Learning to Generate Synthetic Training Data Using Gradient Matching and Implicit Differentiation

Dmitry Medvedev(✉) and Alexander D'yakonov

Lomonosov Moscow State University, Moscow, Russia
dm.medvedev97@gmail.com

Abstract. Using huge training datasets can be costly and inconvenient. This article explores various data distillation techniques that can reduce the amount of data required to successfully train deep networks. Inspired by recent ideas, we suggest new data distillation techniques based on generative teaching networks, gradient matching, and the Implicit Function Theorem. Experiments with the MNIST image classification problem show that the new methods are computationally more efficient than previous ones and allow to increase the performance of models trained on distilled data.

Keywords: Data distillation · Gradient matching · Implicit differentiation · Generative teaching network

1 Introduction

In machine learning, the purpose of data distillation [1] is to compress the original dataset while maintaining the performance of the models trained on it. The generalizability of the dataset is also needed. By this we mean the ability to train models of architectures that were not involved in the distillation process. Since training with less data is usually faster, distillation can be useful in practice. For example, it can be used to speed up a neural architecture search (NAS) task. Acceleration is achieved through the faster training of candidates. In many recent works [1,3,5–7], distillation is formulated as an optimization problem with the objects of a new dataset as parameters for optimization. Therefore, to distill the dataset for an image classification task, pixels of images have to be optimized. First, all new objects are initialized with random noise, then these objects are used to train a student (i.e., a randomly selected network). Then the student misclassification loss is calculated on real data. Finally, a gradient descent step is used to update the synthetic objects. Gradients can be calculated by backpropagating the error through the entire student's learning process. The step of this procedure can be very time-consuming and memory-intensive, so there is a need for an alternative. In [2], the authors use the Implicit Function

E. Burnaev et al. (Eds.): AIST 2021, CCIS 1573, pp. 138–150, 2022.
https://doi.org/10.1007/978-3-031-15168-2_12

Theorem to solve the memory consumption problem. In [3], the data distillation problem has been reformulated to use gradient matching loss and speed up the optimization of synthetic objects and reduce memory usage. There is an alternative to optimizing the pixels of synthetic data. In [4], the authors suggest to optimize parameters of the generator model (a generative teaching network or GTN) to produce synthetic data from noise and labels. The disadvantage is that the authors used backpropagation through the learning process for optimization. Inspired by recent ideas in the field of data distillation, we propose replacing it with gradient matching or with implicit differentiation to make the procedure less computationally expensive. We have found that this allows not only to reduce memory costs but also to create more efficient and generalizable datasets. In addition, we investigate the use of augmentation in the distillation procedure and in models' learning on distilled data.

The paper is divided into 7 sections. We first analyse the first data distillation algorithm [1] and discuss its problems in Sect. 2. A brief description of the algorithms for implicit differentiation [2] and gradient matching [3] can be found in Sects. 3 and 4. Section 5 presents the generative teaching network architecture that we use in our work. Section 6 contains the results of experiments with the MNIST image classification benchmark. In Sect. 6.1 we compare the results of all the described distillation methods, limiting the distillation time to a constant. In Sects. 6.2 and 6.3 we show results of new distillation techniques when training a generator with gradient matching and implicit differentiation, respectively. In Sect. 6.4 we study the use of augmentation by distillation, and in Sect. 6.5 we check the generalization of the data obtained with the new methods. Finally, we present our findings in Sect. 7. The code can be found on our GitHub page.[1]

2 Backpropagation Through the Student's Learning Process

Let λ be teacher parameters. These can be either GTN network's parameters, or synthetic objects' parameters (e.g. pixels of synthetic images). To update λ, we must first train the student network θ on synthetic data, minimizing the task specific loss $\mathcal{L}_{\mathcal{S}}$ (e.g. cross-entropy), and then get the loss on real data $\mathcal{L}_{\mathcal{T}}$. To take care of generalizability, student's initialization goes from preset distribution $p(\theta_0)$. Afterall, the optimization problem for λ can be formulated as follows:

$$\lambda^* := \operatorname*{argmin}_{\lambda}\ \mathbb{E}_{\theta_0\sim p(\theta_0)}\mathcal{L}^*_{\mathcal{T}}, \text{ where} \tag{1}$$
$$\mathcal{L}^*_{\mathcal{T}} :=\mathcal{L}_{\mathcal{T}}(\theta^*(\lambda)), \qquad \theta^*(\lambda) := \operatorname*{argmin}_{\theta}\ \mathcal{L}_{\mathcal{S}}(\lambda,\theta).$$

To resolve the first problem (1) we can calculate gradient of $\mathcal{L}_{\mathcal{T}}$ with respect to λ to do the gradient descent step. In this work, we use cross-entropy loss as

[1] https://github.com/dm-medvedev/EfficientDistillation.

$\mathcal{L}_\mathcal{T}$ and there is an explicit dependence only on θ and parameters of real data, so $\frac{\partial \mathcal{L}_\mathcal{T}}{\partial \lambda} = 0$ and $\frac{\partial \mathcal{L}_\mathcal{T}^*}{\partial \lambda} = \frac{\partial \mathcal{L}_\mathcal{T}}{\partial \theta} \frac{\partial \theta^*}{\partial \lambda}$. Thus, the main part is the calculation of $\frac{\partial \theta^*}{\partial \lambda}$. Where the dependence of θ^* on λ comes from a student's training procedure. The first distillation algorithm was suggested in [1] and it is based on the assumption that the student's learning procedure is differentiable. This means that we can backpropogate gradient through it. We will denote it as **unroll**. This algorithm can be implemented using the Higher library [10]. It allows to backpropogate through many optimizers, in our paper we use SGD with momentum [8]. This distillation method is both time and space consuming. To perform a single step of updating λ it is necessary to perform N student optimization steps, while all intermediate results (copies of the student weights) must be stored in memory. There is also a problem with the generalization of resulting synthetic dataset, the performance of models whose architectures were not involved in the distillation process is much lower. This negative effect can be mitigated by sampling the initialization and student architecture.

Note that the procedure of student's training on the resulting synthetic dataset can be carried out in different ways. New data, parameterized with λ, can be used as a single large batch or it can be split into several smaller ones. This split can be useful to reduce memory consumption per training step. Instead of random sampling of distilled objects, the authors of the original work propose to attach each of them to a specific batch. These batches would have the same order in each epoch. In our paper, we use the same schemes. Let ic (input count) be the number of batches of the synthetic dataset, note that it must be divisor of N. In our experiments we try limit values $ic = 1$ and $ic = 10$.

3 Implicit Differentiation

This method suggested in [2] is based on the Implicit Function Theorem:

Theorem 1 (Cauchy, Implicit Function Theorem). *Let* $\frac{\partial \mathcal{L}_\mathcal{S}}{\partial \theta}(\lambda, \theta) : \varLambda \times \varTheta \to \varTheta$*, be a continuously differentiable function. Fix a point* (λ', θ') *with* $\frac{\partial \mathcal{L}_\mathcal{S}}{\partial \theta}(\lambda', \theta') = 0$*. If the Jacobian matrix* $\frac{\partial^2 \mathcal{L}_\mathcal{S}}{\partial \theta^2}$ *is invertible, then there exists an open set* $U \subseteq \varLambda$ *containing* λ' *such that there exists a unique continuously differentiable function* $\theta^* : U \to \varTheta$*, such that* $\theta^*(\lambda') = \theta'$ *and* $\forall \lambda \in U$, $\frac{\partial \mathcal{L}_\mathcal{S}}{\partial \theta}(\lambda, \theta^*(\lambda)) = 0$*. Moreover, the partial derivatives of* θ^* *in* U *are given by the matrix product:*

$$\frac{\partial \theta^*}{\partial \lambda}(\lambda) = -\left[\frac{\partial^2 \mathcal{L}_\mathcal{S}}{\partial \theta^2}(\lambda, \theta^*(\lambda))\right]^{-1} \frac{\partial^2 \mathcal{L}_\mathcal{S}}{\partial \theta \partial \lambda}(\lambda, \theta^*(\lambda)). \tag{2}$$

So, if there was an efficient way to invert the matrix, we would simply have used (2), after the student θ has reached a local minimum, assuming $\frac{\partial \mathcal{L}_\mathcal{S}}{\partial \theta}(\lambda, \theta^*(\lambda)) \approx 0$. But the inversion operation is time costly, so the authors used the approximation by the Neumann series taking the first few elements and controlling convergence with a hyperparameter α (see (3)).

The resulting algorithm (see Algorithm 1) has no problems with memory consumption since there is no need to store copies of the student θ. And, despite the several subsequent approximations, the experimental results show that the method has a competitive performance (see Table 4). Note that **grad** in Algorithm 1 denotes the dot product between the Jacobian of the given function (**func**) at the given point (**wrt**) and a vector (**vec**). Another interesting detail of this method is that there is no dependence on which optimizer is used to train the student, and on the order (curriculum) of batches of synthetic data. So, in our paper we only use a single large batch of synthetic data. The original work [2] lacks a detailed description of the experimental results, so it can be found in our paper (see Sect. 6.3). We used the open-source code[2] as the basis for the implementing the method.

$$\left[\frac{\partial^2 \mathcal{L}_{\mathcal{S}}}{\partial \theta^2}(\lambda, \theta^*(\lambda))\right]^{-1} \approx \alpha \sum_{j=0}^{N} \left[I - \alpha \frac{\partial^2 \mathcal{L}_{\mathcal{S}}}{\partial \theta^2}(\lambda, \theta^*(\lambda))\right]^{j}. \tag{3}$$

Algorithm 1. Distillation with implicit differentiation.

```
1: Input: teacher's parameters $\lambda$, student's initialization distribution $p(\theta_0)$, the number of distillation epochs $K$, the number of student's learning steps $\zeta_\theta$, real data $\mathcal{T}$, learning rate $\eta$.
2: for $k = 1, ..., K$ do
3:     $\mathcal{B}^{\mathcal{T}} \sim \mathcal{T}, \quad \theta \sim p(\theta_0)$
4:     for $n = 1, ..., \zeta_\theta$ do
5:         $\theta \mathrel{-}= \eta \frac{\partial \mathcal{L}_{\mathcal{S}}(\lambda, \theta)}{\partial \theta}$
6:     $\mathcal{L}_{\mathcal{T}} = ClassificationLoss(\mathcal{B}^{\mathcal{T}}, \theta)$
7:     $v = \frac{\partial \mathcal{L}_{\mathcal{T}}}{\partial \theta}; \qquad p = v$
8:     for $j = 1, ..., N$ do                          ▷ $N$ — number of elements in (3)
9:         $v \mathrel{-}= \alpha \cdot \mathbf{grad}\big(\mathbf{func} = \frac{\partial \mathcal{L}_{\mathcal{S}}}{\partial \theta}, \mathbf{wrt} = \theta, \mathbf{vec} = v\big)$
10:        $p \mathrel{+}= v$
11:    $\nabla_\lambda \mathcal{L}_{\mathcal{T}} = -\alpha \cdot \mathbf{grad}\big(\mathbf{func} = \frac{\partial \mathcal{L}_{\mathcal{S}}}{\partial \theta}, \mathbf{wrt} = \lambda, \mathbf{vec} = p\big)$
12:    Update($\lambda, \nabla_\lambda \mathcal{L}_{\mathcal{T}}$)               ▷ update with any optimizer
   return $\lambda$
```

4 Gradient Matching

The gradient matching method (**GM**) was proposed in [3], and it solves a different problem than the general one (1). The main difference is that we want not only to train the student θ to achieve a good performance on real data but also to get such a solution as if it was trained on real data. To formulate this let $D(\nabla_\theta \mathcal{L}_{\mathcal{S}}, \nabla_\theta \mathcal{L}_{\mathcal{T}})$ be the function of how close one tensor is to another.

[2] https://github.com/AvivNavon/AuxiLearn.

The distance function D is just the sum (in our paper for GTN experiments we used the mean) of the cosine distance functions for each student layer θ^l. Let A and B be gradient tensors with respect to layer parameters. Let i be the index of the output axis (e.g. for a convolutional layer this is the index of the output channel). A_i and B_i are flat gradient vectors corresponding to each output element indexed by i. The most interesting detail here is that the authors [3] suggest to update λ after each step of student optimization, so now we don't need to wait until it reaches a local minimum, as it was before. The authors also propose not to store student copies and to minimize $D\big(\nabla_\theta \mathcal{L}_\mathcal{S}(\lambda, \theta_{t-1}), \nabla_\theta \mathcal{L}_\mathcal{T}(\theta_{t-1})\big)$ for each step separately. So there is no backpropagation through opt_θ. Both of these proposals make the gradient matching method very computational effective.

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}}\, \mathbb{E}_{\theta_0 \sim P_{\theta_0}} \Big[\sum_{n=1}^{N-1} D\big(\nabla_\theta \mathcal{L}_\mathcal{S}(\lambda, \theta_n), \nabla_\theta \mathcal{L}_\mathcal{T}(\theta_n)\big)\Big], \quad \text{where:} \tag{4}$$

$$D(\nabla_\theta \mathcal{L}_\mathcal{S}, \nabla_\theta \mathcal{L}_\mathcal{T}) = \sum_{l=1}^{L} d(\nabla_{\theta^l} \mathcal{L}_\mathcal{S}, \nabla_{\theta^l} \mathcal{L}_\mathcal{T}), \quad d(A, B) = \sum_{i=1}^{\dim(\mathrm{A})} \left(1 - \frac{A_i \cdot B_i}{\|A_i\| \|B_i\|}\right)$$

Algorithm 2. Gradient matching.

```
1: Input: teacher's parameters $\lambda$ and synthetic objects $\mathcal{S}(\lambda)$, student's initialization
   distribution $p(\theta_0)$, the number of distillation epochs $K$, the number of student's
   learning steps $\zeta_\theta$, real data $\mathcal{T}$, learning rate $\eta_\theta$, the number of inner loop steps $N$.
2: for $k = 0, ..., K-1$ do
3:     $\theta_0 \sim p_{\theta_0}$
4:     for $n = 0, ..., N-1$ do
5:         $\mathcal{B}^\mathcal{T} \sim \mathcal{T}, \quad \mathcal{B}^\mathcal{S} \sim \mathcal{S}(\lambda)$
6:         $\mathcal{L}_\mathcal{T} = ClassificationLoss(\mathcal{B}^\mathcal{T}, \theta_n), \quad \mathcal{L}_\mathcal{S} = ClassificationLoss(\mathcal{B}^\mathcal{S}, \theta_n)$
7:         $\mathcal{L}(\lambda) = D(\nabla_\theta \mathcal{L}_\mathcal{S}(\lambda, \theta_n), \nabla_\theta \mathcal{L}_\mathcal{T}(\theta_n))$
8:         Update$(\lambda, \nabla_\lambda \mathcal{L}(\lambda))$
9:         $\theta_{n+1} \leftarrow opt_\theta(\mathcal{L}_\mathcal{S}(\lambda, \theta_n), \zeta_\theta, \eta_\theta)$
10: Output: $\lambda$
```

The peculiarity of this loss function is that the gradient of one synthetic object depends on other objects from the same batch, because of a normalization operation in the d equation (4). It makes the optimization problem harder and can cause negative effects (see Table 2). So authors decided to distill objects separately for each class. Note that the gradient matching is independent of the student training optimization algorithm. There is only one assumption that the direction should be based on the gradient. Another aspect is that the curriculum (the order of the synthetic batches in the student's learning procedure) can be learned with this distillation method. We used an open-source code[3] as the implementation of this method.

5 Generative Teaching Network

The idea first appeared in [4], where the authors suggested to use the generator as the teacher λ. The input of the generator is a concatenation of noise and one hot encoded label (for conditional generation). In the original paper, the authors use backpropagation through the student's learning process to train the generator, which is inconvenient for practical use due to high memory consumption, so in our paper, we show that the same or even better results can be achieved more efficiently by using gradient matching or implicit differentiation. Experimental results in [4] show that using a generator can help to improve students' performance. In our paper, we check if we can improve distillation performance using larger generators. Note that the size of the generator in our experiments is controlled by the k hyperparameter (see Fig. 1). The generator consists of two linear layers and two convolutional layers. The output size of the first layer is k. And $\lfloor k/2 \rfloor \times$ width $\times$ height of picture is the output size of the second layer. $\lfloor k/4 \rfloor$ is the number of output channels of the first convolution. Hereinafter, unless otherwise indicated, we use the following notation: **DD** (Data Distillation) is a distillation, when the parameters of the teacher λ are pixels of synthetic images, and **GTN** is a distillation using a generator. Note that the generator has two modes: **GTN-rnd** is a generator with random noise as input, (**GTN-lrn**) is a generator with a learned input.

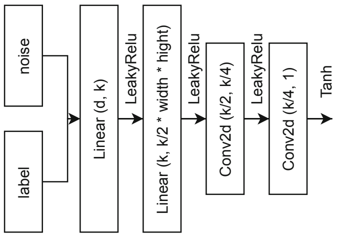

Fig. 1. Generator's architecture; k is a hyperparameter to control network's size, $d = 64$ is a generator's input.

6 Experiments

6.1 Distillation with Time Limit

The neural architecture search (NAS) is one of the most promising areas for distillation and it is important to note that the time spent on distillation should

[3] https://github.com/VICO-UoE/DatasetCondensation.

be added to the time spent on the NAS, this idea was also mentioned in the review[4] of [4]. So, in this section, we check the performance of all known distillation methods. We think that it is fair to distill the data by all methods for the same limited time. We have chosen a time limit of ≈15 min, and it is based on common sense and the time spent on the NAS in similar experiments [3]. Note that this limit may not be accurate, as the distillation takes an integer number of steps, where each step takes a non-deterministic time. To check the performance we use the following scheme. First we train teacher λ with three restarts. The number of steps is determined by the time limit indicated above. Then, to get the final results we train five randomly initialized students θ for each of the three teachers. Each student's training takes 1000 optimization steps. In our work we use the MNIST [9] benchmark and make the same preparations as in [4]. We extract part of the training data for validation (10 thousand images) and use it to get the best teacher hyperparameters. We use $|\mathcal{B}^{\mathcal{T}}| = 256$ batch size of training data. For the most of our experiments we use ConvNet [12] as a student. As student's optimizer we use SGD with momentum with the same parameters as suggested in [3]. We use the same teacher optimizers as in the original papers [1,3,4]. The volume of synthetic data can be controlled by the *ipc* (images per class) parameter. For each table in this paper, the largest numbers in the column are shown in bold.

Table 1. The mean and standard deviation of test accuracy for different distillation algorithms.

Method + Teacher	Accuracy	Params	GPU (MiB)
GM + DD ($K = 60, \zeta_\theta = 50$)	$\mathbf{94.9 \pm 0.1}$	78.4 K	≈2390
Unroll + DD ($ic = 1$)	88.4 ± 0.3	78.4 K	≈4432
Unroll + DD ($ic = 10$)	79.2 ± 0.7	784 K	≈4426
Unroll + GTN-lrn ($ic = 1$)	92.0 ± 0.3	1.646 M	**≈4480**
Unroll + GTN-lrn	91.6 ± 0.5 ($ic = 10$)	**1.704** M	**≈4480**
Unroll + GTN-rnd	91.7 ± 0.3	1.640 M	**≈4480**

Table 1 shows the mean and standard deviation of test accuracy, reached by students trained on distilled data. Note that there is only one difference from previous works: we use time limit for each distillation procedure, so there is a degradation in performance. For this experiment, we use $K = 1000, N = 10$ as default hyperparameters values. To check the memory consumption we use a special tool,[5] which can measure the GPU memory usage. Note that using of the **unroll** distillation procedure consumes memory the most. The third column shows the number of teacher parameters, and although **GTN** ($k = 64$) is twice as large as **DD**, there is not much difference in memory usage.

[4] https://openreview.net/forum?id=HJg_ECEKDr.
[5] https://pytorch.org/docs/stable/cuda.html#torch.cuda.max_memory_reserved.

6.2 Training Generator with Gradient Matching

In this section we explore the use of the gradient matching to train the teacher generator. We first check the hyperparameters for this distillation method. N controls the frequency of the student's reinitialization, ζ_θ controls the speed at which the teacher's parameters are updated. Figure 2 (a–d) shows the non-trivial relationship between performance and the hyperparameter choice. We assume that such a dependence can be caused by the time limit and the fact that increasing the values of these hyperparameters may cause longer convergence. Note that in previous works [1,3,4] where no time limit was used, increasing *ipc* always resulted in better performance.

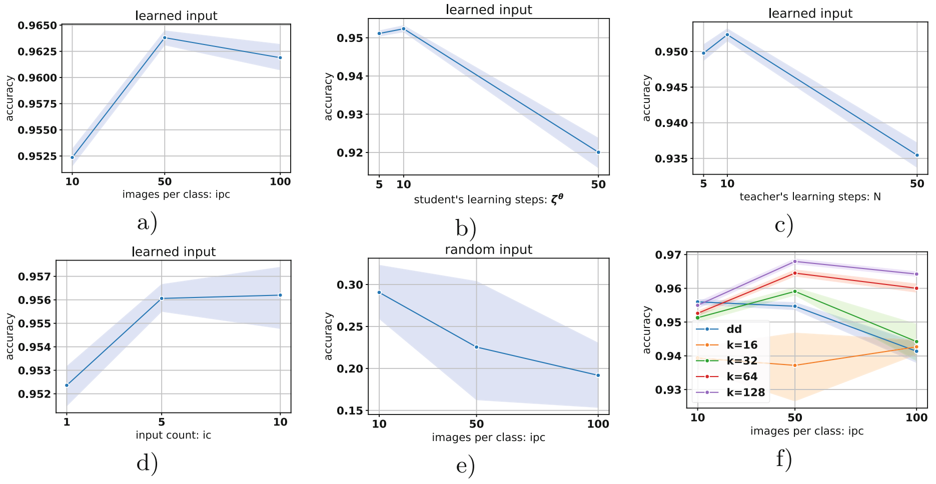

Fig. 2. Dependence of student's performance and hyperparameters of distillation procedure. Next parameters used as default: $ipc = 10, ic = 1, N = 10, \zeta_\theta = 10, k = 64$.

Table 2. Mean and standard deviation of test accuracy for different distillation algorithms.

Method + Teacher	Accuracy	Params	GPU (MiB)
GM + DD	$\mathbf{95.6 \pm 0.1}$	78.4 K	≈2390
GM + DD (not per class)	86.9 ± 1.5	78.4 K	≈2370
GM + GTN-lrn	$\mathbf{95.2 \pm 0.1}$	1.646 M	≈2454
GM + GTN-lrn (not per class)	93.4 ± 0.3	1.646 M	≈2434

Figure 2 (e) shows that the fixation of the generator input is really important for gradient matching distillation because teacher training (optimization of λ) diverges when using random input. Another important aspect mentioned above

Table 3. Mean and standard deviation of test accuracy for different distillation algorithms.

Method + Teacher	Accuracy	Params	GPU (MiB)
GM + GTN-lrn ($k = 16, ipc = 100$)	94.2 ± 0.4	172.2 K	**≈4192**
GM + GTN-lrn ($k = 32, K = 50$)	95.9 ± 0.2	449.7 K	≈3610
GM + GTN-lrn ($K = 50$)	96.4 ± 0.1	1.672 M	≈3640
GM + GTN-lrn ($k = 128, K = 50$)	$\mathbf{96.8 \pm 0.1}$	**6.533** M	≈3770
GM + GTN-rnd ($ipc = 10, K = 110$)	29.0 ± 6.1	1.640 M	≈2454

is that the gradient must be calculated per class. Table 2 shows the results for per class case and not. It seems that per class distillation gives significantly better results. Figure 2 (f) shows the accuracy achieved with data distilled with generators of different sizes (marked with different k), and without a generator (**DD**). This plot depicts the dependency between the number of synthetic images per class (ipc) and student's performance on a test set. It seems that the correct size selection for the generator allows to get a better performance. More detailed results can be found in Tables 2 and 3. For experiment in Table 2, we use $ipc = 10$, $ic = 1$, $N = 10$, $K = 110$, $\zeta_\theta = 10$ and $k = 64$ for GTN as default hyperparameters values. For experiment in Table 3, we use $k = 64, ipc = 50, K = 35, N = 10$, and $\zeta_\theta = 10$. Tables 2 and 3 show the GPU memory usage. It seems that ipc has a greater impact on memory usage than k, which is another benefit of using **GTN**. Note that the memory usage can be reduced by changing the ic value to optimize more synthetic images using smaller batches. Note that such a change can slow down the convergence.

6.3 Distillation with Implicit Differentiation

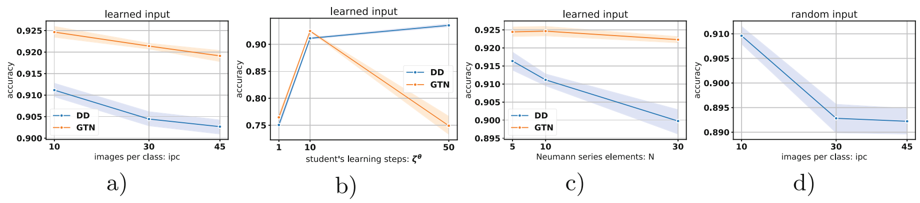

Fig. 3. The relation of the distillation method's hyperparameters and test performance. We use as default: $ipc = 10, N = 10, \zeta_\theta = 10$, and $k = 64$.

The method was proposed in [2], and we will abbreviate it as **IFT** (Implicit Function Theorem). As mentioned above (see Sect. 3), there is no detailed description of the results in the original paper, so they can be found in this section. Figure 3 (a–c) shows the relationship between the hyperparameters of the distillation method and the student's performance on the test. We assume that these results can be explained by the fact that increasing the values of these hyperparameters decreases the frequency of λ update, which negatively affects the performance. The only exception is ζ_θ.

Figure 3 (d) shows results for distillation using a generator with the random input (**GTN-rnd**). Such a generator can produce as much data as we need, but it can not converge when trained with gradient matching. It seems that such distillation becomes possible using implicit differentiation.

Table 4 shows the best results for each method. For this experiment, we use $K = 1080, \zeta_\theta = 50, ipc = 10,$ and $N = 10$ as default hyperparameters values. The performance seems to be the same or even better compared to backpropagation through the training procedure **unroll** (see Table 1). Note the difference in memory usage in both tables. Also note that the implicit differentiation distillation is inferior to the gradient matching distillation.

We think this may be connected with the difference in the frequency of λ update. To do one update using **IFT**, we first have to train the student, which is not needed in case of **GM**. It is also important to note that this method is very sensitive to α and ζ_θ, and in some **DD** cases it starts to diverge after several iterations. Meanwhile the use of **GTN** makes the procedure more stable and allows for a more generalizable dataset (see Table 6).

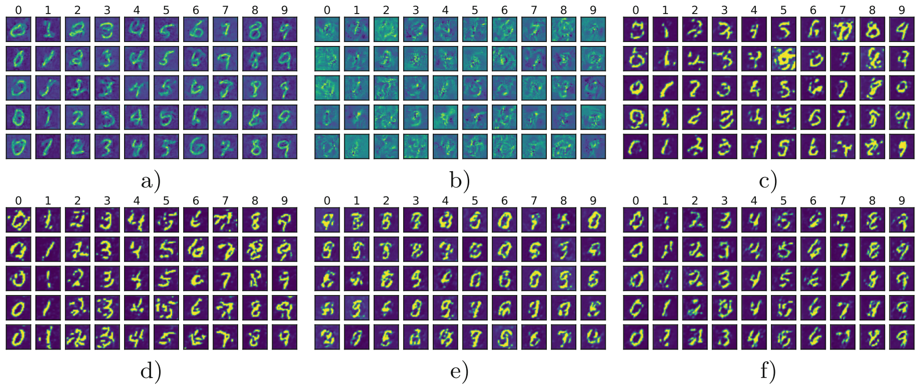

Fig. 4. Synthetic images for MNIST classification task obtained with different distillation methods: a) GM+DD, b) IFT+DD, c) GM+GTN-lrn, d) IFT+GTN-lrn, e) GM+GTN-rnd, f) IFT+GTN-rnd. We use the same hyperparameters as mentioned in Table 5. Hyperparameters for GM+GTN-rnd are described in caption of Table 4.

Table 4. Mean and standard deviation of test accuracy for different distillation algorithms.

Method + Teacher	Accuracy	Params	GPU (MiB)
IFT + DD ($K = 500$)	$\mathbf{93.5 \pm 0.5}$	78.4 K	≈2726
IFT + GTN-lrn ($\zeta_\theta = 10$)	92.4 ± 0.2	**1.646** M	≈2726
IFT + GTN-rnd ($\zeta_\theta = 10$)	90.9 ± 0.3	1.640 M	≈2726

Figure 4 shows part of the final synthetic dataset for **GM** (see a, c and e) and **IFT** (see b, d and f). The greatest difference is obtained when data distilled without a generator (see a, b). Synthetic data obtained using implicit differentiation looks less realistic and therefore can be used for federative learning [13]. Also note that the images distilled using a generator are more contrast.

6.4 Distillation with Augmentation

In previous works, augmentation has been used in different ways. In [4] it takes place during distillation (let's call it train augmentation) by applying transformations to real images $\mathcal{B}^{\mathcal{T}}$. In [1,3] it is used when teaching student on synthetic data (let's call it test augmentation). In our study, we decided to compare augmentation techniques. Table 5 shows the test performance for various distillation and augmentation techniques. It seems that for the MNIST classification problem only test augmentation gives improvements (see Tables 2, 3, 4). To augment images we use random crop and rotation. For this experiment, we use $K = 1080, ipc = 10,\ \zeta_\theta = 10,$ and $N = 10$ as default hyperparameters values.

Table 5. The mean and standard deviation of test accuracy for different distillation algorithms and different augmentations.

Method + Teacher	Test Aug.	Train Aug.	Test + Train Aug.
GM+DD ($ic = 1, K = 110$)	96.1 ± 0.4	94.8 ± 0.1	93.9 ± 0.5
GM+GTN-lrn ($k = 128, ipc = 50, K = 50$)	$\mathbf{97.4 \pm 0.1}$	$\mathbf{96.2 \pm 0.2}$	$\mathbf{95.5 \pm 0.4}$
IFT+DD ($\zeta_\theta = 50, K = 500$)	92.3 ± 0.9	91.4 ± 0.5	89.2 ± 1.5
IFT+GTN-lrn	93.0 ± 0.2	91.4 ± 0.3	91.4 ± 0.4
IFT+GTN-rnd	92.2 ± 0.3	89.7 ± 0.3	90.9 ± 0.6

6.5 Generalizability

The generalization problem of distilled data was first mentioned in [1] and then studied in [4] and [7].

Table 6. The mean and standard deviation of test accuracy for different distillation algorithms and student's architectures.

Method + Teacher	LeNet	AlexNet	VGG11	MLP
GM+DD	94.1 ± 0.6	95.0 ± 0.2	95.8 ± 0.3	$\mathbf{88.6 \pm 0.4}$
GM+GTN-lrn	$\mathbf{95.5 \pm 0.3}$	$\mathbf{96.7 \pm 0.2}$	$\mathbf{97.4 \pm 0.1}$	86.8 ± 0.3
IFT+DD	74.0 ± 7.8	68.6 ± 8.9	86.5 ± 1.6	50.9 ± 8.3
IFT+GTN-lrn	91.5 ± 1.0	82.5 ± 14.9	93.0 ± 0.4	79.9 ± 0.6
IFT+GTN-rnd	88.3 ± 2.3	85.3 ± 3.9	92.1 ± 0.4	74.4 ± 1.1

The problem is that such data can't guarantee convergence for students which didn't participate in the distillation procedure. And this problem is of great importance, since the main practical use of synthetic data is the NAS. For this experiment, we use $K = 1080, ipc = 10,\ \zeta_\theta = 10,$ and $N = 10$ as default hyperparameters values. Table 6 shows the results of students with different architectures trained on data distilled with different methods. For distillation we used ConvNet student's architecture, all results were obtained with test augmentation. It seems that the best generalizability can be obtained using **GTN** and **GM**. For a comparison with ConvNet see the second column of Table 5.

7 Conclusion

This work explores all the latest ideas in dataset distillation field suggested in [1–4]. We honestly compared the performance of all known methods, limiting their running time. We also proposed new methods based on the joint use of generators and memory efficient methods. Experiments with the MNIST benchmark show that selecting the correct size for the generator allows to achieve better performance for gradient matching distillation, and improves the generalizability of implicit differentiation distillation. This paper also presents the results of augmentation impact on distillation. We also provide a detailed description of the experimental results for implicit differentiation distillation, as we could not find them in the original work [2]. As future work, we would like to experiment with much more diverse datasets and architectures. We also want to improve the distilled data generalizing ability using stochastic depth networks [11]. We are also interested in experiments with bringing the distribution of synthetic objects closer to the original one.

Acknowledgments. This research has been supported by the Interdisciplinary Scientific and Educational School of Moscow University "Brain, Cognitive Systems, Artificial Intelligence".

References

1. Wang, T., Zhu, J., Torralba, A., Efros, A.A.: Dataset distillation. CoRR arXiv:1811.10959 (2018)

2. Lorraine, J., Vicol, P., Duvenaud, D.: Optimizing millions of hyperparameters by implicit differentiation. CoRR arXiv:1911.02590 (2019)
3. Zhao, B., Mopuri, K.R., Bilen, H.: Dataset condensation with gradient matching. CoRR arXiv:2006.05929 (2020)
4. Such, F.P., Rawal, A., Lehman, J., Stanley, K.O., Clune, J.: Generative teaching networks: accelerating neural architecture search by learning to generate synthetic training data. CoRR arXiv:1912.07768 (2019)
5. Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based hyperparameter optimization through reversible learning. CoRR arXiv:1502.03492 (2015)
6. Sucholutsky, I., Schonlau, M.: Soft-label dataset distillation and text dataset distillation. CoRR arXiv:1910.02551 (2019)
7. Medvedev, D., D'yakonov, A.: New properties of the data distillation method when working with tabular data. In: van der Aalst, W.M.P., et al. (eds.) AIST 2020. LNCS, vol. 12602, pp. 379–390. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72610-2_29
8. Polyak, B.: Some methods of speeding up the convergence of iteration methods. USSR Comput. Math. Math. Phys. **4**, 1–17 (1964)
9. MNIST Handwritten Digit Database. http://yann.lecun.com/exdb/mnist/. Accessed 17 Apr 2021
10. Grefenstette, E., et al.: Generalized inner loop meta-learning. CoRR arXiv:1910.01727 (2019)
11. Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger K.: Deep networks with stochastic depth. CoRR arXiv:1603.09382 (2016)
12. Gidaris, S., Komodakis, N.: Dynamic few-shot visual learning without forgetting. CoRR arXiv:1804.09458 (2018)
13. Zhou, Y., Pu, G., Ma, X., Li, X., Wu, D.: Distilled one-shot federated learning. CoRR arXiv:2009.07999 (2020)