THE UNIVERSITY *of* EDINBURGH

Data-Efficient Neural Network Training with Dataset Condensation

Bo Zhao

Doctor of Philosophy

Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

2022

Abstract

The state of the art in many data driven fields including computer vision and natural language processing typically relies on training larger models on bigger data. It is reported by OpenAI that the computational cost to achieve the state of the art doubles every 3.4 months in the deep learning era. In contrast, the GPU computation power doubles every 21.4 months[1], which is significantly slower. Thus, advancing deep learning performance by consuming more hardware resources is not sustainable. How to reduce the training cost while preserving the generalization performance is a long standing goal in machine learning. This thesis investigates a largely under-explored while promising solution – dataset condensation which aims to condense a large training set into a small set of informative synthetic samples for training deep models and achieve close performance to models trained on the original dataset. In this thesis, we investigate how to condense image datasets for classification tasks. We propose three methods for image dataset condensation. Our methods can be applied to condense other kinds of datasets for different learning tasks, such as text data, graph data and medical images, and we discuss it in Section 6.1.

First, we propose a principled method that formulates the goal of learning a small synthetic set as a gradient matching problem with respect to the gradients of deep neural network weights that are trained on the original and synthetic data. A new gradient/weight matching loss is designed for robust matching of different neural architectures. We evaluate its performance in several image classification benchmarks and explore the usage of our method in continual learning and neural architecture search.

In the second work, we propose to further improve the data-efficiency of training neural networks with synthetic data by enabling effective data augmentation. Specifically, we propose *Differentiable Siamese Augmentation* and learn better synthetic data that can be used more effectively with data augmentation and thus achieve better performance when training networks with data augmentation. Experiments verify that the proposed method obtains substantial gains over the state of the art.

While training deep models on the small set of condensed images can be extremely fast, their synthesis remains computationally expensive due to the complex

[1] We compare the floating point operations per second (FLOPS) for the double-precision computation of two representative GPU products, namely, GeForce GTX 580 used by Krizhevsky et al. (2012) launched in 2010, and nowadays widely-used A100 launched in 2020.

bi-level optimization. Finally, we propose a simple yet effective method that synthesizes condensed images by matching feature distributions of the synthetic and original training images when being embedded by randomly sampled deep networks. Thanks to its efficiency, we apply our method to more realistic and larger datasets with sophisticated neural architectures and obtain a significant performance boost.

In summary, this manuscript presents several important contributions that improve data efficiency of training deep neural networks by condensing large datasets into significantly smaller synthetic ones. The innovations focus on principled methods based on gradient matching, higher data-efficiency with differentiable Siamese augmentation, and extremely simple and fast distribution matching without bilevel optimization. The proposed methods are evaluated on popular image classification datasets, namely MNIST, FashionMNIST, SVHN, CIFAR10/100 and TinyImageNet. The code is available at https://github.com/VICO-UoE/DatasetCondensation.

Acknowledgements

I am lucky to work with my supervisor, Dr. Hakan Bilen, in this beautiful university. For achieving the Ph.D. degree, I would like to thank him first. His academic pursuit is pure and firm, which deeply inspires me. He can always raise constructive questions and provide substantial help on my research. In daily life, Dr. Hakan Bilen has a good sense of humor, thus talking with him is always comfortable and enjoyable.

Next, I will thank all members in my research group, as they are so kind and helpful. I really enjoy the time whenever we discuss academic problems or chat together. We had multiple warm union activities at home that I will remember forever. I would like to especially thank Weihong Li who helped me a lot before and after I arrived Edinburgh. The valuable information provided by him helped me decide to take Ph.D. study here. With Boyan Gao, the three of us often chatted and had dinner together, which were the main activities for me in the Covid time. The previous PostDoc Konda Reddy Mopuri in our group is my first collaborator in my dataset condensation project. His involvement encouraged me to struggle with this challenging project. I would like to thank Arushi Goel, Octave Mariotti, Taha Kocyigit, Lucas Deecke, Xialei Liu and Yu Yang who were deeply involved in my Ph.D. study. They helped me a lot in various aspects. We had a wonderful time in group meetings and activities.

In my Ph.D. study, Dr. Oisin Mac Aodha and Prof. Timothy Hospedales reviewed my research progress annually. I would like to thank them for their valuable time and constructive comments. I also appreciate that Prof. Iain Murray and Dr. Samuel Albanie can take valuable time to exam my thesis and viva. I learned a lot from them during the viva and thesis correction. My research is mainly funded by China Scholarship Council 201806010331 and the EPSRC programme grant Visual AI EP/T028572/1.

Last but not least, I would like to thank my whole family. Without their support, I cannot achieve this degree. The Covid-19 began soon after I arriving in Edinburgh, which caused my three-year separation from them. Finally, I also want to thank all the other people who helped me but are not listed here.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Bo Zhao*)

Table of Contents

Chapter 1

Introduction

In the past decade, deep learning has continuously pushed forward the state of the art in many fields, for example, computer vision, natural language processing and speech processing. Typically, the new state of the art is achieved by training more sophisticated models (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Szegedy et al., 2015; He et al., 2016; Xie et al., 2020) on larger datasets (Deng et al., 2009; Lin et al., 2014; Guo et al., 2016; Sun et al., 2017; Kwiatkowski et al., 2019; Kuznetsova et al., 2020), thus demanding increasingly more computational resources. It is reported by OpenAI that the computational cost to achieve the state of the art doubles every 3.4 months in the deep learning era (Amodei et al., 2018). In contrast, the GPU computation power doubles every 21.4 months[1], which is significantly slower. Thus, advancing deep learning performance by consuming more hardware resources is not sustainable. How to reduce the computational cost while preserving models' generalization performance is a long-standing and challenging learning problem. In this thesis, we investigate how to reduce the training cost by reducing the training set size.

The traditional solution to reduce the training set size is coreset selection (Agarwal et al., 2004; Har-Peled and Mazumdar, 2004; Feldman et al., 2013; Rebuffi et al., 2017; Toneva et al., 2019; Castro et al., 2018; Aljundi et al., 2019) that aims to select a subset of informative training samples from original training set. However, traditional selection methods suffer from three main drawbacks when being applied to deep learning tasks. First, most selection strategies are designed based on heuristics and implemented with greedy algorithms. The relationship between heuristics and

[1] We compare the floating point operations per second (FLOPS) for the double-precision computation of two representative GPU products, namely, GeForce GTX 580 used by (Krizhevsky et al., 2012) launched in 2010, and nowadays widely-used A100 launched in 2020.

network training dynamics is unclear. Second, these methods assume that a fixed data representation is given. However, deep learning learns the data representation from scratch with random initialization. The subset that is selected with one network as the embedding function may not be compatible with other networks. Third, the effectiveness of a selected subset is bounded by the individual samples in the original training set.

Dataset distillation (Wang et al., 2018) was proposed to distill a small set of synthetic images from the large training data. Thus, it overcomes these drawbacks of coreset selection methods. Dataset distillation formulates the network parameters as a function of synthetic training data, and then minimize the evaluation error on the real training data by optimizing the synthetic data. However, dataset distillation is expensive due to its bilevel optimization and unscalable recursive computation graph. Following (Wang et al., 2018), several methods have been proposed to improve it. Sucholutsky and Schonlau (2019) extend it by learning synthetic images and soft labels simultaneously. Bohdal et al. (2020) replace the SGD optimization in the inner-loop with ridge regression that has a closed-form solution. Nguyen et al. (2021a) propose to replace the inner-loop optimization with the kernel ridge regression, and the neural tangent kernel (Jacot et al., 2018) is used. However, these methods are still expensive and difficult to scale up to large datasets and sophisticated models. More details and analysis are given in the Section 2.2.2.

Inspired by Wang et al. (2018), we focus on the dataset condensation problem which aims to condense the large-scale training set $\mathcal{T} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_{|\mathcal{T}|}, y_{|\mathcal{T}|})\}$ with $|\mathcal{T}|$ image and label pairs into a small synthetic set $\mathcal{S} = \{(\boldsymbol{s}_1, y_1), \dots, (\boldsymbol{s}_{|\mathcal{S}|}, y_{|\mathcal{S}|})\}$ with $|\mathcal{S}|$ synthetic image and label pairs so that models trained on $\mathcal{T}$ and $\mathcal{S}$ obtain comparable performance on unseen testing data:

$$\mathbb{E}_{\boldsymbol{x} \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{T}}}(\boldsymbol{x}), y)] \simeq \mathbb{E}_{\boldsymbol{x} \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{S}}}(\boldsymbol{x}), y)], \tag{1.1}$$

where $P_{\mathcal{D}}$ is the real data distribution, ℓ is the loss function (*i.e.* cross-entropy loss), ϕ is a deep neural network parameterized by $\boldsymbol{\theta}$, and $\phi_{\boldsymbol{\theta}^{\mathcal{T}}}$ and $\phi_{\boldsymbol{\theta}^{\mathcal{S}}}$ are the networks that are trained on $\mathcal{T}$ and $\mathcal{S}$ respectively. The pipeline of dataset condensation is illustrated in Figure 1.1. In this thesis, we present three proposed methods for realizing dataset condensation in practical applications with appropriate computation resources.

In Chapter 3, we propose a principled method that formulates dataset condensation as a gradient matching problem with respect to the gradients of deep neural network parameters that are trained on the original and our synthetic data. We

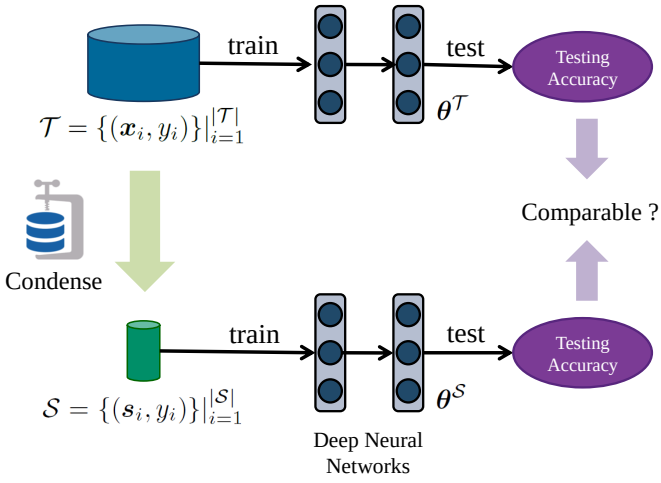


Figure 1.1: Dataset Condensation aims to condense a large training set into a small set of informative synthetic samples for training deep models from scratch and achieving close performance to those trained on the original dataset.

expect the same model trained on a large real dataset and a small synthetic set can produce the same gradients, thus the optimization on two datasets follows the same solution path in the parameter space. The proposed method successfully avoids unrolling the recursive computation graph like previous work, thus is more scalable to large datasets and models. We rigorously evaluate our method in several computer vision benchmarks including MNIST, FashionMNIST, SVHN and CIFAR10, and demonstrate that it significantly outperforms the state-of-the-art methods. Extensive ablation study is implemented, which verify the good generalization ability of the synthetic images generated by our method. Furthermore, we explore the use of our method in continual learning and neural architecture search and report promising gains when limited memory and computations are available.

Data augmentation, a technique to expand a training set with semantic-preserving transformations, is widely applied in many learning tasks. While we could simply apply standard augmentation strategies when learning the condensed synthetic dataset, those naive augmentation strategies lead to either drops or negligible gains of performance because of the different characteristics between real and synthetic images. In Chapter 4, we propose Differentiable Siamese Augmentation (DSA) to learn better synthetic data that can be effectively used with data augmentation for training deep neural networks. Experiments on multiple image classification benchmarks demonstrate that the proposed method obtains substantial gains over the state of the art, 7% improvements on CIFAR10 and CIFAR100 datasets. We show with only less than 1% data that our method achieves 99.6%, 94.9%, 88.5%, 71.5%

relative performance on MNIST, FashionMNIST, SVHN, CIFAR10 respectively. We also explore the use of our method in continual learning and neural architecture search, and show promising results.

While training deep models on a small set of condensed images can be extremely fast, their synthesis remains computationally expensive due to the complex bi-level optimization and second-order derivative computation. In Chapter 5, we propose a simple yet effective method that synthesizes condensed images by matching the feature distributions of the synthetic and original training images in many sampled embedding spaces. We are the first to implement dataset condensation on TinyImageNet, which contains 100,000 training images with 64×64 size from 200 classes. Our method significantly reduces the synthesis cost while achieving comparable or better performance. Thanks to its efficiency, we apply our method to more realistic and larger datasets (*i.e.* TinyImageNet and ImageNet-1K) with sophisticated neural architectures (*i.e.* ResNet) and obtain a significant performance boost. We also show practical benefits of our method in continual learning and neural architecture search.

Our research has important real-world applications including producing efficient memory for continual learning (Zhao et al., 2021; Zhao and Bilen, 2023), accelerating neural architecture or hyper-parameter search (Zhao et al., 2021; Zhao and Bilen, 2021), reducing data transmission in federated learning (Goetz and Tewari, 2020; Hu et al., 2022a; Song et al., 2022) and protecting data privacy (Dong et al., 2022). In this thesis, we also provide some promising experimental results about the application of our method in continual learning and neural architecture search.

1.1 Contributions

The main contributions in this thesis include:

1. In Chapter 3, we propose a principled, effective and scalable solution for dataset condensation, which formulates it as gradient matching problem. Our method successfully avoids unrolling the expensive recursive computation graph, thus is scalable to larger datasets and architectures. We show promising results when applying our dataset condensation method to continual learning and neural architecture search. This work, *Dataset Condensation with Gradient Matching*, has been published by International Conference on Learning

Representations (ICLR) 2021 as an oral presentation (Zhao et al., 2021).

2. In Chapter 4, we find the learned synthetic data do not work better with data augmentation for training deep neural networks. To further improve the data-efficiency, we propose differentiable Siamese augmentation (DSA) that enables effective use of data augmentation to synthesize more informative synthetic images. Significant performance improvement (7%) has been achieved on CIFAR10 and CIFAR100 datasets. This work, *Dataset Condensation with Differentiable Siamese Augmentation*, has been published by International Conference on Machine Learning (ICML) 2021 (Zhao and Bilen, 2021).

3. In Chapter 5, we propose an extremely simple and effective dataset condensation method that is the first method with neither bi-level optimization nor second-order derivatives. We empirically verify that this simple method achieves better performance, scalability and generalization ability than the state of the art. Our method extends the research of dataset condensation to larger settings (*i.e.* learning 1250 images/class for CIFAR10) and datasets (*i.e.* TinyImageNet and ImageNet-1K)\@. This work, *Dataset Condensation with Distribution Matching*, has been accepted by IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) 2023 (Zhao and Bilen, 2023).

Chapter 2

Background and Related Research

2.1 Sample Selection

The traditional solution to reduce the training set size is coreset selection (Agarwal et al., 2004; Har-Peled and Mazumdar, 2004; Feldman et al., 2013). Coreset selection aims to select a subset $\mathcal{T}_s$ of informative samples with a given size from the original training set $\mathcal{T}$, *i.e.* $\mathcal{T}_s \in \mathcal{T}$. In most popular works, coreset selection is implemented with heuristics and greedy selection algorithm. The popular heuristics includes:

1. Geometry similarity between two sets. These methods (Chen et al., 2010; Sener and Savarese, 2018; Sinha et al., 2020; Agarwal et al., 2020) select the subset that can maximize a chosen geometry similarity (or minimize a chosen distance) between the subset and original set. As the optimal subset selection is a NP-hard problem, greedy algorithms are chosen as the approximation. Here, we give two specific selection algorithms that are compared as the baseline in this theses.

   The first one is *herding*, which greedily and incrementally selects the sample that makes the new subset center closest to the original set center (Chen et al., 2010; Rebuffi et al., 2017; Castro et al., 2018; Wu et al., 2019; Belouadah and Popescu, 2020). Initializing the subset $\mathcal{T}_s = \emptyset$, the kth sample is selected by

   $$\boldsymbol{x}_k = \arg\min_{\boldsymbol{x} \in \mathcal{T} - \mathcal{T}_s} |\mu - \frac{1}{k}[\phi(\boldsymbol{x}) + \Sigma_{i=1}^{k-1}\phi(\boldsymbol{x}_i)]|, \tag{2.1}$$

   where $\phi(\boldsymbol{x})$ is the embedding of sample $\boldsymbol{x}$ and $\mu = \frac{1}{|\mathcal{T}|}\Sigma_{i=1}^{|\mathcal{T}|}\phi(\boldsymbol{x}_i)$ is the mean of all samples.

   As the labels of training data are given in image classification tasks, we implement sample selection class by class with balanced number for all

methods in this thesis, which can simplify the research problem and output balanced subset. For example, we select the subset that approaches each individual class center, and then combine all subsets in the herding method.

The second popular method is k-center (Sener and Savarese, 2018) which incrementally selects the sample that has the maximum distance to the current subset $\mathcal{T}_s$:

$$\boldsymbol{x}_k = \arg\max_{\boldsymbol{x}_i \in \mathcal{T} - \mathcal{T}_s} \arg\min_{\boldsymbol{x}_j \in \mathcal{T}_s} \Delta(\phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)), \tag{2.2}$$

where $\Delta(\phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j))$ measures the distance or dissimilarity between samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. However, deep learning learns the data representation from scratch with random initialization. The subset that is selected with one network as the embedding function may not be compatible with other networks. Furthermore, these methods do not clearly demonstrate the relationship between geometry similarity and the training effects.

2. Importance of individual sample. Coleman et al. (2019); Toneva et al. (2019); Paul et al. (2021); Margatina et al. (2021) identify the importance of individual training sample based on model prediction, the produced gradient during training, etc. The importance is measured in either a specific training epoch or averaged over all training epochs. Here, we present the popular forgetfulness method (Toneva et al., 2019) that counts how many times a training sample is previously correctly classified and then incorrectly classified during the training progress as the importance. The k_{th} sample is selected by

$$\boldsymbol{x}_k = \arg\max_{\boldsymbol{x}_i \in \mathcal{T} - \mathcal{T}_s} F(\boldsymbol{x}_i), \tag{2.3}$$

where $F(\boldsymbol{x}_i)$ is the forgetting statistics of sample $\boldsymbol{x}_i$. However, this importance measurement is easily influenced by the noisy data and outliers. Moreover, greedily selecting samples with the largest importance is short-sighted, and thus leads to sub-optimal results.

Other coreset methods include submodularity function based and bilevel optimization based methods. Those submodularity function based methods (Wei et al., 2015; Kaushal et al., 2021; Kothawade et al., 2021) design the submodularity function based on the heuristic metrics, thus have similar drawbacks like above methods. Bilevel optimization based methods (Borsos et al., 2020; Killamsetty et al.,

2021b,a) formulate coreset selection as the problem of optimizing selection weights so that models optimized on the selected subset achieve good performance. They suffer from expensive nested-loop optimization and poor scalability on large datasets and models.

Except the above-mentioned drawbacks, the effectiveness of selected subset is bounded by the individual samples in original training set. In this thesis, we investigate how to synthesize a small set of samples which are more informative than natural samples for training neural networks.

2.2 Sample Synthesis

2.2.1 Generative Models

Generative models aim to learn the distribution of training data. Then, the learned generator can be used to generate unseen data that are from the same distribution. Here, we discuss the generative models for image synthesis. In the NIPS 2016 Tutorial: Generative Adversarial Networks, Goodfellow (2016) introduces a taxonomic tree of generative models that groups methods based on how they approximate the likelihood. Fully Visible Belief Networks (Frey et al., 1995, 1998; Van Den Oord et al., 2016) and Nonlinear Independent Components Analysis (Deco and Brauer, 1994; Dinh et al., 2016) based methods explicitly model the probability density function with tractable models and thus have high design difficulty. Instead, Boltzmann machine (Hinton, 2012; Srivastava and Salakhutdinov, 2012) and Variational Autoencoders (Kingma and Welling, 2013; Rezende et al., 2014) use intractable models as the density function which requires approximation. Different from them, methods like Generative Stochastic Network (Bengio et al., 2014) use implicit density function based on Markov chains. However, Markov chains are usually not scalable to high-dimensional space. To overcome these drawbacks, Goodfellow et al. (2014a) propose Generative Adversarial Networks (GAN) in which a generator is learned to model the data distribution and a discriminator is learned to examine whether a sample is real or fake.

In this thesis, we compare our training set synthesis method to state-of-the-art autoencoders (Kingma and Welling, 2013; Parmar et al., 2021) and GANs (Goodfellow et al., 2014a; Brock et al., 2019) in experiments. We briefly review the basics of autoencoders and GANs. In autoencoders, the encoder $E(\cdot)$ embeds the

high-dimensional image into low-dimensional latent codes, then the decoder $D(\cdot)$ recovers the original image from the latent code for pixel-level fidelity:

$$\arg\min_{D,E} \frac{1}{N}\Sigma_{i=1}^{N}\|D(E(\boldsymbol{x}))-\boldsymbol{x}_i\|_2^2, \tag{2.4}$$

where N is the training set size. These autoencoders are not explicitly optimized for generating more informative synthetic images. Thus, the synthesized images are not more informative than randomly sampled real images for training deep neural networks. We provide the experimental analysis in Appendix B.2.

The objective of GANs is to generate real-looking images with generator G to fool the discriminator D, which can be formulated as

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{z\sim P(z)}[\log(1-D(G(z)))], \tag{2.5}$$

where z is the latent vector sampled from $P(z)$. Similarly, GANs neither generate more efficient training images than randomly sampled real images (Ravuri and Vinyals, 2019). It has been shown in previous work (Ravuri and Vinyals, 2019) that deep models trained on GANs generated images produce much worse testing performance than models trained on real images. Please refer to Appendix B.2 for experimental analysis.

Another generative model that is similar to our distribution matching based method presented in Chapter 5 is the *generative moment matching network* (GMMN) (Li et al., 2015) which aims to learn an image generator that can map a uniform distribution to real image distribution by minimizing the distribution discrepancy with kernel embeddings:

$$\begin{aligned}\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N^2}\Sigma_{i=1}^{N}\Sigma_{j=1}^{N}k(\boldsymbol{x}_i,\boldsymbol{x}_j) - \frac{1}{NM}\Sigma_{i=1}^{N}\Sigma_{j=1}^{M}k(\boldsymbol{x}_i,\phi_{\boldsymbol{\theta}}(\boldsymbol{h}_j)) \\ + \frac{1}{M^2}\Sigma_{i=1}^{M}\Sigma_{j=1}^{M}k(\phi_{\boldsymbol{\theta}}(\boldsymbol{h}_i),\phi_{\boldsymbol{\theta}}(\boldsymbol{h}_j)),\end{aligned} \tag{2.6}$$

where $\phi_{\boldsymbol{\theta}}(\boldsymbol{h}_i)$ is the generator with the latent vector $\boldsymbol{h}_i$ and $k(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is a Gaussian kernel. In other words, the generated images are expected to have close distribution to real image distribtion in the kernel embedding space. Our method significantly differ from GMMN in many ways. First, GMMN aims to generate real-looking images, while our goal is to condense a training set by synthesizing informative training samples that can be used to efficiently train deep networks. Second, our methods learn pixels directly, while GMMN learns a generator network. Third, our methods learn a few condensed synthetic training samples, while GMMN learns to map a uniform distribution to real image distribution.

Although generative models can be trained to produce efficient training samples with adapted objectives, *e.g.* Wang et al. (2018) and our gradient matching in Chapter 3 and distribution matching in Chapter 5, attempts so far have not achieved state-of-the-art results (Such et al., 2020). We leave it as future work.

2.2.2 Dataset Distillation

The pioneer work of Wang et al. (2018) first proposes the problem of distilling a large dataset into a small synthetic one. They formulate it as a meta-learning problem of learning a set of synthetic images $\mathcal{S}$ that can be used to learn neural network $\boldsymbol{\theta}^{\mathcal{S}}$ and achieve good testing performance on real data $\mathcal{T}$:

$$\begin{aligned} \mathcal{S}^* &= \arg\min_{\mathcal{S}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S})) \\ \text{subject to} \quad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) &= \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}). \end{aligned} \tag{2.7}$$

This meta-learning framework suffers from an expensive nested-loop optimization in which appropriately solving the inner-loop optimization $\boldsymbol{\theta}^{\mathcal{S}}$ before implementing outer-loop optimization is expensive or even impossible. The reason is that the function $\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta})$ needs to be expanded as a function with as many terms about synthetic set $\mathcal{S}$ as the training iteration number which can be thousands to millions for deep neural networks. Thus, thousands to millions of intermediate models have to be processed simultaneously in GPU RAM, which is impossible for most computational environments.

Based on this framework, several improvements have been proposed. Sucholutsky and Schonlau (2019) introduce soft-labels into synthetic set optimization. Such et al. (2020) propose to learn a generator (neural network) for synthesizing images instead of optimizing image pixels directly. Bohdal et al. (2020) avoid computing second-order derivatives by introducing ridge regression with a closed-form solution. Nguyen et al. (2021a,b) further introduce kernel ridge regression (KRR) with a neural tangent kernel (Jacot et al., 2018):

$$L(\mathbf{X}^{\mathcal{S}}, \boldsymbol{y}^{\mathcal{S}}) = \frac{1}{2} \| \boldsymbol{y}^{\mathcal{T}} - \mathbf{K}_{\mathbf{X}^{\mathcal{T}}\mathbf{X}^{\mathcal{S}}} (\mathbf{K}_{\mathbf{X}^{\mathcal{S}}\mathbf{X}^{\mathcal{S}}} + \lambda \mathbf{I})^{-1} \boldsymbol{y}^{\mathcal{S}} \|^2, \tag{2.8}$$

where the data pairs $(\mathbf{X}^{\mathcal{S}}, \boldsymbol{y}^{\mathcal{S}})$ and $(\mathbf{X}^{\mathcal{T}}, \boldsymbol{y}^{\mathcal{T}})$ are sampled from $\mathcal{S}$ and $\mathcal{T}$ respectively. $\mathbf{K}_{\mathbf{X}^{\mathcal{S}}\mathbf{X}^{\mathcal{S}}}$ and $\mathbf{K}_{\mathbf{X}^{\mathcal{T}}\mathbf{X}^{\mathcal{S}}}$ are the kernel matrices. The synthetic set $\mathcal{S}$ is learned by minimizing the loss $L(\mathbf{X}^{\mathcal{S}}, \boldsymbol{y}^{\mathcal{S}})$. However, solving KRR with neural tangent kernel (NTK) (Jacot et al., 2018) is expensive, which requires thousands of TPU hours for

condensing the CIFAR10 dataset. In addition, the synthetic set learned with NTK was used to train the modified ConvNet in their papers, which has 1024 filters and additional convolution stem. Thus, we do not directly compare to their results due to the remarkable difference in architecture and training cost.

More recently, Cazenavette et al. (2022a) propose the trajectory matching method that combines the meta-learning and parameter matching, and show significant performance improvement. The new method needs to pretrain and save many models for training synthetic images. Wang et al. (2022) propose a new learning objective that consists of the feature alignment loss and discrimination loss. Dynamic bi-level optimization is designed to prevent the under-/over-fitting in the condensation phase. To further save the storage, Kim et al. (2022) downsample multiple synthetic images and then merge them into one. Before training downstream-task models, they upsample these synthetic images. Although this strategy can further save storage, the data-efficiency of individual synthetic sample (after upper-sampling) is not improved.

2.3 Other Related Research

2.3.1 Few-shot Learning

A related research topic is few-shot learning (Vinyals et al., 2016; Finn et al., 2017; Snell et al., 2017; Sung et al., 2018) which aims to learn a classifier from only a few labeled training samples. Meta-learning (Hospedales et al., 2021) is widely applied in few-shot learning. The dataset condensation problem is essentially different from few-shot learning, although they both try to train neural networks with a few samples. First, the objective of few-shot learning is to learn a better backbone model that can quickly adapt to new concepts with few training samples. In the outer-loop, few-shot learning optimizes the backbone model parameters, while dataset condensation optimizes the synthetic dataset. Second, the few training samples of each task are given in few-shot learning, while dataset condensation learns a small number of more informative training samples.

2.3.2 Knowledge Distillation

Knowledge distillation (Buciluǎ et al., 2006a; Hinton et al., 2015) is proposed to distill the knowledge from one or several (large) models into a single (small) model. The basic idea of knowledge distillation is to align the outputs or intermediate

representations of the teacher and student models, given the same inputs. Dataset condensation (or distillation) is to align the testing performance of the same model(s) trained on the original dataset and the condensed set respectively. Wang et al. (2018) views it as the orthogonal task of knowledge distillation. In this thesis, we show that both model and dataset distillation are important for implementing continual learning (Castro et al., 2018; Wu et al., 2018; Aljundi et al., 2019) which aims to relieve the catastrophic forgetting of knowledge learned from previously tasks without access to previous training data. Please refer to Section 3.3.2 for experimental analysis.

2.3.3 Image Compression

An orthogonal but related research area is image compression (Rabbani and Jones, 1991; Lewis and Knowles, 1992; Rabbani, 2002) which aims to reduce the storage of images with minimal quality loss. Recently, learning-based image compression methods have been proposed (Choi et al., 2019; Mentzer et al., 2019; Agustsson et al., 2019). Typically, an encoder and decoder pair is learned for encoding images to compressed representations, *i.e.* "codes", and then recovering the original images from them. Our dataset condensation is fundamentally different from image compression, as we learn a small set of informative training samples for training deep neural networks instead of compressed representations for recovering visual details. Interestingly, image compression techniques can be applied to our synthetic images and then further reduce the storage size of synthetic images. Schirrmeister et al. (2022) present a demonstration where neural networks trained on simplified synthetic images with smaller bit size have almost no accuracy degradation.

2.3.4 Continual Learning

Our research of dataset condensation can directly benefit continual learning (Rebuffi et al., 2017; Toneva et al., 2019; Castro et al., 2018; Aljundi et al., 2019) in which constructing a memory for preserving history/past knowledge is the key for obtaining good performance (Knoblauch et al., 2020), as the previous training data are no longer accessible in future tasks. In popular continual learning experiment settings (Prabhu et al., 2020), there is a fixed or increasing memory budget for storing history training data. For example, the memory can be a fixed budget of 2000 samples in total or an budget of 20 samples for each new class increasingly. Traditionally, training samples of seen tasks are selected based on coreset selection methods and then used

for joint training with the data of future tasks, *i.e.* "rehearsal". With our dataset condensation methods, we can learn to synthesize more informative training samples as the memory, and thus achieve better continual learning performance. We provide experimental results about continual learning in Section 3.3.2, Section 4.4.5 and Section 5.3.5.

2.3.5 Neural Architecture Search

Neural architecture search (NAS) (Zoph et al., 2018; Elsken et al., 2019; Dong et al., 2021) aims to find the optimal architecture from many candidates. Typically, NAS methods have to evaluate each specific architecture by training and testing it on the whole dataset. This process can be every expensive, because the number of possible architectures can be very large (thousands to millions) even though only a few discrete hyper-parameters are searched. For example, the popular NAS benchmark (Dong et al., 2021) provides the search space of 15,625 candidates for architecture topology and 32,768 candidates for architecture size. Ying et al. (2019) spent 100 TPU years of computation time to implement an exhaustive neural architecture search on CIFAR10 dataset (Krizhevsky et al., 2009). Our methods can be used to construct a small proxy dataset for evaluating neural architectures in a faster way. We provide experimental results about NAS in Section 3.3.2, Section 4.4.6 and Section 5.3.6.

2.3.6 Federated Learning & Data Privacy

Dataset condensation is also a promising solution to reduce the data transmission and protect data privacy for federated learning (Lyu et al., 2020), in which the training data will be transmitted between the clients and servers. There exist some works that use a synthetic dataset to protect the privacy of a medical dataset (Li et al., 2020) and to reduce the communication cost in federated learning (Zhou et al., 2020; Goetz and Tewari, 2020). The pioneering work of Dong et al. (2022) demonstrates that dataset condensation can protect data privacy while preserving training efficiency. Specifically, the experiments demonstrate the difficulty in implementing visual retrieval on condensed images and membership inference attack on models trained on them.

Chapter 3

Dataset Condensation with Gradient Matching

As the state-of-the-art machine learning methods in many fields rely on larger datasets, storing datasets and training models on them become significantly more expensive. This paper proposes a training set synthesis technique for *data-efficient* learning, called *Dataset Condensation*, that learns to condense large dataset into a small set of informative synthetic samples for training deep neural networks from scratch. We formulate this goal as a gradient matching problem between the gradients of deep neural network weights that are trained on the original 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 report promising gains when limited memory and computations are available.

3.1 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. A popular way to deal with large data is data selection – identifying the most representative training samples – that aims at improving *data efficiency* of machine learning techniques. While classical data selection methods, also known as coreset

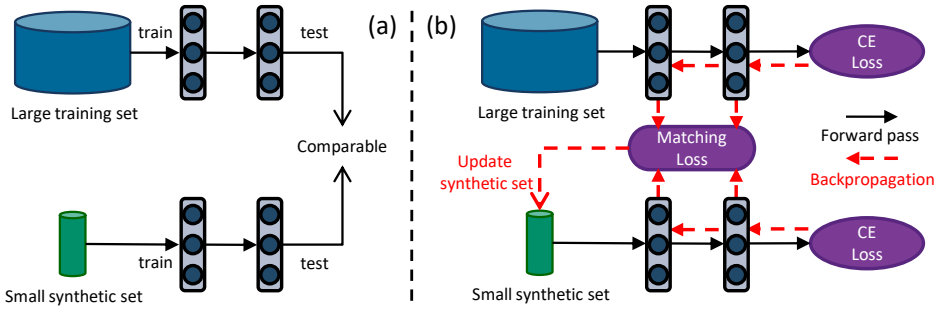


Figure 3.1: Dataset Condensation (left) aims to generate a small set of synthetic images that can match the performance of a network trained on a large image dataset. Our method (right) realizes this goal by learning a synthetic set such that a deep network trained on it and the large set produces similar gradients with respect to its weights. The synthetic data can later be used to train a network from scratch in a small fraction of the original computational load. CE denotes Cross-Entropy.

construction (Agarwal et al., 2004; Har-Peled and 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 and 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 and Savarese, 2018; Aljundi et al., 2019), forgetfulness (Toneva et al., 2019)), then select the representative samples based on the criterion, and 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) and greedy algorithms, which are short-sighted, and thus lead to sub-optimal results, ii) presence of representative samples, which is not guaranteed. A recent method, Dataset Distillation (DD) proposed by 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 with respect to 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 (Wang et al., 2018), we focus on learning to *synthesize informative samples* that are optimized to train neural networks for downstream tasks

and not limited to individual samples in original dataset. Like DD, our goal is to obtain the highest generalization performance with a model trained on a small set of synthetic images, ideally comparable performance to that of a model trained on the original images (see Figure 3.1(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 "condensed" synthetic samples such that a deep neural 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 of the network parameters that are computed for a training loss over a large fixed training set and a learnable condensed set (see Figure 3.1(b)). We show that our method enables effective learning of synthetic images and neural networks trained on them, outperforms the method proposed by 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) techniques (Hinton et al., 2015; Buciluǎ et al., 2006b; Ba and Caruana, 2014; Romero et al., 2014) that transfer the knowledge in 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 and Osindero, 2014; Radford et al., 2015) and Variational AutoEncoders (Kingma and 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 produce image patches by projecting the feature activations back to the input pixel space (Zeiler and Fergus, 2014), reconstruct the input image by matching

the feature activations (Mahendran and Vedaldi, 2015), recover private training images for given training gradients (Zhu et al., 2019; Zhao et al., 2020a), synthesize features from semantic embeddings for zero-shot learning (Sariyildiz and 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 3.2, present and analyze our results in several image recognition benchmarks in section 3.3.1, showcase applications in continual learning and network architecture search in section 3.3.2, and conclude the paper with remarks for future directions in section 3.4.

3.2 Method

3.2.1 Dataset Condensation

Suppose we are given a large dataset consisting of $|\mathcal{T}|$ pairs of a training image and its class label $\mathcal{T} = \{(\boldsymbol{x}_i, y_i)\}|_{i=1}^{|\mathcal{T}|}$ where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$, $y \in \{0, \dots, C-1\}$, $\mathcal{X}$ 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:

$$\boldsymbol{\theta}^{\mathcal{T}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}) \tag{3.1}$$

where $\mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \ell(\phi_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$, $\ell(\cdot,\cdot)$ 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, $\mathcal{S} = \{(\boldsymbol{s}_i, y_i)\}|_{i=1}^{|\mathcal{S}|}$ where $\boldsymbol{s} \in \mathbb{R}^d$ and $y \in \mathcal{Y}$, $|\mathcal{S}| \ll |\mathcal{T}|$. Similar to eq. (3.1), once the condensed set is learned, one can train ϕ on them as follows

$$\boldsymbol{\theta}^{\mathcal{S}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}) \tag{3.2}$$

where $\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{S}|} \sum_{(\boldsymbol{s},y)\in\mathcal{S}} \ell(\phi_{\boldsymbol{\theta}}(\boldsymbol{s}), y)$ 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. (3.2) to be significantly faster than that in eq. (3.1). We also wish the generalization performance of $\phi_{\boldsymbol{\theta}^{\mathcal{S}}}$ to be close 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 was proposed by Wang et al. (2018) and extended by Sucholutsky and Schonlau (2019); Bohdal et al. (2020); Such et al. (2020), is to pose the parameters $\boldsymbol{\theta}^{\mathcal{S}}$ as a function of the synthetic data $\mathcal{S}$:

$$\mathcal{S}^* = \arg\min_{\mathcal{S}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S})) \qquad \text{subject to} \qquad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}). \tag{3.3}$$

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. (3.3) 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 $\boldsymbol{\theta}$ (refer to (Samuel and 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.

3.2.2 Dataset Condensation with Parameter Matching

Here we aim to learn synthetic images $\mathcal{S}$ 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

$$\min_{\mathcal{S}} D(\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\theta}^{\mathcal{T}}) \quad \text{subject to} \quad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}) \tag{3.4}$$

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. (3.4) 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. (3.4) as follows:

$$\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0\sim P_{\boldsymbol{\theta}_0}}[D(\boldsymbol{\theta}^{\mathcal{S}}(\boldsymbol{\theta}_0), \boldsymbol{\theta}^{\mathcal{T}}(\boldsymbol{\theta}_0))] \quad \text{subject to} \quad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}(\boldsymbol{\theta}_0)) \tag{3.5}$$

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

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. (3.5) employs implicit differentiation (refer to (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 proposed by Domke (2012) which re-defines $\boldsymbol{\theta}^{\mathcal{S}}$ as the output of an incomplete optimization:

$$\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}), \varsigma), \tag{3.6}$$

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

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. (3.5). 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 $\boldsymbol{\theta}_0$.

3.2.3 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 decompose eq. (3.5) into multiple subproblems:

$$\begin{aligned} &\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}}[\sum_{t=0}^{T-1} D(\boldsymbol{\theta}_t^{\mathcal{S}}, \boldsymbol{\theta}_t^{\mathcal{T}})] \quad \text{subject to} \\ &\boldsymbol{\theta}_{t+1}^{\mathcal{S}}(\mathcal{S}) = \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t^{\mathcal{S}}), \varsigma^{\mathcal{S}}) \quad \text{and} \quad \boldsymbol{\theta}_{t+1}^{\mathcal{T}} = \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_t^{\mathcal{T}}), \varsigma^{\mathcal{T}}) \end{aligned} \tag{3.7}$$

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 other 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 $\boldsymbol{\theta}_{t+1}^{\mathcal{S}}$, which is parameterized with $\mathcal{S}$, can successfully track $\boldsymbol{\theta}_{t+1}^{\mathcal{T}}$ by updating $\mathcal{S}$ and minimizing $D(\boldsymbol{\theta}_t^{\mathcal{S}}, \boldsymbol{\theta}_t^{\mathcal{T}})$ close to zero.

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

$$\boldsymbol{\theta}_{t+1}^{\mathcal{S}} \leftarrow \boldsymbol{\theta}_t^{\mathcal{S}} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t^{\mathcal{S}}) \quad \text{and} \quad \boldsymbol{\theta}_{t+1}^{\mathcal{T}} \leftarrow \boldsymbol{\theta}_t^{\mathcal{T}} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_t^{\mathcal{T}}), \tag{3.8}$$

where $\eta_{\boldsymbol{\theta}}$ is the learning rate. Based on our observation ($D(\boldsymbol{\theta}_t^{\mathcal{S}}, \boldsymbol{\theta}_t^{\mathcal{T}}) \approx 0$), we simplify the formulation in eq. (3.7) by replacing $\boldsymbol{\theta}_t^{\mathcal{T}}$ with $\boldsymbol{\theta}_t^{\mathcal{S}}$ and use $\boldsymbol{\theta}$ to denote $\boldsymbol{\theta}^{\mathcal{S}}$ in the rest of the paper:

$$\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}} [\sum_{t=0}^{T-1} D(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_t))]. \tag{3.9}$$

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 $\mathcal{L}^{\mathcal{T}}$ with respect to $\boldsymbol{\theta}$ and the gradients for the loss over the condensed samples $\mathcal{L}^{\mathcal{S}}$ with respect to $\boldsymbol{\theta}$ is minimized. In other words, our goal reduces to matching the gradients for the real and synthetic training loss with respect to $\boldsymbol{\theta}$ via updating the condensed samples. This approximation has the key advantage over Wang et al. (2018) and eq. (3.5) 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. (3.9) 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 $\boldsymbol{\theta}$ 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 with respect to $\boldsymbol{\theta}$, then optimize the synthetic samples $\mathcal{S}$ 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 we sample each real and synthetic batch pair from $\mathcal{T}$ and $\mathcal{S}$ containing samples from a single class, and the synthetic data for each class are separately (or parallelly) updated at each iteration (t) for the following reasons: i) this reduces memory use at train time, ii) imitating the mean gradients with respect to the data from single class is easier compared to those of multiple classes. Separately updating synthetic data 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 $\mathcal{S}$ for C classes, 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**

    for $c = 0, \cdots, C-1$ **do**

      Sample a minibatch pair $B_c^{\mathcal{T}} \sim \mathcal{T}$ and $B_c^{\mathcal{S}} \sim \mathcal{S}$ ▷ $B_c^{\mathcal{T}}$ and $B_c^{\mathcal{S}}$ are of the same class c.

      Compute $\mathcal{L}_c^{\mathcal{T}} = \frac{1}{|B_c^{\mathcal{T}}|} \sum_{(\boldsymbol{x},y)\in B_c^{\mathcal{T}}} \ell(\phi_{\boldsymbol{\theta}_t}(\boldsymbol{x}), y)$ and $\mathcal{L}_c^{\mathcal{S}} = \frac{1}{|B_c^{\mathcal{S}}|} \sum_{(\boldsymbol{s},y)\in B_c^{\mathcal{S}}} \ell(\phi_{\boldsymbol{\theta}_t}(\boldsymbol{s}), y)$

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

    Update $\boldsymbol{\theta}_{t+1} \leftarrow \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \varsigma_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\theta}})$ ▷ Use the whole $\mathcal{S}$

Output: $\mathcal{S}$

Gradient Matching Loss. The matching loss $D(\cdot,\cdot)$ in eq. (3.9) measures the distance between the gradients for $\mathcal{L}^{\mathcal{S}}$ and $\mathcal{L}^{\mathcal{T}}$ with respect to $\boldsymbol{\theta}$. $\phi_{\boldsymbol{\theta}}$ is an image classification network with fully connected (FC) and convolutional layers. The gradients correspond to a set of learnable 2D (`out×in`) and 4D (`out×in×h×w`) weights for each fully connected and convolutional layer resp where `out`, `in`, `h`, `w` are number of output and input channels, kernel height and width respectively. The matching loss can be decomposed into a sum of layerwise losses as

$D(\nabla_\theta \mathcal{L}^\mathcal{S}, \nabla_\theta \mathcal{L}^\mathcal{T}) = \sum_{l=1}^{L} d(\nabla_{\theta^{(l)}} \mathcal{L}^\mathcal{S}, \nabla_{\theta^{(l)}} \mathcal{L}^\mathcal{T})$ where l is the layer index, L is the number of layers with weights and

$$d(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{\texttt{out}} \left(1 - \frac{\mathbf{A_i} \cdot \mathbf{B_i}}{\|\mathbf{A_i}\| \|\mathbf{B_i}\|} \right) \tag{3.10}$$

where $\mathbf{A}_{i\cdot}$ and $\mathbf{B}_{i\cdot}$ are flattened vectors of gradients corresponding to each output node i, which is `in` dimensional for FC weights and `in`×`h`×`w` dimensional for convolutional weights. In contrast to Lopez-Paz et al. (2017); Aljundi et al. (2019); Zhu et al. (2019) who ignore the layer-wise structure by flattening tensors over all layers to one vector and then computing the distance between two vectors, we group the gradients for each output node. We found that it is a better distance metric for gradient matching (refer to Appendix A.2) and it enables using a single learning rate across all layers.

3.3 Experiments

3.3.1 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 and Komodakis, 2018), LeNet (LeCun et al., 1998), AlexNet (Krizhevsky et al., 2012), VGG-11 (Simonyan and 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 and 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. The synthetic images can be initialized from Gaussian noise or randomly selected real training images. More details about the datasets, networks and hyper-parameters can be found in Appendix A.2.

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 used in two stages might be different. For the coreset baselines, the coreset is selected in the first stage. We investigate three settings: 1, 10 and 50 image/class learning, which means that the condensed set or coreset contains 1, 10 and 50 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 selects a subset of samples whose center is closet to the center of the original large set (Chen et al., 2010; Rebuffi et al., 2017; Castro et al., 2018; Wu et al., 2019; Belouadah and Popescu, 2020). K-Center (Sener and 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. The 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. Please refer to Section 2.1 for implementation details of sample selection methods.

Comparison to Coreset Methods. We first compare our method to the coreset baselines on MNIST, FashionMNIST, SVHN and CIFAR10 in Table 3.1 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 (98.8%) in case of 50 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 10 and 50

	Img/Cls	Ratio %	Coreset Selection				Ours	Whole Dataset
			Random	Herding	K-Center	Forgetting		
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**	
	50	0.83	97.9±0.2	94.9±0.2	97.4±0.3	88.2±1.2	**98.8±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**	
	50	0.83	82.5±0.7	71.9±0.8	68.3±0.8	55.0±1.1	**83.6±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**	
	50	0.7	70.9±0.9	72.6±0.8	20.1±1.4	27.2±1.5	**82.3±0.3**	
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**	
	50	1	43.4±1.0	40.4±0.6	27.0±1.4	23.3±1.1	**53.9±0.5**	

Table 3.1: 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.

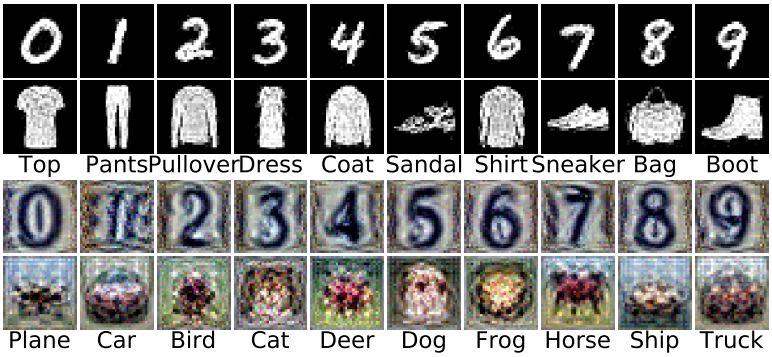


Figure 3.2: Visualization of condensed 1 image/class with ConvNet for MNIST, FashionMNIST, SVHN and CIFAR10.

images per class and (ii) herding method is on average the best coreset technique. We visualize the condensed images produced by our method under 1 image/class setting in Figure 3.2. Interestingly they are interpretable and look like "prototypes" of each class.

Comparison to DD (Wang et al., 2018). Unlike the setting in Table 3.1, 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 the same architectures and report our and their original results in Table 3.3 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.

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	**91.7±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 3.2: Cross-architecture performance in testing accuracy (%) for condensed 1 image/class in MNIST.

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 running time and qualitative comparison can be found in Appendix A.4.

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 1 condensed image per class for MNIST over a diverse set of networks including MLP, ConvNet (Gidaris and Komodakis, 2018), LeNet (LeCun et al., 1998), AlexNet (Krizhevsky et al., 2012), VGG-11 (Simonyan and Zisserman, 2014) and ResNet-18 (He et al., 2016) (see Table 3.2). 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 3.2 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 test performance of a ConvNet trained on them for MNIST, FashionMNIST, SVHN and CIFAR10 for various number of condensed images per class in Figure 3.3 in absolute and relative terms – normalized by its upper-bound. Increasing the number of condensed images improves

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 3.3: 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 3.4: Neural Architecture Search. Methods are compared in performance, ranking correlation, time and memory cost.

the accuracies in all benchmarks and further closes the gap with the upper-bound performance especially in MNIST and FashionMNIST, while the gap remains larger in SVHN and CIFAR10. In addition, our method outperforms the Herding coreset method by a large margin in all cases.

Activation, Normalization & Pooling. We also study the effect of various activation (sigmoid, ReLU (Nair and Hinton, 2010; Zeiler et al., 2013), leaky ReLU (Maas et al., 2013)), pooling (max, average) and normalization functions (batch (Ioffe and Szegedy, 2015), group (Wu and 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, and we suppose that 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 Appendix A.2 for detailed results and discussion.

3.3.2 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 proposed by Castro et al. (2018) that uses a

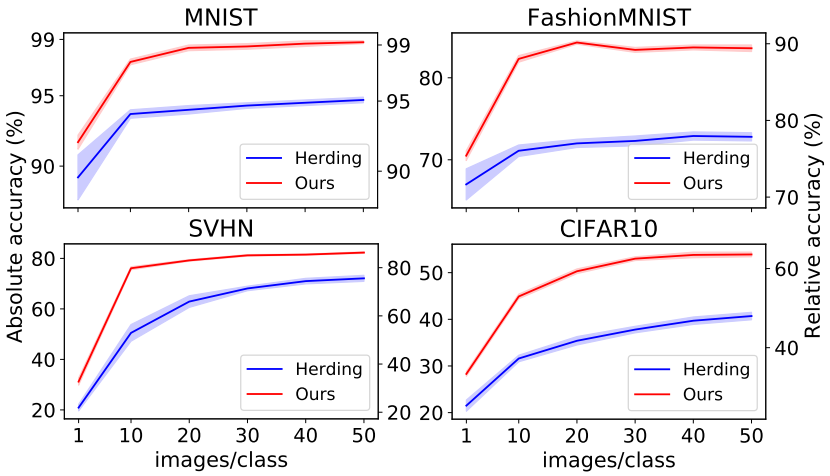


Figure 3.3: Absolute and relative testing accuracies for varying the number of condensed images/class for MNIST, FashionMNIST, SVHN and CIFAR10. The relative accuracy means the ratio compared to its upper-bound, *i.e.* training with the whole dataset.

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 with respect 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 3.4, with and without KD regularization. The experiment contains 3 incremental training stages (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 larger 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 CIFAR10 which typically requires expensive training of numerous architectures multiple times on the whole training set and picking the best performing ones on a validation 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

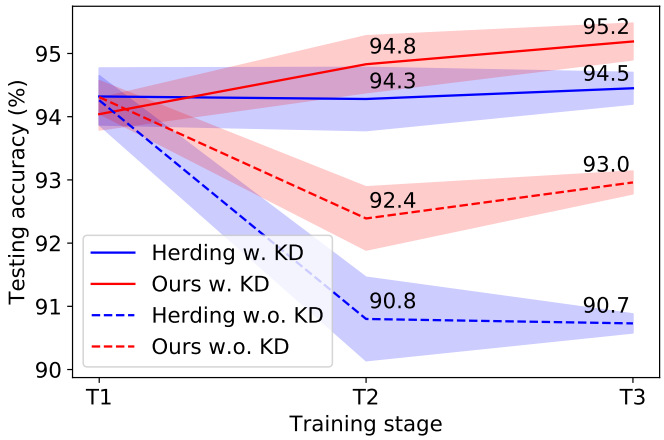


Figure 3.4: Continual learning performance in accuracy (%). Herding denotes the original E2E (Castro et al., 2018). T1, T2, T3 are three learning stages. The performance at each stage is the mean testing accuracy on all learned tasks.

described in Section 3.3.1 by varying hyper-parameters W, N, A, P, D over an uniform grid (see Appendix A.1 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 once only with the default ConvNet architecture and use them to train all kinds of architectures. We also compare to a simple baseline – *early-stopping* (Li and Talwalkar, 2020) which trains the model for specific training steps and then stops before the training loss converges. Specifically, the model is trained on whole training set but with the same number of training iterations as the one required for training the small proxy dataset, in other words, with the same amount of computations.

Table 3.4 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 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 (for $K = 500$) which is a negligible one-off cost when training thousands or even millions of candidate architectures in NAS.

3.4 Conclusion and Limitation

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. Moreover, the condensed images are not architecture dependent, and they 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 respectively.

Limitations. The proposed method has several limitations. First, our method still has the expensive bilevel optimization, although we avoid unrolling the recursive computation graph. The bilevel optimization prevents its application on large-scale datasets such as ImageNet (Deng et al., 2009) that contain higher resolution images with larger variations in appearance and pose of objects, background. In addition, our training algorithm needs to tune the hyper-parameters of inner- and outer-loop iterations, which can be flexible and adaptive in the future. Second, optimization objectives better than our gradient matching are required to achieve better results. Third, we observe the performance varies a lot when training and testing synthetic data on different architectures. More efforts are needed to improve the generalization performance on different architectures. Forth, we find that implementing data augmentation on synthetic data does not obviously improve the performance, which is different from the practice in many learning problems. We leave the investigation of these limitations as future work.

Chapter 4

Dataset Condensation with Differentiable Siamese Augmentation

In many machine learning problems, large-scale datasets have become the de-facto standard to train state-of-the-art deep networks at the price of heavy computation load. In this paper, we focus on condensing large training sets into significantly smaller synthetic sets which can be used to train deep neural networks from scratch with minimum drop in performance. Inspired from the recent training set synthesis methods, we propose Differentiable Siamese Augmentation that enables effective use of data augmentation to synthesize more informative synthetic images and thus achieves better performance when training networks with augmentations. Experiments on multiple image classification benchmarks demonstrate that the proposed method obtains substantial gains over the state-of-the-art of dataset condensation, 7% improvements on CIFAR10 and CIFAR100 datasets. We show with only less than 1% data that our method achieves 99.6%, 94.9%, 88.5%, 71.5% relative performance on MNIST, FashionMNIST, SVHN, CIFAR10 respectively. We also explore the use of our method in continual learning and neural architecture search, and show promising results.

4.1 Introduction

Deep neural networks have become the go-to technique in several fields including computer vision, natural language processing and speech recognition thanks to the recent developments in deep learning (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Szegedy et al., 2015; He et al., 2016) and presence of large-scale

datasets (Deng et al., 2009; Lin et al., 2014; Antol et al., 2015; Abu-El-Haija et al., 2016). However, their success comes at a price, increasing computational expense, as the state-of-the-art models have been primarily fueled by larger models (*e.g.* (Devlin et al., 2018; Radford et al., 2019; Dosovitskiy et al., 2021) and massive datasets (*e.g.* (Kuznetsova et al., 2020; Chen et al., 2020a; Kwiatkowski et al., 2019)). For example, it takes 12.3k TPU days to train EfficientNet-L2 (Xie et al., 2020) on JFT-300M dataset (Sun et al., 2017). To put in a perspective, the energy consumption for training EfficientNet-L2 once is about 29520 kWh, assuming that the TPU training power is 100W. Ideally, the same energy is sufficient to provide a typical UK family with electricity supply for 10 months or total energy supply (including electricity and gas) for 2 months (Ofgem, 2020). More dramatically, the computational cost significantly increases when better neural architectures are searched and designed due to many trials of training and validation on the dataset for different hyper-parameters (Bergstra and Bengio, 2012; Elsken et al., 2019). Significantly decreasing these costs without degrading the performance of the trained models is one of the long-standing goals in machine learning (Agarwal et al., 2004). To address these challenges, this paper focuses on reducing the training data size by learning significantly smaller synthetic data to train deep neural networks with minimum drop in their performance.

The standard way to reduce the training set size is to use a smaller but equally informative portion of data, namely a *coreset*. In literature, there is a large body of coreset selection methods for various target tasks, *e.g.* accelerating model training in neural architecture search (Shleifer and Prokop, 2019; Such et al., 2020), storing previous knowledge compactly in continual learning (Rebuffi et al., 2017; Toneva et al., 2019) and efficient selection of samples to label in active learning (Sener and Savarese, 2018). However, their selection procedures rely on heuristics and greedy algorithms which lead to sub-optimal results. In addition, finding such an informative coreset may not always be possible when the information in the dataset is not concentrated in few samples but uniformly distributed over all of them.

Motivated by these shortcomings, a recent research direction, *training set synthesis* aims at *generating* a small training set which is further used to train deep neural networks for the downstream task (Wang et al., 2018; Sucholutsky and Schonlau, 2019; Bohdal et al., 2020; Such et al., 2020; Nguyen et al., 2021a; Zhao et al., 2021). In particular, Dataset Distillation (DD) proposed by Wang et al. (2018) models the network parameters as a function of synthetic training data, and then minimizes the training loss on the real training data by optimizing the synthetic data.

Sucholutsky and Schonlau (2019) extend DD by learning synthetic images and soft labels simultaneously. Bohdal et al. (2020) simplify DD by only learning the informative soft labels for randomly selected real images. Such et al. (2020) propose to use a generator network instead of directly learning synthetic data. Nguyen et al. (2021a) reformulates DD in a kernel-ridge regression which has a closed-form solution. Zhao et al. (2021) propose Dataset Condensation (DC) that "condenses" the large training set into a small synthetic set by matching the gradients of the network parameters with respect to large-real and small-synthetic training data. The authors show that DC can be trained more efficiently by bypassing the bi-level optimization in DD while significantly outperforming DD in multiple benchmarks. Despite the recent success of the training set synthesis over the coreset techniques, especially in low-data regime, there is still a large performance gap between models trained on the small synthetic set and those trained on the whole training set. For instance, models that are trained on DD and DC synthetic sets obtain 38.3% and 44.9% accuracy respectively with 10 images per class on the CIFAR10 dataset, while a model trained on the whole dataset (5000 images per class) obtains 84.8% .

An orthogonal direction to increase data efficiency and thus generalization performance is data augmentation, a technique to expand training set with semantic-preserving transformations (Krizhevsky et al., 2012; Zhang et al., 2018; Yun et al., 2019; Chen et al., 2020b; Chen and He, 2020). While these strategies can simply be used to augment the synthetic set that are obtained by a training set synthesis method, we show that naive strategies lead to either drops or negligible gains in performance in Section 4.4. This is because the synthetic images i) have substantially different characteristics from natural images, ii) are not learned to train deep neural network under various transformations. Thus we argue that an effective combination of these two techniques is non-trivial and demands careful data augmentation design and a principled learning procedure.

In this paper we propose a principled method to enable learning a synthetic training set that can be effectively used with data augmentation to train deep neural networks. Our main technical contribution is *Differentiable Siamese Augmentation* (DSA), illustrated in Figure 4.1, that applies the same randomly sampled data transformation to both sampled real and synthetic data at each training iteration and also allows for backpropagating the gradient of the loss function with respect to the synthetic data by differentiable data transformations. Applying various data transformations (*e.g.* 15° clockwise rotation) simultaneously to both real and

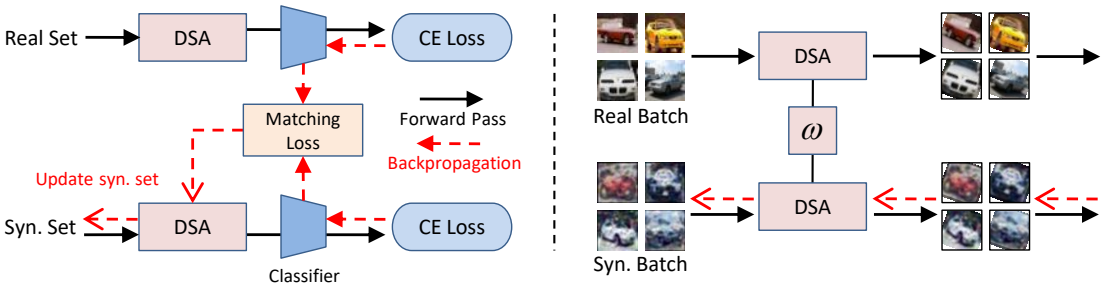


Figure 4.1: Dataset condensation with differentiable Siamese augmentation. Differentiable Siamese augmentation (DSA) applies the same parametric augmentation (*e.g.* rotation) to all data points in the sampled real and synthetic batches in a training iteration. The gradients of network parameters with respect to the sampled real and synthetic batches are matched for updating the synthetic images. A DSA example is given that rotation with the same degree is applied to the sampled real and synthetic batches.

synthetic images in training has three key advantages. First our method can exploit the information in real training images more effectively by augmenting them in several ways and transfer this augmented knowledge to the synthetic images. Second, sharing the same transformation across real and synthetic images allows the synthetic images to learn certain prior knowledge in the real images (*e.g.* the objects are usually horizontally on the ground). Please refer to Section 4.5.3 for detailed analysis. Third, most importantly, once the synthetic images are learned, they can be used with various data augmentation strategies to train different deep neural network architectures. We validate these advantages in multiple image classification benchmarks and show that our method significantly outperforms the state-of-the-art of dataset condensation by a wide margin, around 7% on CIFAR10/100 datasets[1]. Finally we explore the use of our method in continual learning and neural architecture search, and show promising results.

4.2 Related Work

In addition to the coreset selection and training set synthesis methods that are discussed in section 4.1, our method is also related to data augmentation techniques and Generative Adversarial Networks (GANs).

Data Augmentation. Many deep neural networks adopts data transformations for expanding the effective training set size, reducing overfitting and thus improving their performance. The most popular augmentation strategies include color jittering

[1]The implementation is available at https://github.com/VICO-UoE/DatasetCondensation.

(Krizhevsky et al., 2012), cropping (Krizhevsky et al., 2012), cutout (DeVries and Taylor, 2017), flipping, scale, rotation. More elaborate augmentation strategies are Mixup (Zhang et al., 2018) and CutMix (Yun et al., 2019). These augmentation strategies are typically applied to various image recognition problems where the label is invariant to transformations of the input and the transformations do not have to be differentiable with respect to the original input image.While we also use several data augmentation techniques, our focus is to synthesize training images that results in gradients that are equivariant to the ones from real images. In addition, we use differentiable augmentations such that gradients can go through the augmentation function and back-propagate to synthetic data.

Auto-augmentation. This line of work investigates how to automatically find the best augmentation strategy instead of manually designing by either learning a sequence of transformation functions in an adversarial optimization (Ratner et al., 2017), using a reinforcement learning algorithm (Cubuk et al., 2019), or learning the parameters of parametric feature augmentations (Yan et al., 2020). In contrast, our goal is not to find the best augmentation for training data but to synthesize training data that is equipped with the augmentation ability for training downstream-task models.

GANs & Differentiable Augmentation. GANs (Goodfellow et al., 2014a; Mirza and Osindero, 2014; Radford et al., 2015) typically aim at generating real-looking novel images by fooling a discriminator network. Differentiable Augmentation (Zhao et al., 2020b; Tran et al., 2020; Zhao et al., 2020c; Karras et al., 2020) has recently been applied for improving their training and in particular for preventing discriminators from memorizing the limited training set. Though they also apply differentiable augmentation to both real and fake images, augmentations are independently applied to real and fake ones. In contrast we use a Siamese augmentation strategy which is explicitly coordinated to apply the same transformation to both real and synthetic images. In addition, our goal, which is to generate a set of training data that can be used to efficiently train deep neural networks from scratch, differs from theirs and our images do not have to be realistic.

4.3 Method

Here we first review DC (Zhao et al., 2021), then describe our proposed DSA method and its training algorithm.

4.3.1 Dataset Condensation Review

Assume that we are given a large training set $\mathcal{T} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{|\mathcal{T}|}, y_{|\mathcal{T}|})\}$ with $|\mathcal{T}|$ image and label pairs. DC (Zhao et al., 2021) aims at learning a much smaller set with $|\mathcal{S}|$ synthetic image and label pairs $\mathcal{S} = \{(\boldsymbol{s}_1, y_1), \ldots, (\boldsymbol{s}_{|\mathcal{S}|}, y_{|\mathcal{S}|})\}$ from $\mathcal{T}$ such that a deep network trained on $\mathcal{S}$ obtains comparable generalization performance to a deep neural network that is trained on $\mathcal{T}$. Let $\phi_{\boldsymbol{\theta}^\mathcal{T}}$ and $\phi_{\boldsymbol{\theta}^\mathcal{S}}$ denote the deep neural networks with parameters $\boldsymbol{\theta}^\mathcal{T}$ and $\boldsymbol{\theta}^\mathcal{S}$ that are trained on $\mathcal{T}$ and $\mathcal{S}$ respectively. The goal of DC can be formulated as:

$$\mathbb{E}_{\boldsymbol{x}\sim P_\mathcal{D}}[\ell(\phi_{\boldsymbol{\theta}^\mathcal{T}}(\boldsymbol{x}), y)] \simeq \mathbb{E}_{\boldsymbol{x}\sim P_\mathcal{D}}[\ell(\phi_{\boldsymbol{\theta}^\mathcal{S}}(\boldsymbol{x}), y)] \tag{4.1}$$

over the real data distribution $P_\mathcal{D}$ with loss function ℓ (*i.e.* cross-entropy loss) . In practice, their generalization performances are measured on an unseen test set.

A possible way to achieve the comparable performance in eq. (4.1) is obtaining a similar solution to $\boldsymbol{\theta}^\mathcal{T}$, after the parameters of the network are trained on $\mathcal{S}$, *i.e.* $\boldsymbol{\theta}^\mathcal{S} \approx \boldsymbol{\theta}^\mathcal{T}$. However, solving this with respect to $\mathcal{S}$ involves nested loop optimization over network parameters $\boldsymbol{\theta}$ and synthetic data $\mathcal{S}$ which is typically not scalable to large models and multi-step optimization. Instead Zhao et al. (2021) hypothesize that a similar solution can be achieved, when the parameter updates for $\boldsymbol{\theta}_t^\mathcal{T}$ and $\boldsymbol{\theta}_t^\mathcal{S}$ are approximately equal at each training iteration t, given the same initialization $\boldsymbol{\theta}_0^\mathcal{T} = \boldsymbol{\theta}_0^\mathcal{S}$. In addition, assuming that $\boldsymbol{\theta}_t^\mathcal{S} = \boldsymbol{\theta}_t^\mathcal{T}$ can be satisfied at each iteration, the authors simplify the learning by using a single neural network parameterized by $\boldsymbol{\theta}$ and propose the following minimization problem:

$$\min_\mathcal{S} D(\nabla_\boldsymbol{\theta}\mathcal{L}(\mathcal{S}, \boldsymbol{\theta}_t), \nabla_\boldsymbol{\theta}\mathcal{L}(\mathcal{T}, \boldsymbol{\theta}_t)), \tag{4.2}$$

where

$$\mathcal{L}(\mathcal{S}, \boldsymbol{\theta}_t) = \frac{1}{|\mathcal{S}|}\sum_{(\boldsymbol{s},y)\in\mathcal{S}} \ell(\phi_{\boldsymbol{\theta}_t}(\boldsymbol{s}), y)),$$
$$\mathcal{L}(\mathcal{T}, \boldsymbol{\theta}_t) = \frac{1}{|\mathcal{T}|}\sum_{(\boldsymbol{x},y)\in\mathcal{T}} \ell(\phi_{\boldsymbol{\theta}_t}(\boldsymbol{x}), y))$$

and D is a sum of cosine distances between the two gradients of weights associated with each output node at each layer. We refer the readers to Zhao et al. (2021) for more detailed explanation.

4.3.2 Differentiable Siamese Augmentation (DSA)

Here we explain how data augmentation strategies can be effectively used with the DC formulation. One naive way is to apply data augmentation to the synthetic images post-

hoc, after they are learned. However, this strategy results in negligible performance gains (demonstrated in Section 4.4), as the synthetic images are not optimized to be augmented. Hence, a more principled way is to apply data augmentation while learning the synthetic images, which can be formulated by rewriting eq. (4.2):

$$\min_{\mathcal{S}} D(\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{A}(\mathcal{S},\omega^{\mathcal{S}}),\boldsymbol{\theta}_t),\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{A}(\mathcal{T},\omega^{\mathcal{T}}),\boldsymbol{\theta}_t)), \tag{4.3}$$

where $\mathcal{A}$ is a family of image transformations that preserves the semantics of the input (*i.e.* class label) such as cropping, color jittering, flipping that are parameterized with $\omega^{\mathcal{S}}$ and $\omega^{\mathcal{T}}$ for the synthetic and real training sets respectively.

Siamese Augmentation. In the standard data augmentation ω is randomly sampled from a predetermined distribution Ω for each image independently. However, randomly sampling both $\omega^{\mathcal{S}}$ and $\omega^{\mathcal{T}}$ is not meaningful in our case, as this results in ambiguous gradient matching problem in eq. (4.2). For instance, in case of cropping, this would require a particular region of a synthetic image to produce gradients matching to the ones that are generated from different crops of real image at different training iterations. Hence this method results in an averaging affect on the synthetic images and loss of information. To address this issue, we instead use the same transformations across the synthetic and real training sets, *i.e.* $\omega^{\mathcal{S}} = \omega^{\mathcal{T}}$. Thus we use one symbol ω in the remainder of the paper. As two sets have different number of images $|\mathcal{S}| \ll |\mathcal{T}|$ and there is no one-to-one correspondence between them, we randomly sample a single transformation ω and apply it to all images in a minibatch pair at each training iteration. This also avoids the averaging effect in a minibatch. This strategy enables correspondence between the two sets (*e.g.* between 15° clockwise rotation of synthetic and real set) and a more effective way of exploiting the information in the real training images and distilling it to the synthetic images in a more organized way without averaging effect. We illustrate the main idea in Figure 4.1.

Differentiable Augmentation. Solving eq. (4.3) for $\mathcal{S}$ involves computing the gradient for the matching loss D with respect to the synthetic images $\partial D(\cdot)/\partial\mathcal{S}$ by backpropagation:

$$\frac{\partial D(\cdot)}{\partial \mathcal{S}} = \frac{\partial D(\cdot)}{\partial \nabla_{\boldsymbol{\theta}}\mathcal{L}(\cdot)} \frac{\partial \nabla_{\boldsymbol{\theta}}\mathcal{L}(\cdot)}{\partial \mathcal{A}(\cdot)} \frac{\partial \mathcal{A}(\cdot)}{\partial \mathcal{S}}.$$

Thus the transformation $\mathcal{A}$ has to be differentiable with respect to the synthetic images $\mathcal{S}$. Traditionally transformations used for data augmentation are not implemented in a differentiable way, as optimizing input images is not their focus. Note that all the standard data augmentation methods for images are differentiable

and can be implemented as differentiable layers. Thus, we implement them as differentiable functions for deep neural network training and allow the error signal to be backpropagated to the synthetic images.

Algorithm 2: Dataset condensation with differentiable Siamese augmentation.

Input: Training set $\mathcal{T}$

Required: Randomly initialized set of synthetic samples $\mathcal{S}$ for C classes, probability distribution over randomly initialized weights P_{θ_0}, deep neural network $\phi_{\boldsymbol{\theta}}$, number of training iterations 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 $\eta_{\mathcal{S}}$, differentiable augmentation $\mathcal{A}_\omega$ parameterized with ω, augmentation parameter distribution Ω, random augmentation $\mathcal{A}$.

for $k = 0, \cdots, K-1$ **do**
- Initialize $\boldsymbol{\theta}_0 \sim P_{\theta_0}$
- **for** $t = 0, \cdots, T-1$ **do**
  - **for** $c = 0, \cdots, C-1$ **do**
    - Sample $\omega \sim \Omega$ and a minibatch pair $B_c^{\mathcal{T}} \sim \mathcal{T}$ and $B_c^{\mathcal{S}} \sim \mathcal{S}$ ▷ $B_c^{\mathcal{T}}$, $B_c^{\mathcal{S}}$ are of class c.
    - Compute $\mathcal{L}_c^{\mathcal{T}} = \frac{1}{|B_c^{\mathcal{T}}|} \sum_{(\boldsymbol{x},y)\in B_c^{\mathcal{T}}} \ell(\phi_{\boldsymbol{\theta}_t}(\mathcal{A}_\omega(\boldsymbol{x})), y)$ and $\mathcal{L}_c^{\mathcal{S}} = \frac{1}{|B_c^{\mathcal{S}}|} \sum_{(\boldsymbol{s},y)\in B_c^{\mathcal{S}}} \ell(\phi_{\boldsymbol{\theta}_t}(\mathcal{A}_\omega(\boldsymbol{s}), y)$
    - Update $\mathcal{S}_c \leftarrow \mathtt{sgd}_{\mathcal{S}}(D(\nabla_{\boldsymbol{\theta}}\mathcal{L}_c^{\mathcal{S}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}}\mathcal{L}_c^{\mathcal{T}}(\boldsymbol{\theta}_t)), \varsigma_{\mathcal{S}}, \eta_{\mathcal{S}})$
  - Update $\boldsymbol{\theta}_{t+1} \leftarrow \mathtt{sgd}_{\boldsymbol{\theta}}(\mathcal{L}(\boldsymbol{\theta}_t, \mathcal{A}(\mathcal{S})), \varsigma_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\theta}})$ ▷ Use $\mathcal{A}$ for the whole $\mathcal{S}$

Output: $\mathcal{S}$

4.3.3 Training Algorithm

We adopt training algorithm proposed by Zhao et al. (2021) for the proposed DSA, which is depicted in Alg. 2. To enable that the generated synthetic images can train deep neural networks from scratch with any randomly initialized parameters, we use an outer loop with K iterations where at each outer iteration we randomly initialize network parameters (*i.e.* $\boldsymbol{\theta}_0 \sim P_{\theta_0}$) from a distribution P_{θ_0} and train them from scratch. In the inner loop t, we randomly sample an image transformation ω and a minibatch

pair $B_c^{\mathcal{T}}$, $B_c^{\mathcal{S}}$ from the real and synthetic sets that contain samples from only class c, and compute their average cross-entropy loss and gradients with respect to the model parameters separately. Then we compute the gradient matching loss as in eq. (4.3) and update the synthetic data $\mathcal{S}_c$ by using stochastic gradient descent optimization with $\varsigma_{\mathcal{S}}$ gradient descent steps and $\eta_{\mathcal{S}}$ learning rate. We repeat the above steps for every class c in the inner loop t. Alternatively, we update the model parameters $\boldsymbol{\theta}_t$ to minimize the cross-entropy loss on the augmented synthetic data with $\varsigma_{\boldsymbol{\theta}}$ gradient descent steps and $\eta_{\boldsymbol{\theta}}$ learning rate.

Discussion. We observe that using minibatches from multiple classes leads to a slower convergence rate in training. The reason is that when the gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ are averaged over samples from multiple classes, image/class correspondence for synthetic data is harder to retrieve from the gradients.

4.4 Experiments

4.4.1 Datasets & Implementation Details

Datasets. We evaluate our method on 5 image classification datasets, MNIST (LeCun et al., 1990), FashionMNIST (Xiao et al., 2017), SVHN (Netzer et al., 2011), CIFAR10 and CIFAR100 (Krizhevsky et al., 2009). Both the MNIST and FashionMNIST datasets have 60,000 training and 10,000 testing images of 10 classes. SVHN is a real-world digit dataset which has 73,257 training and 26,032 testing images of 10 numbers. CIFAR10 and CIFAR100 both have 50,000 training and 10,000 testing images from 10 and 100 object categories respectively.

Network Architectures. We test our method on a wide range of network architectures, including multilayer perceptron (MLP), ConvNet (Gidaris and Komodakis, 2018), LeNet (LeCun et al., 1998), AlexNet (Krizhevsky et al., 2012), VGG-11 (Simonyan and Zisserman, 2014) and ResNet-18 (He et al., 2016). We use the ConvNet as the default architecture in experiments unless otherwise indicated. The default ConvNet has 3 duplicate convolutional blocks followed by a linear classifier, and each block consists of 128 filters, average pooling, ReLu activation (Nair and Hinton, 2010) and instance normalization (Ulyanov et al., 2016). We refer to Section 3.3.1 for more details about the above-mentioned architectures. The network parameters for all architectures are randomly initialized with Kaiming initialization (He et al., 2015). The labels of synthetic data are pre-defined evenly for

all classes, and the synthetic images are initialized with randomly sampled real images of corresponding class. While our method works well when initialising synthetic images from random noise, initialising them from randomly picked real images leads to better performance. We evaluate the initialization strategies in Section 4.5.2.

Hyper-parameters and Augmentation. For simplicity and generality, we use one set of hyper-parameters and augmentation strategy for all datasets. We set $K = 1000$, $\varsigma_{\mathcal{S}} = 1$, $\eta_\theta = 0.01$, $\eta_{\mathcal{S}} = 0.1$, $T = 1/10/50$ and $\varsigma_\theta = 1/50/10$ for 1/10/50 image(s)/class learning respectively as in (Zhao et al., 2021). The minibatch sizes for both real and synthetic data are 256. When the synthetic set has fewer images than 256, we use all the synthetic images of a class c in each minibatch. For data augmentation, we randomly pick one of several augmentations to implement each time. More details can be found in Section 4.4.4.

Experimental Setting. We evaluate our method at three settings, 1/10/50 image(s) per class learning. Each experiment involves two phases. First, we learn to synthesize a small synthetic set (*e.g.* 10 images/class) from a given large real training set. Then we use the learned synthetic set to train randomly initialized neural networks and evaluate their performance on the real testing set. For each experiment, we learn 5 sets of synthetic images and use each set to train 20 randomly initialized networks, then report mean testing accuracy and its standard deviation over the 100 evaluations.

4.4.2 Comparison to State of the Art

Competitors. We compare our method to several state-of-the-art coreset selection and training set synthesis methods. The coreset selection competitors are *random*, *herding* (Chen et al., 2010; Rebuffi et al., 2017; Castro et al., 2018; Belouadah and Popescu, 2020) and *forgetting* (Toneva et al., 2019). *Random* is a simple baseline that randomly select samples as the coreset. *Herding* is a distance based algorithm that selects samples whose center is close to the distribution center, *i.e.* each class center. *Forgetting* is a statistics based metric that selects samples with the maximum misclassification frequencies during training. Please refer to Section 2.1 for implementation details of sample selection methods. Training set synthesis competitors are Dataset Distillation (DD) proposed by Wang et al. (2018), Label Distillation (LD) proposed by Bohdal et al. (2020), and Dataset Condensation (DC) proposed by Zhao et al. (2021) which we built our method on. We also provide

	Img/Cls	Ratio %	Coreset Selection			Training Set Synthesis				Whole Dataset
			Random	Herding	Forgetting	DD†	LD†	DC	*DSA*	
MNIST	1	0.017	64.9±3.5	89.2±1.6	35.5±5.6		60.9±3.2	**91.7±0.5**	88.7±0.6	
	10	0.17	95.1±0.9	93.7±0.3	68.1±3.3	79.5±8.1	87.3±0.7	97.4±0.2	**97.8±0.1**	99.6±0.0
	50	0.83	97.9±0.2	94.8±0.2	88.2±1.2	-	93.3±0.3	98.8±0.2	**99.2±0.1**	
FashionMNIST	1	0.017	51.4±3.8	67.0±1.9	42.0±5.5	-	-	**70.5±0.6**	**70.6±0.6**	
	10	0.17	73.8±0.7	71.1±0.7	53.9±2.0	-	-	82.3±0.4	**84.6±0.3**	93.5±0.1
	50	0.83	82.5±0.7	71.9±0.8	55.0±1.1	-	-	83.6±0.4	**88.7±0.2**	
SVHN	1	0.014	14.6±1.6	20.9±1.3	12.1±1.7	-	-	**31.2±1.4**	27.5±1.4	
	10	0.14	35.1±4.1	50.5±3.3	16.8±1.2	-	-	76.1±0.6	**79.2±0.5**	95.4±0.1
	50	0.7	70.9±0.9	72.6±0.8	27.2±1.5	-	-	82.3±0.3	**84.4±0.4**	
CIFAR10	1	0.02	14.4±2.0	21.5±1.2	13.5±1.2	-	25.7±0.7	**28.3±0.5**	**28.8±0.7**	
	10	0.2	26.0±1.2	31.6±0.7	23.3±1.0	36.8±1.2	38.3±0.4	44.9±0.5	**52.1±0.5**	84.8±0.1
	50	1	43.4±1.0	40.4±0.6	23.3±1.1	-	42.5±0.4	53.9±0.5	**60.6±0.5**	

Table 4.1: The performance comparison to coreset selection and training set synthesis methods. This table shows the testing accuracies (%) of models trained from scratch on the small coreset or synthetic set. Img/Cls: image(s) per class, Ratio (%): the ratio of condensed images to whole training set. DD† and LD† use LeNet for MNIST and AlexNet for CIFAR10, while the rest use ConvNet for training and testing.

baseline performances for an approximate upperbound that are obtained by training the models on the whole real training set. Note that we report the results of coreset selection methods and upper-bound performances presented in DC (Zhao et al., 2021), as we use the same setting and hyper-parameters, and present the original results for the rest methods.

Results for 10 Category Datasets. Table 4.1 presents the results of different methods on MNIST, FashionMNIST, SVHN and CIFAR10, which all have 10 classes. We first see that *Herding* performs best among the coreset methods for a limited number of images *e.g.* only 1 or 10 image(s)/class and *random* selection performs best for 50 images/class. Overall the training set synthesis methods outperform the coreset methods which shows a clear advantage of synthesizing images for the downstream tasks, especially for 1 or 10 image(s)/class. Our method achieves the best performance in most settings and in particular obtains significant gains when learning 10 and 50 images/class, improves over the state-of-the-art DC by 7.2% and 6.7% in CIFAR10 dataset for 10 and 50 images per class. Remarkably in MNIST with less than 1% data (50 images/class), it achieves 99.2% which on par with the upperbound 99.6%. We also observe that our method obtains comparable or worse performance than DC in case of 1 image/class. We argue that our method acts as a regularizer on DC, as the synthetic images are forced to match the gradients from real training

images under different transformations. Thus we expect that our method works better when the solution space (synthetic set) is larger. Finally, the performance gap between the training set synthesis methods and upperbound gets larger when the task is more challenging. For instance, in the most challenging dataset, CIFAR10, the gap between ours and the upperbound is 24.2%, while it is 0.4% in MNIST in the 50 images/class setting.

Note that we are aware of two recent work, Generative Teaching Networks (GTN) (Such et al., 2020) and Kernel Inducing Point (KIP) (Nguyen et al., 2021a). GTN provides only their performance curve on MNIST for 4,096 synthetic images ($\approx$ 400 images/class) but no numerical results, which is slightly worse than our performance with 50 images/class. KIP achieves 95.7±0.1% and 46.9±0.2% testing accuracies on MNIST and CIFAR10 when learning 50 images/class with kernels and testing with one-layer fully connected network, while our results with ConvNet are 99.2±0.1% and 60.6±0.5% respectively. Though our results are significantly better than theirs, the two methods are not directly comparable, as KIP and our DSA use different training and testing architectures. Please refer to Section 2.2.2 for the details of KIP method.

We visualize the generated 10 images/class synthetic sets of MNIST and CIFAR10 in Figure 4.2. Overall the synthetic images capture diverse appearances in the categories, various writing styles in MNIST and a variety of viewpoints and background in CIFAR10. Although it is not our goal, our images are easily recognizable and more similar to real ones than the ones that are reported by Wang et al. (2018); Such et al. (2020); Nguyen et al. (2021a).

CIFAR100 Results. We also evaluate our method on the more challenging CIFAR100 dataset for which few dataset condensation works report their results. Note that compared to CIFAR10, CIFAR100 is more challenging, as discriminating 10 times more categories requires learning more powerful features and also there are $\frac{1}{10}$ fewer images per class in CIFAR100. We present our results in Table 4.2 and compare to the competitive coreset methods (*random*, *herding*) and train set synthesis methods (*LD*, *DC*). Our method obtains 13.9% and 32.3% testing accuracies for 1 and 10 images/class, which improves over DC by 1.1% and 7.1% respectively. Compared to the 10 category datasets, the relative performance gap between the upperbound (56.2±0.3%) and the best performing method (DSA) is significantly bigger in this dataset.

Figure 4.2: Visualization of the generated 10 images/class synthetic sets of MINIST and CIFAR10. The synthetic images are recognizable for human beings.

Img/Cls	Random	Herding	LD†	DC	*DSA*	Whole Dataset
1	4.2±0.3	8.4±0.3	11.5±0.4	12.8±0.3	**13.9±0.3**	56.2±0.3
10	14.6±0.5	17.3±0.3	-	25.2±0.3	**32.3±0.3**	

Table 4.2: The performance (%) comparison on CIFAR100 dataset. LD† use AlexNet for CIFAR100, while the rest use ConvNet.

C\T	MLP	ConvNet	LeNet	AlexNet	VGG	ResNet
MLP	76.5±1.2	73.1±3.6	80.4±2.9	78.2±6.4	58.7±6.5	78.7±3.9
ConvNet	75.6±1.1	88.8±0.8	84.8±1.5	84.7±1.5	83.5±1.7	89.3±0.8
LeNet	76.5±0.9	86.6±1.5	83.9±1.6	83.9±1.2	81.1±2.3	88.2±0.9
AlexNet	76.1±0.8	87.6±0.8	84.2±1.6	84.6±1.7	82.0±2.1	88.8±0.8
VGG	75.8±1.0	88.9±0.7	84.5±1.6	85.0±1.4	83.2±1.9	88.9±1.0
ResNet	75.8±1.0	88.6±0.8	84.8±1.7	84.8±1.2	82.4±1.9	89.5±1.0

Table 4.3: Cross-architecture performance (%). We learn to condense the training set on one architecture (C), and test it on another architecture (T). We learn 1 image/class condensed set on MNIST.

4.4.3 Cross-architecture Generalization

Here we study the cross-architecture performance of our model and report the results in Table 4.3 in MNIST for 1 image/class. To this end, we use different neural architectures to learn the synthetic images and further use the synthetic images to train classifiers. The rows indicate the architecture which is used to learn the synthetic images and columns show the architectures that we train as classifiers. The results show that synthetic images learned by the convolutional architectures (ConvNet, LeNet, AlexNet, VGG and ResNet) perform best and generalize to the other convolutional ones, while the MLP network produces less informative synthetic images overall. Finally the most competitive architecture, ResNet provides the best results when trained as a classifier on the synthetic images.

4.4.4 Ablation Study

Effectiveness of DSA. Here we study the effect of design choices in the proposed DSA in terms of test performance on CIFAR10 for 10 images/class and report it in Table 4.4. One can apply image transformations to the real and synthetic set while learning the synthetic images, and also to the synthetic images while training a classifier in the second stage. In addition, the same image transformation can be applied to all images in a real and synthetic minibatch pair (denoted as $\mathcal{A}_\omega$) or an independently sampled image transformation can be applied to each image (denoted as $\mathcal{A}$). Note that the former corresponds to our proposed Siamese augmentation and we test different augmentation schemes for cropping, flipping, scaling and rotation.

The results verify that the proposed Siamese augmentation always achieves the best performance when used with individual augmentation. The largest improvement is obtained by applying our Siamese augmentation with cropping. Specifically, using Siamese augmentation with cropping achieves 3.6% improvement compared to no data augmentation (A)\@. Note that (A) corresponds to DC (Zhao et al., 2021) with initialization from real images. While a smaller improvement of 1.4% can be obtained by applying cropping only to synthetic data in the test phase (B), DSA provides a further 2.2% over this. Applying cropping only to the real or synthetic images (C and D) degrades the performance and obtains worse performance than no data augmentation (A)\@. Similarly, applying independent transformations to the real and synthetic images when learning synthetic images, *i.e.* (F), leads to worse performance than (A)\@. Finally, the Siamese augmentation method performs worse than (A)

	Condense		Test	Test Performance (%)			
	Real	Synthetic	Synthetic	Crop	Flip	Scale	Rotation
Ours	$\mathcal{A}_\omega$	$\mathcal{A}_\omega$	$\mathcal{A}$	49.1±0.6	47.9±0.7	46.9±0.5	46.8±0.6
(A)	-	-	-	45.5±0.6	45.5±0.6	45.5±0.6	45.5±0.6
(B)	-	-	$\mathcal{A}$	46.9±0.6	46.1±0.6	45.7±0.5	45.0±0.5
(C)	$\mathcal{A}$	-	$\mathcal{A}$	42.8±0.7	46.2±0.6	44.5±0.6	44.5±0.6
(D)	-	$\mathcal{A}$	$\mathcal{A}$	44.6±0.7	46.8±0.6	45.4±0.6	45.9±0.7
(E)	$\mathcal{A}_\omega$	$\mathcal{A}_\omega$	-	43.4±0.5	46.4±0.6	45.7±0.6	46.3±0.5
(F)	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	44.5±0.5	46.9±0.6	45.7±0.5	45.8±0.5

Table 4.4: Ablation study on augmentation schemes in CIFAR10 for 10 synthetic images/class. $\mathcal{A}_\omega$ denotes Siamese augmentation when applied to both real and synthetic data, while $\mathcal{A}$ denotes augmentation that is not shared across real and synthetic minibatches.

when no data augmentation is used to train the classifier (E)\@. This shows that it is important to apply data augmentation consistently in both stages. The effects on other augmentation strategies may be slightly different but similar to those for cropping augmentation.

Augmentation Strategy. Our method can be used with most common image transformations. Here we investigate the performance of our method with several popular transformations including color jittering, cropping, cutout, flipping, scaling, rotation on MNIST, FashionMNIST, SVHN and CIFAR10 for 10 images/class setting. We also show a simple combination strategy that is used as the default augmentation in experiments by randomly sampling one of these six transformations at each time. The exception is that flipping is not included in the combination for the two number datasets - MNIST and SVHN, as it can change the semantics of a number. Note that our goal is not to exhaustively find the best augmentation strategy but to show that our augmentation scheme can be effectively used for dataset condensation and we leave a more elaborate augmentation strategy for future work.

Table 4.5 depicts the results for no transformation, individual transformations and as well as the combined strategy. We find that applying all the augmentations improve the performance on CIFAR10 compared to the baseline (None). Cropping is the most effective single transformation that can increase the testing accuracy from 45.5% to 49.1%. The combination of these augmentations further improves the performance to 52.1%. Interestingly, cropping and cutout transformations degrade the performance of SVHN\@. We suppose that SVHN images are noisy and some include multiple digits

	Img/Cls	None	Color	Crop	Cutout	Flip	Scale	Rotate	Combination
MNIST	10	96.4±0.1	96.5±0.1	97.2±0.1	96.5±0.1	-	97.2±0.1	97.2±0.1	97.8±0.1
FashionMNIST	10	82.5±0.3	82.9±0.3	83.3±0.3	84.0±0.3	83.1±0.2	84.0±0.4	83.1±0.3	84.6±0.3
SVHN	10	76.7±0.6	77.4±0.5	75.9±0.8	73.1±0.6	-	78.0±0.5	77.4±0.4	79.2±0.5
CIFAR10	10	45.5±0.6	47.6±0.5	49.1±0.6	48.1±0.5	47.9±0.7	46.9±0.5	46.8±0.6	52.1±0.5

Table 4.5: Performance with different augmentation strategies. Flipping is not suitable for number datasets - MNIST and SVHN. Combination means randomly sampling one from the six/five transformations to implement each time.

and these transformations may pick the wrong patch of an image. Nevertheless, we still observe that the combined strategy obtains the best performance in all datasets.

4.4.5 Continual Learning

Here we apply our method to a continual learning task (Rebuffi et al., 2017; Castro et al., 2018; Aljundi et al., 2019) where the tasks are incrementally learned on three digit recognition datasets, SVHN (Netzer et al., 2011), MNIST (LeCun et al., 1998) and USPS (Hull, 1994) as in (Zhao et al., 2021) and the goal is to preserve the performance in the seen tasks while learning new ones. We build our model on the popular continual learning baseline – EEIL (Castro et al., 2018) which leverages memory rehearsal and knowledge distillation (Hinton et al., 2015) to mitigate catastrophic forgetting of old tasks. We replace the sample selection strategy, *i.e.* herding, with our dataset condensation method for memory construction and keep the rest the same. The memory budget is 10 images/class for all seen classes. We refer to Zhao et al. (2021) for more details.

Figure 4.3 depicts the results of EEIL with three memory construction strategies - herding (Castro et al., 2018), DC (Zhao et al., 2021) and our DSA under two settings - with and without knowledge distillation. The results show that DSA always outperforms the other two memory construction methods. Especially, DSA achieves 94.4% testing accuracy after learning all three tasks without knowledge distillation, which surpasses DC and herding by 1.4% and 3.7% respectively. In our experiments, the synthetic images learned by our method are more informative for training models than those produced by competitors.

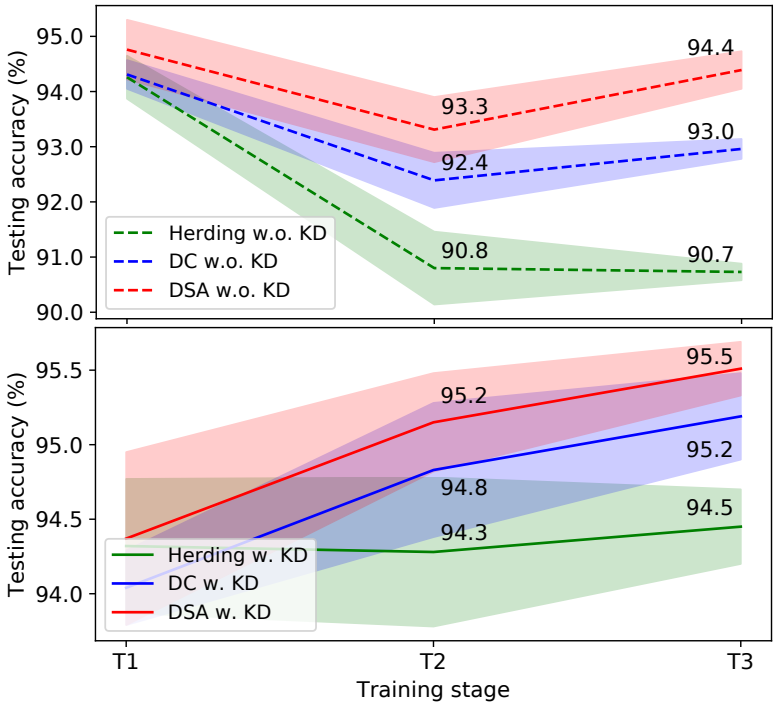


Figure 4.3: Continual learning performance. We compare to the original EEIL (Castro et al., 2018) denoted as Herding and DC (Zhao et al., 2021) under two settings: with and without knowledge distillation. T1, T2, T3 are three learning stages.

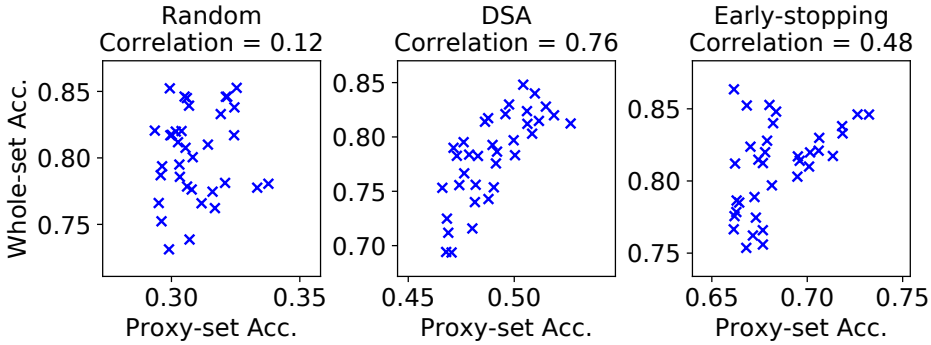


Figure 4.4: The correlation between proxy-set performance and whole-dataset performance on top 5% architectures.

4.4.6 Neural Architecture Search

Our method has substantial practical benefits when one needs to train many neural networks in the same dataset. One such application is neural architecture search (NAS) (Zoph et al., 2018) which aims to search the best network architecture for a given dataset. Here our goal is to verify that a small set of synthetic images learned by our method can be used as a proxy set to efficiently train many networks and the test performances of the neural architectures are correlated to the ones trained on the original training set.

To this end, we design a set of candidate neural network architectures based on a modular ConvNet by varying its depth, width, pooling, activation and normalization layers which produces 720 candidates in total. We refer to Section 3.3.2 for more details. We train these models on the whole CIFAR10 training set for obtaining the ground-truth performance and also three small proxy-sets that are obtained with *random*, *DSA* and *early-stopping* (Li and Talwalkar, 2020). Please refer to Section 3.3.2 for more details about *early-stopping*. We randomly select 10 images/class in *random* and learn 10 images/class condensed set with the default ConvNet in *DSA* as the proxy-sets. We train models for 300 epochs with *random* and *DSA*\@. In *early-stopping* we train models for same amount of iterations to DSA (300 iterations with batch size 100) on the whole original training set. Both *random* and *DSA* use 100 images (0.2% of whole dataset) in total, while *early-stopping* uses 3×10^4 images (60% of whole dataset). For the whole set baseline, we train models for 100 epochs, which is sufficiently long to converge. Finally we pick the best performing architectures on the validation split that are trained on each proxy set, train them on the original training set from scratch and report their test set performance.

We report the results in Table 4.6 in the performance of selected best architecture, correlation between performances of proxy-set and whole-dataset training, training time cost and storage cost. The correlation, *i.e.* Spearman's rank correlation coefficient, is calculated on the top 5% candidate architectures of each proxy-set, which is also illustrated in Figure 4.4. The proxy-set produced by our method achieves the strongest correlation - 0.76, while the time cost of implementing NAS on our proxy-set is only 1.2% of implementing NAS on the whole dataset. This promising performance indicates that our DSA can speed up NAS by training models on small proxy-set. Although the model chosen by *early-stopping* achieves better

	Random	DSA	Early-stopping	Whole Dataset
Performance (%)	78.2	81.3	**84.3**	85.9
Correlation	0.12	**0.76**	0.48	1.00
Time cost (min)	**32.5**	**44.5**	**32.6**	3580.2
Storage (imgs)	$\mathbf{10^2}$	$\mathbf{10^2}$	3×10^4	5×10^4

Table 4.6: Neural architecture search on proxy-sets and whole dataset. The search space is 720 ConvNets. We do experiments on CIFAR10 with 10 images/class randomly selected coreset and synthetic set learned by *DSA*\@. *Early-stopping* means training models on whole dataset but with the same iterations as *random* and *DSA*\@.

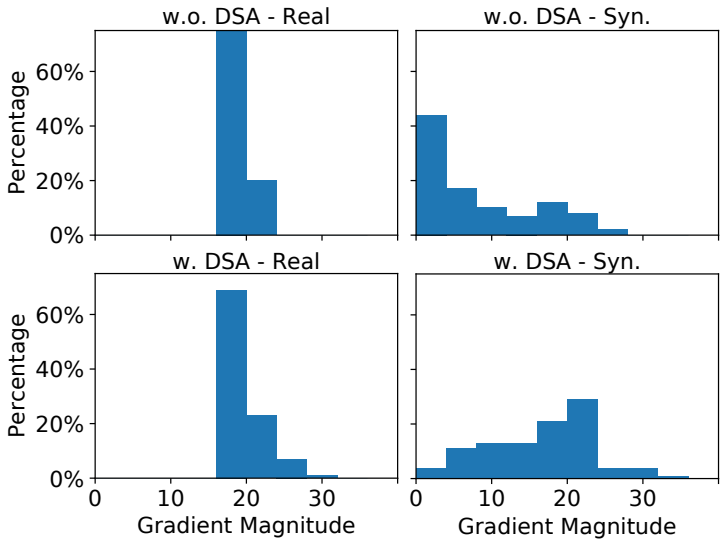


Figure 4.5: Gradient magnitude distribution with respect to real/synthetic data.

performance than ours, *early-stopping* requires two orders of magnitude more training images than ours. In addition, the correlation (0.48) between *early-stopping* performance and whole-dataset training performance is significantly lower than ours (0.76).

4.5 Discussion

4.5.1 Why Does DSA Work?

In this section, we attempt to shed light onto why DSA leads to better synthetic data. We hypothesize that the Siamese augmentation acts as a strong regularizer on the learned high-dimensional synthetic data $\mathcal{S}$ and alleviates its overfitting to the real training set. We refer to Hernández-García and König (2018) for more elaborate

analysis of the relation between data augmentation and regularization. This can be shown more explicitly by reformulating Equation (4.3) over multiple randomly sampled augmentations:

$$\min_{\mathcal{S}} \sum_{\omega \sim \Omega} D(\nabla_{\theta}\mathcal{L}(\mathcal{A}(\mathcal{S},\omega),\theta), \nabla_{\theta}\mathcal{L}(\mathcal{A}(\mathcal{T},\omega),\theta)), \tag{4.4}$$

which forces the synthetic set to match the gradients from the real set under multiple transformations ω when sampled from the distribution Ω and renders the optimization harder and less prone to overfitting.

We also quantitatively analyze this by reporting the gradient magnitude distribution $||\nabla_{\theta}\mathcal{L}(\mathcal{T})||$ and $||\nabla_{\theta}\mathcal{L}(\mathcal{S})||$ for real and synthetic sets respectively in Figure 4.5 when learning 10 images/class synthetic set on CIFAR10 with and without DSA\@. The gradients are obtained at the training iteration $k = 1000$ (see Alg. 2). We see that the gradient magnitudes from the synthetic data quickly vanishes and thus leads to a very small updates in absence of DSA, while the synthetic images can still be learned with DSA\@. Note that as backpropagation involves successive products of gradients, the updates for $\mathcal{S}$ naturally vanishes when multiplied with small $||\nabla_{\theta}\mathcal{L}(\mathcal{S})||$.

4.5.2 Initializing Synthetic Images

In our experiments, we initialize each synthetic image with a randomly sampled real training image (after standard image normalization) from the corresponding category. After the initialization, we update them by using the optimization in Equation (4.3). Once they are trained, they are used to train neural networks without any post-processing. In Figure 4.6, we illustrate the evolution of synthetic data initialized from random noise and real images from car and cat categories through our training in CIFAR10. While we see significant changes over the initialization in both cases, the ones initialized with real images preserve some of their contents such as object pose and color.

4.5.3 Disentangling Semantics and Prior Knowledge

Here, we give an intuitive explanation why our method is better at disentangling semantics and priors. If we randomly crop training images in a batch, for example the "car" images, some images may contain the "land" pattern while the "land" was cropped in other images. In every training iteration, the averaged representations or gradients of a batch will "always" contain a certain ratio of "land" pattern, which is

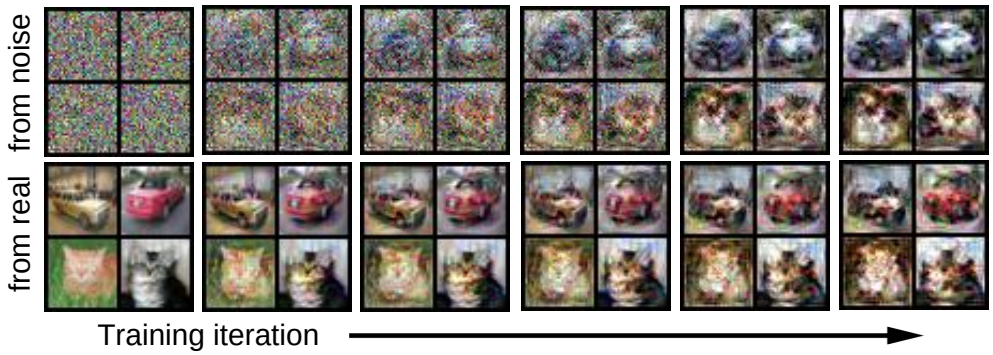


Figure 4.6: The learning/rendering process of two classes in CIFAR10 initialized from random noise and real images respectively.

hard to disentangle from "car" semantics when learning synthetic images by matching the mean gradients. In contrast, exactly the same "cropping" is applied to all real and synthetic images in a batch in our differentiable Siamese augmentation. Thus, for some iteration, the averaged representations or gradients of a batch will explicitly have or not have the "land" pattern. Hence, "car" and "land" are manually disentangled, and thus learning the disentanglement of semantics and priors is much easier.

4.6 Conclusion and Limitation

In this paper, we propose a principled dataset condensation method – Differentiable Siamese Augmentation – to enable learning synthetic training set that can be effectively used with data augmentation when training deep neural networks. Experiments and ablation study show that the learned synthetic training set can be used to train neural networks with data augmentation and achieve significantly better performance (about 7% improvement on CIFAR10/100) than state-of-the-art methods. We also show promising results when applying the proposed method to continual learning and neural architecture search.

Limitations. The proposed method improves the efficiency of synthetic data by enabling them to be used more efficiently with augmentation. However, the limitations of gradient matching method Zhao et al. (2021), such as the scalability, optimization and generalization problems, still remain. In the future, we will further improve the utility of dataset condensation by removing these limitations.

Chapter 5

Dataset Condensation with Distribution Matching

Computational cost of training state-of-the-art deep models in many learning problems is rapidly increasing due to more sophisticated models and larger datasets. A recent promising direction for reducing training cost is dataset condensation that aims to replace the original large training set with a significantly smaller learned synthetic set while preserving the original information. While training deep models on a small set of condensed images can be extremely fast, their synthesis remains computationally expensive due to the complex bi-level optimization and second-order derivative computation. In this work, we propose a simple yet effective method that synthesizes condensed images by matching feature distributions of the synthetic and original training images in many sampled embedding spaces. Our method significantly reduces the synthesis cost while achieving comparable or better performance. Thanks to its efficiency, we apply our method to more realistic and larger datasets with sophisticated neural architectures and obtain a significant performance boost. We also show promising practical benefits of our method in continual learning and neural architecture search.

5.1 Introduction

Computational cost for training a single state-of-the-art model in various fields, including computer vision and natural language processing, doubles every 3.4 months in the deep learning era due to larger models and datasets (Amodei et al., 2018). While training a single model can be expensive, designing new deep learning models

or applying them to new tasks certainly require substantially more computations, as they involve to train multiple models on the same dataset for many times to verify the design choices, such as loss functions, architectures and hyper-parameters (Bergstra and Bengio, 2012; Elsken et al., 2019). For instance, Ying et al. (2019) spend 100 TPU years of computation time conducting an exhaustive neural architecture search on CIFAR10 dataset (Krizhevsky et al., 2009), while training the best-performing architectures take only dozens of TPU minutes. Hence, there is a strong demand for techniques that can reduce the computational cost for training multiple models on the same dataset with minimal performance drop. To this end, this paper focuses on lowering the training cost by reducing the training set size.

The traditional solution to reduce training set size is coreset selection. Typically, coreset selection methods choose samples that are important for training based on heuristic criteria, for example, minimizing distance between coreset and whole-dataset centers (Chen et al., 2010; Rebuffi et al., 2017; Castro et al., 2018; Belouadah and Popescu, 2020), maximizing the diversity of selected samples (Aljundi et al., 2019), discovering cluster centers (Farahani and Hekmatfar, 2009; Sener and Savarese, 2018), counting the mis-classification frequency (Toneva et al., 2019) and choosing samples with the largest negative implicit gradient (Borsos et al., 2020). Although coreset selection methods can be very computationally efficient, they have two major limitations. First most methods incrementally and greedily select samples, which is short-sighted. Second their efficiency is upper bounded by the information in the selected samples in the original dataset.

An effective way of tackling the information bottleneck is *synthesizing* informative samples rather than selecting from given samples. A recent approach, dataset condensation (or distillation) (Wang et al., 2018; Zhao et al., 2021) aims to learn a small synthetic training set so that a model trained on it can obtain comparable testing accuracy to that trained on the original training set. Wang et al. (2018) pose the problem in a learning-to-learn framework by formulating the network parameters as a function of synthetic data and learning them through the network parameters to minimize the training loss over the original data. An important shortcoming of this method is the expensive optimization procedure that involves updating network weights for multiple steps for each outer iteration and unrolling its recursive computation graph. Zhao et al. (2021) propose to match the gradients with respect to the network weights giving real and synthetic training images that successfully avoids the expensive unrolling of the computational graph. Another efficiency improvement

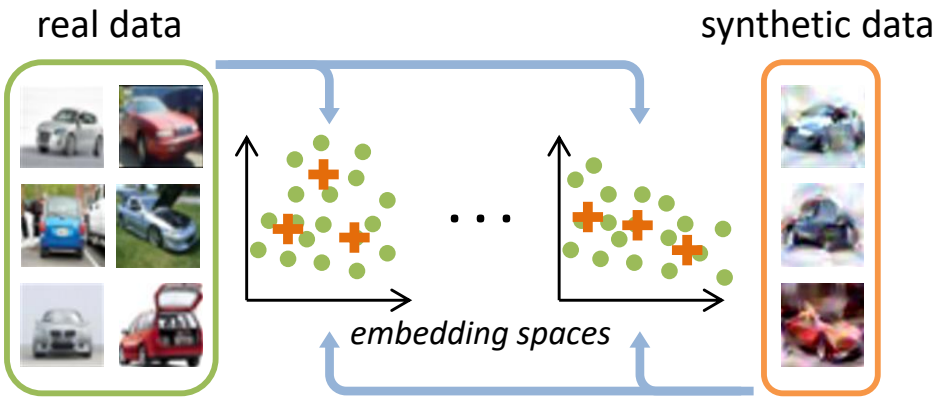


Figure 5.1: Dataset Condensation with Distribution Matching. We randomly sample real and synthetic data, and then embed them with the randomly sampled deep neural networks. We learn the synthetic data by minimizing the distribution discrepancy between real and synthetic data in these sampled embedding spaces.

is a closed form optimizer by posing the classification task as a ridge regression problem to simplify the inner-loop model optimization (Bohdal et al., 2020; Nguyen et al., 2021a). In spite of recent progress, dataset condensation still requires solving an expensive bi-level optimization which jeopardizes their goal of reducing training time due to the expensive image synthesis process. For instance, the state-of-the-art (Zhao and Bilen, 2021) requires 15 hours of GPU time to learn 500 synthetic images on CIFAR10 which equals to the cost of training 6 deep networks on the same dataset. In addition, these methods also require tuning multiple hyper-parameters, *e.g.* the steps to update synthetic set and network parameters respectively in each iteration, that can be different for different settings such as sizes of synthetic sets.

In this paper, we propose a novel training set synthesis technique that combines the advantages of previous coreset and dataset condensation methods while avoiding their limitations. Unlike the former and like the latter, our method is not limited to individual samples from original dataset and can synthesize training images. Like the former and unlike the latter, our method can very efficiently produce a synthetic set and avoid expensive bi-level optimization. In particular, we pose this task as a distribution matching problem such that the synthetic data are optimized to match the original data distribution in a family of embedding spaces by using the maximum mean discrepancy, namely MMD measurement (Gretton et al., 2012) (see Figure 5.1). Distance between data distributions is commonly used as the criterion for coreset selection (Chen et al., 2010; Farahani and Hekmatfar, 2009; Wang and Ye, 2015; Sener and Savarese, 2018), however, it has not been used to synthesize training data

before. We show that the family of embedding spaces can be efficiently obtained by sampling randomly initialized deep neural networks. Hence, our method is significantly faster (*e.g.* 45× in CIFAR10 when synthesizing 500 images) than the state-of-the-art (Zhao and Bilen, 2021) and involves tuning only one hyperparameter (learning rate for synthetic images), while obtaining comparable or better results. In addition, the training of our method can be independently run for each class in parallel and its computation load can be distributed. Finally, our method provides a different training cost/performance tradeoff for large-scale settings. In contrast to prior works (Wang et al., 2018; Nguyen et al., 2021a) that are limited to learning small synthetic sets on small datasets, our method can be successfully applied in more realistic settings, *i.e.* synthesizing 1250 images per class for CIFAR10 (Krizhevsky et al., 2009), and larger datasets, *i.e.* TinyImageNet (Le and Yang, 2015) and ImageNet-1K (Deng et al., 2009). We also validate these benefits in two downstream tasks by producing more data-efficient memory for continual learning and generating more representative proxy dataset for accelerating neural architecture search.

5.2 Methodology

5.2.1 Dataset Condensation Problem

Dataset condensation aims to condense the large training set $\mathcal{T} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_{|\mathcal{T}|}, y_{|\mathcal{T}|})\}$ with $|\mathcal{T}|$ image and label pairs into a small synthetic set $\mathcal{S} = \{(\boldsymbol{s}_1, y_1), \dots, (\boldsymbol{s}_{|\mathcal{S}|}, y_{|\mathcal{S}|})\}$ with $|\mathcal{S}|$ synthetic image and label pairs so that models trained on each $\mathcal{T}$ and $\mathcal{S}$ obtain comparable performance on unseen testing data:

$$\mathbb{E}_{\boldsymbol{x} \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{T}}}(\boldsymbol{x}), y)] \simeq \mathbb{E}_{\boldsymbol{x} \sim P_{\mathcal{D}}}[\ell(\phi_{\boldsymbol{\theta}^{\mathcal{S}}}(\boldsymbol{x}), y)], \tag{5.1}$$

where $P_{\mathcal{D}}$ is the real data distribution, ℓ is the loss function (*i.e.* cross-entropy loss), ϕ is a deep neural network parameterized by $\boldsymbol{\theta}$, and $\phi_{\boldsymbol{\theta}^{\mathcal{T}}}$ and $\phi_{\boldsymbol{\theta}^{\mathcal{S}}}$ are the networks that are trained on $\mathcal{T}$ and $\mathcal{S}$ respectively.

Existing Solutions. Previous works (Wang et al., 2018; Sucholutsky and Schonlau, 2019; Such et al., 2020; Bohdal et al., 2020; Nguyen et al., 2021a,b) formulate the dataset condensation as a learning-to-learn problem, pose the network parameters $\boldsymbol{\theta}^{\mathcal{S}}$ as a function of synthetic data $\mathcal{S}$ and obtain a solution for $\mathcal{S}$ by minimizing the training

loss $\mathcal{L}^{\mathcal{T}}$ over the original data $\mathcal{T}$:

$$\begin{aligned} \mathcal{S}^* &= \arg\min_{\mathcal{S}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S})) \\ \text{subject to} \quad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) &= \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}). \end{aligned} \tag{5.2}$$

Recently the authors of (Zhao et al., 2021; Zhao and Bilen, 2021) show that a similar goal can be achieved by matching gradients of the losses over the synthetic and real training data respectively with respect to the network parameters $\boldsymbol{\theta}$, while optimizing $\boldsymbol{\theta}$ and the synthetic data $\mathcal{S}$ in an alternating way:

$$\begin{aligned} \mathcal{S}^* &= \arg\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}} \Big[\sum_{t=0}^{T-1} D(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_t)) \Big] \\ \text{subject to} \quad \boldsymbol{\theta}_{t+1} &\leftarrow \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \varsigma_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\theta}}), \end{aligned} \tag{5.3}$$

where $P_{\boldsymbol{\theta}_0}$ is the distribution of parameter initialization, T is the outer-loop iteration for updating synthetic data, $\varsigma_{\boldsymbol{\theta}}$ is the inner-loop iteration for updating network parameters, $\eta_{\boldsymbol{\theta}}$ is the parameter learning rate and $D(\cdot, \cdot)$ measures the gradient matching error. Note that all the training algorithms (Wang et al., 2018; Zhao et al., 2021; Zhao and Bilen, 2021) have another loop of sampling $\boldsymbol{\theta}_0$ over the bi-level optimization.

Dilemma. The learning problems in eq. (5.2) and eq. (5.3) involve solving an expensive bi-level optimization: first optimizing the model $\boldsymbol{\theta}^{\mathcal{S}}$ in eq. (5.2) or $\boldsymbol{\theta}_t$ in eq. (5.3) at the inner loop, then optimizing the synthetic data $\mathcal{S}$ along with additional second-order derivative computation at the outer loop. For example, training 50 images/class synthetic set $\mathcal{S}$ by using the method in (Zhao et al., 2021) requires 500K epochs of updating network parameters $\boldsymbol{\theta}_t$ on $\mathcal{S}$, in addition to the 50K updating of $\mathcal{S}$. Furthermore, Zhao et al. (2021) need to tune the hyper-parameters of the outer and inner loop optimization (*i.e.* how many steps to update $\mathcal{S}$ and $\boldsymbol{\theta}_t$) for different learning settings, which requires cross-validating them and hence multiplies the cost for training synthetic images.

5.2.2 Dataset Condensation with Distribution Matching

Our goal is to synthesize data that can accurately approximate the distribution of the real training data in a similar spirit to coreset techniques (*e.g.* (Chen et al., 2010; Sener and Savarese, 2018)). However, to this end, we do not limit our method to select a subset of the training samples but to synthesize them as in Wang et al. (2018); Zhao et al. (2021). As the training images are typically very high dimensional, estimating

the real data distribution $P_{\mathcal{D}}$ can be expensive and inaccurate. Instead, we assume that each training image $\boldsymbol{x} \in \Re^d$ can be embedded into a lower dimensional space by using a family of parametric functions $\psi_{\boldsymbol{\vartheta}} : \Re^d \rightarrow \Re^{d'}$ where $d' \ll d$ and $\boldsymbol{\vartheta}$ is the parameter. In other words, each embedding function ψ can be seen as providing a partial interpretation of its input, while their combination provides a complete one.

Now we can estimate the distance between the real and synthetic data distribution with commonly used maximum mean discrepancy, namely MMD (Gretton et al., 2012):

$$\sup_{\|\psi_{\boldsymbol{\vartheta}}\|_{\mathcal{H}} \leq 1} (\mathbb{E}[\psi_{\boldsymbol{\vartheta}}(\mathcal{T})] - \mathbb{E}[\psi_{\boldsymbol{\vartheta}}(\mathcal{S})]), \tag{5.4}$$

where $\mathcal{H}$ is reproducing kernel Hilbert space. As we do not have access to ground-truth data distributions, we use the empirical estimate of the MMD:

$$\mathbb{E}_{\boldsymbol{\vartheta} \sim P_{\boldsymbol{\vartheta}}} \| \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \psi_{\boldsymbol{\vartheta}}(\boldsymbol{x}_i) - \frac{1}{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{S}|} \psi_{\boldsymbol{\vartheta}}(\boldsymbol{s}_j) \|^2, \tag{5.5}$$

where $P_{\boldsymbol{\vartheta}}$ is the distribution of network parameters.

Following Chapter 4, we also apply the differentiable Siamese augmentation $\mathcal{A}(\cdot, \omega)$ to real and synthetic data that implements the same randomly sampled augmentation to the real and synthetic minibatch in training, where $\omega \sim \Omega$ is the augmentation parameter such as the rotation degree. Thus, the learned synthetic data can benefit from semantics-preserving transformations (*e.g.* cropping) and learn prior knowledge about spatial configuration of samples while training deep neural networks with data augmentation. Finally, we solve the following optimization problem:

$$\min_{\mathcal{S}} \mathbb{E}_{\substack{\boldsymbol{\vartheta} \sim P_{\boldsymbol{\vartheta}} \\ \omega \sim \Omega}} \| \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \psi_{\boldsymbol{\vartheta}}(\mathcal{A}(\boldsymbol{x}_i, \omega)) - \frac{1}{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{S}|} \psi_{\boldsymbol{\vartheta}}(\mathcal{A}(\boldsymbol{s}_j, \omega)) \|^2. \tag{5.6}$$

We learn the synthetic data $\mathcal{S}$ by minimizing the discrepancy between two distributions in various embedding spaces by sampling $\boldsymbol{\vartheta}$. Importantly eq. (5.6) can be efficiently solved, as it requires only optimizing $\mathcal{S}$ but no model parameters and thus avoids expensive bi-level optimization. This is in contrast to the existing formulations (see eq. (5.2) and eq. (5.3)) that involve bi-level optimizations over network parameters $\boldsymbol{\theta}$ and the synthetic data $\mathcal{S}$.

Note that, as we target image classification problems, we minimize the discrepancy between the real and synthetic samples of the same class only. We assume that each real training sample is labelled and we also set a label to each synthetic sample and keep it fixed during training.

5.2.3 Training Algorithm

We depict the mini-batch based training algorithm in Algorithm 3. We train the synthetic data for K iterations. In each iteration, we randomly sample the model $\psi_{\boldsymbol{\vartheta}}$ with parameter $\boldsymbol{\vartheta} \sim P_{\boldsymbol{\vartheta}}$. Then, we sample a pair of real and synthetic data batches ($B_c^{\mathcal{T}} \sim \mathcal{T}$ and $B_c^{\mathcal{S}} \sim \mathcal{S}$) and augmentation parameter $\omega_c \sim \Omega$ for every class c. The mean discrepancy between the augmented real and synthetic batches of every class is calculated and then summed as loss $\mathcal{L}$. The synthetic data $\mathcal{S}$ is updated by minimizing $\mathcal{L}$ with stochastic gradient descent and learning rate η.

Algorithm 3: Dataset condensation with distribution matching

Input: Training set $\mathcal{T}$

Required: Randomly initialized set of synthetic samples $\mathcal{S}$ for C classes, deep neural network $\psi_{\boldsymbol{\vartheta}}$ parameterized with $\boldsymbol{\vartheta}$, probability distribution over parameters $P_{\boldsymbol{\vartheta}}$, differentiable augmentation $\mathcal{A}_\omega$ parameterized with ω, augmentation parameter distribution Ω, training iterations K, learning rate η.

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

- Sample $\boldsymbol{\vartheta} \sim P_{\boldsymbol{\vartheta}}$
- Sample mini-batch pairs $B_c^{\mathcal{T}} \sim \mathcal{T}$ and $B_c^{\mathcal{S}} \sim \mathcal{S}$ and $\omega_c \sim \Omega$ for every class c
- Compute $\mathcal{L} = \sum_{c=0}^{C-1} \| \frac{1}{|B_c^{\mathcal{T}}|} \sum_{(\boldsymbol{x},y)\in B_c^{\mathcal{T}}} \psi_{\boldsymbol{\vartheta}}(\mathcal{A}_{\omega_c}(\boldsymbol{x})) - \frac{1}{|B_c^{\mathcal{S}}|} \sum_{(\boldsymbol{s},y)\in B_c^{\mathcal{S}}} \psi_{\boldsymbol{\vartheta}}(\mathcal{A}_{\omega_c}(\boldsymbol{s})) \|^2$
- Update $\mathcal{S} \leftarrow \mathcal{S} - \eta \nabla_{\mathcal{S}} \mathcal{L}$

Output: $\mathcal{S}$

5.2.4 Discussion

Randomly Initialized Networks. The family of embedding functions $\psi_{\boldsymbol{\vartheta}}$ can be designed in different ways. Here we use a deep neural network with different random initializations rather than sampling its parameters from a set of pre-trained networks which is more computationally expensive to obtain. We experimentally validate that our random initialization strategy produces better or comparable results with the more expensive strategy of using pretrained networks in Section 5.3.4. However, one may still question why randomly initialized networks provide meaningful embeddings for distribution matching. Here we list two reasons based on the observations from previous work. First, randomly initialized networks are reported to produce powerful representations for multiple computer vision tasks (Saxe et al., 2011; Cao et al., 2018;

Amid et al., 2022). Second, such random networks are showed to perform a distance-preserving embedding of the data, *i.e*. smaller distances between samples of same class and larger distances across samples of different classes (Giryes et al., 2016). In addition, the combination of many weak embeddings provides a complete interpretation of the inputs.

Connection to Gradient Matching. While we match the mean features of the real and synthetic image batches, Zhao et al. (2021) match the mean gradients of network weights over the two batches. We find that, given a batch of data from the same class, the mean gradient vector with respect to each output neuron in the last layer of a network is equivalent to a weighted mean of features where the weights are a function of classification probabilities predicted by the network and proportional to the distance between prediction and ground-truth. In other words, while our method weighs each feature equally, Zhao et al. (2021) assign larger weights to samples whose predictions are inaccurate. Note that these weights dynamically vary for different networks and training iterations. We provide the derivation in Appendix B.4.

Generative Models. The classic image synthesizing techniques, includes AutoEncoders (Kingma and Welling, 2013) and Generative Adversarial Networks (GANs) (Goodfellow et al., 2014a), aim to synthesize real-looking images, while our goal is to generate data-efficient training samples. Regularizing the images to look real may limit the data-efficiency. Previous work (Zhao et al., 2021) showed that the images synthesized by cGAN (Mirza and Osindero, 2014) are not better than the randomly selected real images for training networks. We further provide the comparison to state-of-the-art VAE and GAN models and GMMN method (Li et al., 2015) in Appendix B.2. Although generative models can be trained to produce data-efficient training samples with adapted objectives, *e.g*. Wang et al. (2018) and our gradient matching in Chapter 3 and distribution matching in Chapter 5, attempts so far have not achieved state-of-the-art results (Such et al., 2020). We leave it as future work.

5.3 Experiments

5.3.1 Experimental Settings

Datasets. We evaluate the classification performance of deep networks that are trained on the synthetic images generated by our method. We conduct experiments on

five datasets including MNIST (LeCun et al., 1998), CIFAR10, CIFAR100 (Krizhevsky et al., 2009), TinyImageNet (Le and Yang, 2015) and ImageNet-1K (Deng et al., 2009). MNIST consists of 60K 28×28 gray-scale training images of 10 classes. CIFAR10 and CIFAR100 contain 50k 32×32 training images from 10 and 100 object categories respectively. TinyImageNet and ImageNet-1K have 100K training images from 200 categories and 1.3M training images from 1K categories respectively. We resize these ImageNet images with 64×64 resolution. These two datasets are significantly more challenging than MNIST and CIFAR10/100 due to more diverse classes and higher image resolution.

Experimental Settings. We first learn 1/10/50 image(s) per class synthetic sets for all datasets by using the same ConvNet architecture in (Zhao et al., 2021). Then, we use the learned synthetic sets to train randomly initialized ConvNets from scratch and evaluate them on real test data. The default ConvNet includes three repeated convolutional blocks, and each block involves a 128-kernel convolution layer, instance normalization layer (Ulyanov et al., 2016), ReLU activation function (Nair and Hinton, 2010) and average pooling. Note that four-block ConvNets are used to adjust to the larger input size (64×64) of TinyImageNet and ImageNet-1K images. In each experiment, we learn one synthetic set and use it to test 20 randomly initialized networks. We repeat each experiment for 5 times and report the mean testing accuracy of the 100 trained networks. We also do cross-architecture experiments in Section 5.3.3. where we learn the synthetic set on one network architecture and use them to train networks with different architectures.

Hyper-parameters. Like the standard neural network training, dataset condensation also involves tuning a set of hyperparameters. Our method needs to tune **only one hyper-parameter**, *i.e.* the learning rate for the synthetic images, for learning different sizes of synthetic sets, while existing methods (Wang et al., 2018; Zhao et al., 2021; Nguyen et al., 2021a; Wang et al., 2022; Cazenavette et al., 2022a) have to tune more hyper-parameters such as the steps to update synthetic images and network parameters respectively. We use a fixed learning rate of 1 for optimizing synthetic images for all 1/10/50 images/class learning on all datasets. When learning larger synthetic sets such as 100/200/500/1,000 images per class, we use a larger learning rate (*i.e.* 10) due to the relatively smaller distribution matching loss. We train synthetic images for 20,000 iterations on MNIST, CIFAR10/100 and 10,000 iterations on TinyImageNet and ImageNet-1K respectively. The mini-batch size for sampling real data is 256. We initialize the synthetic images using randomly sampled real images with corresponding

labels. All synthetic images of a class are used to compute the class mean. We use the same augmentation strategy proposed in Chapter 4.

5.3.2 Comparison to State of the Art

Competitors. We compare our method to three standard coreset selection methods, namely, Random Selection, Herding (Chen et al., 2010; Rebuffi et al., 2017; Castro et al., 2018; Belouadah and Popescu, 2020) and Forgetting (Toneva et al., 2019). Herding method greedily adds samples into the coreset so that the mean vector is approaching the whole dataset mean. Toneva et al. (2019) count how many times a training sample is learned and then forgotten during network training. The samples that are less forgetful can be dropped. Please refer to Section 2.1 for implementation details of sample selection methods. We also compare to four state-of-the-art training set synthesis methods, namely, DD (Wang et al., 2018), LD (Bohdal et al., 2020), DC (Zhao et al., 2021) and DSA (Zhao and Bilen, 2021). Please refer to Section 3.2 and Section 4.3 for implementation details of DC and DSA respectively. Note that we are aware of concurrent works (Lee et al., 2022; Kim et al., 2022; Cazenavette et al., 2022a) that largely improves the existing bilevel optimization based dataset condensation solutions. Unlike them, we contribute the first solution that has neither bi-level optimization nor second-order derivative, and provide a different training cost/performance tradeoff. Compared to them, our method is significantly simpler and faster. Thus, it is able to scale to large settings *i.e.* learning 1250 images per class for CIFAR10 and large datasets *i.e.* ImageNet-1K. More detailed comparison and discussion to other methods (Such et al., 2020; Nguyen et al., 2021a,b), MMD baseline (Gretton et al., 2012) and generative baselines including DC-VAE (Parmar et al., 2021), BigGAN (Brock et al., 2019) and GMMN (Li et al., 2015) can be found in Appendix B.2.

Performance Comparison. Here we evaluate our method on MNIST, CIFAR10 and CIFAR100 datasets and report the results in Table 5.1. Among the coreset selection methods, Herding performances the best in most settings. Especially, when small synthetic sets are learned, the Herding method performs significantly better. For example, Herding achieves 8.4% testing accuracy when learning 1 image/class synthetic set on CIFAR100, while Random and Forgetting obtains only 4.2% and 4.5% testing accuracies respectively.

Training set synthesis methods have clear superiority over coreset selection

	Img/Cls	Ratio %	Coreset Selection			Training Set Synthesis					Whole-Dataset
			Random	Herding	Forgetting	DD†	LD†	DC	DSA	*DM*	
MNIST	1	0.017	64.9±3.5	89.2±1.6	35.5±5.6		60.9±3.2	**91.7±0.5**	88.7±0.6	89.7±0.6	99.6±0.0
	10	0.17	95.1±0.9	93.7±0.3	68.1±3.3	79.5±8.1	87.3±0.7	97.4±0.2	**97.8±0.1**	97.5±0.1	
	50	0.83	97.9±0.2	94.8±0.2	88.2±1.2	-	93.3±0.3	98.8±0.2	**99.2±0.1**	98.6±0.1	
CIFAR10	1	0.02	14.4±2.0	21.5±1.2	13.5±1.2	-	25.7±0.7	**28.3±0.5**	**28.8±0.7**	26.0±0.8	84.8±0.1
	10	0.2	26.0±1.2	31.6±0.7	23.3±1.0	36.8±1.2	38.3±0.4	44.9±0.5	**52.1±0.5**	48.9±0.6	
	50	1	43.4±1.0	40.4±0.6	23.3±1.1	-	42.5±0.4	53.9±0.5	60.6±0.5	**63.0±0.4**	
CIFAR100	1	0.2	4.2±0.3	8.4±0.3	4.5±0.2	-	11.5±0.4	12.8±0.3	**13.9±0.3**	11.4±0.3	56.2±0.3
	10	2	14.6±0.5	17.3±0.3	15.1±0.3	-	-	25.2±0.3	**32.3±0.3**	29.7±0.3	
	50	10	30.0±0.4	33.7±0.5	30.5±0.3	-	-	-	42.8±0.4	**43.6±0.4**	
Tiny-ImageNet	1	0.2	1.4±0.1	2.8±0.2	1.6±0.1	-	-	-	-	**3.9±0.2**	37.6±0.4
	10	2	5.0±0.2	6.3±0.2	5.1±0.2	-	-	-	-	**12.9±0.4**	
	50	10	15.0±0.4	16.7±0.3	15.0±0.3	-	-	-	-	**24.1±0.3**	

Table 5.1: Comparing to coreset selection and training set synthesis methods. We first learn the synthetic data and then evaluate them by training neural networks from scratch and testing on real testing data. The testing accuracies (%) are reported. Img/Cls: image(s) per class. Ratio (%): the ratio of condensed set size to the whole training set size. Note: DD† and LD† use different architectures *i.e.* LeNet for MNIST and AlexNet for CIFAR10. The rest methods all use ConvNet.

methods, as the synthetic training data are not limited to a set of real images. The best results were obtained either by DSA or our method. While DSA produces more data-efficient samples with a small number of synthetic samples (1/10 image(s) per class), our method outperforms DSA at 50 images/class setting in CIFAR10 and CIFAR100. The possible reason is that the inner-loop model optimization in DSA with limited number of steps is more effective to fit the network parameters on smaller synthetic data (see eq. (5.3)). In case of bigger learned synthetic data, the solution obtained in the inner-loop becomes less accurate as it can use only limited number of steps to keep the algorithm scalable. In contrast, our method is robust to increasing synthetic data size, and can be efficiently optimized significantly faster than DSA\@.

TinyImageNet and ImageNet-1K. Due to higher image resolution and more diverse classes, prior bilevel optimization based methods do not scale to TinyImageNet and ImageNet-1K. Our method takes 27 hours with one Tesla V100 GPU to condense TinyImageNet into three condensed sets (1/10/50 images/class synthetic sets), and it takes 28 hours with ten GTX 1080 GPUs to condense ImageNet-1K into these three sets. As shown in Table 5.1, our method achieves 3.9%, 12.9% and 24.1% testing accuracies when learning 1, 10 and 50 images/class synthetic sets for TinyImageNet, and recovers 60% classification performance of the baseline that is trained on the whole original training set with only 10% of data. Our

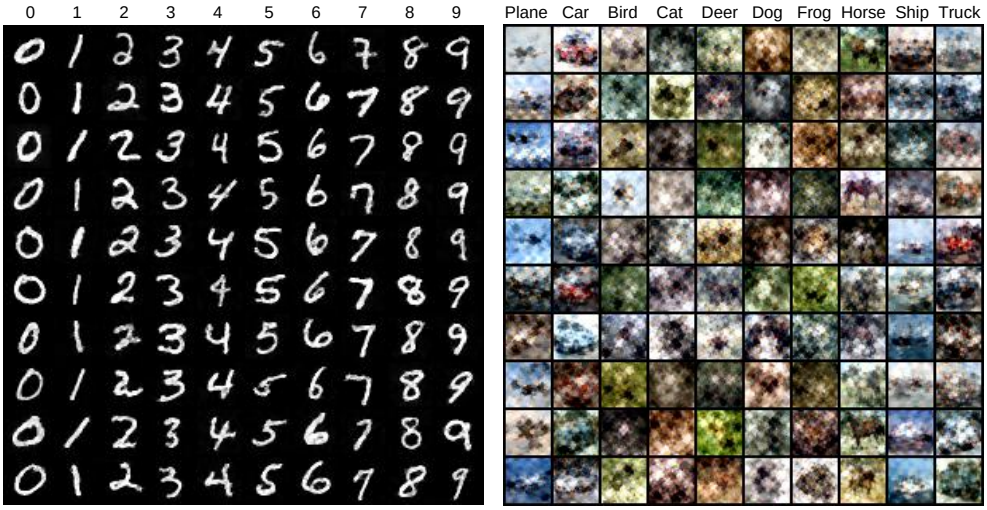


Figure 5.2: Visualization of generated 10 images per class synthetic sets of MNIST and CIFAR10 datasets.

method significantly outperforms the best coreset selection method, Herding, which obtains 2.8%, 6.3% and 16.7% testing accuracies. On the ImageNet-1K dataset, our method achieves 1.3%, 5.7% and 11.4% testing accuracies when learning 1, 10 and 50 images/class synthetic sets, which outperforms random selection (0.52%, 1.94% and 7.54%) by large margins.

Visualization. The learned synthetic images of MNIST and CIFAR10 are visualized in Figure 5.2. We find that the synthetic MNIST images are clear and noise free, while the number images synthesized by previous methods contain obvious noise and some unnatural strokes. The synthetic images of CIFAR10 dataset are also visually recognizable and diverse. It is easy to distinguish the background and foreground object.

Figure 5.3 depicts the feature distribution of the (50 images/class) synthetic sets learned by DC, DSA and our method (DM)\@. We use a network trained on the whole training set to extract features and visualize the features with T-SNE (Van der Maaten and Hinton, 2008). We find that the synthetic images learned by DC and DSA do not cover the real image distribution. In contrast, our synthetic images successfully cover the real image distribution. Furthermore, fewer outlier synthetic samples are produced by our method.

Learning with Batch Normalization. Zhao et al. (2021) showed that instance normalization (Ulyanov et al., 2016) works better than batch normalization, namely BN (Ioffe and Szegedy, 2015), when learning small synthetic sets, because the synthetic data number is too small to calculate stable running mean and standard

	InstanceNorm		BatchNorm	
	DSA	DM	DSA	DM
CIFAR10	60.6±0.5	**63.0±0.4**	59.9±0.8	**65.2±0.4**
CIFAR100	42.8±0.4	**43.6±0.4**	44.6±0.5	**48.0±0.4**
TinyImageNet	-	**24.1±0.3**	-	**28.2±0.5**

Table 5.2: 50 images/class learning with Batch Normalization.

	C\T	ConvNet	AlexNet	VGG	ResNet
DSA	ConvNet	59.9±0.8	53.3±0.7	51.0±1.1	47.3±1.0
DM	ConvNet	65.2±0.4	61.3±0.6	59.9±0.8	57.0±0.9
	AlexNet	60.5±0.4	59.8±0.6	58.9±0.4	54.6±0.7
	VGG	54.2±0.6	52.6±1.0	52.8±1.1	49.1±1.0
	ResNet	52.2±1.0	50.9±1.4	49.6±0.9	52.2±0.4

Table 5.3: Cross-architecture testing performance (%) on CIFAR10. The 50 img/cls synthetic set is learned on one architecture (C), and then tested on another architecture (T).

deviation (std). When learning with batch normalization, they first pre-set the BN mean and std using many real training data and then freeze them for synthetic data. Thus, the inaccurate mean and std will make optimization difficult (Ioffe and Szegedy, 2015). In contrast, we estimate running mean and std by inputting augmented synthetic data from all classes. Hence, our method benefits from the true mean and std of synthetic data. Table 5.2 show that using ConvNet with BN can further improve our performance. Specifically, our method with BN achieves 65.2%, 48.0% and 28.2% testing accuracies when learning 50 images/class synthetic sets on CIFAR10, CIFAR100 and TinyImageNet respectively, which means 2.2%, 4.4% and 4.1% improvements over our method with the default instance normalization, and also outperforms DSA with BN by 5.3% and 3.4% on CIFAR10 and CIFAR100 respectively.

Training Cost Comparison. Our method is significantly more efficient than bi-level optimization based methods. We compare the training time of ours and DSA in the setting of learning 50 images/class synthetic data on CIFAR10. Figure 5.4 shows that our method needs less than 20 minutes to reach the performance of DSA trained for 15 hours, which means less than 2.2% training cost. Note that we run the two methods in the same computation environment with one GTX 1080 GPU.

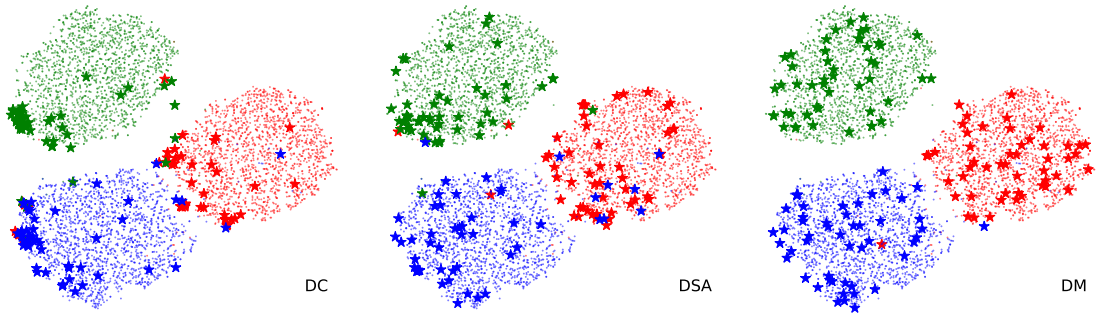


Figure 5.3: Distributions of synthetic images learned by DC, DSA and DM\@. The red, green and blue points are the real images of first three classes in CIFAR10. The stars are corresponding learned synthetic images.

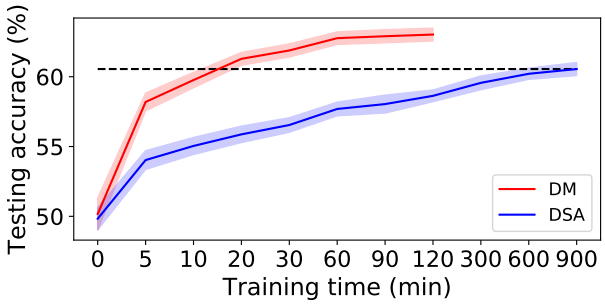


Figure 5.4: Training time comparison to DSA when learning 50 img/cls synthetic sets on CIFAR10.

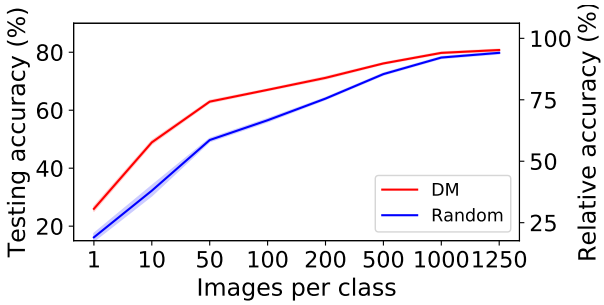


Figure 5.5: Learning larger synthetic sets on CIFAR10.

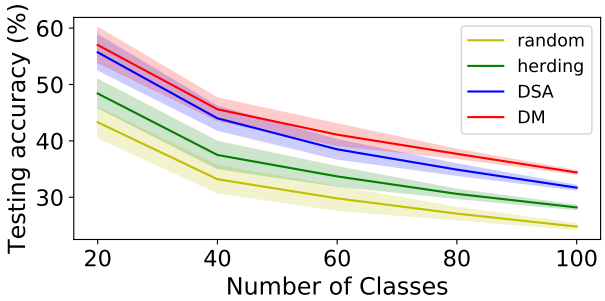


Figure 5.6: 5-step class-incremental learning on CIFAR100.

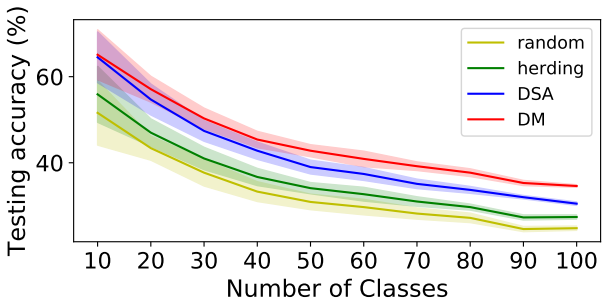


Figure 5.7: 10-step class-incremental learning on CIFAR100.

Learning Larger Synthetic Sets We show that our method can also be used to learn larger synthetic sets, while the bi-level optimization based methods typically requires more training time and elaborate hyper-parameter tuning for larger settings. Figure 5.5 compares our method to random selection baseline in CIFAR10 in terms of absolute and relative performance with respect to whole dataset training performance. Clearly our method outperforms the random baseline at all operating points which means that our synthetic set is more data-efficient. The advantage of our method is remarkable in challenging settings, *i.e.* settings with small data budgets. Our method obtains $67.0 \pm 0.3\%$, $71.2 \pm 0.4\%$, $76.1 \pm 0.3\%$, $79.8 \pm 0.3\%$ and $80.8 \pm 0.3\%$ testing accuracies when learning 100, 200, 500, 1000 and 1250 images/class synthetic sets on CIFAR10 dataset respectively, which means we can recover 79%, 84%, 90%, 94% and 95% relative performance using only 2%, 4%, 10%, 20% and 25% training data compared to whole dataset training. We see that the performance gap between the two methods narrows when we learn larger synthetic set. This is somewhat expected, as randomly selecting more samples will approach the whole dataset training which can be considered as the upper-bound. As we initialize synthetic images from random real images, the initial distribution discrepancy becomes tiny when the synthetic set is large.

5.3.3 Cross-architecture Generalization

Zhao et al. (2021); Zhao and Bilen (2021) verified the cross-architecture generalization ability of synthetic data in an easy setting – learning 1 image/class for MNIST dataset. In this paper, we implement a more challenging cross-architecture experiment – learning 50 images/class for CIFAR10 dataset. In Table 5.3, the synthetic data are learned with one architecture (denoted as C) and then be evaluated on another architecture (denoted as T) by training a model from scratch and testing on real testing data. We test several sophisticated neural architectures namely ConvNet,

AlexNet (Krizhevsky et al., 2012), VGG-11 (Simonyan and Zisserman, 2014) and ResNet-18 (He et al., 2016). Batch Normalization is used in all architectures.

Table 5.3 shows that learning and evaluating synthetic set on ConvNet achieves the best performance 65.2%. Comparing with DSA, the synthetic data learned by our method with ConvNet have better generalization performance than that learned by DSA with the ConvNet. Specifically, our method outperforms DSA by 8.0%, 8.9% and 9.7% when testing with AlexNet, VGG and ResNet respectively. These results indicate that the synthetic images learned with distribution matching have better generalization performance on unseen architectures than those learned with gradient matching. The learning of synthetic set can be worse with more sophisticated architecture such as ResNet. It is reasonable that the synthetic data fitted on sophisticated architecture will contain some bias that doesn't exist in other architectures, therefore cause worse cross-architecture generalization performance. We also find that the evaluation of the same synthetic set on more sophisticated architectures will be worse. The reason may be that sophisticated architectures are under-fitted using small synthetic set.

5.3.4 Ablation Study on Network Distribution

Here we study the effect of using different network distributions while learning 1/10/50 image(s)/class synthetic sets on CIFAR10 with ConvNet architecture. Besides sampling randomly-initialized network parameters, we also construct a set of networks that are pre-trained on the original training set. In particular, we train 1,000 ConvNets with different random initializations on the whole original training set and also store their intermediate states. We roughly divide these networks into nine groups according to their validation accuracies, sample networks from each group, learn the synthetic data on them and use learned synthetic data to train randomly initialized neural networks. Interestingly we see in Table 5.4 that our method works well with all nine network distributions and the performance variance is small. The visualization and analysis about synthetic images learned with different network distributions are provided in Appendix B.3.

5.3.5 Continual Learning

We also use our method to store more efficient training samples in the memory for relieving the catastrophic forgetting problem in continual (incremental) learning (Rebuffi et al., 2017). We set up the baseline based on GDumb (Prabhu et al., 2020)

	Random	10-20	20-30	30-40	40-50	50-60	60-70	≥70	All
1	26.0	26.2	25.9	26.1	26.7	26.8	27.3	26.5	26.4
10	48.9	48.7	48.1	50.7	51.1	49.9	48.6	48.2	50.7
50	63.0	62.7	62.1	62.8	63.0	61.9	60.6	60.0	62.5

Table 5.4: The performance of synthetic data learned on CIFAR10 with different network distributions. All standard deviations in this table are < 1. These networks are trained on the whole training set and grouped based on the validation accuracy (%).

which stores training samples in memory greedily and keeps class-balance. The model is trained from scratch on the latest memory only. Hence, the continual learning performance completely depends on the quality of the memory construction. We compare our memory construction method *i.e.* training set condensation to the *random* selection that is used in (Prabhu et al., 2020), *herding* (Chen et al., 2010; Rebuffi et al., 2017; Castro et al., 2018; Belouadah and Popescu, 2020) and *DSA* (Zhao and Bilen, 2021). We implement class-incremental learning on CIFAR100 dataset with an increasing memory budget of 20 images/class. We implement 5 and 10 step learning, in which we randomly and evenly split the 100 classes into 5 and 10 learning steps *i.e.* 20 and 10 classes per step respectively. The default ConvNet is used in this experiment.

As depicted in Figure 5.6 and Figure 5.7, we find that our method GDumb + DM outperforms others in both two settings, which means that our method can produce the best condensed set as the memory. The final performances of ours, DSA, herding and random are 34.4%, 31.7%, 28.2% and 24.8% in 5-step learning and 34.6%, 30.5%, 27.4% and 24.8% in 10-step learning. We find that ours and random selection performances are not influenced by how the classes are split namely how many new training classes and images occur in each learning step, because both two methods learn/generate the sets independently for each class. However, DSA and herding methods perform worse when the training classes are densely split into more learning steps. The reason is that DSA and herding needs to learn/generate sets based on the model(s) trained on the current training data, which is influenced by the data split. More details can be found in Appendix B.1.

	Random	DSA	DM	Early-stopping	Whole Dataset
Performance (%)	84.0	82.6	82.8	**84.3**	85.9
Correlation	-0.04	0.68	**0.76**	0.11	1.00
Time cost (min)	142.6	142.6	142.6	142.6	3580.2
Storage (imgs)	**500**	**500**	**500**	5×10^4	5×10^4

Table 5.5: We implement neural architecture search on CIFAR10 dataset with the search space of 720 ConvNets.

5.3.6 Neural Architecture Search

The synthetic sets can also be used as a proxy set to accelerate model evaluation in Neural Architecture Search (NAS) (Elsken et al., 2019). Following Zhao et al. (2021), *i.e*. Chapter 3, we implement NAS on CIFAR10 with the search space of 720 ConvNets varying in network depth, width, activation, normalization and pooling layers. Please refer to Section 3.3.2 for more details. We train all architectures on the learned 50 images/class synthetic set, *i.e*. 1% size of the whole dataset, from scratch and then rank them based on the accuracy on a small validation set. We compare to *random*, *DSA* and *early-stopping* methods. The same size of real images are selected as the proxy set in *random*. *DSA* means that we use the synthetic set learned by DSA in the same setting. In *early-stopping*, we use the whole training set to train the model but with the same training iterations like training on the proxy datasets. Therefore, all these methods have the same training time. We train models on the proxy sets for 200 epochs and whole dataset for 100 epochs. The best model is selected based on validation accuracies obtained by different methods. The Spearman's rank correlation between performances of proxy-set and whole-dataset training is computed for the top 5% architectures selected by the proxy-set.

The NAS results are provided in Table 5.5. Although the architecture selected by early-stopping achieves the best performance (84.3%), its performance rank correlation (0.11) is remarkably lower than DSA (0.68) and DM (0.76). In addition, early-stopping needs to use the whole training set, while other proxy-set methods need only 500 training samples. The performance rank correlation of Random (-0.04) is too low to provide a reliable ranking for the architectures. Our method (DM) achieves the highest performance rank correlation (0.76), which means that our method can produce reliable ranking for those candidate architectures while using only around $\frac{1}{25}$ training time of whole dataset training. Although our method needs 72 min to obtain the condensed set, it is negligible compared to whole-dataset training

(3580.2 min). More implementation details and analysis can be found in Appendix B.1.

5.4 Conclusion and Limitation

In this paper, we propose an efficient dataset condensation method based on distribution matching. To our knowledge, it is the first solution that has neither bi-level optimization nor second-order derivatives. Thus, the synthetic data of different classes can be learned independently and in parallel. Thanks to its efficiency, we can apply our method to more challenging datasets – TinyImageNet and ImageNet-1K, and learn larger synthetic sets – 1250 images/class on CIFAR10. Our method is 45 times faster than the state-of-the-art for learning 50 images/class synthetic set on CIFAR10. We also empirically demonstrate that our method can produce a more informative memory for continual learning and better proxy set for speeding up model evaluation in NAS\@.

Limitations. Though remarkable progress has been seen in this area since the pioneering work of Wang et al. (2018), dataset condensation is still in its early stage. The state of the art of dataset condensation is still much lower than the upperbound, *i.e*. whole-dataset training, especially on large-scale datasets, such as ImageNet-1K\@. In the future, we will further improve dataset condensation performance and extend it to more complex vision tasks.

Chapter 6

Conclusion and Future Work

Here, we summarize our three works, discuss their limitations, and provide several promising future work directions.

In Chapter 3, we propose a principled, effective and scalable solution to dataset condensation problem. We expect a model trained on synthetic dataset should follow similar solution path to that trained on the original dataset. Thus, we learn synthetic data by minimizing the matching loss of two gradients of network parameters with respect to the real and synthetic training data. Extensive experiments and an ablation study have been implemented to verify the effectiveness of the proposed method. We also show promising results in applications of continual learning and neural architecture search.

In Chapter 4, we investigate the reason why data augmentation does not improve the efficiency of synthetic training data for training deep neural networks. To solve this problem, we design the differentiable Siamese augmentation for learning more informative synthetic images that can be use to train deep neural networks more efficiently with data augmentation. Remarkable performance improvements have been achieved by applying the proposed augmentation to the state-of-the-art method.

Although training deep models on the small set of condensed images can be extremely fast, the dataset condensation procedure is expensive due to the bilevel optimization. In Chapter 5, we propose a simple yet effective method that synthesizes condensed images by matching feature distributions of the synthetic and original training images in many sampled embedding spaces. To our knowledge, this is the first dataset condensation method that has neither bilevel optimization nor second-order derivative. Thanks to its simplicity, we can implement dataset condensation on larger datasets and synthesize larger synthetic sets.

6.1 Impact

Our works have achieved significant impact on the community. We for the first time make dataset condensation an easy-to-use tool for downstream tasks such as continual learning (Zhao et al., 2021; Zhao and Bilen, 2023), neural architecture search (Zhao et al., 2021; Zhao and Bilen, 2021), federated learning (Goetz and Tewari, 2020; Hu et al., 2022a; Song et al., 2022) and privacy protection (Dong et al., 2022).

Many followup works have been accepted by top conferences and journals. For example, Cazenavette et al. (2022a) propose the trajectory matching and achieve the new state of the art by combining our parameter matching method and meta-learning method (Wang et al., 2018). Based on our gradient matching, Wang et al. (2022) design the feature alignment and discrimination loss and dynamic bilevel optimization. Lee et al. (2022) introduce the contrastive signal and improve the robustness of gradient matching. Kim et al. (2022) improves data efficiency of gradient matching method by multi-formation that combines multiple low-resolution synthetic images into one. Jin et al. (2022b,a) adapt our method for graph data and achieves good performance. Dong et al. (2022) introduce our distribution matching (Zhao and Bilen, 2023) into privacy protection and prove that our method can preserve data privacy and training efficiency simultaneously. Wiewel and Yang (2021) extend our method by learning shared components and apply it in continual learning. Our designed gradient matching loss is also verified effective in (Hu et al., 2022b). Schirrmeister et al. (2022) find that our synthetic images can be further simplified with lower bits and achieve almost the same performance for training deep neural networks. Zhao and Bilen (2022); Lee et al. (2022) integrate our distribution matching method into generative models and further reduce the storage of synthetic images.

Though we only implement image dataset condensation in this thesis, the proposed methods can be easily adapted for condensing other kinds of datasets. For example, Sucholutsky and Schonlau (2019) embed the discrete text data (words) into continuous vectorized representations using off-the-shelf embedding functions (Devlin et al., 2018), and then pad/truncate each sentence to the same length. After learning a small number of condensed embeddings, the closest discrete words can be found by searching their nearest neighbors in the embedding space. Similarly, Jin et al. (2022b) embed the nodes in a graph dataset into features, and then learn the condensed node features by adapting our gradient matching method.

6.2 Limitations

Though remarkable progress (Wang et al., 2018; Zhao et al., 2021; Zhao and Bilen, 2021, 2023; Nguyen et al., 2021a,b; Wang et al., 2022; Cazenavette et al., 2022a) has been achieved in this research area since the pioneering work of Wang et al. (2018), dataset condensation research is still at an early stage. The current research still has the following limitations:

1. Low performance and limited utility. Although we can achieve 99.2% testing accuracy by training models on 50 images/class condensed MNIST images, the state-of-the-art performance on CIFAR10 is around 70%, which is far from the upper-bound about 95% obtained by training models on the whole dataset (5K images/class). The performance gap is larger if we condense more challenging dataset, *e.g*. TinyImageNet and ImageNet-1K\@. The low performance hinders its application in real-world tasks. In addition, current dataset condensation research focuses on classification tasks, and more efforts are needed to extend it to complex tasks, *e.g*. object detection and semantic segmentation.

2. Limited generalization ability. The model architectures used in the condensation and evaluation can be different. In this thesis, we show the performance varies when learning and evaluating synthetic data on different architectures. Specifically, Table 5.3 shows that learning synthetic data with a more sophisticated architecture would produce worse performance for the same evaluating architecture. Vice versa, with the synthetic data learned on the same architecture, evaluating it on more sophisticated architecture would produce worse performance.

3. Poor scalability. Most of the existing condensation methods are based on bilevel optimization, which have poor scalability. The training time and memory cost will increase along with the size of image resolution, original dataset, synthetic dataset and the model. Thus, it is challenging to learn large synthetic set from large dataset with sophisticated architectures.

6.3 Future Work

6.3.1 Large-scale and Complex Tasks

Dataset condensation for large-scale datasets, *e.g*. ImageNet (Deng et al., 2009), is still challenging. ImageNet is more challenging due to the higher resolution images with larger variations in pose, appearance of objects, background and more categories. These challenges hinder the effective and efficient learning of condensed images. Thus, future work is needed to deal with the large variations of images in many categories. In addition, how to improve the dataset condensation speed is also important. The new condensation methods should be able to run in parallel, so that the huge computation can be distributed on multiple machines.

There exist a few early works about synthesizing training data for complex tasks such as semantic segmentation (Zhang et al., 2021; Yang et al., 2022). The two works focus on learning an annotator that can label GANs generated images, thus the annotator together with GANs can produce numerous training data pairs for segmentation task. These generated training data are not guaranteed to be more informative than real data. How to improve the informativeness of synthesized segmentation training samples needs further research.

6.3.2 More Applications

Existing works have applied dataset condensation to image classification (Wang et al., 2018; Zhao et al., 2021), text classification (Sucholutsky and Schonlau, 2019; Li and Li, 2021), privacy protection (Dong et al., 2022; Tsilivis et al., 2022), federated learning (Goetz and Tewari, 2020; Hu et al., 2022a; Song et al., 2022), continual learning (Zhao et al., 2021; Zhao and Bilen, 2021, 2023), graph model (Jin et al., 2022b,a), neural architecture search (Such et al., 2020; Zhao et al., 2021; Zhao and Bilen, 2021), medical application (Li et al., 2020, 2022) and art (Cazenavette et al., 2022b). The preliminary results have shown that dataset condensation techniques are promising for advancing these applications by providing informative synthetic training samples. In the future, more efforts are required to design better and customized dataset condensation methods for these applications. For example, in continual learning, the joint use of real and synthetic data should be investigated. Applying dataset condensation in other tasks such as video (Tavakolian et al., 2019b,a; Qiu et al., 2021) and speech classification (Baevski et al., 2021; Nassif et al.,

2019) is also worth exploring.

6.3.3 Dataset Condensation with Generative Models

The well-studied generative models, *e.g*. GAN and auto-encoders, can synthesize high-resolution and real-looking images. A nature idea is to integrate dataset condensation with these classic generative models. Ideally, the combination can benefit from the powerful generation ability of generative models and achieve much better performance than optimizing image pixels directly. However, the previous attempt (Such et al., 2020) does not show much advantage. Zhao and Bilen (2022) integrate distribution matching and a pre-trained generator. Their results show that, under the same storage budget, storing informative latent vectors and the generator can achieve better performance compared to storing the learned synthetic images directly. Despite the promising results, the informativeness of individual images learned with generator is still lower than that learned by optimizing image pixels directly. The underlying reason and how to improve the data efficiency demands more investigation.

6.3.4 Joint Use of Real and Synthetic Data

Another important research direction is the joint use of real and synthetic data. To our knowledge, existing works on dataset condensation use only synthetic data to train models. However, the joint use of real and synthetic data is important in practical learning scenarios. For example, in continual learning, the learned synthetic images of old tasks will be used together with real images of new tasks. In this scenario, the synthesis based condensation of old-task images should be regularized by the new-task real images. In such task, the imbalance of number and informativeness between synthetic and real images is worth investigating.

Due to the limitation of computation resources and the difficulty of optimization, how to jointly synthesize and select images in one algorithm for dataset condensation is an interesting but challenging problem. Previous work (Zhao and Bilen, 2023) has shown that learning a large synthetic set is challenging and the performance advantage over sample selection narrows quickly. The joint learning algorithm is expected to significantly reduce the training cost while preserving performance by balancing sample synthesis and selection.

Appendix A

Appendix: Dataset Condensation with Gradient Matching

A.1 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 η_{θ} and number of optimization steps ς_{θ} for the model weights. In all experiments, we set $K = 1000$, $\eta_{\mathcal{S}} = 0.1$, $\eta_{\theta} = 0.01$, $\varsigma_{\mathcal{S}} = 1$ 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 3.2), as MLP requires a slightly different treatment. Note that while K is the maximum number of outer-loop steps, the optimization can early-stop automatically if it converges before K steps. For the remaining hyperparameters, we use different sets for 1, 10 and 50 image(s)/class learning. We set $T = 1$, $\varsigma_{\theta} = 1$ for 1 image/class, $T = 10$, $\varsigma_{\theta} = 50$ for 10 images/class, $T = 50$, $\varsigma_{\theta} = 10$ for 50 images/class learning. Note that when $T = 1$, it is not required to update the model parameters (Step 9 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 $\varsigma_{\theta} = 500/T$, *e.g.* $T = 20$, $\varsigma_{\theta} = 25$ for 20 images/class learning. The ablation study on hyper-parameters are given in Appendix A.2 which shows that our method is not sensitive to varying hyper-parameters.

We do separate-class mini-batch sampling for Step 6 in Algorithm 1. Specifically, we sample a mini-batch pair $B_c^{\mathcal{T}}$ and $B_c^{\mathcal{S}}$ that contain real and synthetic images from the same class c at each inner iteration. Then, the matching loss for each class is computed with the sampled mini-batch pair and used to update corresponding synthetic images $\mathcal{S}_c$ by back-propogation (Step 7 and 8). This is repeated separately (or parallelly given enough computational resources) for every class. Training as such is not slower than using mixed-class batches. Although our method still works well when we randomly sample the real and synthetic mini-batches with mixed labels, we found that separate-class strategy is faster to train as matching gradients with respect to data from single class is easier compared to those of multiple classes. In experiments, we randomly sample 256 real images of a class as a mini-batch to calculate the mean gradient and match it with the mean gradient that is averaged over all synthetic samples with the same class label. The performance is not sensitive to the size of real-image mini-batch if it is greater than 64.

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 overall comparable performances in different settings and datasets. Then, we used Gaussian noise for initialization in experiments.

	USPS	MNIST	FashionMNIST	SVHN	CIFAR10	CIFAR100
Train	7,291	60,000	60,000	73,257	50,000	50,000
Test	2,007	10,000	10,000	26,032	10,000	10,000

Table T1: Train/test statistics for USPS, MNIST, FashionMNIST, SVHN, CIFAR10 and CIFAR100 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

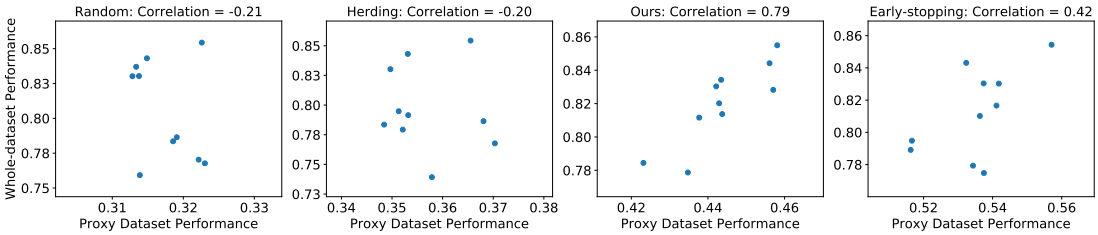


Figure F1: The performance correlation between the training on proxy dataset and whole-dataset. For each proxy dataset, the best 10 models are selected based on validation set performance. In the figure, each point represents an architecture.

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 with respect to 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 3.4.

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

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.

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 F1. 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\@. Note that the NAS performance is sensitive to hyper-parameters of model training, *e.g.* learning rate, training epoch and data augmentation. Our experiments demonstrate that dataset condensation techniques can be a promising solution to speeding up NAS\@.

A.2 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. We also study the selection of hyper-parameters and the gradient distance metric. 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

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.

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 with respect 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 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.

Ablation Study on Normalization Functions. Next we study the performance of four normalization options – No normalization, Batch (Ioffe and Szegedy, 2015), Layer (Ba et al., 2016), Instance (Ulyanov et al., 2016) and Group Normalization (Wu and 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.

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.

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 and 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, when learning 10 images/class condensed sets, is depicted in Figure F2. 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.

Ablation Study on Gradient Distance Metric. To prove the effectiveness and robustness of the proposed distance metric for gradients (or weights), we compare to the traditional ones (Lopez-Paz et al., 2017; Aljundi et al., 2019; Zhu et al., 2019)

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.

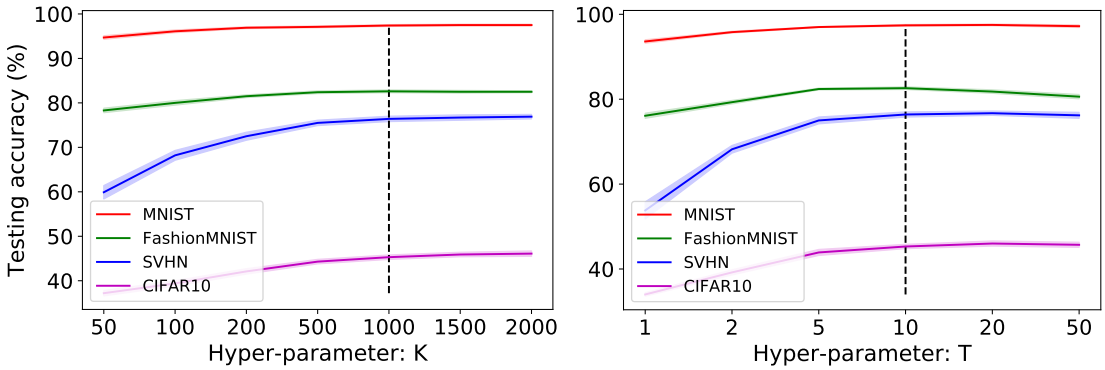


Figure F2: Ablation study on the hyper-parameters K and T when learning 10 images/class condensed sets.

which vectorize and concatenate the whole gradient, $\mathbf{G}^{\mathcal{T}}, \mathbf{G}^{\mathcal{S}} \in \mathbb{R}^D$, and compute the squared Euclidean distance $\|\mathbf{G}^{\mathcal{T}} - \mathbf{G}^{\mathcal{S}}\|^2$ and the Cosine distance $1 - \cos(\mathbf{G}^{\mathcal{T}}, \mathbf{G}^{\mathcal{S}})$, where D is the number of all network parameters. We do 1 image/class learning experiment on MNIST with different architectures. For simplicity, the synthetic images are learned and tested on the same architecture in this experiment. Table T7 shows that the proposed gradient distance metric remarkably outperforms others on complex architectures (*e.g.* LeNet, AlexNet, VGG and ResNet) and achieves the best performances in most settings, which means it is more effective and robust than the traditional ones. Note that we set $\eta_{\mathcal{S}} = 0.1$ for MLP-Euclidean and MLP-Cosine because it works better than $\eta_{\mathcal{S}} = 0.01$.

Further Qualitative Analysis We first depict the condensed images that are learned on MNIST, FashionMNIST, SVHN and CIFAR10 datasets in one experiment using the default ConvNet in 10 images/class setting in Figure F3. It is interesting that the 10 images/class results in Figure F3 are diverse which cover the main variations, while the condensed images for 1 image/class setting (see Figure 3.2) look like the "prototype" of each class. For example, in Figure F3 (a), the ten images of "four" indicate ten different styles. The ten "bag" images in Figure F3 (b) are significantly different from

each other, similarly "wallet" (1st row), "shopping bag" (3rd row), "handbag" (8th row) and "schoolbag" (10th row). Figure F3 (c) also shows the diverse house numbers with different shapes, colors and shadows. Besides, different poses of a "horse" have been learned in Figure F3 (d).

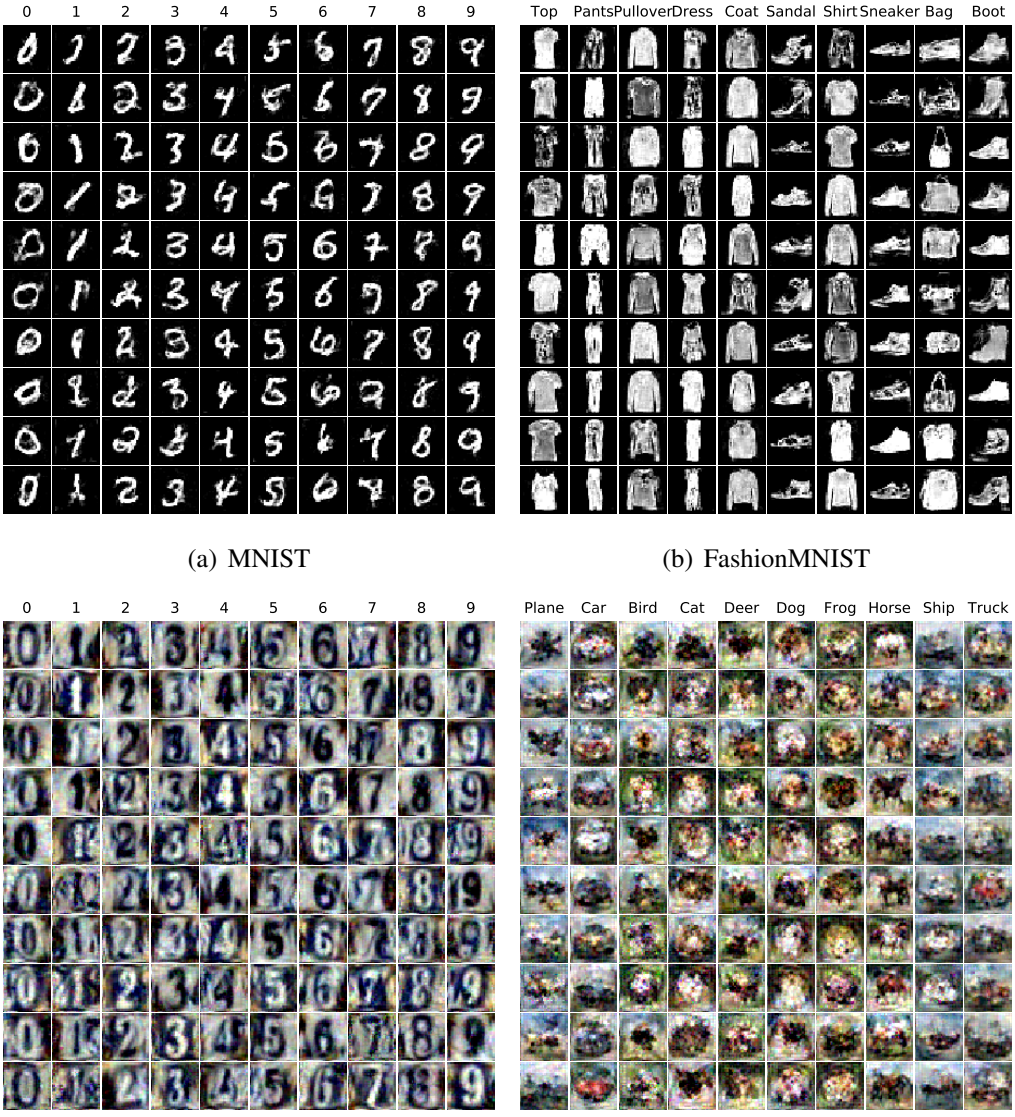
A.3 Comparison to More Baselines

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 T8 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.

Generative Model. We also compare to the popular generative model, *namely*, Conditional Generative Adversarial Networks (cGAN) (Mirza and 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 T8 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.

	MLP	ConvNet	LeNet	AlexNet	VGG	ResNet
Euclidean	69.3±0.9	**92.7±0.3**	65.0±5.1	66.2±5.6	57.1±7.0	68.0±5.2
Cosine	45.2±3.6	69.2±2.7	61.1±8.2	58.3±4.1	55.0±5.0	68.8±7.8
Ours	**70.5±1.2**	91.7±0.5	**85.0±1.7**	**82.7±2.9**	**81.7±2.6**	**89.4±0.9**

Table T7: Ablation study on different gradient distance metrics. Obviously, the proposed distance metric is more effective and robust. Euclidean: squared Euclidean distance, Cosine: Cosine distance.



(a) MNIST (b) FashionMNIST

(c) SVHN (d) CIFAR10

Figure F3: The synthetic images for MNIST, FashionMNIST, SVHN and CIFAR10 produced by our method with ConvNet under 10 images/class setting.

	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 T8: 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.

Analysis of Coreset Performances. We find that K-Center (Sener and 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.

Performance on CIFAR100. We provide the performance comparison on CIFAR100 dataset which includes 10 times as many classes as other benchmarks. More classes while fewer images per class makes CIFAR100 significantly more challenging than other datasets. We use the same set of hyper-parameters for CIFAR100 as other datasets. Table T9 depicts the performances of coreset selection methods, Label Distillation (LD) (Bohdal et al., 2020) and ours. Our method achieves 12.8% and 25.2% testing accuracies on CIFAR100 when learning 1 and 10 images per class, which are the best compared with others.

	Img/Cls	Ratio %	Core-set Selection				LD†	Ours	Whole Dataset
			Random	Herding	K-Center	Forgetting			
CIFAR100	1	0.2	4.2±0.3	8.4±0.3	8.3±0.3	3.5±0.3	11.5±0.4	**12.8±0.3**	56.2±0.3
	10	2	14.6±0.5	17.3±0.3	7.1±0.3	9.8±0.2	-	**25.2±0.3**	

Table T9: The performance comparison on CIFAR100. This table shows the testing accuracies (%) of different methods. ConvNet is used for training and testing except that LD† uses AlexNet. Img/Cls: image(s) per class, Ratio (%): the ratio of condensed images to whole training set.

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 T10: 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.

A.4 Further Comparison to Dataset Distillation

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 T10. We see 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.

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 T11: Time and memory use for training DD and our method in 10 images/class setting.

Qualitative Comparison. We also provide a qualitative comparison to DD in terms of image quality in Figure F4. 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 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 T11 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 using smaller hyper-parameters, *e.g.* K, T and the batch size of sampled real images, with a slight performance decline (refer to Figure F2).

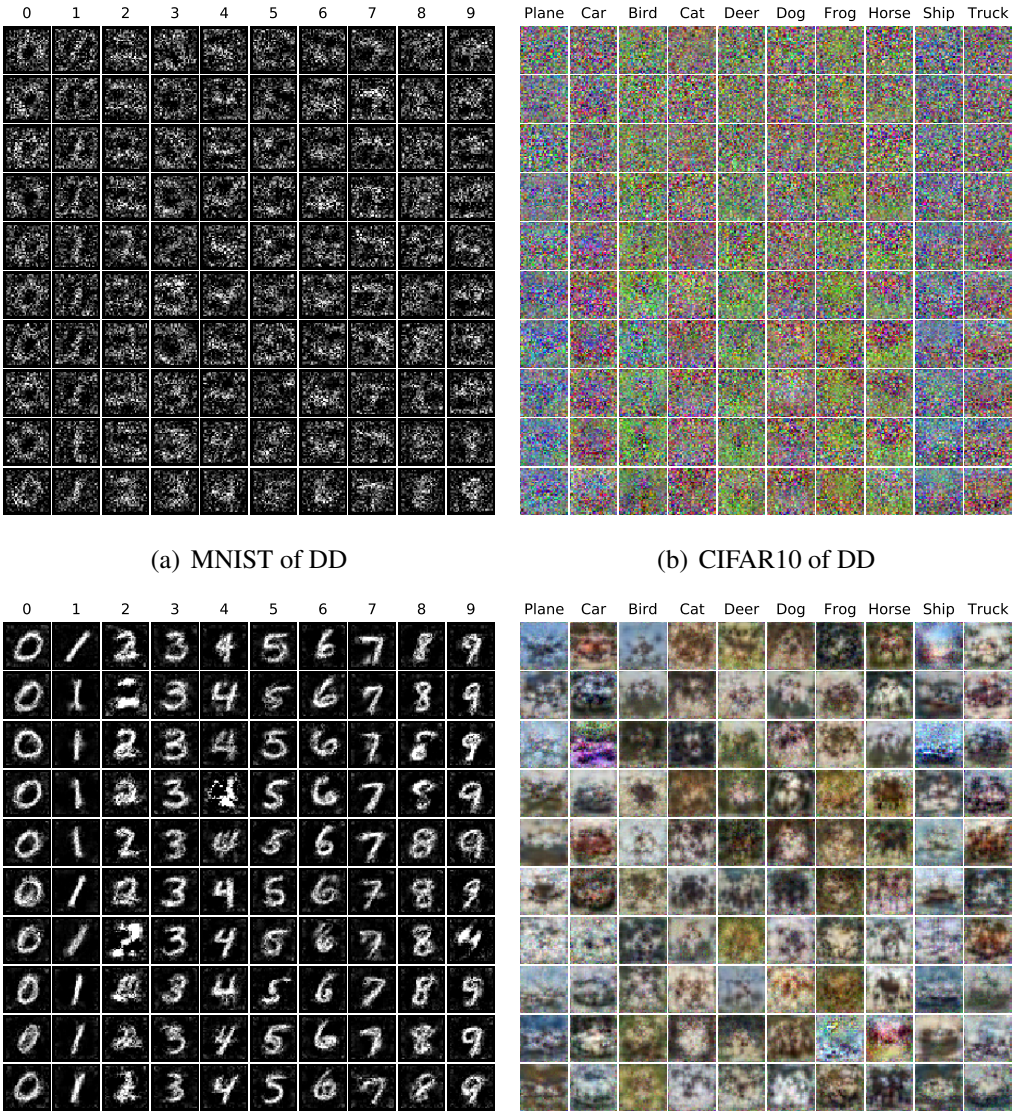
A.5 Extended Related Work

Variations of Dataset Distillation. There exists recent work that extends Dataset Distillation (Wang et al., 2018). For example, Sucholutsky and Schonlau (2019); Bohdal et al. (2020) aim to improve DD by learning soft labels with/without synthetic images. Such et al. (2020) 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 thoroughly 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.

Data Privacy & Federated Learning. Synthetic dataset is also a promising solution to protecting data privacy and enabling safe federated learning. There exists some work that uses synthetic dataset to protect the privacy of medical dataset (Li et al., 2020) and reduce the communication rounds in federated learning (Zhou et al., 2020). Although transmitting model weights or gradients (Zhu et al., 2019; Zhao et al., 2020a) may increase the transmission security, the huge parameters of modern deep neural networks are prohibitive to transmit frequently. In contrast, transmitting small-scale synthetic dataset between clients and server is low-cost (Goetz and Tewari, 2020).



(a) MNIST of DD

(b) CIFAR10 of DD

(c) MNIST of ours

(d) CIFAR10 of ours

Figure F4: 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.

Appendix B

Appendix: Dataset Condensation with Distribution Matching

B.1 Implementation details

Dataset Condensation. As Zhao and Bilen (2021) didn't report 50 images/class learning performance on CIFAR100, we obtain the result in Table 1 by running their released code and coarsely searching the hyper-parameters (outer and inner loop steps). Then, we set both outer and inner loop to be 10 steps. The rest hyper-parameters are the default ones in their released code. To obtain the DSA results with batch normalization in Table 2 and Table 3, we also run DSA code and set batch normalization in ConvNet.

We follow the modification of ResNet in (Zhao et al., 2021). They replace the $stride = 2$ convolution layer with $stride = 1$ convolution layer followed by an average pooling layer in the ResNet architecture that is used to learn the synthetic data. This modification enables smooth error back-propagation to the input images. We directly use their released ResNet architecture.

Continual Learning. Prabhu et al. (2020) use cutmix (Yun et al., 2019) augmentation strategy for training models. Different from them, we follow (Zhao and Bilen, 2021) and use the default DSA augmentation strategy in order to be consistent with other experiments in this paper. We run DSA training algorithm on the new training classes and images only in every learning step. It is not a easy work to take old model and memory into DSA training and achieve better performance. The synthetic data learned with old model can also be biased to it, and thus perform worse. Similarly, we train the embedding function (ConvNet) for herding method on

the new training classes and images only.

Neural Architecture Search. We randomly select 10% training samples in CIFAR10 dataset as the validation set. The rest are the training set. The batch size is 250, then one training epoch on the small (50 images/class) proxy sets includes 2 batches. The DSA augmentation strategy is applied to all proxy-set methods and early-stopping. We train each model 5 times and report the mean accuracies. We do NAS experiment on one Tesla v100 GPU.

We visualize the performance rank correlation between proxy-set and whole-dataset training in Figure F1. The top 5% architectures are selected based on the validation accuracies of models trained on each proxy-set. Each point represents a selected architecture. The horizontal and vertical axes are the testing accuracies of models trained on the proxy-set and the whole dataset respectively. The figure shows that our method can produce better proxy set to obtain more reliable performance ranking of candidate architectures.

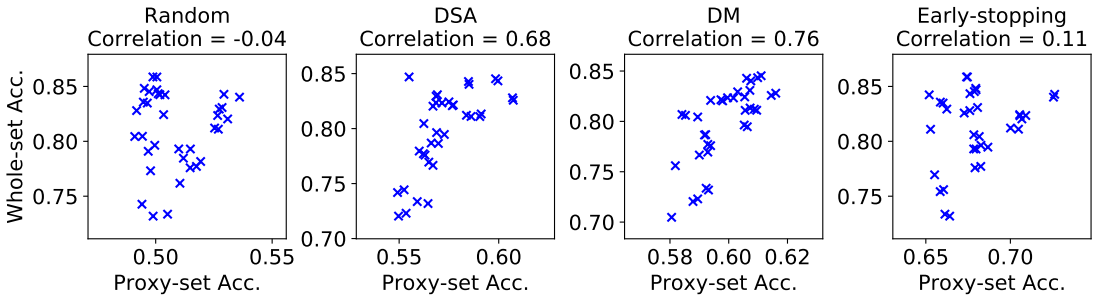


Figure F1: Performance rank correlation between proxy-set and whole-dataset training.

B.2 Comparison to More Baselines and Related Works

Comparison to Generative Models. In this subsection, we compare the data-efficiency of samples generated by our dataset condensation method to those generated by traditional generative models, namely VAE and GAN\@. Specifically, we choose the state-of-the-art DC-VAE (Parmar et al., 2021) and BigGAN (Brock et al., 2019). The BigGAN model is trained with the differentiable augmentation (Zhao et al., 2020b). In addition, we also compare to a related generative model GMMN (Li et al., 2015) which aims to learn an image generator that can map a uniform distribution to real image distribution. Our method differs from GMMN in many ways significantly. First, GMMN aims to generate real-looking images, while

our goal is to condense a training set by synthesizing informative training samples that can be used to efficiently train deep networks through MMD\@. Second, our method learns pixels directly, while GMMN learns a generator network. Third, our method learns a few synthetic samples to approximate the distribution of large real training set in any embedding space with any augmentation, while GMMN learns to map a uniform distribution to real image distribution which is an easier task.

We train these generative models on CIFAR10 dataset. ConvNets are trained on these synthetic images and then evaluated on real testing images. The results in Table T1 verify that our method outperforms them by large margins, indicating that our synthetic images are more informative for training deep neural networks. The comparison to random baseline indicates that the images generated by traditional generative models are not more informative than randomly selected real images.

Img/Cls	Random	GMMN	VAE	BigGAN	MMD	*DM*
1	14.4±2.0	16.1±2.0	15.7±2.1	15.8±1.2	22.7±0.6	**26.0±0.8**
10	26.0±1.2	32.2±1.3	29.8±1.0	31.0±1.4	34.9±0.3	**48.9±0.6**
50	43.4±1.0	45.3±1.0	44.0±0.8	46.2±0.9	50.9±0.3	**63.0±0.4**

Table T1: Comparison to traditional generative models and MMD baseline. Random means randomly selected real images. The experiments are implemented with ConvNets on CIFAR10 dataset.

Comparison to MMD Baseline. Another baseline is to learn synthetic images by distribution matching with vanilla MMD in the pixel space. This baseline can also been considered as the ablation study of the embedding function and differentiable augmentation in our method. We try this baseline with linear, polynomial, RBF and Laplacian kernels and with various kernel hyper-parameters. We find that only MMD with linear kernel can achieve better synthetic images, *i.e.* better than randomly selected real images. The performance of MMD with linear kernel in the pixel space is presented in Table T1, which outperforms all generative models while is inferior to our method. This result also verifies that the distribution matching mechanism enables learning more informative synthetic samples.

Comparison to GTN and KIP Methods. We notice the recent works Generative Teaching Networks (GTN) (Such et al., 2020) and Kernel Inducing Point (KIP) (Nguyen et al., 2021a,b) on dataset condensation. Such et al. (2020) propose to learn a generative network that outputs condensed training samples by minimizing the

meta-loss on real data. They report the performance of 4,096 synthetic images learned on MNIST which is comparable to our 50 images/class synthetic set (*i.e*. 500 images in total) performance.

Nguyen et al. (2021a,b) propose to replace the neural network optimization in the bi-level optimization (Wang et al., 2018) with kernel ridge regression which has a closed-form solution. Zero Component Analysis (ZCA) (Krizhevsky et al., 2009) is applied for pre-processing images. Although Nguyen et al. (2021b) report the results on 1024-width neural networks while we train and test 128-width neural networks, our results still outperform theirs in some settings, for example $98.6 \pm 0.1\%$ v.s. $98.3 \pm 0.1\%$ when learning 50 images/class on MNIST and $29.7 \pm 0.3\%$ v.s. $28.3 \pm 0.1\%$ when learning 10 images/class on CIFAR100. Note that they achieve those results by leveraging distributed computation environment and training for thousands of GPU hours. In contrast, our method can learn synthetic sets with one GTX 1080 GPU in dozens of minutes, which is significantly more efficient.

B.3 Extended Visualization and Analysis

We visualize the 10 images/class synthetic sets learned on CIFAR10 dataset with different network parameter distributions in Figure F2. It is interesting that images learned with "poor" networks that have lower validation accuracies look blur. We can find obvious checkerboard patterns in them. In contrast, images learned with "good" networks that have higher validation accuracies look colorful. Some twisty patterns can be found in these images. Although synthetic images learned with different network parameter distributions look quite different, they have similar generalization performance. We think that these images are mainly different in terms of their background patterns but similar in semantics. It means that our method can produce synthetic images with similar network optimization effects while significantly different visual effects. Our method may have promising applications in protecting data privacy and federated learning (Lyu et al., 2020).

B.4 Connection to Gradient Matching

In this section, we show the connection between gradient matching (Zhao et al., 2021) and our method. Both (Zhao et al., 2021) and our training algorithm sample real and synthetic image batches from one class in each iteration, which is denoted as class

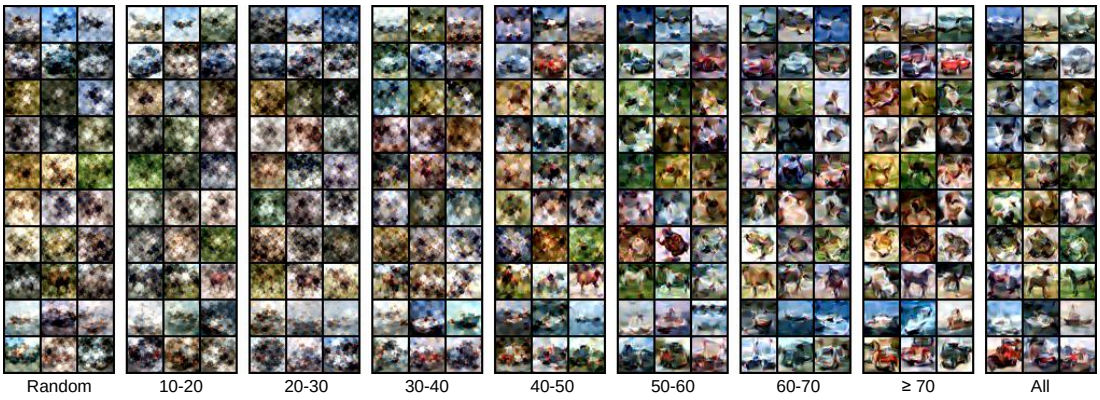


Figure F2: Synthetic images of CIFAR10 dataset learned with different network parameter distributions, *i.e.* networks with different validation accuracies (%). Each row represents a class.

y. We embed each training sample $(\boldsymbol{x}_i, y)$ and obtain the feature $\boldsymbol{e}_i$ using a neural network $\psi_{\boldsymbol{\vartheta}}$ followed a linear classifier $\mathbf{W} = [\boldsymbol{w}_0, ..., \boldsymbol{w}_{C-1}]$, where $\boldsymbol{w}_j$ is the weight vector connected to the j^{th} output neuron and C is the number of all classes. Note that the weight and its gradient vector are organized in the same way in (Zhao et al., 2021). We focus on the weight and gradient of the linear classification layer (*i.e.* the last layer) of a network in this paper. The classification loss J_i of each sample is denoted as

$$J_i = -\log \frac{\exp\left(\boldsymbol{w}_y^T \cdot \boldsymbol{e}_i\right)}{\Sigma_k \exp\left(\boldsymbol{w}_k^T \cdot \boldsymbol{e}_i\right)}. \tag{B.1}$$

Then, we compute the partial derivative with respect to each weight vector,

$$\boldsymbol{g}_{i,j} = \frac{\partial J_i}{\partial \boldsymbol{w}_j} = \begin{cases} -\boldsymbol{e}_i + \dfrac{\exp\left(\boldsymbol{w}_y^T \cdot \boldsymbol{e}_i\right)}{\Sigma_k \exp\left(\boldsymbol{w}_k^T \cdot \boldsymbol{e}_i\right)} \cdot \boldsymbol{e}_i, & j = y \\ \dfrac{\exp\left(\boldsymbol{w}_j^T \cdot \boldsymbol{e}_i\right)}{\Sigma_k \exp\left(\boldsymbol{w}_k^T \cdot \boldsymbol{e}_i\right)} \cdot \boldsymbol{e}_i, & j \neq y \end{cases} \tag{B.2}$$

This equation can be simplified using the predicted probability $p_{i,j} = \frac{\exp\left(\boldsymbol{w}_j^T \cdot \boldsymbol{e}_i\right)}{\Sigma_k \exp\left(\boldsymbol{w}_k^T \cdot \boldsymbol{e}_i\right)}$ that classifies sample $\boldsymbol{x}_i$ into category j:

$$\boldsymbol{g}_{i,j} = \begin{cases} (p_{i,y} - 1) \cdot \boldsymbol{e}_i, & j = y \\ (p_{i,j} - 0) \cdot \boldsymbol{e}_i, & j \neq y \end{cases} \tag{B.3}$$

Eq. B.3 shows that the last-layer gradient vector $\boldsymbol{g}_{i,j}$ is equivalent to a weighted feature vector $\boldsymbol{e}_i$ and vice versa. The weight is a function of classification probability. Generally speaking, the weight is large when the difference between predicted probability $p_{i,j}$ and ground-truth one-hot label (1 or 0) is large.

As the real and synthetic samples in each training iteration are from the same class y, we can obtain the mean gradient over a data batch by averaging the corresponding gradient components:

$$\frac{1}{N}\Sigma_i^N \boldsymbol{g}_{i,j} = \begin{cases} \frac{1}{N}\Sigma_i^N (p_{i,y} - 1) \cdot \boldsymbol{e}_i, & j = y \\ \frac{1}{N}\Sigma_i^N (p_{i,j} - 0) \cdot \boldsymbol{e}_i, & j \neq y \end{cases} \tag{B.4}$$

N is the batch size. Thus, last-layer mean gradient is equivalent to the weighted mean feature, and the mean gradient matching is equivalent to the matching of weighted mean feature.

Our method can learn synthetic images with randomly initialized networks. Given networks with random parameters, we assume that the predicted probability is uniform over all categories, *i.e.* $p_{i,j} = \frac{1}{C}$. Then, the mean gradient is

$$\frac{1}{N}\Sigma_i^N \boldsymbol{g}_{i,j} = \begin{cases} \frac{1-C}{C} \cdot \frac{1}{N}\Sigma_i^N \boldsymbol{e}_i, & j = y \\ \frac{1}{C} \cdot \frac{1}{N}\Sigma_i^N \boldsymbol{e}_i, & j \neq y \end{cases} \tag{B.5}$$

which is equivalent to the mean feature with a constant weight. Thus, with randomly initialized networks, the last-layer mean gradient matching is equivalent to mean feature matching multiplied by a constant.

Bibliography

Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B., and Vijayanarasimhan, S. (2016). Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*.

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

Agarwal, S., Arora, H., Anand, S., and Arora, C. (2020). Contextual diversity for active learning. In *European Conference on Computer Vision*, pages 137–153. Springer.

Agustsson, E., Tschannen, M., Mentzer, F., Timofte, R., and Gool, L. V. (2019). Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 221–231.

Aljundi, R., Lin, M., Goujaud, B., and Bengio, Y. (2019). Gradient based sample selection for online continual learning. *Advances in Neural Information Processing Systems*, 32.

Amid, E., Anil, R., Kotłowski, W., and Warmuth, M. K. (2022). Learning from randomly initialized neural network features. *arXiv preprint arXiv:2202.06438*.

Amodei, D., Hernandez, D., Sastry, G., Clark, J., Brockman, G., and Sutskever, I. (2018). AI and compute. In *OpenAI Blog*.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433.

Ba, J. and Caruana, R. (2014). Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Baevski, A., Hsu, W.-N., Conneau, A., and Auli, M. (2021). Unsupervised speech recognition. *Advances in Neural Information Processing Systems*, 34:27826–27839.

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

Bengio, Y., Laufer, E., Alain, G., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234. PMLR.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bohdal, O., Yang, Y., and Hospedales, T. (2020). Flexible dataset distillation: Learn labels instead of images. *Neural Information Processing Systems Workshops*.

Borsos, Z., Mutny, M., and Krause, A. (2020). Coresets via bilevel optimization for continual learning and streaming. *Advances in Neural Information Processing Systems*, 33:14879–14890.

Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*.

Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006a). Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–541.

Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006b). Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–541.

Cao, W., Wang, X., Ming, Z., and Gao, J. (2018). A review on neural networks with random weights. *Neurocomputing*, 275:278–287.

Castro, F. M., Marín-Jiménez, M. J., Guil, N., Schmid, C., and Alahari, K. (2018). End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 233–248.

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

Cazenavette, G., Wang, T., Torralba, A., Efros, A. A., and Zhu, J.-Y. (2022b). Wearable imagenet: Synthesizing tileable textures via dataset distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Workshops*.

Chen, H., Xie, W., Vedaldi, A., and Zisserman, A. (2020a). VGGSound: A large-scale audio-visual dataset. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020b). A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607. PMLR.

Chen, X. and He, K. (2020). Exploring simple siamese representation learning. *arXiv preprint arXiv:2011.10566*.

Chen, Y., Welling, M., and Smola, A. (2010). Super-samples from kernel herding. *The Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence*.

Choi, Y., El-Khamy, M., and Lee, J. (2019). Variable rate deep image compression with a conditional autoencoder. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3146–3154.

Coleman, C., Yeh, C., Mussmann, S., Mirzasoleiman, B., Bailis, P., Liang, P., Leskovec, J., and Zaharia, M. (2019). Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2019). Autoaugment: Learning augmentation policies from data. *IEEE Conference on Computer Vision and Pattern Recognition*.

Deco, G. and Brauer, W. (1994). Higher order statistical decorrelation without information loss. *Advances in Neural Information Processing Systems*, 7.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Domke, J. (2012). Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326.

Dong, T., Zhao, B., and Liu, L. (2022). Privacy for free: How does dataset condensation help privacy? In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 5378–5396.

Dong, X., Liu, L., Musial, K., and Gabrys, B. (2021). Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations (ICLR)*.

Elsken, T., Metzen, J. H., Hutter, F., et al. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Farahani, R. Z. and Hekmatfar, M. (2009). *Facility location: concepts, models, algorithms and case studies*. Springer Science & Business Media.

Feldman, D., Schmidt, M., and Sohler, C. (2013). Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In *Proceedings*

of the Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1434–1453. SIAM.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR.

Frey, B. J., Brendan, J. F., and Frey, B. J. (1998). *Graphical models for machine learning and digital communication*. MIT press.

Frey, B. J., Hinton, G. E., and Dayan, P. (1995). Does the wake-sleep algorithm produce good density estimators? *Advances in Neural Information Processing Systems*, 8.

Gidaris, S. and Komodakis, N. (2018). Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375.

Giryes, R., Sapiro, G., and Bronstein, A. M. (2016). Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13):3444–3457.

Goetz, J. and Tewari, A. (2020). Federated learning via synthetic data. *arXiv preprint arXiv:2008.04489*.

Goodfellow, I. (2016). NeurIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773.

Guo, Y., Zhang, L., Hu, Y., He, X., and Gao, J. (2016). Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *European Conference on Computer Vision*, pages 87–102. Springer.

Har-Peled, S. and Mazumdar, S. (2004). On coresets for k-means and k-median clustering. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Hernández-García, A. and König, P. (2018). Data augmentation instead of explicit regularization. *arXiv preprint arXiv:1806.03852*.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hinton, G. E. (2012). A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*, pages 599–619. Springer.

Hospedales, T. M., Antoniou, A., Micaelli, P., and Storkey, A. J. (2021). Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Hu, S., Goetz, J., Malik, K., Zhan, H., Liu, Z., and Liu, Y. (2022a). Fedsynth: Gradient compression via synthetic data in federated learning. *arXiv preprint arXiv:2204.01273*.

Hu, S., Wang, R., Hong, L., Li, Z., Hsieh, C.-J., and Feng, J. (2022b). Generalizing few-shot nas with gradient matching. *arXiv preprint arXiv:2203.15207*.

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167.

Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems*, 31.

Jin, W., Tang, X., Jiang, H., Li, Z., Zhang, D., Tang, J., and Ying, B. (2022a). Condensing graphs via one-step gradient matching. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Jin, W., Zhao, L., Zhang, S., Liu, Y., Tang, J., and Shah, N. (2022b). Graph condensation for graph neural networks. In *International Conference on Learning Representations*.

Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., and Aila, T. (2020). Training generative adversarial networks with limited data. *Neural Information Processing Systems*.

Kaushal, V., Kothawade, S., Ramakrishnan, G., Bilmes, J., and Iyer, R. (2021). Prism: A unified framework of parameterized submodular information measures for targeted data subset selection and summarization. *arXiv preprint arXiv:2103.00128*.

Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., and Iyer, R. (2021a). Glister: Generalization based data subset selection for efficient and robust learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8110–8118.

Killamsetty, K., Zhao, X., Chen, F., and Iyer, R. (2021b). Retrieve: Coreset selection for efficient and robust semi-supervised learning. *arXiv preprint arXiv:2106.07760*.

Kim, J.-H., Kim, J., Oh, S. J., Yun, S., Song, H., Jeong, J., Ha, J.-W., and Song, H. O. (2022). Dataset condensation via efficient synthetic-data parameterization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 11102–11118.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.

Knoblauch, J., Husain, H., and Diethe, T. (2020). Optimal continual learning has perfect memory and is np-hard. In *International Conference on Machine Learning*, pages 5327–5337. PMLR.

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

Kothawade, S., Beck, N., Killamsetty, K., and Iyer, R. (2021). Similar: Submodular information measures based active learning in realistic scenarios. *arXiv preprint arXiv:2107.00717*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. Technical report.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., Duerig, T., and Ferrari, V. (2020). The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *International Journal of Computer Vision*.

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Devlin, J., Lee, K., et al. (2019). Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Le, Y. and Yang, X. (2015). Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3.

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

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605.

Lee, S., Chun, S., Jung, S., Yun, S., and Yoon, S. (2022). Dataset condensation with contrastive signals. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 12352–12364.

Lewis, A. S. and Knowles, G. (1992). Image compression using the 2-d wavelet transform. *IEEE Transactions on image Processing*, 1(2):244–250.

Li, G., Togo, R., Ogawa, T., and Haseyama, M. (2020). Soft-label anonymous gastric x-ray image distillation. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 305–309.

Li, G., Togo, R., Ogawa, T., and Haseyama, M. (2022). Compressed gastric image generation based on soft-label dataset distillation for medical data sharing. *Computer Methods and Programs in Biomedicine*.

Li, L. and Talwalkar, A. (2020). Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR.

Li, Y. and Li, W. (2021). Data distillation for text classification. *arXiv preprint arXiv:2104.08448*.

Li, Y., Swersky, K., and Zemel, R. (2015). Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727. PMLR.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer.

Lopes, R. G., Fenu, S., and Starner, T. (2017). Data-free knowledge distillation for deep neural networks. In *LLD Workshop at Neural Information Processing Systems (NIPS)*.

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

Lyu, L., Yu, H., and Yang, Q. (2020). Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*.

Mahendran, A. and Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5188–5196.

Margatina, K., Vernikos, G., Barrault, L., and Aletras, N. (2021). Active learning by acquiring contrastive examples. *arXiv preprint arXiv:2109.03764*.

Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., and Gool, L. V. (2019). Practical full resolution learned lossless image compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10629–10638.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814.

Nassif, A. B., Shahin, I., Attili, I., Azzeh, M., and Shaalan, K. (2019). Speech recognition using deep neural networks: A systematic review. *IEEE Access*, 7:19143–19165.

Nayak, G. K., Mopuri, K. R., Shaj, V., Radhakrishnan, V. B., and Chakraborty, A. (2019). Zero-shot knowledge distillation in deep networks. In *Proceedings of the 36th International Conference on Machine Learning*.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

Nguyen, T., Chen, Z., and Lee, J. (2021a). Dataset meta-learning from kernel-ridge regression. In *International Conference on Learning Representations*.

Nguyen, T., Novak, R., Xiao, L., and Lee, J. (2021b). Dataset distillation with infinitely wide convolutional networks. *arXiv preprint arXiv:2107.13034*.

Ofgem (2020). Typical domestic consumption values 2020 decision letter.

Parmar, G., Li, D., Lee, K., and Tu, Z. (2021). Dual contradistinctive generative autoencoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 823–832.

Paul, M., Ganguli, S., and Dziugaite, G. K. (2021). Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34:20596–20607.

Prabhu, A., Torr, P. H., and Dokania, P. K. (2020). Gdumb: A simple approach that questions our progress in continual learning. In *European Conference on Computer Vision*, pages 524–540. Springer.

Qiu, Z., Yao, T., Shu, Y., Ngo, C.-W., and Mei, T. (2021). Condensing a sequence to one informative frame for video recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16311–16320.

Rabbani, M. (2002). Book review: JPEG2000: Image compression fundamentals, standards and practice.

Rabbani, M. and Jones, P. W. (1991). *Digital image compression techniques*, volume 7. SPIE press.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Ratner, A. J., Ehrenberg, H. R., Hussain, Z., Dunnmon, J., and Ré, C. (2017). Learning to compose domain-specific transformations for data augmentation. *Advances in Neural Information Processing Systems*, 30:3239.

Ravuri, S. and Vinyals, O. (2019). Seeing is not necessarily believing: Limitations of biggans for data augmentation. *International Conference on Learning Representations Workshops*.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2001–2010.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286. PMLR.

Rifai, S., Bengio, Y., Dauphin, Y., and Vincent, P. (2012). A generative process for sampling contractive auto-encoders. *arXiv preprint arXiv:1206.6434*.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2014). Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*.

Samuel, K. G. and Tappen, M. F. (2009). Learning optimized map estimates in continuously-valued MRF models. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 477–484.

Sariyildiz, M. B. and Cinbis, R. G. (2019). Gradient matching generative networks for zero-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2168–2178.

Saxe, A. M., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. (2011). On random weights and unsupervised feature learning. In *International Conference on Machine Learning*.

Schirrmeister, R. T., Liu, R., Hooker, S., and Ball, T. (2022). When less is more: Simplifying inputs aids neural network understanding. *arXiv preprint arXiv:2201.05610*.

Sener, O. and Savarese, S. (2018). Active learning for convolutional neural networks: A core-set approach.

Shleifer, S. and Prokop, E. (2019). Using small proxy datasets to accelerate hyperparameter search. *arXiv preprint arXiv:1906.04887*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sinha, S., Zhang, H., Goyal, A., Bengio, Y., Larochelle, H., and Odena, A. (2020). Small-gan: Speeding up GAN training using core-sets. In *International Conference on Machine Learning*.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087.

Song, R., Liu, D., Chen, D. Z., Festag, A., Trinitis, C., Schulz, M., and Knoll, A. (2022). Federated learning via decentralized dataset distillation in resource-constrained edge environments. *arXiv preprint arXiv:2208.11311*.

Srivastava, N. and Salakhutdinov, R. R. (2012). Multimodal learning with deep boltzmann machines. *Advances in Neural Information Processing Systems*, 25.

Such, F. P., Rawal, A., Lehman, J., Stanley, K. O., and Clune, J. (2020). Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *International Conference on Machine Learning (ICML)*.

Sucholutsky, I. and Schonlau, M. (2019). Soft-label dataset distillation and text dataset distillation. *arXiv preprint arXiv:1910.02551*.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 843–852.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. (2018). Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Tavakolian, M., Sabokrou, M., and Hadid, A. (2019a). Avd: Adversarial video distillation. *arXiv preprint arXiv:1907.05640*.

Tavakolian, M., Tavakoli, H. R., and Hadid, A. (2019b). Awsd: Adaptive weighted spatiotemporal distillation for video representation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8020–8029.

Toneva, M., Sordoni, A., Combes, R. T. d., Trischler, A., Bengio, Y., and Gordon, G. J. (2019). An empirical study of example forgetting during deep neural network learning.

Tran, N.-T., Tran, V.-H., Nguyen, N.-B., Nguyen, T.-K., and Cheung, N.-M. (2020). On data augmentation for GAN training. *IEEE Transactions on Image Processing*.

Tsilivis, N., Su, J., and Kempe, J. (2022). Can we achieve robustness from data alone? *arXiv preprint arXiv:2207.11727*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*.

Van Den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756. PMLR.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11).

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*.

Wang, K., Zhao, B., Peng, X., Zhu, Z., Yang, S., Wang, S., Huang, G., Bilen, H., Wang, X., and You, Y. (2022). Cafe: Learning to condense dataset by aligning features.

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

Wang, Z. and Ye, J. (2015). Querying discriminative and representative samples for batch mode active learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(3):1–23.

Wei, K., Iyer, R., and Bilmes, J. (2015). Submodularity in data subset selection and active learning. In *International Conference on Machine Learning*. PMLR.

Wiewel, F. and Yang, B. (2021). Condensed composite memory continual learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Wu, C., Herranz, L., Liu, X., van de Weijer, J., Raducanu, B., et al. (2018). Memory replay gans: Learning to generate new categories without forgetting. *Advances in Neural Information Processing Systems*, 31.

Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., and Fu, Y. (2019). Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 374–382.

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

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xie, Q., Luong, M.-T., Hovy, E., and Le, Q. V. (2020). Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698.

Yan, B., Zhou, C., Zhao, B., Guo, K., Yang, J., Li, X., Zhang, M., and Wang, Y. (2020). Augmented bi-path network for few-shot learning. *International Conference on Pattern Recognition (ICPR)*.

Yang, Y., Cheng, X., Bilen, H., and Ji, X. (2022). Learning to annotate part segmentation with gradient matching. In *International Conference on Learning Representations*.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR.

Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer.

Zeiler, M. D., Ranzato, M., Monga, R., Mao, M. Z., Yang, K., Le, Q. V., Nguyen, P., Senior, A. W., Vanhoucke, V., Dean, J., and Hinton, G. E. (2013). On rectified linear units for speech processing. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2018). mixup: Beyond empirical risk minimization. *International Conference on Learning Representations (ICLR)*.

Zhang, Y., Ling, H., Gao, J., Yin, K., Lafleche, J.-F., Barriuso, A., Torralba, A., and Fidler, S. (2021). Datasetgan: Efficient labeled data factory with minimal human effort. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10145–10155.

Zhao, B. and Bilen, H. (2021). Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*.

Zhao, B. and Bilen, H. (2022). Synthesizing informative training samples with GAN. *NeurIPS 2022 Workshops SyntheticData4ML*.

Zhao, B. and Bilen, H. (2023). Dataset condensation with distribution matching. *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*.

Zhao, B., Mopuri, K. R., and Bilen, H. (2020a). idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*.

Zhao, B., Mopuri, K. R., and Bilen, H. (2021). Dataset condensation with gradient matching. In *International Conference on Learning Representations*.

Zhao, S., Liu, Z., Lin, J., Zhu, J.-Y., and Han, S. (2020b). Differentiable augmentation for data-efficient GAN training. *Neural Information Processing Systems*.

Zhao, Z., Zhang, Z., Chen, T., Singh, S., and Zhang, H. (2020c). Image augmentations for GAN training. *arXiv preprint arXiv:2006.02595*.

Zhou, Y., Pu, G., Ma, X., Li, X., and Wu, D. (2020). Distilled one-shot federated learning. *arXiv preprint arXiv:2009.07999*.

Zhu, L., Liu, Z., and Han, S. (2019). Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pages 14747–14756.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710.