Delving into Effective Gradient Matching for Dataset Condensation

Zixuan Jiang, Jiaqi Gu, Mingjie Liu, David Z. Pan
Chandra Family Department of Electrical and Computer Engineering, The University of Texas at Austin
Austin, TX, United States 78703
{zixuan, jqgu, jay_liu}@utexas.edu, dpan@ece.utexas.edu

2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS) | 979-8-3503-4647-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/COINS57856.2023.10189244

***Abstract*—As deep learning models and datasets rapidly scale up, model training is extremely time-consuming and resource-costly. Instead of training on the entire dataset, learning with a small synthetic dataset becomes an efficient solution. Extensive research has been explored in the direction of dataset condensation, among which gradient matching achieves state-of-the-art performance. The gradient matching method directly targets the training dynamics by matching the gradient when training on the original and synthetic datasets. However, there are limited deep investigations into the principle and effectiveness of this method. In this work, we delve into the gradient matching method from a comprehensive perspective and answer the critical questions of what, how, and where to match. We propose to match the multi-level gradients to involve both intra-class and inter-class gradient information. We demonstrate that the distance function should focus on the angle, considering the magnitude simultaneously to delay the overfitting. An overfitting-aware adaptive learning step strategy is also proposed to trim unnecessary optimization steps for algorithmic efficiency improvement. Ablation and comparison experiments demonstrate that our proposed methodology shows superior accuracy, efficiency, and generalization compared to prior work.**

I. Introduction

Large datasets are critical for the success of deep learning at the cost of computation and memory. The high cost is unbearable when we train deep learning models with limited training time or memory budget. For example, quick training is needed when training is a subtask. When we perform neural architecture search [8], hyper-parameter optimization [10, 3], or training algorithm design and validation, we expect to obtain training performance quickly. Another example is the training with limited storage space. To overcome catastrophic forgetting in continual learning, we usually save partial samples for future training [12]. There is a harsh constraint on the memory space when training on edge devices [22]. Above all, it is a critical problem to achieve data efficiency in deep learning training.

As a traditional method to reduce the size of the training dataset, coreset construction defines a criterion for representativeness [9, 16, 5, 1, 17] and then selects samples based on the criterion. Coreset construction is used in many efficient and quick training tasks, e.g., accelerating hyperparameter search [18], continual learning [4]. Unlike the coreset construction method, dataset synthesis generates a small dataset, which is directly optimized for the downstream task. Since it does not rely on representative samples, the dataset synthesis outperforms the coreset construction in the corresponding downstream task.

979-8-3503-4647-3/23/$31.00 ©2023 IEEE

Wang et al. [21] formulate the network parameters as a function of the synthetic training set and formulate the dataset condensation task as a bi-level optimization problem. Specifically, the ultimate target is to train deep learning models on the synthetic training set from scratch such that the trained model can generalize to the original training dataset. The authors minimize the training loss on the original large training data by optimizing the synthetic data. Based on the formulation of the bi-level optimization problem, Sucholutsky and Schonlau [20] extend the method by distilling both input and their soft labels. Such et al. [19] propose to learn a generative teaching network, which generates synthetic data for training student networks. Nguyen et al. [15] use kernel ridge-regression to compress training datasets, enhancing the dataset distillation.

Zhao et al. [26] propose to match gradients w.r.t. parameters when training examples come from synthetic and original datasets, respectively, to solve the bi-level optimization problem. This method mimics the first-order loss landscape when the real training set is used and intuitively maximizes the landscape similarity via gradient matching. By directly targeting the training dynamics, this optimization-aware methodology achieves the current state-of-the-art performance on dataset condensation. However, the previous method does not deeply investigate the working principle in gradient matching, and the current matching flow has limited effectiveness and learning efficiency.

In this paper, we analyze the gradient matching method from a comprehensive perspective, including what, how, and where to match. We enhance the gradient matching algorithm with three essential techniques to achieve higher efficiency and better task-specific performance. We highlight our contributions as follows.

- *Multi-level matching.* We jointly explore intra-class and inter-class gradient matching to improve performance without extra gradient computation.
- *Overfitting delaying.* We propose to adopt a new type of gradient matching function to mitigate the overfitting issue on the synthetic training set to facilitate the optimization. We concentrate on the angle between the gradients, considering the magnitude simultaneously.

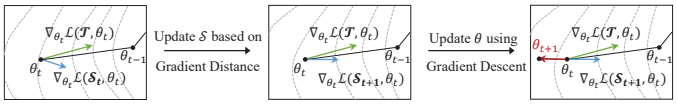

Fig. 1. One simplified inner loop iteration of dataset condensation with gradient matching algorithm. We first update $\mathcal{S}$ to minimize the gradient distance. Then network parameter is updated to imitate the training process.

- *Adaptive learning.* We update synthetic data against the parameter where overfitting happens. Thus, we can achieve the same performance with fewer parameter updates.

II. Background

In this section, we first introduce the background of dataset condensation. Then we describe the working principle of the gradient matching method as we will analyze and extend it in Section III. Similar to the previous work, we take the classification task with balanced class distribution as an example. The algorithm can be easily extended to other problems.

Dataset condensation. Dataset condensation is a task to generate a small synthetic training dataset $\mathcal{S}$ to mimic the model optimization behavior with the original training set $\mathcal{T}$. Specifically, the network parameter θ is formulated as a function of the synthetic training set $\mathcal{S}$ [21].

$$\theta(\mathcal{S}) = \arg\min_{\theta} \mathcal{L}(\mathcal{S}, \theta) \tag{1}$$

$\mathcal{L}(\mathcal{S},\theta) = \frac{1}{|\mathcal{S}|}\sum_{(x,y)\in\mathcal{S}} \ell(f_\theta(x), y)$ is the loss with synthetic dataset $\mathcal{S}$ and model parameter θ. ℓ is the task specific loss function, such as cross entropy loss in the classification task. f_θ represents a deep learning model with parameter θ. Thus, the dataset synthesis problem can be written as the following bi-level optimization problem.

$$\min_{\mathcal{S}} \mathcal{L}\big(\mathcal{T}, \theta(\mathcal{S})\big) \quad \text{s.t. } \theta(\mathcal{S}) = \arg\min_{\theta} \mathcal{L}(\mathcal{S}, \theta) \tag{2}$$

We train a deep learning model f using the synthetic training set $\mathcal{S}$ from scratch and obtain the optimal parameter $\theta(\mathcal{S})$. The objective is to minimize $\mathcal{L}\big(\mathcal{T}, \theta(\mathcal{S})\big)$, the loss on the original large training set $\mathcal{T}$. In other words, the original dataset $\mathcal{T}$ is the test dataset to verify the model $f_{\theta(\mathcal{S})}$.

Gradient matching algorithm. Among all prior work on dataset condensation, the state-of-the-art performance has been achieved by gradient matching [26], which directly encourages the training dynamics on the synthetic set to mimic that on the real training set. The distance between gradients $\nabla_{\theta_t}\mathcal{L}(\mathcal{S}_t, \theta_t)$ and $\nabla_{\theta_t}\mathcal{L}(\mathcal{T}, \theta_t)$ is minimized, where $t = 0, ..., T$ is the time step. If the gradients match, the training trajectories will be the same using gradient-based optimization methods.

Algorithm 1 shows the original gradient matching method. For each iteration of the outer loop, the model parameters θ_0 are initialized following the distribution of P_{θ_0}. Lines 5 $\sim$ 13 correspond one inner loop iteration, which is visualized in Figure 1. The mini-batches are sampled in the same class when calculating the distance between these two gradients $\nabla_{\theta_t}\mathcal{L}(\mathcal{S}_t, \theta_t)$ and $\nabla_{\theta_t}\mathcal{L}(\mathcal{T}, \theta_t)$. Afterwards, the distances for C classes are accumulated to update the synthetic set,

$$\mathcal{S}_{t+1} = \mathcal{S}_t - \eta_{\mathcal{S}} \nabla_{\mathcal{S}_t} D\big(\nabla_{\theta_t}\mathcal{L}(\mathcal{S}_t, \theta_t), \nabla_{\theta_t}\mathcal{L}(\mathcal{T}, \theta_t)\big) \tag{3}$$

where $D(a, b)$ is a function to measure the distance between two tensors, $\eta_{\mathcal{S}}$ is the learning rate.

In Lines 10 $\sim$ 13, the parameter θ_t is updated for ζ_θ times to imitate the training process. The gradient $\nabla_{\theta_t}\mathcal{L}(\mathcal{S}_{t+1}, \theta_t)$ instead of $\nabla_{\theta_t}\mathcal{L}(\mathcal{S}_t, \theta_t)$ or $\nabla_{\theta_t}\mathcal{L}(\mathcal{T}, \theta_t)$ is used to mimic the real training step to update the synthetic set. The red arrow in Figure 1 represents a gradient descent step.

The gradient matching method has several extensions. Zhao and Bilen [25] extend it with differentiable data augmentation. Wiewel and Yang [23] propose to learn a weighted combination of shared components to increase memory efficiency. These extensions are orthogonal to our analysis and enhancements so that our method can be easily integrated into these extensions.

III. Method

In this section, we present our analysis and describe our improvement on the original algorithm. We answer the following questions. *What, how, and where do we match in this gradient matching algorithm?*

A. What we match: multi-level gradient matching

The original algorithm matches gradients of the mini-batch that samples in the same class. Specifically, we sample mini-batches $B_c^{\mathcal{S}} \sim \mathcal{S}$, $B_c^{\mathcal{T}} \sim \mathcal{T}$ in the c-th class and calculate the gradients $g_c^{\mathcal{S}} = \nabla_{\theta_t}\mathcal{L}(B_c^{\mathcal{S}}, \theta_t), g_c^{\mathcal{T}} = \nabla_{\theta_t}\mathcal{L}(B_c^{\mathcal{T}}, \theta_t)$, respectively. We minimize the distance between these intra-class gradients with accumulation.

$$\texttt{loss}_{\texttt{intra}} = \sum_{c=0}^{C-1} D(g_c^{\mathcal{S}}, g_c^{\mathcal{T}}) \tag{4}$$

Therefore, only the intra-class gradients are matched by using these intra-class mini-batches, missing the inter-class gradient information. However, when we use either $\mathcal{S}$ or $\mathcal{T}$ to train the model, we usually use mini-batches that sample across different classes. To mimic the realistic training process, we also need to match the gradients of these inter-class mini-batches. We propose to match the gradients of these inter-class mini-batches in the following efficient way.

Since $\{g_c^{\mathcal{S}}\}_{c=0}^{C-1}$ has already been computed when calculating the intra-gradient distance, we can directly use them to compute the gradients for the mini-batches $\bigcup_{c=0}^{C-1} B_c^{\mathcal{S}}$ as follows.

$$g_{\cup}^{\mathcal{S}} = \nabla_{\theta_t}\mathcal{L}(\bigcup_{c=0}^{C-1} B_c^{\mathcal{S}}, \theta_t) = \frac{\sum_{c=0}^{C-1} |B_c^{\mathcal{S}}| g_c^{\mathcal{S}}}{\sum_{c=0}^{C-1} |B_c^{\mathcal{S}}|} \tag{5}$$

If the mini-batch $B_c^{\mathcal{S}}$ shares the same size, we can further simplify Equation (5) and obtain $g_{\cup}^{\mathcal{S}} = \frac{1}{C}\sum_{c=0}^{C-1} g_c^{\mathcal{S}}$. We can also assign different weights to different mini-batches $B_c^{\mathcal{S}}$ to mimic the original training set $\mathcal{T}$ if the class distribution is not balanced in $\mathcal{T}$. $g_{\cup}^{\mathcal{T}} = \nabla_{\theta_t}\mathcal{L}(\bigcup_{c=0}^{C-1} B_c^{\mathcal{T}}, \theta_t)$ can be computed

Algorithm 1: Gradient matching algorithm. Our proposed methods are highlighted with color.

Input : Original training set $\mathcal{T}$

Initialize $\mathcal{S}_0$ following Gaussian distribution;

for $k = 0, 1, ..., K-1$ **do** // outer loop: explore with different initialization

  Initialize model parameters $\theta_0 \sim P_{\theta_0}$;

  for $t = 0, 1, ..., T-1$ **do** // inner loop

    for $c = 0, 1, ..., C-1$ **do**

      Sample mini-batches $B_c^{\mathcal{S}_t} \sim \mathcal{S}_t$, $B_c^{\mathcal{T}} \sim \mathcal{T}$ in the c-th class;

      Compute gradients $g_c^{\mathcal{S}_t} = \nabla_{\theta_t}\mathcal{L}(B_c^{\mathcal{S}_t}, \theta_t)$, $g_c^{\mathcal{T}} = \nabla_{\theta_t}\mathcal{L}(B_c^{\mathcal{T}}, \theta_t)$;

    $\texttt{loss} = \sum_{c=0}^{C-1} \mathcal{D}(g_c^{\mathcal{S}_t}, g_c^{\mathcal{T}}) + \lambda\mathcal{D}\big(\frac{1}{C}\sum_{c=0}^{C-1} g_c^{\mathcal{S}_t}, \frac{1}{C}\sum_{c=0}^{C-1} g_c^{\mathcal{T}}\big)$;

    $\mathcal{S}_{t+1} = \mathcal{S}_t - \eta_{\mathcal{S}}\nabla_{\mathcal{S}_t}\texttt{loss}$;

    $\theta_t^0 = \theta_t$;

    for $i = 0, 1, ..., \zeta_\theta(t) - 1$ **do** // update θ with adaptive learning steps

      $\theta_t^{i+1} = \theta_t^i - \eta_\theta \nabla_{\theta_t^i}\mathcal{L}(\mathcal{S}_{t+1}, \theta_t^i)$;

    $\theta_{t+1} = \theta_t^{i+1}$;

Output: Synthetic training set $\mathcal{S}$

in the same way. In this way, we do not perform extra forward and backward computations to calculate gradients for the inter-class mini-batches $\bigcup_{c=0}^{C-1} B_c^{\mathcal{S}}$ and $\bigcup_{c=0}^{C-1} B_c^{\mathcal{T}}$.

With these inter-class gradients, we add a new term in the gradient matching loss as shown in Equation (6).

$$\underbrace{\sum_{c=0}^{C-1} D(g_c^{\mathcal{S}_t}, g_c^{\mathcal{T}})}_{\texttt{loss}_{\texttt{intra}}} + \lambda \underbrace{D\big(\frac{1}{C}\sum_{c=0}^{C-1} g_c^{\mathcal{S}_t}, \frac{1}{C}\sum_{c=0}^{C-1} g_c^{\mathcal{T}}\big)}_{\texttt{loss}_{\texttt{inter}}} \tag{6}$$

The first term is the intra-class gradient matching loss $\texttt{loss}_{\texttt{intra}}$, which is used in the original method. We add a new term of inter-class gradient matching loss, with λ being the weight to balance these two terms. In this multi-level gradient matching loss, we consider both the intra-class and inter-class information. Through experiments, we find that the multi-level gradient matching has better performance than either the intra-class or inter-class counterpart. Experimental results are shown in Section IV-B.

B. How we match: angle and magnitude

In the original gradient matching algorithm [26], the authors propose to decompose the matching loss layer by layer

$$\mathcal{D}\big(\nabla_\theta\mathcal{L}(\mathcal{S}, \theta), \nabla_\theta\mathcal{L}(\mathcal{T}, \theta)\big) = \sum_{l=1}^{L} d\big(\nabla_{\theta^l}\mathcal{L}(\mathcal{S}, \theta), \nabla_{\theta^l}\mathcal{L}(\mathcal{T}, \theta)\big) \tag{7}$$

where L is the number of layers. For each layer, negative cosine similarity is used as the distance between two tensors,

$$d(A, B) = \sum_{i=1}^{\texttt{out}} \big(1 - \frac{A_i \cdot B_i}{\|A_i\|\|B_i\|}\big), \tag{8}$$

where $\texttt{out}$ is the number of output channels. For example, the weights and the corresponding gradients of a 2D convolution layer has the shape of (out, in/groups, h, w) [1]. We reshape the gradients as (out, in/groups × h × w) and compute the cosine similarity for each output channels. This distance function considers the layer-wise structure and output channels, enabling a single learning rate across all layers. By maximizing the cosine similarity between gradients, the $\mathcal{S}$ is expected to lead the parameter in a correct direction.

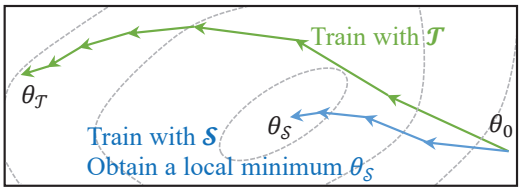

Fig. 2. Optimization trajectories of training from scratch using $\mathcal{S}$ and $\mathcal{T}$.

However, in this distance matching loss, only the angle between gradients is considered, with the magnitude ignored. This is a critical issue when we train a network f from scratch for evaluation using the resultant $\mathcal{S}$. Figure 2 visualizes the training process using $\mathcal{S}$ and $\mathcal{T}$ respectively. Since $|\mathcal{S}|$ is usually very small, it is a severe challenge that the deep learning model can easily remember the samples, which induces overfitting and a bad generalization. The norm of gradient $\|\nabla_\theta\mathcal{L}(\mathcal{S}, \theta)\|$ degrades quickly during the training process. In few gradient descent steps, we will be stuck in a local minimum, where $\nabla_\theta\mathcal{L}(\mathcal{S}, \theta) = 0$.

It is meaningless to match the angle when either of the gradient norm are small. The right direction cannot help us escape the local minimum. Thus, we have to consider the magnitude of the gradient vectors in this distance function. For example, we can consider the Euclidean distance between two vectors A_i and B_i,

$$d(A, B) = \sum_{i=1}^{\texttt{out}} \big(1 - \frac{A_i \cdot B_i}{\|A_i\|\|B_i\|} + \|A_i - B_i\|\big) \tag{9}$$

We also try other distance functions that consider the magnitude and show the results in Section IV-C.

[1] out and in are the number of output channels and input channels. groups is the number of blocked connections from input channels to output channels. h and w are kernel height and width, respectively.

C. Where we match: adaptive learning steps

In the t-th iteration of the inner loop, θ_t is used to update $\mathcal{S}_t$, and $\mathcal{S}_{t+1}$ is used to update θ_t. Here comes the question in this sequential update. When we update $\mathcal{S}_t$, how many gradient descent steps do we perform such that $\mathcal{S}_{t+1}$ is a *good* training set for θ_t? Similarly, how many gradient descent steps do we conduct when we update θ_t such that θ_{t+1} is a *good* point for $\mathcal{S}_{t+1}$?

In the original algorithm, the authors provide their answers by setting the number of gradient descent steps $\zeta_{\mathcal{S}}$, ζ_θ empirically. Namely, we have the following update flows.

$$\mathcal{S}_t = \mathcal{S}_t^0 \rightarrow \mathcal{S}_t^1 \rightarrow \mathcal{S}_t^2 \rightarrow ... \rightarrow \mathcal{S}_t^{\zeta_{\mathcal{S}}} = \mathcal{S}_{t+1} \tag{10}$$

$$\theta_t = \theta_t^0 \rightarrow \theta_t^1 \rightarrow \theta_t^2 \rightarrow ... \rightarrow \theta_t^{\zeta_\theta} = \theta_{t+1} \tag{11}$$

We present our understanding of these two hyperparameters. A change on $\mathcal{S}$ may induce a non-negligible update in the gradient $\nabla_\theta \mathcal{L}(\mathcal{S}, \theta)$, which is large enough for a update in the parameter. Moreover, $\mathcal{S}$ should not be updated many times at one parameter θ to avoid overfitting. Hence, the original setting of $\zeta_{\mathcal{S}} = 1$ is a good choice.

Updating θ_t is an imitation of the training process. The synthetic dataset $\mathcal{S}$ is the training dataset, while the original training dataset $\mathcal{T}$ serves as the validation dataset. In particular, after one update from θ_t^i to θ_t^{i+1}, we usually have a smaller training loss $\mathcal{L}(\mathcal{S}_{t+1}, \theta_t^i) > \mathcal{L}(\mathcal{S}_{t+1}, \theta_t^{i+1})$. However, it is unknown how the validation loss change. The relationship between $\mathcal{L}(\mathcal{T}, \theta_t^i)$ and $\mathcal{L}(\mathcal{T}, \theta_t^{i+1})$ is uncertain. We can check if there exists overfitting after updating θ_t. The naive method of detecting overfitting is making comparison between $\mathcal{L}(\mathcal{T}, \theta_t^i)$ and $\mathcal{L}(\mathcal{T}, \theta_t^{i+1})$. We can also check the gap between two loss terms $\mathcal{L}(\mathcal{T}, \theta_t^{i+1}) - \mathcal{L}(\mathcal{S}_{t+1}, \theta_t^{i+1})$ to decide whether if overfitting happens.

Ideally, we should update network parameters θ until overfitting happens. If there is no overfitting, the $\mathcal{S}$ leads the parameter as $\mathcal{T}$ does, which means $\mathcal{S}$ is a good approximation of $\mathcal{T}$ in the perspective of gradient matching. We do not need to update $\mathcal{S}$ in this case. If overfitting happens, the $\mathcal{S}$ needs to be updated against the current parameter since the $\mathcal{S}$ has divergence from $\mathcal{T}$. Hence, it is better to use dynamic and adaptive learning steps, which help us locate where we need to update $\mathcal{S}$ and improve the algorithm efficiency.

Nevertheless, there is an overhead to detect the overfitting in real implementation. For instance, we have to compute the loss term $\mathcal{L}(\mathcal{T}, \theta_t^{i+1})$ as the extra computation. Therefore, we propose to run preliminary experiments and find when overfitting usually happens. With the preliminary results, we make a schedule for ζ_θ. In other words, let ζ_θ be a function of the index of the current inner loop t and we define this function from preliminary results to avoid the extra computation on overfitting detection. This $\zeta_\theta(t)$ could help us locate where overfitting happens approximately. Figure 3 shows this improvement.

D. Improved gradient matching flow

Algorithm 1 shows our improvement on the gradient matching algorithm, with changes highlighted. We match the multi-level gradients to consider both intra-class and inter-class information without extra gradient computation, as shown in Line 8. We apply the new distance function, which considers magnitude, to avoid small gradients, which delays the overfitting. We use a dynamic number of steps in Line 11 to improve the algorithm efficiency. Ideally, the θ should be updated until overfitting happens where $\mathcal{S}$ diverges from $\mathcal{T}$. We use a schedule $\zeta_\theta(t)$ to help us approximate when overfitting occurs.

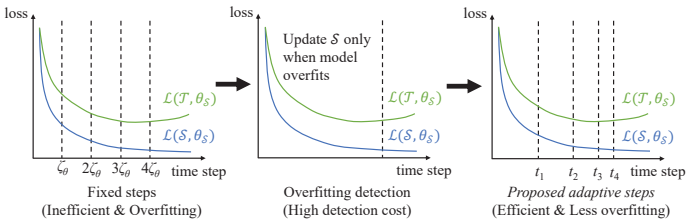

Fig. 3. *Left*. The original method updates $\mathcal{S}$ after updating network parameter θ for a fixed number of steps ζ_θ [26]. *Middle*. Ideally, $\mathcal{S}$ should be updated right after the model overfits. *Right*. To avoid overhead on detecting overfitting, we propose to use adaptive learning steps $\zeta_\theta(t)$.

TABLE I
TEST ACCURACY (%) WITH MATCHING DIFFERENT GRADIENTS. NEW DISTANCE FUNCTION AND ADAPTIVE LEARNING STEPS ARE DISABLED.

Dataset	#Image/Class	intra-class	inter-class	interleaved	multi-level
MNIST	1	**91.7±0.5**	88.8±0.7	**91.7±0.4**	90.9±0.5
	10	97.4±0.2	96.9±0.1	97.1±0.1	**97.6±0.1**
FashionMNIST	1	70.5±0.6	70.2±0.7	**70.6±0.6**	**70.6±0.7**
	10	82.3±0.4	82.4±0.3	83.1±0.3	**84.4±0.3**
SVHN	1	31.2±1.4	29.8±0.7	30.8±1.6	**32.9±1.2**
	10	**76.1±0.6**	72.7±1.0	75.4±0.7	75.5±0.7
CIFAR-10	1	28.3±0.5	**29.7±0.7**	28.6±0.7	**29.7±0.7**
	10	44.9±0.5	46.7±0.5	45.9±0.6	**48.6±0.5**

IV. EXPERIMENTS

We show an ablation study on our proposed techniques to validate their superiority to other variants and compare the test accuracy with prior arts.

A. Settings

We follow the same settings with the original work for all the experiments. Specifically, we use the same network architectures, datasets, and hyperparameters. The difference is the improvement highlighted in Algorithm 1. We use five image classification datasets, MNIST [7], FashionMNIST [24], SVHN [14], CIFAR-10 [13] and CIFAR-100. These datasets have a balanced class distribution. There are 100 classes in the CIFAR-100 dataset, while other datasets have 10 classes.

We refer to the original work [26] and implementation [2] for more details regarding experimental settings. The results of the baseline method are from the original paper.

There are two phases in every single experiment. We first use our algorithm to obtain a small synthetic training set $\mathcal{S}$ with a *source* model. In the second phase, we train a *target* model with $\mathcal{S}$ from scratch and test the trained model on the original testing dataset. For every experiment, we generate 2 sets of synthetic images and train 50 target networks. The average and standard deviation of the test accuracy over these 100 evaluations are reported.

[2]Link to the implementation

TABLE II
TEST ACCURACY (%) WITH DIFFERENT DISTANCE FUNCTIONS.

Distance Function	Test Accuracy
$d_1(a,b) = 1 - (a \cdot b)/(\|a\|\|b\|)$	32.9±1.2
$d_2(a,b) = \|a-b\|$	23.6±2.1
$d_3(a,b) = \|a-b\|^2$	24.3±1.8
$100d_4(a,b) = 100d_3(a,b)/len(a)$	23.5±1.4
$d_1 + d_2$	**34.5±1.9**
$d_1 + d_3$	34.1±2.0
$d_1 + 100d_4$	34.0±1.4

B. Different gradient matching methods

Our first improvement is to match the multi-level gradients, as discussed in Section III-A. To demonstrate the efficacy of our proposed method, we make comparisons on four settings: (1) intra-class gradient matching, (2) inter-class gradient matching, (3) matching these two gradients in an interleaved way[3], and (4) multi-level gradient matching. We use the same hyperparameters in these experiments, with λ being the number of classes C, disabling the new distance function and adaptive learning steps. Table I lists the results on four datasets. In most cases, the multi-level gradient matching achieves the best results. Focusing on either intra-class gradient matching or inter-class gradient matching misses the other information. Compared with minimizing the two matching losses in an interleaved way, the multi-level gradient matching is much more stable.

C. Different distance functions

Our second improvement is to use a new distance function. We make comparisons on the following distance functions with multi-level gradients enabled. (1) Negative cosine similarity $d_1(a,b) = 1 - (a \cdot b)/(\|a\|\|b\|)$, (2) Euclidean distance $d_2(a,b) = \|a-b\|$, (3) sum of the squared error $d_3(a,b) = \|a-b\|^2$, (4) mean squared error $d_4(a,b) = d_3(a,b)/len(a)$, Note that the d_1 focuses on the angle only, while the other functions consider angle and magnitude simultaneously.

We take the SVHN dataset with 1 image per class as an example. Table II lists the test accuracy when using these distance functions to generate synthetic sets. We directly assign the same weight to these distance functions except that we set the weight of 100 for d_4.

We find that when using only one of these distance functions, the test accuracy with d_1 is the highest. Our explanation is that the angle of the gradient is much more important than the magnitude when (stochastic) gradient descent and its variants are used. However, when we combine d_1 and other magnitude-related distance functions, we get improvement compared with pure d_1. Namely, we concentrate on the gradient direction while considering the magnitude to avoid being stuck in traps where the gradient norm is small.

D. Adaptive learning steps

Ideally, we should update $\mathcal{S}$ when it is no longer a good approximation of $\mathcal{T}$ in terms of gradient matching. Thus, a criterion to detect overfitting is needed. We try the naive overfitting criterion $\mathcal{L}(\mathcal{T}, \theta_t^i) < \mathcal{L}(\mathcal{T}, \theta_t^{i+1})$. In other words, if the validation loss increases, we will stop the parameter update and proceed to update $\mathcal{S}$. With this setting, we have improved the test accuracy from 44.9% to 45.7% for 10 images per class of CIFAR-10. However, we notice that this improvement is at the cost of overfitting detection, which is nontrivial in real implementation. Therefore, we define a pre-defined schedule $\zeta_\theta(t)$ by observing when overfitting happens in the CIFAR-10 experiment above.

$$\zeta_\theta(t) = \begin{cases} 50 - 10t, & t < 4 \\ 10, & 4 \le t < 10 \\ 5, & t \ge 10 \end{cases} \tag{12}$$

The reason $\zeta_\theta(t)$ is non-increasing is that we may encounter overfitting issues more frequently as training proceeds. Hence, we need to update $\mathcal{S}$ at shorter intervals.

With this schedule, we can first proceed to where overfitting happens and then stay in this area. Another advantage of this schedule is that it reduces the number of model parameter updates. In the baseline method, the authors set $T = 1, 10, 50, \zeta_\theta = 1, 50, 10$ for synthesizing $1, 10, 50$ images per class. Taking $T = 10$ as an example, the original algorithm updates θ 450 times in one iteration of the outer loop, while we only update it 190 times.

E. Comparison with prior work

In Table III, we perform an ablation study on our methods with four settings to demonstrate the effectiveness of our proposed enhancement. We name the four settings as *Ours-M*, *Ours-MD*, *Ours-MDO*, and *Ours-MDA*, where M, D, O, A stand for **m**ulti-level gradient matching, new **d**istance function, updating θ until **o**verfitting, and **a**daptive steps, respectively. We also add two coreset selection methods for comparison. *Random* means that samples are randomly selected as the coreset. *Herding* [6, 2] selects samples whose center is close to the distribution center. For a fair comparison, we evaluate our method on the same ConvNet model [11] as used in the original work [26]. Both the source network and the target network are the ConvNet model.

For our method, we use the settings mentioned above. We match the multi-level gradients as shown in Equation (6) with $\lambda = C$, the number of classes. We use the distance in Equation (9), the overfitting criterion $\mathcal{L}(\mathcal{T}, \theta_t^i) < \mathcal{L}(\mathcal{T}, \theta_t^{i+1})$, and the adaptive learning step in Equation (12). [4] We use the same settings for all the benchmarks without further tuning [5]. It is expected that we can achieve better results with better hyperparameters tuned for each benchmark. For instance, we can tune the distance function and the learning steps $\zeta_\theta(t)$.

Since the algorithm only runs a single inner loop, i.e., $T = 1$, when the condensed dataset contains one image per

[3]We match intra-class gradients in one iteration, and match inter-class gradients in the next iteration.

[4]Equation (12) is from the observation on a CIFAR-10 experiment and we generalize the setting to other benchmarks.

[5]An exception is that we use $d = d_1 + 0.1d_2 = 1 - (a \cdot b)/(\|a\|\|b\|) + 0.1\|a-b\|$ for 50 images per class with CIFAR-10.

TABLE III

ABLATION STUDY IN TERMS OF THE TEST ACCURACY (%). IPC IS THE NUMBER OF IMAGE PER CLASS IN $\mathcal{S}$. *Random* MEANS THAT SAMPLES ARE RANDOMLY SELECTED AS THE CORESET. *Herding* SELECTS SAMPLES WHOSE CENTER IS CLOSE TO THE DISTRIBUTION CENTER. *DC baseline* REFERS TO THE ORIGINAL WORK ON DATASET CONDENSATION. M, D, O, A REPRESENT MULTI-LEVEL GRADIENT MATCHING, NEW DISTANCE FUNCTION, UPDATING θ UNTIL OVERFITTING, AND ADAPTIVE STEPS, RESPECTIVELY. O AND A ARE NOT APPLICABLE WHEN IPC IS 1.

Dataset	IPC	Random	Herding	DC Baseline	Ours-M	Ours-MD	Ours-MDO	Ours-MDA	Whole Training Set
MNIST	1	64.9±3.5	89.2±1.6	91.7±0.5	90.9±0.5	**91.9±0.4**	-	-	99.6±0.0
	10	95.1±0.9	93.7±0.3	97.4±0.2	97.6±0.1	**97.9±0.1**	**97.9±0.1**	**97.9±0.2**	
	50	97.9±0.2	94.8±0.2	**98.8±0.2**	98.0±0.1	98.6±0.1	98.6±0.1	98.5±0.1	
FashionMNIST	1	51.4±3.8	67.0±1.9	70.5±0.6	70.6±0.7	**71.4±0.6**	-	-	93.5±0.1
	10	73.8±0.7	71.1±0.7	82.3±0.4	84.4±0.3	**85.4±0.3**	84.6±0.3	84.2±0.3	
	50	82.5±0.7	71.9±0.8	83.6±0.4	87.8±0.2	87.4±0.2	**87.9±0.2**	**87.9±0.2**	
SVHN	1	14.6±1.6	20.9±1.3	31.2±1.4	32.9±1.2	**34.5±1.9**	-	-	95.4±0.1
	10	35.1±4.1	50.5±3.3	76.1±0.6	75.5±0.7	75.9±0.7	**76.2±0.7**	75.9±0.7	
	50	70.9±0.9	72.6±0.8	82.3±0.3	82.2±0.2	82.9±0.2	**83.8±0.3**	83.2±0.3	
CIFAR-10	1	14.4±2.0	21.5±1.2	28.3±0.5	29.5±0.7	**30.0±0.6**	-	-	84.8±0.1
	10	26.0±1.2	31.6±0.7	44.9±0.5	48.6±0.5	49.5±0.5	49.9±0.6	**50.2±0.6**	
	50	43.4±1.0	40.4±0.6	53.9±0.5	58.5±0.5	58.6±0.4	**60.0±0.4**	58.3±0.5	
CIFAR-100	1	4.2±0.3	8.4±0.3	**12.8±0.3**	12.4±0.3	12.7±0.4	-	-	56.2±0.3
	10	14.6±0.5	17.3±0.3	25.2±0.3	30.8±0.3	28.0±0.4	29.5±0.3	**31.1±0.3**	

class, the method of adaptive step has no impact in this setting. In terms of test accuracy, our proposed multi-level gradient matching and angle-magnitude distance function outperform the baseline gradient matching method [26] in most benchmarks. Our proposed adaptive learning step technique is a good approximation of overfitting detector since the results of *Ours-MDO* and *Ours-MDA* are similar. With *Ours-MDA*, we cut down unnecessary steps in the later optimization stage, leading to higher learning efficiency while maintaining our advantages in test accuracy.

The usage of multi-level gradients and new distance functions introduces less than 1% extra computation time. The adaptive learning step can reduce the computation time by $25\% \sim 30\%$ for the experiments with 10 images per class.

V. CONCLUSION

In this paper, we present our analysis of the gradient matching method for the dataset condensation problem. Based on our analysis, we further extend the original algorithm. We provide our answers to the question of *what, how, and where we match in this gradient matching algorithm.* We match the multi-level gradients to involve both intra-class and inter-class gradient information. A new distance function is proposed to mitigate the overfitting issue. We use adaptive learning steps to improve algorithm efficiency. The effectiveness and efficiency of our proposed improvements are shown in the experiments.

REFERENCES

[1] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *arXiv preprint arXiv:1903.08671*, 2019.

[2] E. Belouadah and A. Popescu. Scail: Classifier weights scaling for class incremental learning. In *WACV*, Los Alamitos, CA, USA, mar 2020. IEEE Computer Society.

[3] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*, pages 115–123. PMLR, 2013.

[4] Zalán Borsos, Mojmir Mutny, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. In *NeurIPS*, 2020.

[5] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *ECCV*, 2018.

[6] Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. In *UAI*, 2010.

[7] Yann Le Cun, John S. Denker, and Sara A. Solla. *Optimal Brain Damage*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1558601007.

[8] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

[9] Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *SODA*, 2013.

[10] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated machine learning*, pages 3–33. Springer, Cham, 2019.

[11] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *CVPR*, 2018.

[12] James Kirkpatrick et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

[13] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[14] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[15] Timothy Nguyen, Zhourung Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. *arXiv preprint arXiv:2011.00050*, 2020.

[16] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental classifier and representation learning. In *CVPR*, pages 2001–2010, 2017.

[17] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[18] Sam Shleifer and Eric Prokop. Using small proxy datasets to accelerate hyperparameter search. *arXiv preprint arXiv:1906.04887*, 2019.

[19] Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth Stanley, and Jeffrey Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *ICML*, 2020.

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

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

[22] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020.

[23] Felix Wiewel and Bin Yang. Condensed composite memory continual learning. *arXiv preprint arXiv:2102.09890*, 2021.

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

[25] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *ICML*, 2021.

[26] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. *ICLR*, 2021.