" have been learned in Figure F7 (d).

C Comparison to More Baselines

Optimal Random Selection. One interesting and strong baseline is Optimal Random Selection (ORS) that we implement random selection experiments for 1,000 times and pick the best ones. Table T7 presents the performance comparison to the selected Top 1000 (all), Top 100 and Top 10 coresets. These optimal coresets are selected by ranking their performance. Obviously, the condensed set generated by our method surpasses the selected Top 10 of 1000 coresets with a large margin on all four datasets.

	Img/Cls	Ratio %	Optimal Random Selection			cGAN	Ours	Whole Dataset
			Top 1000	Top 100	Top 10			
MNIST	1	0.017	64.3±6.1	74.4±1.8	78.2±1.7	64.0±3.2	**91.7±0.5**	99.6±0.0
	10	0.17	94.8±0.7	96.0±0.2	96.4±0.1	94.9±0.6	**97.4±0.2**	
FashionMNIST	1	0.017	51.3±5.4	59.6±1.3	62.4±0.9	51.1±0.8	**70.5±0.6**	93.5±0.1
	10	0.17	73.8±1.6	76.4±0.6	77.6±0.2	73.9±0.7	**82.3±0.4**	
SVHN	1	0.014	14.3±2.1	18.1±0.9	19.9±0.2	16.1±0.9	**31.2±1.4**	95.4±0.1
	10	0.14	34.6±3.2	40.3±1.3	42.9±0.9	33.9±1.1	**76.1±0.6**	
CIFAR10	1	0.02	15.0±2.0	18.5±0.8	20.1±0.5	16.3±1.4	**28.3±0.5**	84.8±0.1
	10	0.2	27.1±1.6	29.8±0.7	31.4±0.2	27.9±1.1	**44.9±0.5**	

Table T7: The performance comparison to optimal random selection (ORS) and conditional generative adversarial networks (cGAN) baselines. This table shows the testing accuracies (%) of different methods on four datasets. ConvNet is used for training and testing. Img/Cls: image(s) per class, Ratio (%): the ratio of condensed images to whole training set. Top 1000, Top 100 and Top 10 means the selected 1000, 100 and 10 optimal coresets by ranking their performances.

Generative Model. We also compare to the popular generative model, *namely*, Conditional Generative Adversarial Networks (cGAN) (Mirza & Osindero, 2014). The generator has two blocks which consists of the Up-sampling (scale_factor=2), Convolution (stride=1), BatchNorm and LeakyReLu layers. The discriminator has three blocks which consists of Convolution (stride=2), BatchNorm and LeakyReLu layers. In additional to the random noise, we also input the class label as the condition. We generate 1 and 10 images per class for each dataset with random noise. Table T7 shows that the images produced by cGAN have similar performances to those randomly selected coresets (*i.e*. Top 1000). It is reasonable, because the aim of cGAN is to generate real-look images. In contrast, our method aims to generate images that can train deep neural networks efficiently.

Analysis of coreset performances We find that K-Center (Wolf, 2011; Sener & Savarese, 2018) and Forgetting (Toneva et al., 2019) don't work as well as other general coreset methods, namely Random and Herding (Rebuffi et al., 2017), in this experimental setting. After analyzing the algorithms and coresets, we find two main reasons. 1) K-Center and Forgetting are not designed for training deep networks from scratch, instead they are for active learning and continual learning respectively. 2) The two algorithms both tend to select "hard" samples which are often outliers when only a small number of images are selected. These outliers confuse the training, which results in worse performance. Specifically, the first sample per class in K-Center coreset is initialized by selecting the one closest to each class center. The later ones selected by the greedy criterion that pursues maximum coverage are often outliers which confuse the training.

D FURTHER COMPARISON TO DD (WANG ET AL., 2018)

Next we compare our method to DD (Wang et al., 2018) first quantitatively in terms of cross-architecture generalization, then qualitatively in terms of synthetic image quality, and finally in terms of computational load for training synthetic images. Note that we use the original source code to obtain the results for DD that is provided by the authors of DD in the experiments.

Generalization ability comparison. Here we compare the generalization ability across different deep network architectures to DD. To this end, we use the synthesized 10 images/class data learned with LeNet on MNIST to train MLP, ConvNet, LeNet, AlexNet, VGG11 and ResNet18 and report the results in Table T8. We see that that the condensed set produced by our method achieves good classification performances with all architectures, while the synthetic set produced by DD perform poorly when used to trained some architectures, *e.g*. AlexNet, VGG and ResNet. Note that DD generates learning rates to be used in every training step in addition to the synthetic data. This is in contrast to our method which does not learn learning rates for specific training steps. Although the tied learning rates improve the performance of DD while training and testing on the same architecture, they will hinder the generalization to unseen architectures.

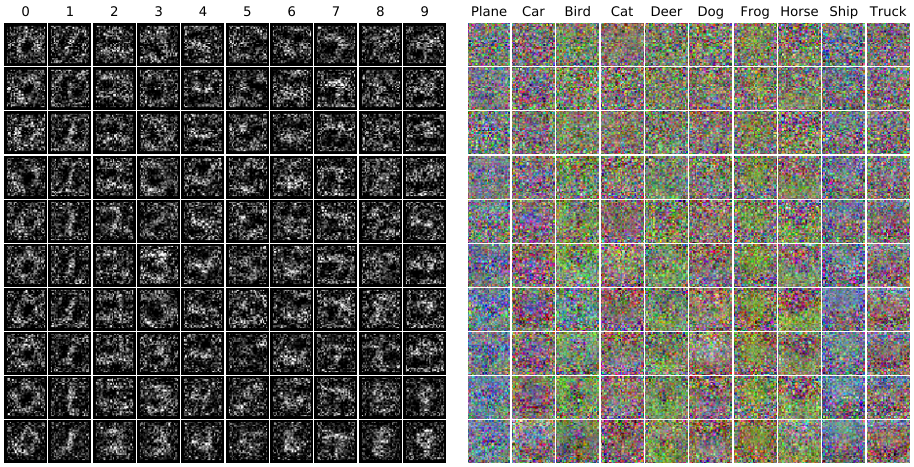
Qualitative comparison. We also provide a qualitative comparison to to DD in terms of image quality in Figure F8. Note that both of the synthetic sets are trained with LeNet on MNIST and AlexCifarNet on CIFAR10. Our method produces more interpretable and realistic images than DD, although it is not our goal. The MNIST images produced by DD are noisy, and the CIFAR10 images

Method	MLP	ConvNet	LeNet	AlexNet	VGG	ResNet
DD	72.7±2.8	77.6±2.9	79.5±8.1	51.3±19.9	11.4±2.6	63.6±12.7
Ours	**83.0±2.5**	**92.9±0.5**	**93.9±0.6**	**90.6±1.9**	**92.9±0.5**	**94.5±0.4**

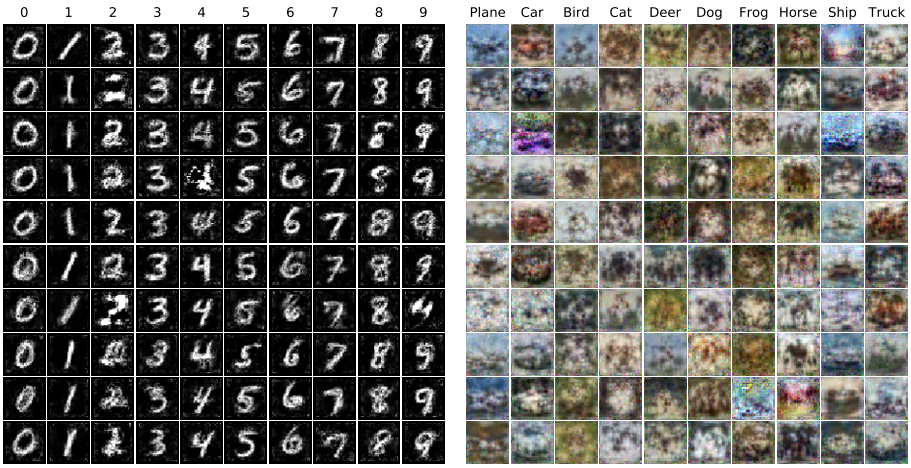
Table T8: Generalization ability comparison to DD. The 10 condensed images per class are trained with LeNet, and tested on various architectures. It shows that condensed images generated by our method have better generalization ability.

Method	Dataset	Architecture	Memory (MB)	Time (min)	Test Acc.
DD	MNIST	LeNet	785	160	79.5±8.1
Ours	MNIST	LeNet	653	46	93.9±0.6
DD	CIFAR10	AlexCifarNet	3211	214	36.8±1.2
Ours	CIFAR10	AlexCifarNet	1445	105	39.1±1.2

Table T9: Time and memory use for training DD and our method in 10 images/class setting.



(a) MNIST of DD (b) CIFAR10 of DD



(c) MNIST of ours (d) CIFAR10 of ours

Figure F8: Qualitative comparison between the condensed images produced by DD and ours under 10 images/class setting. LeNet and AlexCifarNet are utilized for MNIST and CIFAR10 respectively.

produced by DD do not show any clear structure of the corresponding class. In contrast, the MNIST and CIFAR10 images produced by our method are both visually meaningful and diverse.

Training memory and time. One advantage of our method is that we decouple the model weights from its previous states in training, while DD requires to maintain the recursive computation graph which is not scalable to large models and inner-loop optimizers with many steps. Hence, our method requires less training time and memory cost. We compare the training time and memory cost required by DD and our method with one NVIDIA GTX1080-Ti GPU. Table T9 shows that our method requires significantly less memory and training time than DD and provides an approximation reduction of 17% and 55% in memory and 71% and 51% in train time to learn MNIST and CIFAR10 datasets respectively. Furthermore, our training time and memory cost can be significantly decreased by use smaller hyper-parameters, *e.g.* K, T and the batch size of sampled real images with a slight performance decline (refer to Figure F6).

E EXTENDED RELATED WORK

Variations of Dataset Distillation. There exists recent work that extends Dataset Distillation (Wang et al., 2018). For example, (Sucholutsky & Schonlau, 2019; Bohdal et al., 2020) aims to improve DD by learning soft labels with/without synthetic images. (Such et al., 2019) utilizes a generator to synthesize images instead of directly updating image pixels. However, the reported quantitative and qualitative improvements over DD are minor compared to our improvements. In addition, none of these methods have seriously verified the cross-architecture generalization ability of the synthetic images.

Zero-shot Knowledge Distillation. Recent zero-shot KD methods (Lopes et al., 2017; Nayak et al., 2019) aim to perform KD from a trained model in the absence of training data by generating synthetic data as the intermediate production to further use. Unlike them, our method does not require pretrained teacher models to provide the knowledge, *i.e.* to obtain the features and labels.